



PX4 Development Guide



制作组: 无人机组

制作人: 张晓辉

刘晓楠

廖鑫淼

制作日期: 2018年3月7日

声 明

本文件资料制作不易, 仅限无人机组内部使用, 禁止外传, 或上传网络共享, 以及做其它商业用途!

CONTENTS

1.	introduction	1
	Forums and Chat.....	1
	Contributing	1
	Translations	2
	Licence.....	2
2.	Getting Started	3
2.1	Initial Configuration & Setup	3
	Basic Equipment.....	3
	Vehicle Configuration	3
2.2	Installing Files and Code	4
	Supported Targets	4
	Development Environment	4
2.2.1	Development Environment on Mac	5
	Homebrew Installation.....	5
	Common Tools.....	5
	Ground Control Software	5
	Editor / IDE	6
	Next Steps.....	6
2.2.2	Development Environment on Linux	6
	Development Toolchain.....	7
	Pixhawk/NuttX (and jMAVSim).....	7
	Snapdragon Flight or Raspberry Pi	7
	Parrot Bepop	8
	jMAVSim/Gazebo Simulation	8
	Gazebo with ROS.....	8
	Ground Control Software	8
	Editor / IDE	9
	Next Steps.....	9
2.2.3	Development Environment on Ubuntu LTS / Debian Linux	9
	Convenience Bash Scripts	10
	How to use the scripts.....	10
	Permission Setup	11
	Remove the modemmanager.....	11
	Common Dependencies	11
	Ninja Build System	11
2.2.4	Fast RTPS installation	12
	Simulation Dependencies	13
	jMAVSim	13
	Gazebo	13
	ROS/Gazebo	13

NuttX-based Hardware	15
Snapdragon Flight	16
(Cross) Toolchain installation	16
Sysroot Installation	16
Update ADSP firmware	17
References	17
Raspberry Pi Hardware	17
clang	17
Parrot Bebop.....	18
Ground Control Software	18
Editor / IDE	19
Next Steps.....	19
Development Environment on CentOS.....	19
Common Dependencies	19
GCC Toolchain Installation.....	19
Ninja Build System	20
Development Environment on ArchLinux	21
Permissions	21
Script-based Installation	21
Manual Installation	21
Common Dependencies	21
GCCE Compiler	22
Advanced Linux Installation Use-Cases	23
Using JTAG Programming Adapters.....	23
Windows Installation Instructions	23
Toolchain Installation	24
Native toolchain.....	24
Bash on Windows (NEW!)	24
Ground Control Software	26
Editor / IDE	26
Next Steps.....	26
2.3 Building PX4 Software	26
Downloading PX4 Source Code	26
First Build (Using the jMAVSim Simulator)	28
NuttX / Pixhawk Based Boards	30
Building	30
Uploading Firmware (Flashing the board).....	30
Other Boards	31
Raspberry Pi 2/3 Boards	31
Parrot Bebop	32
OcPoC-Zynq Mini.....	34
QuRT / Snapdragon Based Boards	34
Compiling in a Graphical IDE.....	35
Qt Creator Functionality	35

Qt Creator on Linux.....	36
Qt Creator on Windows	36
Qt Creator on Mac OS	36
2.4 First App Tutorial (Hello Sky).....	37
Prerequisites	37
Minimal Application	37
Build the Application/Firmware.....	40
Test App (Hardware)	40
Upload the firmware to your board	40
Connect the Console	41
Test App (SITL).....	42
Subscribing to Sensor Data.....	42
Testing the uORB Subscription	43
Publishing Data	43
Full Example Code	44
Running the Complete Example.....	47
Wrap-Up	47
2.5 Advanced Configuration.....	47
Setting Configuration Parameters	47
2.6 Contributing	48
Support.....	48
Weekly Dev Call.....	48
Tests Flights.....	49
Have a problem?	50
Calendar & Events	50
Contribution	51
Code of Conduct.....	51
Source Code Management	51
Code Style Formatting	51
Commits and Commit Messages	52
2.6.1GIT Examples	52
Contributing code to PX4	52
Update Submodule	55
Do a PR for a submodule update.....	55
Checkout pull requests.....	55
Common pitfalls	56
Force push to forked repository.....	56
Rebase merge conflicts	56
Pull merge conflicts	56
2.6.2Contributing to Documentation.....	56
Quick Changes.....	57
Adding New Content - Big Changes.....	57
What Goes Where?	57
Gitbook Documentation Toolchain	57

Style guide	58
Translations	58
Starting a new language translation	59
Updating translations	60
Tracking changes	60
Licence.....	60
2.7 Terminology.....	61
Notation.....	61
Acronyms	61
Symbols	62
Subscripts / Indices	64
Superscripts / Indices	64
Decorators.....	65
2.8 Licenses	65
3. Concepts.....	66
3.1 PX4 Architectural Overview	66
High-Level Software Architecture	66
Flight Stack.....	67
Middleware	68
Update Rates.....	68
Runtime Environment.....	68
Background Tasks	69
OS-Specific Information	69
Controller Diagrams.....	69
Multicopter Position Controller	70
Fixed-Wing Attitude Controller	70
3.2 Dronecode Platform Hardware/Software Architecture	71
3.3 Flight Modes	72
Flight Mode Summary	73
Manual flight modes	73
Assisted flight modes	74
Auto flight modes.....	74
Flight Mode Evaluation Diagram.....	76
3.4 Mixing and Actuators.....	77
Control Pipeline.....	77
Control Groups.....	77
Virtual Control Groups	79
Mapping	80
PX4 mixer definitions.....	80
Syntax	80
3.5 PWM_limit State Machine	82
Quick Summary.....	82
State Transition Diagram	84
4. Simulation	85

Supported Simulators	85
Simulator MAVLink API.....	86
Default PX4 MAVLink UDP Ports.....	88
SITL Simulation Environment	88
Starting/Building SITL Simulation	89
Init File Location.....	90
Example Startup File.....	91
HITL Simulation Environment	92
Joystick/Gamepad Integration	92
Camera Simulation.....	93
4.1 jMAVSim with SITL	93
Simulation Environment	94
Running SITL	94
Important Files	94
Taking it to the Sky	95
Simulating a Wifi Drone	95
Extending and Customizing	96
Interfacing to ROS	96
4.2 Gazebo Simulation.....	96
Installation	97
Running the Simulation	97
Quadrotor	97
Quadrotor with Optical Flow	97
3DR Solo.....	97
Standard VTOL	98
Tailsitter VTOL	99
Ackerman vehicle (UGV/Rover)	100
HippoCampus TUHH (UUV: Unmanned Underwater Vehicle)	100
Change World.....	101
Taking it to the Sky	101
Usage/Configuration Options	102
Set Custom Takeoff Location	102
Using a Joystick.....	102
Simulating GPS Noise	103
Starting Gazebo and PX4 Separately	103
Video Streaming	104
Prerequisites	104
Enable GStreamer Plugin	104
How to View Gazebo Video	105
Gazebo GUI to Start/Stop Video Streaming	105
Extending and Customizing	106
Interfacing to ROS	106
Further Information.....	107
4.3 AirSim Simulation	107

4.4	Multi-Vehicle Simulation	107
	Required	107
	Build and test	107
	What's happening?	108
	Additional Resources.....	110
4.5	Hardware in the Loop Simulation (HITL)	110
	HITL-Compatible Airframes.....	110
	HITL Simulation Environment	111
	JMAVSim/Gazebo HITL Environment.....	111
	X-Plane HITL Environment.....	111
	HITL vs SITL	112
	Setting up HITL.....	112
	PX4 Configuration	112
	Simulator Setup.....	114
	Fly an Autonomous Mission in HITL	118
5.	Hardware	119
5.1	PX4 Reference Flight Controller Design	119
	Binary Compatibility.....	119
	Reference Design Generations.....	119
5.2	Flight Controller Porting Guide	120
	Architecture	120
	NuttX Boards	120
	NuttX Menuconfig	120
	QuRT / Hexagon	120
	Linux Boards	121
	Related Information	121
5.3	Airframes	121
	Airframes Reference.....	122
	Copter.....	122
	Coaxial Helicopter	122
	Quadrotor asymmetric	129
	Adding a New Airframe Configuration	138
	Configuration File Overview	139
	Config File	139
	Mixer File	141
	Tuning Gains.....	143
	Getting a New Airframe to Show in QGroundControl	143
5.4	Driver Development.....	144
	Creating a Driver.....	144
	Core Architecture.....	144
	Device IDs	144
	Decoding example	145
	Device ID Encoding	145
5.5	Telemetry Radios/Modems	146

Supported Radio Systems	146
Integrating Telemetry Systems	147
SiK Radio	147
Build Instructions	147
Mac OS	147
Configuration Instructions	148
5.6 Sensor and Actuator I/O	148
I2C Bus Overview	148
Integrating I2C Devices	149
I2C Driver Examples.....	149
Further Information.....	149
UAVCAN Introduction	149
Initial Setup	150
Wiring.....	150
Firmware Setup.....	151
Upgrading Node Firmware	151
Enumerating and Configuring Motor Controllers.....	151
Useful links.....	151
UAVCAN Bootloader Installation	151
Overview	151
Prerequisites	152
Device Preparation	152
Installation	152
BlackMagic Probe	152
ST-Link v2.....	153
Segger J-Link Debugger.....	153
Erasing Flash with SEGGER JLink Debugger	154
UAVCAN Firmware Upgrading	154
Vectorcontrol ESC Codebase (Pixhawk ESC 1.6 and S2740VC).....	154
Flashing the UAVCAN Bootloader.....	154
Compiling the Main Binary	154
Sapog Codebase (Pixhawk ESC 1.4 and Zubax Orel 20).....	155
Flashing the UAVCAN Bootloader.....	155
Compiling the Main Binary.....	155
Zubax GNSS	155
Firmware Installation on the Autopilot	156
Placing the binaries in the PX4 ROMFS	156
Starting the Firmware Upgrade process	157
UAVCAN Enumeration and Configuration.....	157
Various Notes	158
Arm but motors not spinning	158
Debugging with Zubax Babel	158
5.7 RTK GPS (Background)	158
Overview	158

Supported RTK GPS modules.....	159
Automatic Configuration	159
RTCM messages.....	159
Uplink datarate.....	160
5.8 Camera Trigger.....	161
Trigger Modes/Configuration.....	161
Trigger Modes	161
Trigger Hardware Configuration	161
Trigger Interface Backends	162
Other Parameters	162
Trigger Command Interface.....	163
MAV_CMD_DO_TRIGGER_CONTROL	163
MAV_CMD_DO_DIGICAM_CONTROL	163
MAV_CMD_DO_SET_CAM_TRIGG_DIST	164
MAV_CMD_DO_SET_CAM_TRIGG_INTERVAL	164
Testing Trigger Functionality.....	164
Examples	165
Sony QX-1 example (Photogrammetry).....	165
Camera-IMU Sync Example (VIO).....	167
5.9 Gimbal Control Setup	169
Parameters	169
AUX output.....	170
Customizing the mixer configuration.....	170
SITL.....	171
Testing	171
5.10 Companion Computer for Pixhawk class	171
Pixhawk setup	171
Companion computer setup	172
Hardware setup	172
Software setup on Linux	172
6. Middleware	174
6.1 uORB Messaging.....	174
Introduction	174
Adding a new topic	174
Publishing.....	174
Listing Topics and Listening in.....	175
urb top Command.....	176
Multi-instance	176
Troubleshooting and common Pitfalls.....	176
6.2 uORB Publication/Subscription Graph	177
Graph Properties	177
6.3 MAVLink Messaging	178
Create Custom MAVLink Messages.....	178
Sending Custom MAVLink Messages	178

Receiving Custom MAVLink Messages.....	180
Alternative to Creating Custom MAVLink Messages	181
General.....	181
Set streaming rate	181
6.4 RTPS/ROS2 Interface: PX4-FastRTPS Bridge	182
When should RTPS be used?	182
Architectural overview.....	183
Code generation.....	183
Supported uORB messages	184
Client (PX4 Firmware).....	184
Agent (Off Board FastRTPS Interface)	185
Creating a FastRTPS Listener application.....	186
Examples/tests.....	188
Troubleshooting.....	188
Client reports that selected UART port is busy.....	188
Agent not built/fastrtpsgen is not found	188
Enable UART on Raspberry Pi.....	188
Additional information	189
6.5 Micro RTPS Throughput Test.....	189
Create the uORB message.....	189
Disable automatic bridge code generation	190
Generate the bridge code	190
Modify the client code	190
Result.....	191
6.6 Manually Generate Client and Agent Code	191
Disable automatic bridge code generation	192
Using generate_microRTPS_bridge.py	192
Generated code	193
uORB serialization code.....	193
RTPS message IDL files.....	193
Verify code generation.....	194
Build and use the code.....	195
7. Modules & Commands Reference	196
Categories.....	196
7.1 Modules Reference: Command	196
bl_update.....	196
config	197
dumpfile	197
esc_calib	197
hardfault_log	198
led_control	199
Description	199
Examples.....	199
listener	200

mixer.....	200
Description	200
motor_ramp	201
Description	201
Example	201
motor_test	201
mtd.....	202
nshterm	202
param	203
Description	203
Examples.....	203
perf	204
pwm	205
Description	205
Examples.....	205
reboot.....	207
sd_bench.....	207
top.....	207
usb_connected.....	207
ver.....	208
7.2 Modules Reference: Communication	209
frsky_telemetry	209
mavlink.....	209
Description	209
Implementation	209
Examples	210
micrortps_client	211
uorb	212
Description	212
Implementation	212
Examples	212
7.3 Modules Reference: Driver	213
fmu.....	213
Description	213
Implementation	213
Examples	214
gpio_led	215
Description	215
Implementation	215
Examples	215
gps.....	216
Description	216
Implementation	216
Examples	216

	vmount	217
	Description	217
	Implementation	217
	Examples	217
7.4	Modules Reference: Estimator	218
	ekf2	218
	Description	218
	Usage	218
7.5	Modules Reference: Controller	219
	fw_att_control	219
	Description	219
	Usage	219
	fw_pos_control_l1	219
	Description	219
	Usage	219
7.6	Modules Reference: System	220
	dataman	220
	Description	220
	Implementation	220
	Usage	220
	land_detector	221
	Description	221
	Implementation	221
	Usage	222
	load_mon	222
	Description	222
	Usage	222
	logger	222
	Description	223
	Implementation	223
	Examples	223
	Usage	223
	module	224
	Description	224
	Implementation	224
	Examples	224
	Usage	224
	replay	225
	Description	225
	Usage	225
	send_event	226
	Description	226
	Usage	226
	sensors	226

	Description	226
	Implementation	227
	Usage	227
8.	Robotics	228
8.1	Offboard Control	228
	Offboard Control Firmware Setup	228
	Map an RC switch to offboard mode activation	228
	Enable the companion computer interface	228
	Hardware setup	229
	Serial radios	229
	On-board processor	229
	On-board processor and wifi link to ROS (<i>Recommended</i>)	230
8.2	Robotics using ROS	231
8.2.1	MAVROS	232
	Installation	232
	Binary installation (Debian / Ubuntu).....	232
	Source installation.....	232
8.2.2	MAVROS <i>Offboard</i> control example	233
	Code.....	233
	Code explanation	236
8.2.3	ROS with Gazebo Simulation.....	238
	Installing ROS and Gazebo.....	239
	Launching ROS/Simulation	239
	Launching Gazebo with ROS Wrappers	239
	What's Happening Behind the Scenes.....	240
8.2.4	OctoMap 3D Models with ROS/Gazebo	241
	Installation	241
	Running the Simulation	242
8.2.5	Raspberry Pi - ROS installation	242
	Prerequisites	242
	Installation	243
	Errors when installing packages.....	243
8.2.6	Using Vision or Motion Capture Systems	243
	LPE Tuning for Vision or Mocal.....	244
	Enabling external pose input	244
	Asserting on Reference Frames	245
	Using Mavros	245
	First Flight	246
8.3	Using DroneKit to communicate with PX4	247
	Setting up DroneKit with PX4.....	247
	Full mission example	248
9	Debugging Topics	253
9.1	Frequently Asked Questions.....	253
	Build Errors	253

	Flash Overflow	253
	USB Errors.....	253
	The upload never succeeds.....	253
9.2	PX4 System Console	253
	System Console vs. Shell	254
	Snapdragon Flight: Wiring the Console	254
	Pixracer / Pixhawk v3: Wiring the Console	254
	Pixhawk v1: Wiring the Console	254
	Connecting via Dronecode Probe	255
	Connecting via FTDI 3.3V Cable	255
	Opening the Console.....	256
	Linux / Mac OS: Screen	256
	Windows: PuTTY	256
	Getting Started on the Console.....	257
	MAVLink Shell	257
	Snapdragon DSP Console	257
9.3	Embedded Debugging	258
	Identifying large memory consumers.....	258
	Heap allocations	258
	Debugging Hard Faults in NuttX	259
9.4	Sensor/Topic Debugging using the Listener Command.....	261
9.5	Simulation Debugging	262
	CLANG Address Sanitizer (Mac OS, Linux)	262
	Valgrind	262
	Start combinations	262
9.6	Compiler optimization	263
9.7	Send and Receive Debug Values.....	264
	Mapping between MAVLink Debug Messages and uORB Topics.....	264
	Tutorial: Send String / Float Pairs	264
	Tutorial: Receive String / Float Pairs.....	266
9.8	System-wide Replay	267
	Prerequisites	267
	Usage	267
	Important Notes	268
	EKF2 Replay.....	268
	Behind the Scenes	269
9.9	Poor Man's Sampling Profiler.....	270
	Approach.....	270
	Implementation.....	270
	Understanding the Output	271
	Possible Issues.....	271
9.10	Logging.....	271
	Usage	272
	Configuration.....	272

	Scripts.....	272
	Dropouts.....	272
	SD Cards	273
	Log Streaming	273
	Diagnostics	274
9.11	Flight Log Analysis	275
	Reporting Flights	275
	Structured Analysis	275
	Ruling Out Power Failures	275
	Analysis Tools	276
	Flight Review (Online Tool)	276
	FlightPlot (Desktop).....	277
	PX4Tools.....	277
9.12	ULog File Format.....	278
	Data types.....	278
	File structure.....	279
	Header Section	279
	Definitions Section	279
	Data Section	283
	Requirements for Parsers	284
	Known Implementations	285
	File Format Version History	285
	Changes in version 2	285
10	Tutorials	286
10.1	QGroundControl.....	286
	Planning Missions	286
	Flying Missions	286
	Setting parameters	287
	Installation	288
	Building from source	288
10.2	Video streaming in QGroundControl	288
	Install Linux environment in Odroid C1	289
	Set up alternative power connection	289
	Enable WiFi connection for Odroid C1	289
	Configure as WiFi Access Point	289
	Onboard Computer as Access Point.....	290
	gstreamer Installation	292
10.3	Long-distance Video Streaming in QGroundControl.....	293
	Wifibroadcast Overview	293
	Hardware Setup.....	293
	Hardware Modification.....	294
	Software Setup.....	294
	Generate Encryption Keys.....	294
	UAV Setup (TX).....	294

	Ground Station Setup (RX)	295
	Enhanced setup with RX antenna array, FPV goggles and OSD	295
	FAQ	295
	TODO	298
10.4	Optical Flow	298
	Setup	299
	Cameras	299
	Range Finder	302
	Estimators	302
	Extended Kalman Filter (EKF2)	302
	Local Position Estimator (LPE)	302
10.5	Using the ecl EKF	302
	What is the ecl EKF?	302
	What sensor measurements does it use?	303
	IMU	303
	Magnetometer	304
	Height	304
	GPS	304
	Range Finder	305
	Airspeed	305
	Synthetic Sideslip	305
	Drag Specific Forces	305
	Optical Flow	305
	External Vision System	306
	How do I use the 'ecl' library EKF?	306
	What are the advantages and disadvantages of the ecl EKF over other estimators?	306
	Disadvantages	306
	Advantages	307
	How do I check the EKF performance?	307
	Output Data	308
	States	308
	State Variances	308
	Observation Innovations	308
	Observation Innovation Variances	309
	Output Complementary Filter	309
	EKF Errors	310
	Observation Errors	310
	GPS Quality Checks	310
	EKF Numerical Errors	311
	What should I do if the height estimate is diverging?	311
	What should I do if the position estimate is diverging?	312
	Determination of Excessive Vibration	314
	Determination of Excessive Gyro Bias	314
	Determination of Poor Yaw Accuracy	315

	Determination of Poor GPS Accuracy	315
	Determination of GPS Data Loss	317
10.6	Preflight Sensor and EKF Checks	318
	COM_ARM_WO_GPS	319
	COM_ARM_EKF_POS	319
	COM_ARM_EKF_VEL	319
	COM_ARM_EKF_HGT	319
	COM_ARM_EKF_YAW	319
	COM_ARM_EKF_AB	320
	COM_ARM_EKF_GB	320
	COM_ARM_IMU_ACC	320
	COM_ARM_IMU_GYR	320
10.7	Land Detector Configuration	320
	Configuring Auto-Disarming	320
	Multicopter Land Detector Configuration	320
	Land detector states	321
	Fixed Wing Land Detector Configuration	322
10.8	Flying with Motion Capture (VICON, Optitrack)	322
	Computing Architecture	323
	Coordinate Frames	323
	Estimator choice	323
10.9	S.Bus Driver for Linux	324
	Signal inverter circuit	324
	Required components	324
	Circuit diagram/Connections	324
	Breadboard image	325
	Source code	325
	Usage	326
11	Advanced Topics	327
11.1	System Startup	327
	Debugging the System Boot	327
	Common boot failure causes	327
	Replacing the System Startup	327
	Customizing the System Startup	328
	Customizing the Configuration (config.txt)	328
	Starting additional applications	328
	Starting a custom mixer	329
11.2	Parameters & Configurations	329
	Command Line usage	329
	Getting and Setting Parameters	329
	Exporting and Loading Parameters	330
	C / C++ API	330
	Parameter Meta Data	331
11.3	Parameter Reference	332

Attitude Q estimator	332
Battery Calibration	333
Camera Control.....	337
Camera trigger	337
Circuit Breaker.....	339
Commander	342
Data Link Loss.....	350
EKF2	352
FW Attitude Control	372
FW L1 Control.....	380
FW Launch detection	385
FW TECS	386
Follow target.....	392
GND Attitude Control.....	393
GND POS Control	397
GPS.....	398
GPS Failure Navigation	399
Geofence.....	400
Land Detector	402
MAVLink.....	411
MKBLCTRL Testmode	413
MPU9x50 Configuration	413
Mission	414
Mount	421
Payload drop.....	439
Position Estimator INAV	451
Precision Land	452
RC Receiver Configuration.....	457
Radio Calibration	458
Radio Switches.....	458
Return To Land	486
Runway Takeoff.....	498
SD LoggingSITL.....	499
Sensor Calibration.....	503
Sensors.....	504
Snapdragon UART ESC.....	521
Subscriber Example	521
Syslink	522
System	522
VTOL Attitude Control.....	546
11.4 Installing driver on Ubuntu for Intel RealSense R200	552
Installing Ubuntu 14.04.3 LTS in Virtual Box.....	553
Installing ROS Indigo.....	554
Installing camera driver.....	554

11.5	Switching State Estimators	555
	Available estimators	555
	How to enable different estimators	555
	Usage	556
11.6	STM32 Bootloader	557
	Supported Boards	557
	Building the Bootloader	557
	Flashing the Bootloader	557
	Black Magic / Dronocode Probe	557
	J-Link.....	558
	Troubleshooting	559
12	Platform Testing and Continuous Integration.....	560
12.1	Unit Tests.....	560
	Writing a Test	560
	Testing on the local machine	562
12.2	PX4 Continuous Integration	562
	Travis-ci	562
	Semaphore.....	563
	CircleCI.....	563
12.3	Jenkins CI	563
	Overview	563
	Test Execution	563
	Server Setup.....	564
	Installation	564
	Configuration	564
	Integration Testing.....	564
	ROS / MAVROS Tests	564
	Execute Tests	564
	Write a new MAVROS test (Python).....	565
12.4	PX4 Docker Containers.....	567
	Prerequisites	567
	Container Hierarchy	567
	Use the Docker Container	568
	Helper Script (docker_run.sh).....	568
	Calling Docker Manually	569
	Re-enter the Container	570
	Clearing the Container.....	570
	QGroundControl	570
	Troubleshooting	571
	Virtual Machine Support	571
	Legacy	572
12.5	Maintenance notes	572
	Analyze churn	572

1. introduction

This guide is for primarily for software developers and (new) hardware integrators. To fly, build and modify vehicles using supported hardware see the [PX4 User Guide](#).

This guide explains how to:

- Get a [minimum developer setup](#), [build PX4 from source](#) and deploy on [numerous supported autopilots](#).
- Understand the [PX4 System Architecture](#) and other core concepts.
- Learn how to modify the flight stack and middleware:
 - Modify flight algorithms and add new [flight modes](#).
 - Support new [airframes](#).
- Learn how to integrate PX4 with new hardware:
 - Support new sensors and actuators, including cameras, rangefinders, etc.
 - Modify PX4 to run on new autopilot hardware.
- [Simulate](#), [test](#) and [debug/log](#) PX4.
- Communicate/integrate with external robotics APIs.

Forums and Chat

The core development team and community are active on the following forums and chat channels.

- [PX4 Discuss](#) (*recommended*)
- [Slack](#) (sign up)
- [Google+](#)

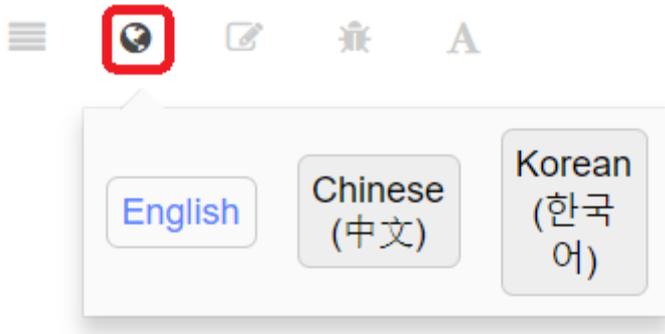
Developers who want to [contribute](#) to the platform are also most welcome to attend the [weekly dev call](#) and our other [developer events](#).

Contributing

[Contributing & Dev Call](#) explains how to work with our source codelines. [Documentation](#) explains how and where documentation changes can be made.

Translations

There are Chinese and Korean [translations](#) of this guide. You can access these by clicking the language-switcher icon:



Licence

The code is free to use and modify under terms of the permissive [BSD 3-clause license](#). The documentation is licensed under [CC BY 4.0](#). For more information see: [Licences](#).

2. Getting Started

This section contains topics related to getting a [minimum developer setup](#), [build PX4 from source](#) and contributing to PX4.

2.1 Initial Configuration & Setup

Before starting to work on PX4 we recommend that developers obtain the basic equipment described below (or similar), and initially use a "default" configuration for their [airframe](#).

PX4 can be used with a much wider range of equipment, but new developers will benefit from going with one of the standard setups, and a Taranis RC plus a Note 4 tablet make up for a very inexpensive field kit.

Basic Equipment

The equipment below is highly recommended:

- A Taranis Plus remote control for the safety pilot (or equivalent)
- A development computer
 - MacBook Pro or Air with OSX 10.10 or newer
 - Modern laptop with Ubuntu Linux (14.04 or newer)
- A ground control station device
 - Any MacBook or Ubuntu Linux laptop (can be the development computer)
 - Samsung Note 4 or equivalent (any recent Android tablet or phone with a large enough screen to run QGroundControl effectively).
 - iPad (requires Wifi telemetry adapter)
- Safety glasses
- For multicopters - tether for more risky tests

Vehicle Configuration

Download the [QGroundControl Daily Build](#) for your development platform.

QGroundControl for a desktop OS is required for vehicle configuration. The daily build is highly recommended in order to take advantage of the latest features in PX4.

Follow the video instructions below to set up your vehicle. For more detailed/written instructions see the sidebar topics in the [PX4 User Guide > Basic Configuration](#) section.

2.2 Installing Files and Code

PX4 code can be developed on [Linux](#) or [Mac OS](#). We recommend [Ubuntu Linux LTS edition](#) as this enables building [all PX4 targets](#), and using most [simulators](#) and [ROS](#).

A [Windows](#) toolchain also exists but is not officially supported (we highly discourage its use). It is possible to build PX4 on Windows using a virtual machine running Ubuntu Linux, but this may not provide a reliable platform for Simulation. Before starting to develop on Windows, consider installing a dual-boot environment with [Ubuntu](#).

Supported Targets

The table below show what PX targets you can build on each OS.

Target	Linux (Ubuntu)	Mac	Windows
NuttX based hardware: Pixhawk Series , Crazyflie , Intel® Aero Ready to Fly Drone	X	X	X
Qualcomm Snapdragon Flight hardware	X		
Linux-based hardware: Raspberry Pi 2/3 , Parrot Bebop	X		
Simulation: jMAVSim SITL	X	X	X
Simulation: Gazebo SITL	X	X	
Simulation: ROS with Gazebo	X		

Development Environment

The installation of the development environment is covered below:

- [Mac OS](#)
- [Linux](#)
- [Windows](#) (not recommended!)

If you're familiar with Docker you can also use one of the prepared containers: [Docker Containers](#)

2.2.1 Development Environment on Mac

Mac OS X is the main development platform for PX4. The following instructions explain how to set up a development environment for building NuttX-based hardware (Pixhawk, etc.) and Simulation (jMAVSim/Gazebo) targets. For other targets see: [Toolchain Installation > Supported Targets](#).

Homebrew Installation

The installation of Homebrew is quick and easy: [installation instructions](#).

Common Tools

After installing Homebrew, run these commands in your shell to install the common tools:

```
brew tap PX4/px4
brew install px4-dev
# Optional, but recommended additional simulation tools:
brew install px4-sim
```

If the installation outputs an error message about missing requirements follow the instructions. Your system will be missing Java and Quartz:

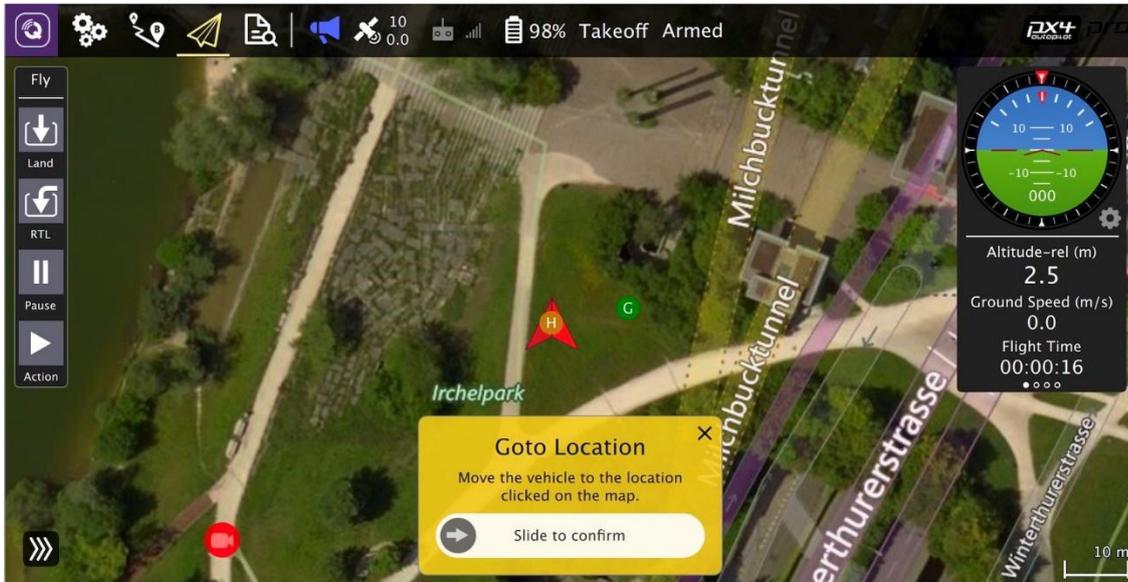
```
brew cask install xquartz java
```

Install pip if you don't already have it and use it to install the required packages:

```
sudo easy_install pip
sudo -H pip install pyserial empy toml numpy pandas jinja2
```

Ground Control Software

Download and install the [QGroundControl Daily Build](#).



Editor / IDE

The development team often use:

- [Sublime Text](#): a fast and lean text editor.
- [Qt Creator](#): A popular open-source IDE.

Next Steps

Once you have finished setting up the environment, continue to the [build instructions](#).

2.2.2 Development Environment on Linux

Linux allows you to build for [all PX4 targets](#) (NuttX based hardware, Qualcomm Snapdragon Flight hardware, Linux-based hardware, Simulation, ROS).

We have standardized on Debian / [Ubuntu Linux LTS](#) (16.04) as the supported Linux distribution. Instructions are also provided for [CentOS](#) and [Arch Linux](#).

The following instructions explain how to set up a development environment on Ubuntu LTS using convenience bash scripts. Instructions for *manually installing* these and additional targets can be found in [Ubuntu/Debian Linux](#).

Development Toolchain

The instructions below show how you can use our [convenience bash scripts](#) to setup the developer toolchain on Ubuntu LTS. All the scripts install the *Qt Creator IDE*, [Ninja Build System](#), [Common Dependencies](#), [FastRTPS](#), and also download the PX4 source to your computer (`~/src/Firmware`).

The scripts have been tested on a clean Ubuntu LTS 16.04 installation. They *may* not work as expected if installed on top of an existing system or on another Ubuntu release. If you have any problems then follow the [manual installation instructions](#).

First make the user a member of the group "dialout"

1. On the command prompt enter:

```
sudo usermod -a -G dialout $USER
```

2. Logout and login again (the change is only made after a new login).

Then follow the instructions for your development target in the sections below.

Pixhawk/NuttX (and jMAVSim)

To install the development toolchain:

1. Download [ubuntu_sim_nuttx.sh](#).
2. Run the script in a bash shell:

```
source ubuntu_sim_nuttx.sh
```

You may need to acknowledge some prompts as the script progresses.

3. Restart the computer on completion.

Snapdragon Flight or Raspberry Pi

To install the development toolchain:

1. Download [ubuntu_sim_common_deps.sh](#) (this contains the jMAVSim simulator and common toolchain dependencies).
2. Run the script in a bash shell:

```
source ubuntu_sim_common_deps.sh
```

You may need to acknowledge some prompts as the script progresses.

-
3. Follow the platform-specific instructions in [Ubuntu/Debian Linux](#) for your target:
 - o [Snapdragon Flight](#)
 - o [Raspberry Pi](#)

Parrot Bepop

Follow the (manual) instructions here: [Ubuntu/Debian Linux > Parrot Bebop](#).

jMAVSim/Gazebo Simulation

To install the Gazebo and jMAVSim simulators:

1. Download [ubuntu_sim.sh](#).
2. Run the script in a bash shell:

```
source ubuntu_sim.sh
```

You may need to acknowledge some prompts as the script progresses.

If you just need jMAVSim, instead download and run [ubuntu_sim_common_deps.sh](#).

Gazebo with ROS

To install the development toolchain:

1. Download [ubuntu_sim_ros_gazebo.sh](#).
2. Run the script in a bash shell:

```
source ubuntu_sim_ros_gazebo.sh
```

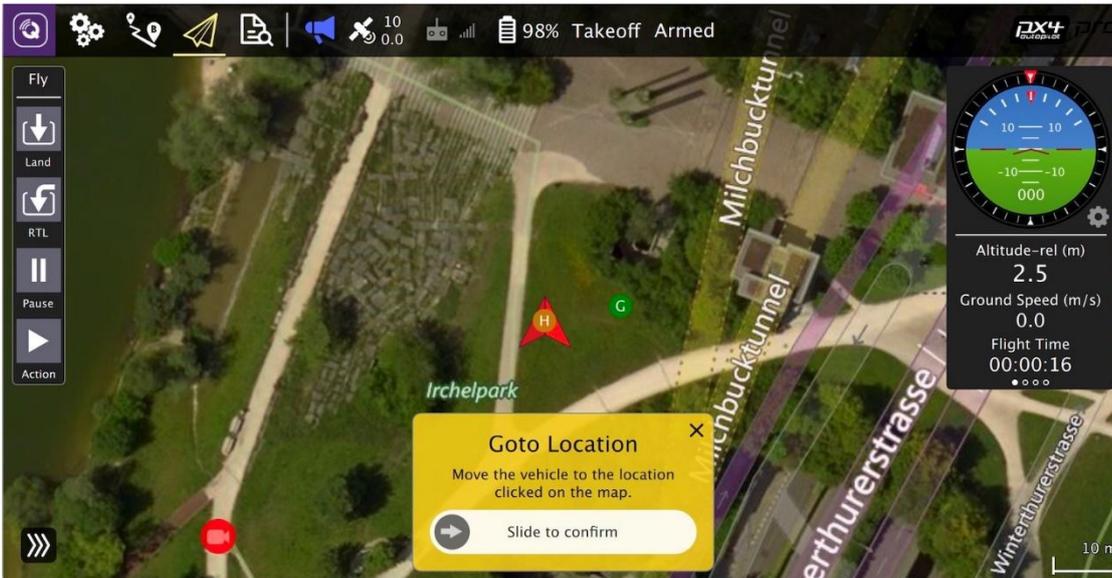
You may need to acknowledge some prompts as the script progresses.

Note:

- ROS is installed with Gazebo7 by default (we have chosen to use the default rather than Gazebo8 to simplify ROS development).
- Your catkin (ROS build system) workspace is created at `~/catkin_ws/`.

Ground Control Software

Download and install the [QGroundControl Daily Build](#).



Editor / IDE

The development team often use:

- [Sublime Text](#): a fast and lean text editor.
- [Qt Creator](#): A popular open-source IDE.

The installation scripts automatically install *Qt Creator* as part of the common dependencies. You can launch it by entering `qtcreator` in a bash terminal.

Next Steps

Once you have finished setting up the environment, continue to the [build instructions](#).

2.2.3 Development Environment on Ubuntu LTS / Debian Linux

[Ubuntu Linux LTS](#) (16.04) is the standard/preferred Linux development OS. It allows you to build for [all PX4 targets](#) (NuttX based hardware, Qualcomm Snapdragon Flight hardware, Linux-based hardware, Simulation, ROS).

The following instructions explain how to *manually* set up a development environment each of the supported targets.

We recommend that you use the [Convenience bash scripts](#) to install the Simulators and/or NuttX toolchain (this is easier than typing in the instructions below). Then follow

just the additional instructions for other targets (e.g. Qualcomm Snapdragon Flight, Bebop, Raspberry Pi, etc.)

Convenience Bash Scripts

We've created a number of bash scripts that you can use to install the Simulators and/or NuttX toolchain. All the scripts install the *Qt Creator IDE*, [Ninja Build System](#), [Common Dependencies](#), [FastRTPS](#), and also download the PX4 source to your computer (`~/src/Firmware`).

The scripts have been tested on a clean Ubuntu 16.04 LTS installation. They *may* not work as expected if installed on top of an existing system.

The scripts are:

- [ubuntu_sim_common_deps.sh](#): [Common Dependencies](#), [jMAVSim](#) simulator
 - This script contains the common dependencies for all PX4 build targets. It is automatically downloaded and run when you call any of the other scripts.
 - You can run this before installing the remaining dependencies for [Qualcomm Snapdragon Flight](#) or [Raspberry Pi/Parrot Bebop](#).
- [ubuntu_sim.sh](#): `ubuntu_sim_common_deps.sh` + [Gazebo8](#) simulator.
- [ubuntu_sim_nuttX.sh](#): `ubuntu_sim.sh` + NuttX tools.
 - *This requires computer restart on completion.*
- [ubuntu_sim_ros_gazebo.sh](#): `ubuntu_sim_common_deps.sh` + [ROS/Gazebo and MAVROS](#).
 - ROS Kinetic is installed with Gazebo7 by default (we have chosen to use the default rather than Gazebo 8 to simplify ROS development).
 - Your catkin (ROS build system) workspace is created at `~/catkin_ws/`.

How to use the scripts

To use the scripts:

1. Make the user a member of the group "dialout" (this only has to be done once):
 - i. Open a terminal and enter the following command:

```
sudo usermod -a -G dialout $USER
```

- ii. Logout and login again (the change is only made after a new login).
2. Download the desired script
3. Run the script in a bash shell (e.g. to run `ubuntu_sim.sh`):

```
source ubuntu_sim.sh
```

Acknowledge any prompts as the scripts progress.

Permission Setup

Never ever fix permission problems by using `sudo`. It will create more permission problems in the process and require a system re-installation to fix them.

The user needs to be part of the group "dialout":

```
sudo usermod -a -G dialout $USER
```

Then logout and login again (the change is only made after a new login).

Remove the modemmanager

Ubuntu comes with a serial modem manager which interferes heavily with any robotics related use of a serial port (or USB serial). It can be removed/deinstalled without side effects:

```
sudo apt-get remove modemmanager
```

Common Dependencies

Update the package list and install the following dependencies for all PX4 build targets.

```
sudo apt-get update -y
sudo apt-get install git zip qtcreator cmake \
    build-essential genromfs ninja-build -y
# Required python packages
sudo apt-get install python-argparse \
    python-empy python-toml python-numpy \
    python-dev python-pip -y
sudo -H pip install --upgrade pip
sudo -H pip install pandas jinja2 pyserial
```

You may also wish to install [pyulog](#). This is a useful python package that contains scripts to parse *ULog* files and display them.

```
# optional python tools
sudo -H pip install pyulog
```

Ninja Build System

[Ninja](#) is a faster build system than *Make* and the PX4 *CMake* generators support it.

On Ubuntu Linux you can install this automatically from normal repos.

```
sudo apt-get install ninja-build -y
```

Other systems may not include Ninja in the package manager. In this case an alternative is to download the binary and add it to your path:

```
mkdir -p $HOME/ninja
cd $HOME/ninja
wget https://github.com/martine/ninja/releases/download/v1.6.0/ninja-linux.zip
unzip ninja-linux.zip
rm ninja-linux.zip
exportline="export PATH=$HOME/ninja:\$PATH"
if grep -Fxq "$exportline" ~/.profile; then echo nothing to do ; else echo
$exportline >> ~/.profile; fi
. ~/.profile
```

2.2.4 Fast RTPS installation

[eProsima Fast RTPS](#) is a C++ implementation of the RTPS (Real Time Publish Subscribe) protocol. FastRTPS is used, via the [RTPS/ROS2 Interface: PX4-FastRTPS Bridge](#), to allow PX4 uORB topics to be shared with offboard components.

The following instructions can be used to install the FastRTPS 1.5 binaries to your home directory.

```
wget http://www.eprosima.com/index.php/component/ars/repository/eprosima-fast-
rtps/eprosima-fast-rtps-1-5-0/eprosima_fastrtps-1-5-0-linux-tar-gz -O
eprosima_fastrtps-1-5-0-linux.tar.gz
tar -xzf eprosima_fastrtps-1-5-0-linux.tar.gz eProsima_FastRTPS-1.5.0-Linux/
tar -xzf eprosima_fastrtps-1-5-0-linux.tar.gz requiredcomponents
tar -xzf requiredcomponents/eProsima_FastCDR-1.0.7-Linux.tar.gz
```

In the following lines where we compile the FastCDR and FastRTPS libraries, the `make` command is issued with the `-j2` option. This option defines the number of parallel threads (or jobs) that are used to compile the source code. Change `-j2` to `-j<number_of_cpu_cores_in_your_system>` to speed up the compilation of the libraries.

```
cd eProsima_FastCDR-1.0.7-Linux; ./configure --libdir=/usr/lib; make -j2; sudo make
install
cd ..
cd eProsima_FastRTPS-1.5.0-Linux; ./configure --libdir=/usr/lib; make -j2; sudo make
install
cd ..
rm -rf requiredcomponents eprosima_fastrtps-1-5-0-linux.tar.gz
```

More "generic" instructions, which additionally cover installation from source, can be found here: [Fast RTPS installation](#).

Simulation Dependencies

The dependencies for the Gazebo and jMAVSim simulators listed below. You should minimally install jMAVSim to make it easy to test the installation. Additional information about these and other supported simulators is covered in: [Simulation](#).

jMAVSim

Install the dependencies for [jMAVSim Simulation](#).

```
# jMAVSim simulator
sudo apt-get install ant openjdk-8-jdk openjdk-8-jre -y
```

Gazebo

If you're going work with ROS then follow the [ROS/Gazebo](#) instructions in the following section (these install Gazebo automatically, as part of the ROS installation).

Install the dependencies for [Gazebo Simulation](#).

```
# Gazebo simulator
sudo apt-get install protobuf-compiler libeigen3-dev libopencv-dev -y
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable
`lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
## Setup keys
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
## Update the debian database:
sudo apt-get update -y
## Install Gazebo8
sudo apt-get install gazebo8 -y
## For developers (who work on top of Gazebo) one extra package
sudo apt-get install libgazebo8-dev
```

ROS/Gazebo

Install the dependencies for [ROS/Gazebo](#) ("Kinetic"). These include Gazebo7 (at time of writing, the default version that comes with ROS). The instructions come from the ROS Wiki [Ubuntu page](#).

```
# ROS Kinetic/Gazebo
## Gazebo dependencies
sudo apt-get install protobuf-compiler libeigen3-dev libopencv-dev -y
```

```

## ROS Gazebo: http://wiki.ros.org/kinetic/Installation/Ubuntu
## Setup keys
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key
421C365BD9FF1F717815A3895523BAEED01FA116
## For keyserver connection problems substitute hkp://pgp.mit.edu:80 or
hkp://keyserver.ubuntu.com:80 above.
sudo apt-get update
## Get ROS/Gazebo
sudo apt-get install ros-kinetic-desktop-full -y
## Initialize rosdep
sudo rosdep init
rosdep update
## Setup environment variables
rossource="source /opt/ros/kinetic/setup.bash"
if grep -Fxq "$rossource" ~/.bashrc; then echo ROS setup.bash already in .bashrc;
else echo "$rossource" >> ~/.bashrc; fi
source ~/.bashrc
## Get rosinstall
sudo apt-get install python-roinstall -y

```

Install the [MAVROS \(MAVLink on ROS\)](#) package. This enables MAVLink communication between computers running ROS, MAVLink enabled autopilots, and MAVLink enabled GCS.

MAVROS can be installed as an ubuntu package or from source. Source is recommended for developers.

```

## Create catkin workspace (ROS build system)
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws

## Install dependencies
sudo apt-get install python-wstool python-roinstall-generator python-catkin-tools -y

## Initialise wstool
wstool init ~/catkin_ws/src

## Build MAVROS
### Get source (upstream - released)
roinstall_generator --upstream mavros | tee /tmp/mavros.roinstall
### Get latest released mavlink package
roinstall_generator mavlink | tee -a /tmp/mavros.roinstall
### Setup workspace & install deps
wstool merge -t src /tmp/mavros.roinstall

```

```
wstool update -t src
rosdep install --from-paths src --ignore-src --rosdistro kinetic -y
If you use an ubuntu-based distro and the command rosdep install --from-paths src --
ignore-src --rosdistro kinetic -y fails, you can try to force the command to run by
executing rosdep install --from-paths src --ignore-src --rosdistro kinetic -y --os
ubuntu:xenial

## Build!
catkin build
## Re-source environment to reflect new packages/build environment
catkin_ws_source="source ~/catkin_ws/devel/setup.bash"
if grep -Fxq "$catkin_ws_source" ~/.bashrc; then echo ROS catkin_ws setup.bash
already in .bashrc;
else echo "$catkin_ws_source" >> ~/.bashrc; fi
source ~/.bashrc
```

NuttX-based Hardware

Install the following dependencies to build for NuttX based hardware: Pixhawk, Pixfalcon, Pixracer, Pixhawk 3, Intel® Aero Ready to Fly Drone.

Packages with specified versions should be installed with the specified package version.

```
sudo apt-get install python-serial openocd \
    flex bison libncurses5-dev autoconf texinfo \
    libftdi-dev libtool zlib1g-dev -y
```

Remove any old versions of the arm-none-eabi toolchain.

```
sudo apt-get remove gcc-arm-none-eabi gdb-arm-none-eabi binutils-arm-none-eabi gcc-
arm-embedded
sudo add-apt-repository --remove ppa:team-gcc-arm-embedded/ppa
```

Execute the script below to install GCC 7-2017-q4:

```
pushd .
cd ~
wget https://armkeil.blob.core.windows.net/developer/Files/downloads/gnu-rm/7-
2017q4/gcc-arm-none-eabi-7-2017-q4-major-linux.tar.bz2
tar -jxf gcc-arm-none-eabi-7-2017-q4-major-linux.tar.bz2
exportline="export PATH=$HOME/gcc-arm-none-eabi-7-2017-q4-major/bin:\$PATH"
if grep -Fxq "$exportline" ~/.profile; then echo nothing to do ; else echo
$exportline >> ~/.profile; fi
popd
```

Now restart your machine.

Troubleshooting

Check the version by entering the following command:

```
arm-none-eabi-gcc --version
```

The output should be something similar to:

```
arm-none-eabi-gcc (GNU Tools for Arm Embedded Processors 7-2017-q4-major) 7.2.1
20170904 (release) [ARM/embedded-7-branch revision 255204]
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Snapdragon Flight

(Cross) Toolchain installation

```
sudo apt-get install android-tools-adb android-tools-fastboot \
  fakechroot fakeroot unzip xz-utils wget python python-empy -y
```

Please follow the instructions on https://github.com/ATLFlight/cross_toolchain for the toolchain installation.

Load the new configuration:

```
source ~/.bashrc
```

Sysroot Installation

A sysroot is required to provide the libraries and header files needed to cross compile applications for the Snapdragon Flight applications processor.

The qrlSDK sysroot provides the required header files and libraries for the camera, GPU, etc.

Download the file [Flight 3.1.3 qrlSDK.tgz](#) and save it in `cross_toolchain/download/`.

```
cd cross_toolchain
```

```
unset HEXAGON_ARM_SYSROOT
```

```
./qrlinux_sysroot.sh
```

Append the following to your `~/.bashrc`:

```
export HEXAGON_ARM_SYSROOT=${HOME}/Qualcomm/qrlinux_v3.1.1_sysroot
```

Load the new configuration:

```
source ~/.bashrc
```

For more sysroot options see [Sysroot Installation](#)

Update ADSP firmware

Before building, flashing and running code, you'll need to update the [ADSP firmware](#).

References

There is a an external set of documentation for Snapdragon Flight toolchain and SW setup and verification: [ATLFlightDocs](#)

Messages from the DSP can be viewed using mini-dm.

```
${HEXAGON_SDK_ROOT}/tools/debug/mini-dm/Linux_Debug/mini-dm
```

Note: Alternatively, especially on Mac, you can also use [nano-dm](#).

Raspberry Pi Hardware

Developers working on Raspberry Pi hardware need to download a ARMv7 cross-compiler, either GCC or clang. The recommended toolchain for raspbian is GCC 4.8.3 and can be cloned from <https://github.com/raspberrypi/tools.git>.

The PATH environmental variable should include the path to the gcc cross-compiler collection of tools (e.g. gcc, g++, strip) prefixed with `arm-linux-gnueabihf-`.

```
git clone https://github.com/raspberrypi/tools.git ${HOME}/rpi-tools
```

```
# test compiler
```

```
${HOME}/rpi-tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf-gcc -v
```

```
# permanently update PATH variable by modifying ~/.profile
```

```
echo 'export PATH=$PATH:${HOME}/rpi-tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin' >> ~/.profile
```

```
# update PATH variable only for this session
```

```
export PATH=$PATH:${HOME}/rpi-tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin
```

clang

In order to use clang, you also need GCC.

Download clang for your specific distribution from [LLVM Download page](#) and unpack it. Assuming that you've unpacked clang to `CLANG_DIR`, and `clang` binary is available in `CLANG_DIR/bin`, and you have the GCC cross-compiler in `GCC_DIR`, you will need to setup the symlinks for clang in the `GCC_DIR` bin dir, and add `GCC_DIR/bin` to PATH.

Example below for building PX4 firmware out of tree, using CMake.

```
ln -s <CLANG_DIR>/bin/clang <GCC_DIR>/bin/clang
ln -s <CLANG_DIR>/bin/clang++ <GCC_DIR>/bin/clang++
export PATH=<GCC_DIR>/bin:$PATH

cd <PATH-TO-PX4-SRC>
mkdir build/posix_rpi_cross_clang
cd build/posix_rpi_cross_clang
cmake \
-G"Unix Makefiles" \
-DCONFIG=posix_rpi_cross \
-DCMAKE_C_COMPILER=clang \
-DCMAKE_CXX_COMPILER=clang++ \
..
```

Parrot Bebop

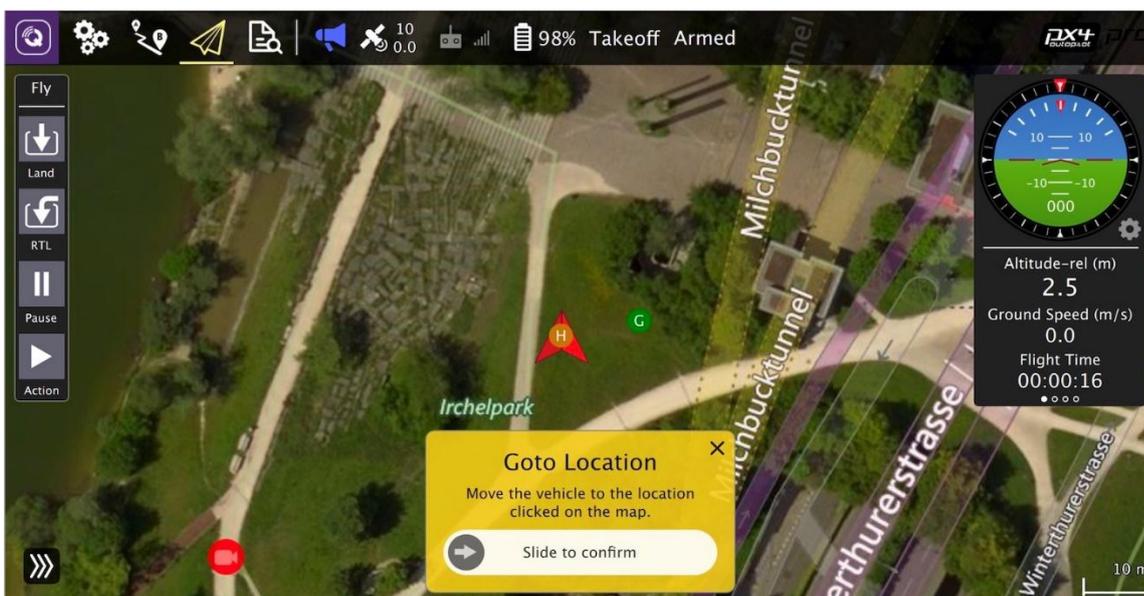
Developers working with the Parrot Bebop should install the RPi Linux Toolchain. Follow the description under [Raspberry Pi hardware](#).

Next, install ADB.

```
sudo apt-get install android-tools-adb -y
```

Ground Control Software

Download and install the [QGroundControl Daily Build](#).



Editor / IDE

The development team often use:

- [Sublime Text](#): a fast and lean text editor.
- [Qt Creator](#): A popular open-source IDE.

Next Steps

Once you have finished setting up the environment, continue to the [build instructions](#).

Development Environment on CentOS

These instructions have not been tested with recent builds of PX4. We hope to provide fully tested instructions with the supported toolchain in the near future.

The build requires Python 2.7.5. Therefore as of this writing Centos 7 should be used. (For earlier Centos releases a side-by-side install of python v2.7.5 may be done. But it is not recommended because it can break yum.)

Common Dependencies

The EPEL repositories are required for openocd libftdi-devel libftdi-python

```
wget https://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-5.noarch.rpm
sudo yum install epel-release-7-5.noarch.rpm
yum update
yum groupinstall "Development Tools"
yum install python-setuptools python-numpy
easy_install pyserial
easy_install pexpect
easy_install toml
yum install openocd libftdi-devel libftdi-python python-argparse flex bison-devel
ncurses-devel ncurses-libs autoconf texinfo libtool zlib-devel cmake
```

You may want to also install python-pip and screen

GCC Toolchain Installation

Execute the script below to install GCC 7-2017-q4:

```
pushd .
```

```
cd ~
wget https://armkeil.blob.core.windows.net/developer/Files/downloads/gnu-rm/7-2017q4/gcc-arm-none-eabi-7-2017-q4-major-linux.tar.bz2
tar -jxf gcc-arm-none-eabi-7-2017-q4-major-linux.tar.bz2
exportline="export PATH=$HOME/gcc-arm-none-eabi-7-2017-q4-major/bin:\$PATH"
if grep -Fxq "$exportline" ~/.profile; then echo nothing to do ; else echo
$exportline >> ~/.profile; fi
popd
```

Now restart your machine.

Troubleshooting

Check the version by entering the following command:

```
arm-none-eabi-gcc --version
```

The output should be something similar to:

```
arm-none-eabi-gcc (GNU Tools for Arm Embedded Processors 7-2017-q4-major) 7.2.1
20170904 (release) [ARM/embedded-7-branch revision 255204]
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Ninja Build System

[Ninja](#) is a faster build system than *Make* and the PX4 *CMake* generators support it.

On Ubuntu Linux you can install this automatically from normal repos.

```
sudo apt-get install ninja-build -y
```

Other systems may not include Ninja in the package manager. In this case an alternative is to download the binary and add it to your path:

```
mkdir -p $HOME/ninja
cd $HOME/ninja
wget https://github.com/martine/ninja/releases/download/v1.6.0/ninja-linux.zip
unzip ninja-linux.zip
rm ninja-linux.zip
exportline="export PATH=$HOME/ninja:\$PATH"
if grep -Fxq "$exportline" ~/.profile; then echo nothing to do ; else echo
$exportline >> ~/.profile; fi
. ~/.profile
```

Development Environment on ArchLinux

These instructions allow you to build PX4 (without RTPS) for NuttX targets, using an unsupported version of GCCE from the package manager. The instructions have been tested on Antergos (an Arch Linux based distribution) as it is easier to set up than Arch Linux. We hope to provide fully tested instructions with the supported toolchain in the near future.

Permissions

The user needs to be added to the group "uucp":

```
sudo usermod -a -G uucp $USER
```

Then log out and log in for changes to take effect.

Script-based Installation

This script installs the (unsupported) latest GCCE from the package manager. MicroRTPS is not built.

Once ArchLinux is installed you can use the docker script [archlinux_install_script.sh](#) to install all dependencies required for building PX4 firmware.

To install using this script, enter the following in a terminal:

```
wget https://raw.githubusercontent.com/PX4/containers/master/docker/px4-dev/scripts/archlinux_install_script.sh
sudo -s
source ./archlinux_install_script.sh
```

Manual Installation

Common Dependencies

To install the dependencies manually, enter the following lines into a terminal.

```
# Common dependencies for all targets
sudo pacman -Sy --noconfirm \
    base-devel make cmake ccache git \
    ninja python-pip tar unzip zip vim wget

# Install Python dependencies
```

```
pip install serial empy numpy toml jinja2
```

```
# Install genromfs
```

```
wget https://sourceforge.net/projects/romfs/files/genromfs/0.5.2/genromfs-0.5.2.tar.gz
```

```
tar zxvf genromfs-0.5.2.tar.gz
```

```
cd genromfs-0.5.2 && make && make install && cd ..
```

```
rm genromfs-0.5.2.tar.gz genromfs-0.5.2 -r
```

genromfs is also available in the [Archlinux User Repository](#) (AUR). To use this package, install [yaourt](#) (Yet AnOther User Repository Tool) and then use it to download, compile and install *genromfs* as shown:

```
yaourt -S genromfs
```

GCCE Compiler

A GCC compiler is required to build for NuttX targets. Enter the command below to install the latest version from the package manager (unsupported).

```
# Compiler from package manager (unsupported)
```

```
sudo pacman -Sy --noconfirm \
```

```
    arm-none-eabi-gcc arm-none-eabi-newlib
```

Alternatively, the standard instructions for installing the **official** version are listed below.

These are untested. Attempt them at your own risk!

Execute the script below to install GCC 7-2017-q4:

```
pushd .
```

```
cd ~
```

```
wget https://armkeil.blob.core.windows.net/developer/Files/downloads/gnu-rm/7-2017q4/gcc-arm-none-eabi-7-2017-q4-major-linux.tar.bz2
```

```
tar -jxf gcc-arm-none-eabi-7-2017-q4-major-linux.tar.bz2
```

```
exportline="export PATH=$HOME/gcc-arm-none-eabi-7-2017-q4-major/bin:$PATH"
```

```
if grep -Fxq "$exportline" ~/.profile; then echo nothing to do ; else echo
```

```
$exportline >> ~/.profile; fi
```

```
popd
```

Now restart your machine.

Troubleshooting

Check the version by entering the following command:

```
arm-none-eabi-gcc --version
```

The output should be something similar to:

```
arm-none-eabi-gcc (GNU Tools for Arm Embedded Processors 7-2017-q4-major) 7.2.1
20170904 (release) [ARM/embedded-7-branch revision 255204]
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Advanced Linux Installation Use-Cases

Using JTAG Programming Adapters

Linux users need to explicitly allow access to the USB bus for JTAG programming adapters.

For Archlinux: replace the group `plugdev` with `uucp` in the following commands

Run a simple `ls` in `sudo` mode to ensure the commands below succeed:

```
sudo ls
```

Then with `sudo` rights temporarily granted, run this command:

```
cat > $HOME/rule.tmp <<_EOF
# All 3D Robotics (includes PX4) devices
SUBSYSTEM=="usb", ATTR{idVendor}=="26AC", GROUP="plugdev"
# FTDI (and Black Magic Probe) Devices
SUBSYSTEM=="usb", ATTR{idVendor}=="0483", GROUP="plugdev"
# Olimex Devices
SUBSYSTEM=="usb", ATTR{idVendor}=="15ba", GROUP="plugdev"
_EOF
sudo mv $HOME/rule.tmp /etc/udev/rules.d/10-px4.rules
sudo /etc/init.d/udev restart
```

The user needs to be added to the group `plugdev`:

```
sudo usermod -a -G plugdev $USER
```

Windows Installation Instructions

Although a Windows toolchain is available, it is not officially supported (and we discourage its use). It is unbearably slow during Firmware compilation and does not support new boards like Snapdragon Flight. It also cannot run the standard robotics software packages many developers use to prototype computer vision and navigation. Before starting to develop on Windows, consider installing a dual-boot environment with [Ubuntu](#).

Toolchain Installation

There are a number of ways you can set up a Windows development toolchain for PX4.

- The native toolchain allows you to build for NuttX/Pixhawk and jMAVSim simulator targets.
- The Windows Bash toolchain allows you to build for NuttX/Pixhawk targets (only).

Native toolchain

Download and install these on your system:

- [PX4 Toolchain Installer v14 for Windows Download](#) (32/64 bit systems, complete build system, drivers)
- [PX4 USB Drivers](#) (32/64 bit systems)

Bash on Windows (NEW!)

Windows users can alternatively install a *slightly modified* Ubuntu Linux PX4 development environment within [Bash on Windows](#), and use it to build firmware for NuttX/Pixhawk targets. We have provided a script below that makes this easy.

This approach does not currently support simulation because *Bash on Windows* does not enable Linux UI applications.

The script has been updated to [install Fast RTPS from \(Linux\) binaries](#).

To use the build script:

1. Install [Bash on Windows](#).
2. Download the [windows_bash_nutt.sh](#) script.
3. Open the bash shell and navigate to the directory containing the script.
4. Run the script using the command below (acknowledging any prompts as required):

```
source windows_bash_nutt.sh
```

5. Test the script by building the firmware:

```
cd $src/Firmware
make px4fmu-v2_default
```

On successful completion you'll find the firmware here: `Firmware/build/px4fmu-v2/src/firmware/nuttX/px4fmu-v2_default.px4`

-
6. You can flash the custom firmware on Windows using *QGroundControl* or *Mission Planner* (it is not yet possible to directly flash the firmware from within the bash shell).

Build script details

The [windows_bash_nutt.sh](#) build script modifies the Ubuntu build instructions to remove Ubuntu-specific and UI-dependent components, including the *Qt Creator* IDE and the simulators.

In addition, it uses a [64 bit arm-none-eabi compiler](#) since BashOnWindows doesn't run 32 bit ELF programs (and the default compiler from <https://launchpad.net/gcc-arm-embedded> is 32 bit).

To add this compiler to your environment manually:

1. Download the compiler:

```
wget https://github.com/SolinGuo/arm-none-eabi-bash-on-win10-/raw/master/gcc-arm-none-eabi-5_4-2017q2-20170512-linux.tar.bz2
```

2. Unpack it using this command line in the Bash On Windows console:

```
tar -xvf gcc-arm-none-eabi-5_4-2017q2-20170512-linux.tar.bz2
```

This will unpack the arm gcc cross-compiler to:

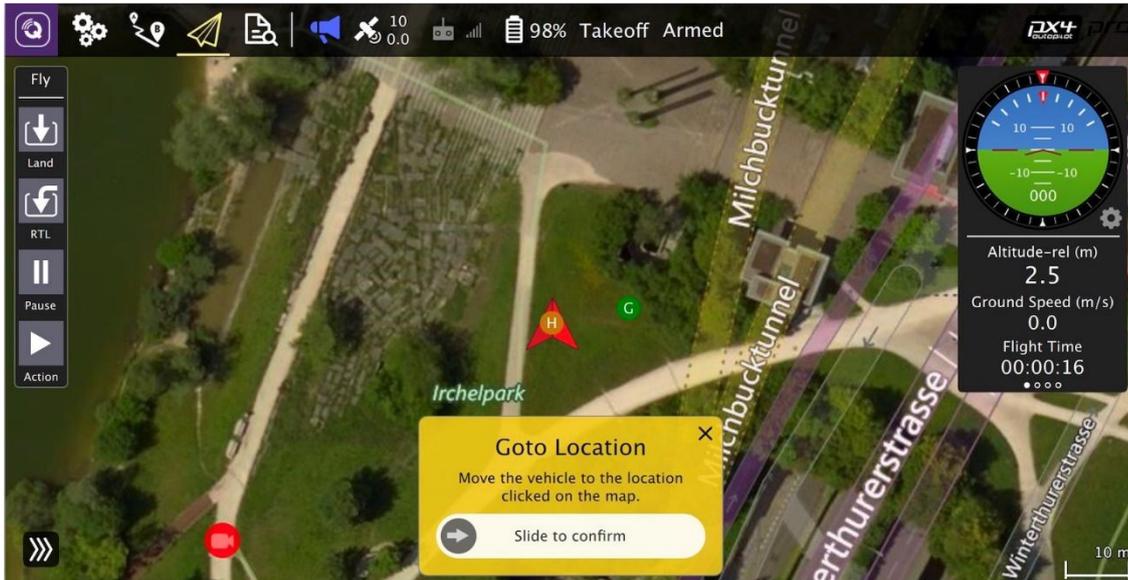
```
gcc-arm-none-eabi-5_4-2017q2/bin
```

3. Add the to the environment (add the line to your bash profile to make the change permanent)

```
export PATH=$HOME/gcc-arm-none-eabi-5_4-2017q2/bin:~$PATH
```

Ground Control Software

Download and install the [QGroundControl Daily Build](#).



Editor / IDE

The development team often use:

- [Sublime Text](#): a fast and lean text editor.
- [Qt Creator](#): A popular open-source IDE.

Next Steps

Once you have finished setting up the environment, continue to the [build instructions](#).

2.3 Building PX4 Software

PX4 can be built on the console or in an IDE, for both simulated and hardware targets.

Downloading PX4 Source Code

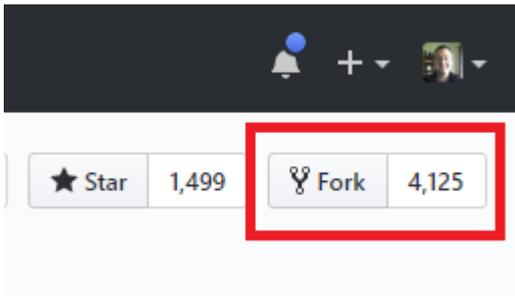
The PX4 source code is stored on Github in the [PX4/Firmware](#) repository. We recommend that you [fork](#) this repository (creating a copy associated with your own Github account), and then [clone](#) the source to your local computer.

Forking the repository allows you to better manage your custom code. Later on you will be able to use *git* to share changes with the main project.

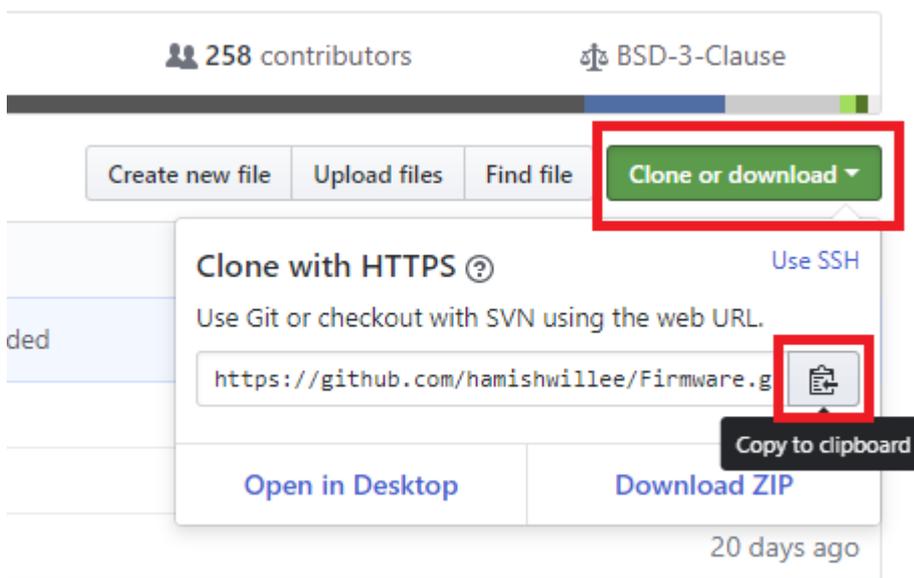
The steps to fork and clone the project source code are:

[Sign up](#) to Github.

Go to the [Firmware](#) repository and click the **Fork** button near the upper right corner. This will create and open the forked repository.



Copy the repository URL for your *Firmware* repository fork. The easiest way to do this is to click the **Clone or download** button and then copy the URL:



Open a command prompt/terminal on your computer

On OS X, hit \mathbb{C} -space and search for 'terminal'.

On Ubuntu, click the launch bar and search for 'terminal'.

On Windows, find the PX4 folder in the start menu and click on 'PX4 Console'.

Clone the repository fork using the copied URL. This will look something like:

```
git clone https://github.com/<youraccountname>/Firmware.git
```

Windows users [refer to the Github help](#). You can use a *git* command line client as above or instead perform the same actions with the *Github for Windows* app.

This will copy *most* of PX4 onto your computer (the rest of the code is automatically fetched from other [git submodules](#) when you build PX4).

If you're just experimenting (and don't want to make any sort of permanent changes) you can simply clone the main Firmware repository as shown:

```
git clone https://github.com/PX4/Firmware.git
cd Firmware
```

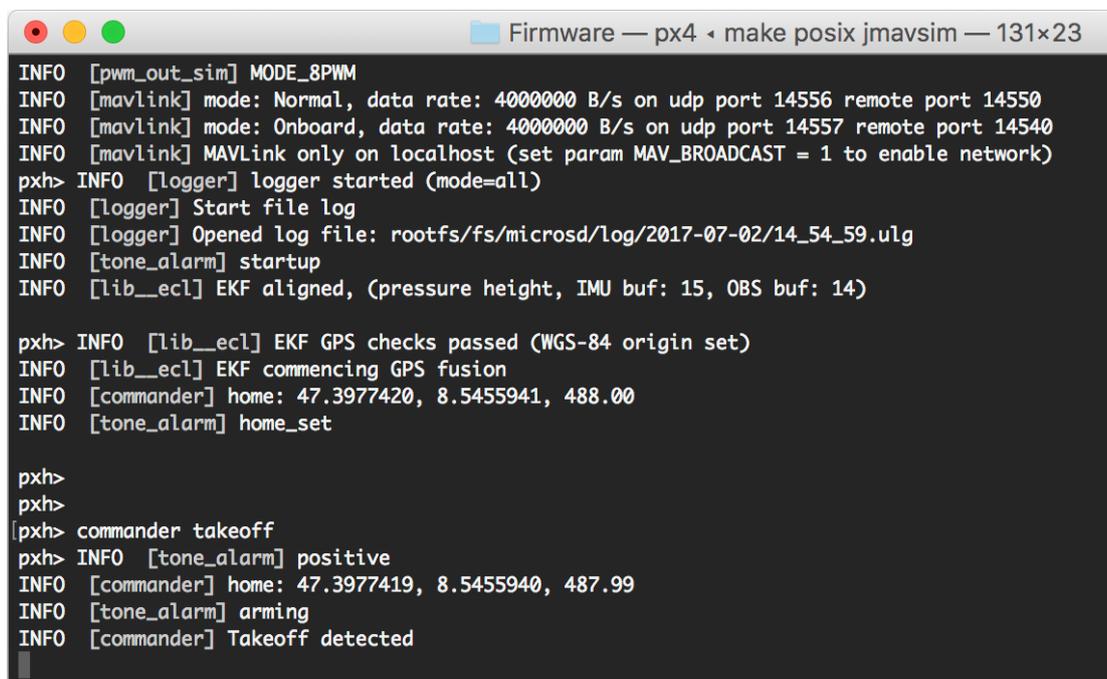
First Build (Using the jMAVSim Simulator)

For the first build we'll build for a simulated target using a console environment. This allows us to validate the system setup before moving on to real hardware and an IDE.

Navigate into the **Firmware** directory and start [jMAVSim](#) using the following command:

```
make posix jmavsim
```

This will bring up the PX4 console below:



```
Firmware — px4 ◀ make posix jmavsim — 131x23
INFO [pwm_out_sim] MODE_8PWM
INFO [mavlink] mode: Normal, data rate: 4000000 B/s on udp port 14556 remote port 14550
INFO [mavlink] mode: Onboard, data rate: 4000000 B/s on udp port 14557 remote port 14540
INFO [mavlink] MAVLink only on localhost (set param MAV_BROADCAST = 1 to enable network)
pxh> INFO [logger] logger started (mode=all)
INFO [logger] Start file log
INFO [logger] Opened log file: rootfs/fs/microsd/log/2017-07-02/14_54_59.u1g
INFO [tone_alarm] startup
INFO [lib_ecl] EKF aligned, (pressure height, IMU buf: 15, OBS buf: 14)

pxh> INFO [lib_ecl] EKF GPS checks passed (WGS-84 origin set)
INFO [lib_ecl] EKF commencing GPS fusion
INFO [commander] home: 47.3977420, 8.5455941, 488.00
INFO [tone_alarm] home_set

pxh>
pxh>
pxh> commander takeoff
pxh> INFO [tone_alarm] positive
INFO [commander] home: 47.3977419, 8.5455940, 487.99
INFO [tone_alarm] arming
INFO [commander] Takeoff detected
```

The drone can be flown by typing:

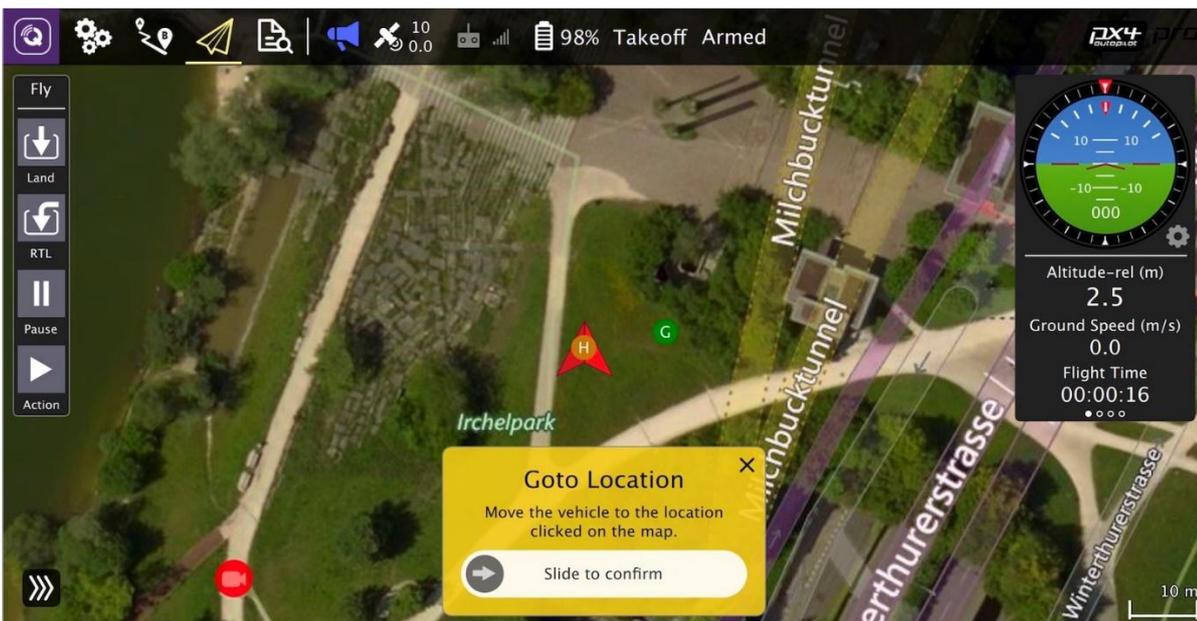
```
pxh> commander takeoff
```



The drone can be landed by typing `commander land` and the whole simulation can be stopped by doing **CTRL+C** (or by entering `shutdown`).

The simulation setup is documented in full detail here: [jMAVSim Simulation](#).

Flying the simulation with the ground control station is closer to the real operation of the vehicle. Click on a location in the map while the vehicle is flying (takeoff flight mode) and enable the slider. This will reposition the vehicle.



NuttX / Pixhawk Based Boards

Building

To build for NuttX- or Pixhawk- based boards, navigate into the **Firmware** directory and then call `make` with the build target for your board.

In the example below the first part of the build target `px4fmu-v2` is the autopilot hardware version and `default` is the configuration name (in this case the "default" configuration). All PX4 build targets follow this logic).

For example, to build for *Pixhawk 1* you would use the following command:

```
cd Firmware
make px4fmu-v2_default
```

A successful run will end with this output:

```
[100%] Linking CXX executable firmware_nuttX
[100%] Built target firmware_nuttX
Scanning dependencies of target build/firmware_px4fmu-v2
[100%] Generating nuttX-px4fmu-v2-default.px4
[100%] Built target build/firmware_px4fmu-v2
```

The following list shows the build commands for common boards:

- [Pixhawk 1](#): `make px4fmu-v2_default`
- [HKPilot32](#): `make px4fmu-v2_default`
- [Pixfalcon](#): `make px4fmu-v2_default`
- [Dropix](#): `make px4fmu-v2_default`
- [mRo Pixhawk](#): `make px4fmu-v3_default` (supports 2MB Flash)
- [mRo X-2.1](#): `make auav-x21_default`
- [Pixhawk 2](#): `make px4fmu-v3_default`
- [Pixracer](#): `make px4fmu-v4_default`
- [MindPX/MindRacer](#): `make mindpx-v2_default`
- [Pixhawk Mini](#): `make px4fmu-v3_default`
- [Pixhawk 3 Pro](#): `make px4fmu-v4pro_default`
- [Crazyflie 2.0](#): `make crazyflie_default`
- [Intel® Aero Ready to Fly Drone](#): `make aerofc-v1_default`
- Pixhawk 4: `make px4fmu-v5_default`
- [AUAV-X2 \(Discontinued\)](#): `make px4fmu-v2_default`

Uploading Firmware (Flashing the board)

Append `upload` to the `make` commands to upload the compiled binary to the autopilot hardware via USB. For example

```
make px4fmu-v2_default upload
```

A successful run will end with this output:

```
Erase : [=====] 100.0%
Program: [=====] 100.0%
Verify : [=====] 100.0%
Rebooting.
```

```
[100%] Built target upload
```

Other Boards

The following boards have more complicated build and/or deployment instructions.

Raspberry Pi 2/3 Boards

The command below builds the target for [Raspberry Pi 2/3 Navio2](#).

Cross-compiler Build

```
cd Firmware
```

```
make posix_rpi_cross # for cross-compiler build
```

The "px4" executable file is in the directory **build/posix_rpi_cross/**. Make sure you can connect to your RPi over ssh, see [instructions how to access your RPi](#).

Then set the IP (or hostname) of your RPi using:

```
export AUTOPILOT_HOST=192.168.X.X
```

And upload it with:

```
cd Firmware
```

```
make posix_rpi_cross upload # for cross-compiler build
```

Then, connect over ssh and run it with (as root):

```
sudo ./px4 px4.config
```

Native Build

If you're building *directly* on the Pi, you will want the native build target (posix_rpi_native).

```
cd Firmware
```

```
make posix_rpi_native # for native build
```

The "px4" executable file is in the directory **build/posix_rpi_native/**. Run it directly with:

```
sudo ./build/posix_rpi_native/px4 ./posix-configs/rpi/px4.config
```

A successful build followed by executing px4 will give you something like this:

```
____ _  _  _  _
|  _  \ \ \ / / / |
| |_/ / \ v / / / |
|  _/ / \ / / / |
| | / / ^ \ \ _ |
\_| \ \ \ \ \ |_/
```

```
px4 starting.
```

```
pxh>
```

Autostart

To autostart px4, add the following to the file **/etc/rc.local** (adjust it accordingly if you use native build), right before the `exit 0` line:

```
cd /home/pi && ./px4 -d px4.config > px4.log
```

Parrot Bebop

Support for the [Parrot Bebop](#) is at an early stage and should be used very carefully.

Build

```
cd Firmware
make posix_bebop_default
```

Turn on your Bebop and connect your host machine with the Bebop's wifi. Then, press the power button four times to enable ADB and to start the telnet daemon.

```
make posix_bebop_default upload
```

This will upload the PX4 mainapp into `/usr/bin` and create the file `/home/root/parameters` if not already present. In addition, we need the Bebop's mixer file and the `px4.config`. Currently, both files have to be copied manually using the following commands.

```
adb connect 192.168.42.1:9050
adb push ROMFS/px4-fmu_common/mixers/bebop.main.mix /home/root
adb push posix-configs/bebop/px4.config /home/root
adb disconnect
```

Run

Connect to the Bebop's wifi and press the power button four times. Next, connect with the Bebop via telnet or adb shell and run the commands bellow.

```
telnet 192.168.42.1
```

Kill the Bebop's proprietary driver with

```
kk
```

and start the PX4 mainapp with:

```
px4 /home/root/px4.config
```

In order to fly the Bebop, connect a joystick device with your host machine and start QGroundControl. Both, the Bebop and the joystick should be recognized. Follow the instructions to calibrate the sensors and setup your joystick device.

Autostart

To auto-start PX4 on the Bebop at boot, modify the init script `/etc/init.d/rcS_mode_default`. Comment the following line:

```
DragonStarter.sh -out2null &
```

Replace it with:

```
px4 -d /home/root/px4.config > /home/root/px4.log
```

Enable adb server by pressing the power button 4 times and connect to adb server as described before:

```
adb connect 192.168.42.1:9050
```

Re-mount the system partition as writeable:

```
adb shell mount -o remount,rw /
```

In order to avoid editing the file manually, you can use this

one : <https://gist.github.com/mhkabir/b0433f0651f006e3c7ac4e1cbd83f1e8>

Save the original one and push this one to the Bebop

```
adb shell cp /etc/init.d/rcS_mode_default /etc/init.d/rcS_mode_default_backup
```

```
adb push rcS_mode_default /etc/init.d/
```

Sync and reboot:

```
adb shell sync
```

```
adb shell reboot
```

OcPoC-Zynq Mini

Build instructions for the [OcPoC-Zynq Mini](#) are covered in:

- [Aerotenna OcPoC-Zynq Mini Flight Controller > Building PX4 for OcPoC-Zynq](#) (PX4 User Guide)
- [OcPoC PX4 Setup Page](#)

QuRT / Snapdragon Based Boards

This section shows how to build for the [Qualcomm Snapdragon Flight](#).

Build

If you use the [Qualcomm ESC board](#) (UART-based), then please follow their instructions [here](#). If you use normal PWM-based ESCs boards, then you may continue to follow the instructions on this page.

The commands below build the targets for the Linux and the DSP side. Both executables communicate via [muORB](#).

```
cd Firmware
make eagle_default
```

To load the SW on the device, connect via USB cable and make sure the device is booted. Run this in a new terminal window:

```
adb shell
```

Go back to previous terminal and upload:

```
make eagle_default upload
```

Note that this will also copy (and overwrite) the two config files [mainapp.config](#) and [px4.config](#) to the device. Those files are stored under `/usr/share/data/adsp/px4.config` and `/home/linaro/mainapp.config` respectively if you want to edit the startup scripts directly on your vehicle.

The mixer currently needs to be copied manually:

```
adb push ROMFS/px4fmu_common/mixers/quad_x.main.mix /usr/share/data/adsp
```

Run

Run the DSP debug monitor:

```
${HEXAGON_SDK_ROOT}/tools/debug/mini-dm/Linux_Debug/mini-dm
```

Note: alternatively, especially on Mac, you can also use [nano-dm](#).

Go back to ADB shell and run px4:

```
cd /home/linaro
./px4 mainapp.config
```

Note that the px4 will stop as soon as you disconnect the USB cable (or if you ssh session is disconnected). To fly, you should make the px4 auto-start after boot.

Autostart

To run the px4 as soon as the Snapdragon has booted, you can add the startup to `rc.local`:

Either edit the file `/etc/rc.local` directly on the Snapdragon:

```
adb shell
vim /etc/rc.local
```

Or copy the file to your computer, edit it locally, and copy it back:

```
adb pull /etc/rc.local
gedit rc.local
adb push rc.local /etc/rc.local
```

For the auto-start, add the following line before `exit 0`:

```
(cd /home/linaro && ./px4 mainapp.config > mainapp.log)
```

```
exit 0
```

Make sure that the `rc.local` is executable:

```
adb shell
chmod +x /etc/rc.local
```

Then reboot the Snapdragon:

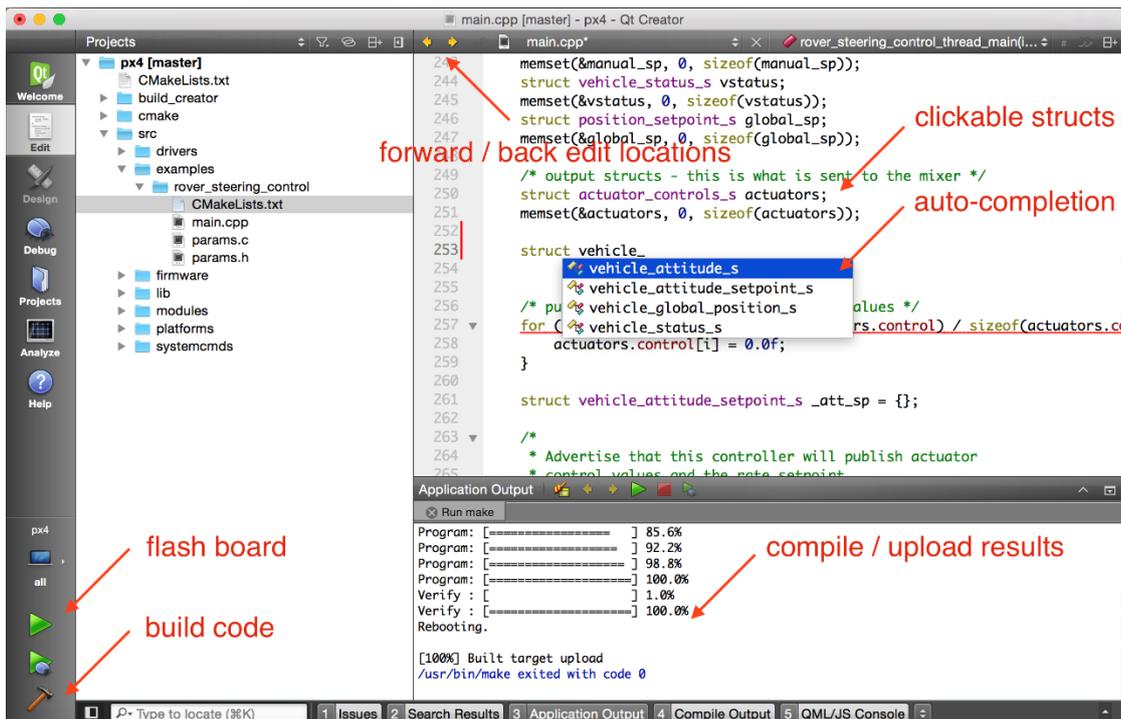
```
adb reboot
```

Compiling in a Graphical IDE

The PX4 system supports Qt Creator, Eclipse and Sublime Text. Qt Creator is the most user-friendly variant and hence the only officially supported IDE. Unless an expert in Eclipse or Sublime, their use is discouraged. Hardcore users can find an [Eclipse project](#) and a [Sublime project](#) in the source tree.

Qt Creator Functionality

Qt creator offers clickable symbols, auto-completion of the complete codebase and building and flashing firmware.



Qt Creator on Linux

Before starting Qt Creator, the [project file](#) needs to be created:

```
cd ~/src/Firmware
mkdir ../Firmware-build
cd ../Firmware-build
cmake ../Firmware -G "CodeBlocks - Unix Makefiles"
```

Then load the CMakeLists.txt in the root firmware folder via File -> Open File or Project -> Select the CMakeLists.txt file.

After loading, the 'play' button can be configured to run the project by selecting 'custom executable' in the run target configuration and entering 'make' as executable and 'upload' as argument.

Qt Creator on Windows

Qt Creator on Mac OS

Before starting Qt Creator, the [project file](#) needs to be created:

```
cd ~/src/Firmware
mkdir -p build/creator
cd build/creator
```

```
cmake ../../.. -G "CodeBlocks - Unix Makefiles"
```

That's it! Start Qt Creator, then complete the steps in the video below to set up the project to build.

2.4 First App Tutorial (Hello Sky)

This tutorial explains in detail how to create and run a new onboard application.

Prerequisites

You will require the following:

- [PX4 SITL Simulator](#) or a [PX4-compatible flight controller](#).
- [PX4 Development Toolchain](#) for the desired target.
- [Download the PX4 Source Code](#) from Github

The source code [Firmware/src/examples/px4_simple_app](#) directory contains a completed version of this tutorial that you can review if you get stuck.

- Rename (or delete) the **px4_simple_app** directory.

Minimal Application

In this section we create a *minimal application* that just prints out `Hello Sky!`. This consists of a single `C` file and a `cmake` definition (which tells the toolchain how to build the application).

1. Create a new directory **Firmware/src/examples/px4_simple_app**.
2. Create a new `C` file in that directory named **px4_simple_app.c**:
 - Copy in the default header to the top of the page. This should be present in all contributed files!

```
○ /*****  
○ *  
○ * Copyright (c) 2012-2016 PX4 Development Team. All rights reserved.  
○ *  
○ * Redistribution and use in source and binary forms, with or without  
○ * modification, are permitted provided that the following conditions  
○ * are met:  
○ *  
○ * 1. Redistributions of source code must retain the above copyright  
○ * notice, this list of conditions and the following disclaimer.  
○ * 2. Redistributions in binary form must reproduce the above copyright
```

```

○ * notice, this list of conditions and the following disclaimer in
○ * the documentation and/or other materials provided with the
○ * distribution.
○ * 3. Neither the name PX4 nor the names of its contributors may be
○ * used to endorse or promote products derived from this software
○ * without specific prior written permission.
○ *
○ * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
○ * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
○ * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
○ * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
○ * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
○ * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
○ * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
○ * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
○ * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
○ * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
○ * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
○ * POSSIBILITY OF SUCH DAMAGE.
○ *
○ *****/

```

○ Copy the following code below the default header. This code style should be used for all files.

```

○ /**
○  * @file px4_simple_app.c
○  * Minimal application example for PX4 autopilot
○  *
○  * @author Example User <mail@example.com>
○  */
○
○ #include <px4_log.h>
○
○ __EXPORT int px4_simple_app_main(int argc, char *argv[]);
○
○ int px4_simple_app_main(int argc, char *argv[])
○ {
○     PX4_INFO("Hello Sky!");
○     return OK;
○ }

```

The main function must be named `<module_name>_main` and exported from the module as shown.

PX4_INFO is the equivalent of `printf` for the PX4 shell (included from `px4_log.h`). There are different log levels: `PX4_INFO`, `PX4_WARN`, `PX4_ERR`, `PX4_DEBUG`. Warnings and errors are additionally added to the [ULog](#) and shown on [Flight Review](#).

3. Create and open a new `cmake` definition file named **CMakeLists.txt**. Copy in the text below:

```
#####  
#  
# Copyright (c) 2015 PX4 Development Team. All rights reserved.  
#  
# Redistribution and use in source and binary forms, with or without  
# modification, are permitted provided that the following conditions  
# are met:  
#  
# 1. Redistributions of source code must retain the above copyright  
# notice, this list of conditions and the following disclaimer.  
# 2. Redistributions in binary form must reproduce the above copyright  
# notice, this list of conditions and the following disclaimer in  
# the documentation and/or other materials provided with the  
# distribution.  
# 3. Neither the name PX4 nor the names of its contributors may be  
# used to endorse or promote products derived from this software  
# without specific prior written permission.  
#  
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE  
# COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,  
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,  
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS  
# OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED  
# AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT  
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN  
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE  
# POSSIBILITY OF SUCH DAMAGE.  
#  
#####  
px4_add_module(  
  MODULE examples__px4_simple_app  
  MAIN px4_simple_app  
  STACK_MAIN 2000  
  SRCS  
    px4_simple_app.c
```

```
DEPENDS
    platforms__common
)
```

The `px4_add_module()` method builds a static library from a module description. The `MAIN` block lists the name of the module - this registers the command with NuttX so that it can be called from the PX4 shell or SITL console.

The `px4_add_module()` format is documented in [Firmware/cmake/common/px4_base.cmake](#).

Build the Application/Firmware

The application is now complete. In order to run it you first need to make sure that it is built as part of PX4. Applications are added to the build/firmware in the appropriate board-level `cmake` file for your target:

- Posix SITL (Simulator): [Firmware/cmake/configs/posix_sitl_default.cmake](#)
- Pixhawk v1/2: [Firmware/cmake/configs/nuttx_px4fmu-v2_default.cmake](#)
- Pixracer: [Firmware/cmake/configs/nuttx_px4fmu-v4_default.cmake](#)
- `cmake` files for other boards can be found in [Firmware/cmake/configs/](#)

To enable the compilation of the application into the firmware create a new line for your application somewhere in the `cmake` file:

```
examples/px4_simple_app
```

The line will already be present for most files, because the examples are included in firmware by default.

Build the example using the board-specific command:

- jMAVSim Simulator: `make posix_sitl_default jmavsim`
- Pixhawk v1/2: `make px4fmu-v2_default`
- Pixhawk v3: `make px4fmu-v4_default`
- Other boards: [Building the Code](#)

Test App (Hardware)

Upload the firmware to your board

Enable the uploader and then reset the board:

- Pixhawk v1/2: `make px4fmu-v2_default upload`
- Pixhawk v3: `make px4fmu-v4_default upload`

It should print before you reset the board a number of compile messages and at the end:

```
Loaded firmware for X,X, waiting for the bootloader...
```

Once the board is reset, and uploads, it prints:

```
Erase : [=====] 100.0%
Program: [=====] 100.0%
Verify : [=====] 100.0%
Rebooting.
```

```
[100%] Built target upload
```

Connect the Console

Now connect to the [system console](#) either via serial or USB. Hitting **ENTER** will bring up the shell prompt:

```
nsh>
```

Type "help" and hit ENTER

```
nsh> help
```

```
help usage: help [-v] [<cmd>]
```

[df	kill	mkfifo	ps	sleep
?	echo	losetup	mkrd	pwd	test
cat	exec	ls	mh	rm	umount
cd	exit	mb	mount	rmdir	unset
cp	free	mkdir	mv	set	usleep
dd	help	mkfatfs	mw	sh	xd

Builtin Apps:

```
reboot
perf
top
..
px4_simple_app
..
sercon
serdis
```

Note that `px4_simple_app` is now part of the available commands. Start it by typing `px4_simple_app` and ENTER:

```
nsh> px4_simple_app
```

```
Hello Sky!
```

The application is now correctly registered with the system and can be extended to actually perform useful tasks.

Test App (SITL)

If you're using SITL the *PX4 console* is automatically started (see [Building the Code > First Build \(Using the jMAVSim Simulator\)](#)). As with the *nsh console* (see previous section) you can type `help` to see the list of built-in apps.

Enter `px4_simple_app` to run the minimal app.

```
pxh> px4_simple_app
INFO [px4_simple_app] Hello Sky!
```

The application can now be extended to actually perform useful tasks.

Subscribing to Sensor Data

To do something useful, the application needs to subscribe inputs and publish outputs (e.g. motor or servo commands).

The benefits of the PX4 hardware abstraction comes into play here! There is no need to interact in any way with sensor drivers and no need to update your app if the board or sensors are updated.

Individual message channels between applications are called [topics](#). For this tutorial, we are interested in the [sensor_combined](#) topic, which holds the synchronized sensor data of the complete system.

Subscribing to a topic is straightforward:

```
#include <uORB/topics/sensor_combined.h>
..
int sensor_sub_fd = orb_subscribe(ORB_ID(sensor_combined));
```

The `sensor_sub_fd` is a topic handle and can be used to very efficiently perform a blocking wait for new data. The current thread goes to sleep and is woken up automatically by the scheduler once new data is available, not consuming any CPU cycles while waiting. To do this, we use the [poll\(\)](#) POSIX system call.

Adding `poll()` to the subscription looks like (*pseudocode, look for the full implementation below*):

```
#include <poll.h>
#include <uORB/topics/sensor_combined.h>
..
int sensor_sub_fd = orb_subscribe(ORB_ID(sensor_combined));

/* one could wait for multiple topics with this technique, just using one here */
px4_pollfd_struct_t fds[] = {
```

```

    { .fd = sensor_sub_fd,    .events = POLLIN },
};

while (true) {
uORB/* wait for sensor update of 1 file descriptor for 1000 ms (1 second) */
uORBint poll_ret = px4_poll(fds, 1, 1000);
..
    if (fds[0].revents & POLLIN) {
        /* obtained data for the first file descriptor */
        struct sensor_combined_s raw;
        /* copy sensors raw data into local buffer */
        orb_copy(ORB_ID(sensor_combined), sensor_sub_fd, &raw);
        PX4_INFO("Accelerometer:\t%8.4f\t%8.4f\t%8.4f",
                (double)raw.accelerometer_m_s2[0],
                (double)raw.accelerometer_m_s2[1],
                (double)raw.accelerometer_m_s2[2]);
    }
}

```

Compile the app again by entering:

```
make
```

Testing the uORB Subscription

The final step is to start your application as a background process/task by typing the following in the nsh shell:

```
px4_simple_app &
```

Your app will display 5 sensor values in the console and then exit:

```

[px4_simple_app] Accelerometer:   0.0483           0.0821           0.0332
[px4_simple_app] Accelerometer:   0.0486           0.0820           0.0336
[px4_simple_app] Accelerometer:   0.0487           0.0819           0.0327
[px4_simple_app] Accelerometer:   0.0482           0.0818           0.0323
[px4_simple_app] Accelerometer:   0.0482           0.0827           0.0331
[px4_simple_app] Accelerometer:   0.0489           0.0804           0.0328

```

The [Firmware/src/examples/px4_daemon_app](#) example shows how to write a daemon (background process) that can be controlled from the command line.

Publishing Data

To use the calculated outputs, the next step is to *publish* the results. Below we show how to publish the attitude topic.

We've chosen `attitude` because we know that the *mavlink* app forwards it to the ground control station - providing an easy way to look at the results.

The interface is pretty simple: initialize the `struct` of the topic to be published and advertise the topic:

```
#include <uORB/topics/vehicle_attitude.h>
..
/* advertise attitude topic */
struct vehicle_attitude_s att;
memset(&att, 0, sizeof(att));
orb_advert_t att_pub_fd = orb_advertise(ORB_ID(vehicle_attitude), &att);
```

In the main loop, publish the information whenever its ready:

```
orb_publish(ORB_ID(vehicle_attitude), att_pub_fd, &att);
```

Full Example Code

The [complete example code](#) is now:

```
/*
 * Copyright (c) 2012-2016 PX4 Development Team. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in
 * the documentation and/or other materials provided with the
 * distribution.
 * 3. Neither the name PX4 nor the names of its contributors may be
 * used to endorse or promote products derived from this software
 * without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
```

```

* AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
* ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF SUCH DAMAGE.
*
*****/

/**
 * @file px4_simple_app.c
 * Minimal application example for PX4 autopilot
 *
 * @author Example User <mail@example.com>
 */

#include <px4_config.h>
#include <px4_tasks.h>
#include <px4_posix.h>
#include <unistd.h>
#include <stdio.h>
#include <poll.h>
#include <string.h>
#include <math.h>

#include <uORB/uORB.h>
#include <uORB/topics/sensor_combined.h>
#include <uORB/topics/vehicle_attitude.h>

__EXPORT int px4_simple_app_main(int argc, char *argv[]);

int px4_simple_app_main(int argc, char *argv[])
{
    PX4_INFO("Hello Sky!");

    /* subscribe to sensor_combined topic */
    int sensor_sub_fd = orb_subscribe(ORB_ID(sensor_combined));
    /* limit the update rate to 5 Hz */
    orb_set_interval(sensor_sub_fd, 200);

    /* advertise attitude topic */
    struct vehicle_attitude_s att;
    memset(&att, 0, sizeof(att));
    orb_advert_t att_pub = orb_advertise(ORB_ID(vehicle_attitude), &att);

    /* one could wait for multiple topics with this technique, just using one here */

```

```

px4_pollfd_struct_t fds[] = {
    { .fd = sensor_sub_fd, .events = POLLIN },
    /* there could be more file descriptors here, in the form like:
    * { .fd = other_sub_fd, .events = POLLIN },
    */
};

int error_counter = 0;

for (int i = 0; i < 5; i++) {
    /* wait for sensor update of 1 file descriptor for 1000 ms (1 second) */
    int poll_ret = px4_poll(fds, 1, 1000);

    /* handle the poll result */
    if (poll_ret == 0) {
        /* this means none of our providers is giving us data */
        PX4_ERR("Got no data within a second");
    } else if (poll_ret < 0) {
        /* this is seriously bad - should be an emergency */
        if (error_counter < 10 || error_counter % 50 == 0) {
            /* use a counter to prevent flooding (and slowing us down) */
            PX4_ERR("ERROR return value from poll(): %d", poll_ret);
        }

        error_counter++;
    } else {

        if (fds[0].revents & POLLIN) {
            /* obtained data for the first file descriptor */
            struct sensor_combined_s raw;
            /* copy sensors raw data into local buffer */
            orb_copy(ORB_ID(sensor_combined), sensor_sub_fd, &raw);
            PX4_INFO("Accelerometer:\t%8.4f\t%8.4f\t%8.4f",
                (double)raw.accelerometer_m_s2[0],
                (double)raw.accelerometer_m_s2[1],
                (double)raw.accelerometer_m_s2[2]);

            /* set att and publish this information for other apps
            the following does not have any meaning, it's just an example
            */
            att.q[0] = raw.accelerometer_m_s2[0];
            att.q[1] = raw.accelerometer_m_s2[1];
        }
    }
}

```

```

    att.q[2] = raw.accelerometer_m_s2[2];

    orb_publish(ORB_ID(vehicle_attitude), att_pub, &att);
}

/* there could be more file descriptors here, in the form like:
 * if (fds[1..n].revents & POLLIN) {}
 */
}
}

PX4_INFO("exiting");

return 0;
}

```

Running the Complete Example

And finally run your app:

```
px4_simple_app
```

If you start *QGroundControl*, you can check the sensor values in the real time plot (**Widgets > Analyze**).

Wrap-Up

This tutorial covered everything needed to develop a "grown up" PX4 autopilot application. Keep in mind that the full list of uORB messages/topics is [available here](#) and that the headers are well documented and serve as reference.

Further information and troubleshooting/common pitfalls can be found here: [uORB](#).

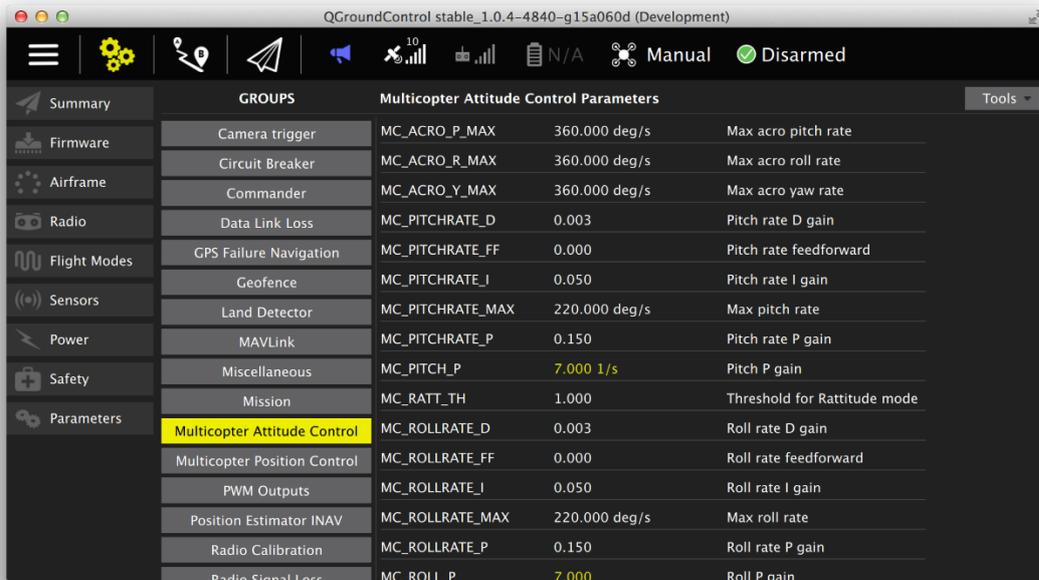
2.5 Advanced Configuration

The advanced system configuration is performed through [QGroundControl](#).

[Download the DAILY BUILD of QGroundControl](#) in regular intervals to stay up to date.

Setting Configuration Parameters

The parameter menu is in the cogwheel tab at the bottom. Parameters are grouped per module, e.g. system parameters or attitude controller parameters. Clicking on the value opens a menu with further explanations and the option to change it.



2.6 Contributing

Contact information for the core dev team and community can be [found here](#). Developers are also most welcome to attend the [weekly dev call](#) and other [developer events](#).

Support

Weekly Dev Call

The PX4 dev team syncs up on platform technical details and in-depth analysis. There is also space in the agenda to discuss pull requests, major impacting issues and Q&A.

Who should attend:

- Core project maintainers
- Component maintainers
- Test team lead
- Dronecode members
- Community members

The dev call is open to all interested developers (not just the core dev team). This is a great opportunity to meet the team and contribute to the ongoing development of the platform.

Schedule

- TIME: Wednesday 5PM CET, 11AM EST, 8AM PST ([subscribe to calendar](#))
- **Join the call:** <https://zoom.us/j/625711763>
- **Meeting ID:** 625 711 763
- **Dial(for higher quality, dial a number based on your current location):**
 - **Switzerland:** +41 (0) 31 528 0988
 - **US:** +1 646 876 9923 or +1 669 900 6833 or +1 408 740 3766
 - **Germany:** +49 (0) 30 3080 6188
 - **Mexico:** +52 554 161 4288
 - **Australia:** +61 (0) 2 8015 2088
 - **United Kingdom:** +44 (0) 20 3695 0088
 - **South Korea:** +82 (0) 2 6022 2322
 - **Spain:** +34 91 198 0188
 - [International numbers available](#)
- Agenda is published before the call on [PX4 Discuss - weekly-dev-call](#)
- To nominate Issues and PRs for the call you can use the [devcall](#) label to flag them for discussion.
-

Tests Flights

Test flights are important for quality assurance. The Dronecode test team can help review (test flight) your pull requests and provide feedback and logs.

How to request test flights

- Assign the test team [@PX4/testflights](#) as a reviewer to the pull request or issue
- Add a complete and thorough description of your changes
- Wait for feedback from the test team
- The test team will [add your PR/issue to their queue](#)

Response times

- Multi-Copter: up to 48 hours (typically within 24 hours)
- VTOL, Fixed Wing: up to 4 days (typically 2 days)

Have a problem?

Help diagnosing problems

If you are unsure what the problem is and you need help diagnosing

- Upload logs to [Flight Log Review](#)
- Open a discussion on [PX4 Discuss](#) with a flight report and links to logs.
- If you find an issue or bug with PX4 [open a Github Issue](#)

Issue & Bug reporting

- Upload logs to [Flight Log Review](#)
- [Open a Github Issue](#) with a flight report with as much detail as possible and links to logs.

General support

- [Join our Slack community](#)
- [Open a discussion](#)
- [Open Github Issue](#)

Calendar & Events

The *Dronecode Calendar* shows important events for platform developers and users. Select the links below to display the calendar in your timezone (and to add it to your own calendar):

- [Switzerland – Zurich](#)
- [Pacific Time – Tijuana](#)
- [Australia – Melbourne/Sydney/Hobart](#)

Note: calendar defaults to CET.

Contribution

Code of Conduct

We pledge to adhere to the [PX4 code of conduct](#). This code aims to foster an open and welcoming environment.

Source Code Management

The PX4 project uses a three-branch Git branching model:

- [master](#) is by default unstable and sees rapid development.
- [beta](#) has been thoroughly tested. It's intended for flight testers.
- [stable](#) points to the last release.

We try to retain a [linear history through rebases](#) and avoid the [Github flow](#). However, due to the global team and fast moving development we might resort to merges at times.

To contribute new functionality, [sign up for Github](#), then [fork](#) the repository, [create a new branch](#), add your changes, and finally [send a pull request](#). Changes will be merged when they pass our [continuous integration](#) tests.

All code contributions have to be under the permissive [BSD 3-clause license](#) and all code must not impose any further constraints on the use.

Code Style Formatting

PX4 uses [astyle](#) for code formatting. Valid versions are

- [astyle 2.06](#) (recommended)
- [astyle 3.0](#)
- [astyle 3.01](#)

Once installed, formatting can be checked with `./Tools/astyle/check_code_style_all.sh`. The output should be `Format checks passed` on a clean master. If that worked, `make format` can be used in the future to check and format all files automatically.

Commits and Commit Messages

Please use descriptive, multi-paragraph commit messages for all non-trivial changes. Structure them well so they make sense in the one-line summary but also provide full detail.

```
Component: Explain the change in one sentence. Fixes #1234
```

```
Prepend the software component to the start of the summary line, either by the module name or a description of it. (e.g. "mc_att_ctrl" or "multicopter attitude controller").
```

```
If the issue number is appended as <Fixes #1234>, Github will automatically close the issue when the commit is merged to the master branch.
```

```
The body of the message can contain several paragraphs. Describe in detail what you changed. Link issues and flight logs either related to this fix or to the testing results of this commit.
```

```
Describe the change and why you changed it, avoid to paraphrase the code change (Good: "Adds an additional safety check for vehicles with low quality GPS reception". Bad: "Add gps_reception_check() function").
```

```
Reported-by: Name <email@px4.io>
```

Use `git commit -s` **to sign off on all of your commits.** This will add `signed-off-by:` with your name and email as the last line.

This commit guide is based on best practices for the Linux Kernel and other [projects maintained](#) by Linus Torvalds.

2.6.1 GIT Examples

Contributing code to PX4

Adding a feature to PX4 follows a defined workflow. In order to share your contributions on PX4, you can follow this example.

-
- [Sign up](#) for github if you haven't already
 - Fork the Firmware (see [here](#))
 - Clone your forked repository to your local computer

```
cd ~/wherever/  
git clone https://github.com/<your git name>/Firmware.git
```

- Go into the new directory, initialize and update the submodules, and add the original upstream Firmware

```
cd Firmware  
git submodule update --init --recursive  
git remote add upstream https://github.com/PX4/Firmware.git
```

- You should have now two remote repositories: One repository is called upstream that points to the PX4 Firmware, and one repository that points to your forked repository of the PX4 repository.
- This can be checked with the following command:

```
git remote -v
```

- Make the changes that you want to add to the current master.
- Create a new branch with a meaningful name that represents your feature

```
git checkout -b <your feature branch name>
```

you can use the command `git branch` to make sure you're on the right branch.

- Add your changes that you want to be part of the commit by adding the respective files

```
git add <file name>
```

If you prefer having a GUI to add your files see [Gitk](#) or `git add -p`.

- Commit the added files with a meaningful message explaining your changes

```
git commit -m "<your commit message>"
```

For a good commit message, please refer to [Contributing](#) section.

-
- Some time might have passed and the [upstream master](#) has changed. PX4 prefers a linear commit history and uses [git rebase](#). To include the newest changes from upstream in your local branch, switch to your master branch

```
git checkout master
```

Then pull the newest commits from upstream master

```
git pull upstream master
```

Now your local master is up to date. Switch back to your feature branch

```
git checkout <your feature branch name>
```

and rebase on your updated master

```
git rebase master
```

- Now you can push your local commits to your forked repository

```
git push origin <your feature branch name>
```

- You can verify that the push was successful by going to your forked repository in your browser: <https://github.com/<your git name>/Firmware.git>
There you should see the message that a new branch has been pushed to your forked repository.
- Now it's time to create a pull request (PR). On the right hand side of the "new branch message" (see one step before), you should see a green button saying "Compare & Create Pull Request". Then it should list your changes and you can (must) add a meaningful title (in case of a one commit PR, it's usually the commit message) and message (**explain what you did for what reason**. Check [other pull requests](#) for comparison)
- You're done! Responsible members of PX4 will now have a look at your contribution and decide if they want to integrate it. Check if they have questions on your changes every once in a while.

Update Submodule

There are several ways to update a submodule. Either you clone the repository or you go in the submodule directory and follow the same procedure as in [Contributing code to PX4](#).

Do a PR for a submodule update

This is required after you have done a PR for a submodule X repository and the bug-fix / feature-add is in the current master of submodule X. Since the Firmware still points to a commit before your update, a submodule pull request is required such that the submodule used by the Firmware points to the newest commit.

```
cd Firmware
```

- Make a new branch that describes the fix / feature for the submodule update:

```
git checkout -b pr-some-fix
```

- Go to submodule subdirectory

```
cd <path to submodule>
```

- PX4 submodule might not necessarily point to the newest commit. Therefore, first checkout master and pull the newest upstream code.

```
git checkout master  
git pull upstream master
```

- Go back to Firmware directory, and as usual add, commit and push the changes.

```
cd -  
git add <path to submodule>  
git commit -m "Update submodule to include ..."  
git push upstream pr-some-fix
```

Checkout pull requests

You can test someone's pull request (changes are not yet merged) even if the branch to merge only exists on the fork from that person. Do the following

```
git fetch upstream pull/<PR ID>/head:<branch name>
```

PR ID is the number right next to the PR's title (without the #) and the `<branch name>` can also be found right below the PR ID, e.g. `<the other persons git name>:<branch name>`. After that you can see the newly created branch locally with

```
git branch
```

Then switch to that branch

```
git checkout <branch name>
```

Common pitfalls

Force push to forked repository

After having done the first PR, people from the PX4 community will review your changes. In most cases this means that you have to fix your local branch according to the review. After changing the files locally, the feature branch needs to be rebased again with the most recent upstream/master. However, after the rebase, it is no longer possible to push the feature branch to your forked repository directly, but instead you need to use a force push:

```
git push --force-with-lease origin <your feature branch name>
```

Rebase merge conflicts

If a conflict occurs during a `git rebase`, please refer to [this guide](#).

Pull merge conflicts

If a conflict occurs during a `git pull`, please refer to [this guide](#).

2.6.2 Contributing to Documentation

Contributions to the Dronecode guides, including the PX4 developer and user guides are very welcome! This article explains how you can make changes, add content, and create translations.

You will need a (free) [Github](#) account to contribute to the guide.

Quick Changes

Fixing typos or editing an *existing page* is easy:

1. Click the **Edit** toolbar icon at the top of the relevant page in the guide.



This will open the page for editing (in Github).

2. Make the desired change.
3. At the bottom of the page you'll be prompted to create a separate branch and then guided to submit a *pull request*.

The documentation team reviews submitted pull requests and will either merge it or work with you to update it.

Adding New Content - Big Changes

What Goes Where?

The *Developer Guide* is for documentation that is relevant to *software developers*. This includes users who need to:

- Add or modify platform features - modules, flight modes, etc.
- Add support/integrate with new hardware - flight controllers, peripherals, airframes, etc.
- Communicate with the platform from an external source - e.g. a companion computer.
- Understand the architecture

The *User Guide*, by contrast, is *primarily* for users who want to:

- Fly a vehicle using PX4
- Build, modify, or configure a vehicle using PX4 on a supported/existing airframe.

For example, detailed information about how to build/configure an existing airframe are in the User Guide, while instructions for defining a *new* airframe are in the Developer Guide.

Gitbook Documentation Toolchain

The guide uses the [Gitbook](#) toolchain. Change requests can be either done on the Gitbook website using the [Gitbook editor](#) or locally (more flexible, but less user-friendly).

Everything you need to install and build Gitbook locally is explained in the [toolchain documentation](#). In overview:

- Pages are written in separate files using markdown (almost the same syntax used by Github wiki).
- The *structure* of the book is defined in a file named **SUMMARY.md**.
- This is a [multilingual](#) book, so there is a **LANGS.md** file in the root directory defining what languages are supported. Pages for each language are stored in the folder named for the associated language code (e.g. "zh" for Chinese, "en" for English).
- A file named **book.json** defines any dependencies of the build.
- A web hook is used to track whenever files are merged into the master branch on this repository, causing the book to rebuild.

Style guide

1. Files/file names

- Put new files in an appropriate sub-folder
- Use descriptive names. In particular, image filename should describe what they contain.
- Use lower case and separate words using underscores "_"

2. Images

- Use the smallest size and lowest resolution that makes the image still useful.
- New images should be created in a sub-folder of **/assets/** by default (so they can be shared between translations).

3. Content:

- Use "style" (bold, emphasis, etc) consistently. **Bold** for button presses and menu definitions. *Emphasis* for tool names. Otherwise use as little as possible.
- Headings and page titles should use "First Letter Capitalisation"
- The page title should be a first level heading (#). All other headings should be h2 (##) or lower.
- Don't add any style to headings.
- Don't translate the *first part* of a note, tip or warning declaration (e.g. > ****Note****) as this precise text is required to render the note properly.

Translations

We have recently started adding translated versions of all the PX4/Dronecode guides! If you would like to help, contact us on [our support channels](#).

Gitbook supports translation [as described here](#):

- Each language is independent and keeps all its documents in its own directory (named using it's international code - "en" for English, "es" for Spanish, etc.)

-
- The **LANGS.md** file in the root directory lists the language folders that Gitbook must build.

In order to keep all language-versions of the guide up to date and synchronised, we have the following policy/guidelines:

- This is an **English-first** book.
- Any *technical* changes should be made in the English tree *first* (including both updates and new pages). After the English change has been accepted in the main repo then submit the translation. This approach ensures that changes will propagate through to all the other translations!
- Improvements to translations that don't change "technical information" won't need to be made in the English tree.
- The structure and documents should be the same for all languages (i.e. based on the English version).
- All languages share the same images by default (do not duplicate the */image* folder, unless you're changing/translating the image).
- Translation changes are submitted to the repo in the same way as any other changes (fork the repo, make a branch for your changes, and create PRs of the branches to submit them into this repo).
- Translation teams can organise themselves however they like as long as PRs are submitted using the above approach.

Starting a new language translation

The process straightforward:

1. Fork the documentation repo.
2. Create and checkout a new branch for your language.

```
checkout -b add_translation_language_yourlanguagename
```

3. Copy the whole English folder (*/en*) and rename it to the appropriate language code (e.g. "es" for Spanish).

This ensures that you keep the same structure and documents as the original version.

4. Update **LANGS.md** with your language.
5. Translate the content in your language tree.

Minimally complete the home page and the **SUMMARY.md** before submitting any PR request. Ideally do more!

6. Commit the changes and push them back to your own fork repo.

```
git add *
git commit -m "Created a your_new_language translation"
git push origin add_translation_language_yourlanguagename
```

7. On the Github interface, create a PR to submit your branch back to the master repo (a banner appears on Github that you can click when you visit the repo).

Updating translations

Translations can be updated like any other change to documentation: fork the repo, create a branch for your changes in your fork, then submit them back to the main repo as PRs.

Tracking changes

We hope that translation owners will track changes in the English version and propagate them through to their translations.

Git/Github have excellent mechanisms for tracking changes. We recommend that when you add [document front matter](#) to your translation with the commit information for the page you translated. This allows anyone to go back later and find out whether the text has changed since it was last translated. For example:

```
---
translated_page:
https://github.com/PX4/Devguide/blob/master/en/setup/config_initial.md
translated_sha: 95b39d747851dd01c1fe5d36b24e59ec865e323e
translated: false
---
```

The *translated_sha* is the full SHA of the commit that you translated. Find this by opening the source page on github, press the **History** button. Find the commit of the document you are translating from (ideally the most recent) and press the "Copy the full SHA" icon associated with that commit.

Licence

All PX4/Dronecode documentation is free to use and modify under terms of the permissive [CC BY 4.0](#) licence.

2.7 Terminology

The following terms, symbols, and decorators are used in text and diagrams throughout this guide.

Notation

- Bold face variables indicate vectors or matrices and non-bold face variables represent scalars.
- The default frame for each variable is the local frame \mathcal{L} . Right [superscripts](#) represent the coordinate frame. If no right superscript is present, then the default frame is assumed. An exception is given by Rotation Matrices, where the lower right subscripts indicates the current frame and the right superscripts the target frame.
- Variables and subscripts can share the same letter, but they always have different meaning.

Acronyms

Acronym	Expansion
AOA	Angle Of Attack. Also named <i>alpha</i> .
AOS	Angle Of Sideslip. Also named <i>beta</i> .
FRD	Coordinate system where the X-axis is pointing towards the Front of the vehicle, the Y-axis is pointing Right and the Z-axis is pointing Down, completing the right-hand rule.
FW	Fixed-Wing.
MC	MultiCopter.
MPC or MCPC	MultiCopter Position Controller. MPC is also used for Model Predictive Control.
NED	Coordinate system where the X-axis is pointing towards the true North, the Y-axis is pointing East and the Z-axis is pointing Down, completing the right-hand rule.
PID	Controller with Proportional, Integral and Derivative actions.

Symbols

Variable	Description
	Translation along coordinate axis x,y and z respectively.
	Position vector .
	Velocity vector .
	Acceleration vector .
	Angle of attack (AOA).
	Wing span (from tip to tip).
	Wing area.
	Aspect ratio. .
	Angle of sideslip (AOS).
	Wing chord length.
	Aerodynamic control surface angular deflection. A positive deflection generates a negative moment.
	Euler angles roll (=Bank), pitch and yaw (=Heading).
	Attitude vector. .
	Forces along coordinate axis x,y and z.
	Force vector .
	Drag force.
	Cross-wind force.

Variable	Description
	Lift force.
	Gravity.
	Moments around coordinate axis x,y and z.
	Moment vector .
	Mach number. Can be neglected for scale aircrafts.
	Vector part of Quaternion.
	Hamiltonian attitude quaternion. . describes the attitude relative to the local frame . To represent a vector in local frame given a vector in body frame, the following operation can be used: (or instead of if is not unitary). represents a <i>quaternionized</i> vector: .
	Rotation matrix. Rotates a vector from frame to frame . .
	Leading-edge sweep angle.
	Taper ratio .
	Wind velocity.
	Angular rates around body axis x,y and z.
	Angular rate vector in body frame .
	General state vector.

Superscripts / Indices

Superscripts / Indices	Description
	Aileron.
	Elevator.
	Rudder.
	Aerodynamic.
	Thrust force.
	Relative airspeed.
	Component of vector along coordinate axis x, y and z.
	Component of vector along global north, east and down direction.

Superscripts / Indices

Superscripts / Indices	Description
	Local-frame. Default for PX4 related variables.
	Body-frame.
	Wind-frame.

Decorators

Decorator	Description
	Complex conjugate.
	Time derivative.
	Estimate.
	Mean.
	Matrix inverse.
	Matrix transpose.
	Quaternion.

2.8 Licenses

All code contributions must be made under the permissive [BSD 3-clause license](#) and must not impose any further constraints on its use.

This page documents the licenses of various components in the system.

- [PX4 Flight Stack](#) — BSD
- [PX4 Middleware](#) — BSD
- [Pixhawk Hardware](#) — CC-BY-SA 3.0
- Snapdragon Hardware — Qualcomm Proprietary
- Documentation
 - [PX4 Development Guide](#) — [CC BY 4.0](#).
 - [PX4 User Guide](#) — [CC BY 4.0](#).

3. Concepts

This section contains topics about the [PX4 System Architecture](#) and other core concepts.

3.1 PX4 Architectural Overview

PX4 consists of two main layers: the [flight stack](#) is an estimation and flight control system, and the [middleware](#) is a general robotics layer that can support any type of autonomous robot, providing internal/external communications and hardware integration.

All PX4 [airframes](#) share a single codebase (this includes other robotic systems like boats, rovers, submarines etc.). The complete system design is [reactive](#), which means that:

- All functionality is divided into exchangeable and reusable components
- Communication is done by asynchronous message passing
- The system can deal with varying workload

High-Level Software Architecture

The diagram below provides a detailed overview of the building blocks of PX4. The top part of the diagram contains middleware blocks, while the lower section shows the components of the flight stack.

The source code is split into self-contained modules/programs (shown in `monospace` in the diagram). Usually a building block corresponds to exactly one module.

At runtime, you can inspect which modules are executed with the `top` command in shell, and each module can be started/stopped individually via `<module_name> start/stop`. While `top` command is specific to NuttX shell, the other commands can be used in the SITL shell (`pxh>`) as well. For more information about each of these modules see the [Modules & Commands Reference](#).

The arrows show the information flow for the *most important* connections between the modules. In reality, there are many more connections than shown, and some data (e.g. for parameters) is accessed by most of the modules.

Modules communicate with each other through a publish-subscribe message bus named [uORB](#). The use of the publish-subscribe scheme means that:

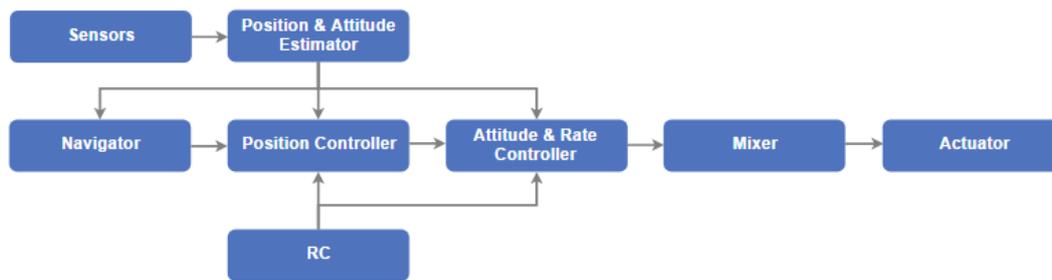
-
- The system is reactive — it is asynchronous and will update instantly when new data is available
 - All operations and communication are fully parallelized
 - A system component can consume data from anywhere in a thread-safe fashion

This architecture allows every single one of these blocks to be rapidly and easily replaced, even at runtime.

Flight Stack

The flight stack is a collection of guidance, navigation and control algorithms for autonomous drones. It includes controllers for fixed wing, multirotor and VTOL airframes as well as estimators for attitude and position.

The following diagram shows an overview of the building blocks of the flight stack. It contains the full pipeline from sensors, RC input and autonomous flight control (Navigator), down to the motor or servo control (Actuators).



An **estimator** takes one or more sensor inputs, combines them, and computes a vehicle state (for example the attitude from IMU sensor data).

A **controller** is a component that takes a setpoint and a measurement or estimated state (process variable) as input. Its goal is to adjust the value of the process variable such that it matches the setpoint. The output is a correction to eventually reach that setpoint. For example the position controller takes position setpoints as inputs, the process variable is the currently estimated position, and the output is an attitude and thrust setpoint that move the vehicle towards the desired position.

A **mixer** takes force commands (e.g. turn right) and translates them into individual motor commands, while ensuring that some limits are not exceeded. This translation is specific for a vehicle type and depends on various factors, such as the motor arrangements with respect to the center of gravity, or the vehicle's rotational inertia.

Middleware

The [middleware](#) consists primarily of device drivers for embedded sensors, communication with the external world (companion computer, GCS, etc.) and the uORB publish-subscribe message bus.

In addition, the middleware includes a [simulation layer](#) that allows PX4 flight code to run on a desktop operating system and control a computer modeled vehicle in a simulated "world".

Update Rates

Since the modules wait for message updates, typically the drivers define how fast a module updates. Most of the IMU drivers sample the data at 1kHz, integrate it and publish with 250Hz. Other parts of the system, such as the `navigator`, don't need such a high update rate, and thus run considerably slower.

The message update rates can be [inspected](#) in real-time on the system by running `uorb top`.

Runtime Environment

PX4 runs on various operating systems that provide a POSIX-API (such as Linux, macOS, NuttX or QuRT). It should also have some form of real-time scheduling (e.g. FIFO).

The inter-module communication (using [uORB](#)) is based on shared memory. The whole PX4 middleware runs in a single address space, i.e. memory is shared between all modules.

The system is designed such that with minimal effort it would be possible to run each module in separate address space (parts that would need to be changed include `uORB`, `parameter interface`, `dataman` and `perf`).

There are 2 different ways that a module can be executed:

- **Tasks:** The module runs in its own task with its own stack and process priority (this is the more common way).
- **Work queues:** The module runs on a shared task, meaning that it does not own a stack. Multiple tasks run on the same stack with a single priority per work queue.

A task is scheduled by specifying a fixed time in the future. The advantage is that it uses less RAM, but the task is not allowed to sleep or poll on a message.

Work queues are used for periodic tasks, such as sensor drivers or the land detector.

Tasks running on a work queue do not show up in `top` (only the work queues themselves can be seen - e.g. as `lpwork`).

Background Tasks

`px4_task_spawn_cmd()` is used to launch new tasks (NuttX) or threads (POSIX - Linux/macOS) that run independently from the calling (parent) task:

```
independent_task = px4_task_spawn_cmd(
    "commander",           // Process name
    SCHED_DEFAULT,        // Scheduling type (RR or FIFO)
    SCHED_PRIORITY_DEFAULT + 40, // Scheduling priority
    3600,                 // Stack size of the new task or thread
    commander_thread_main, // Task (or thread) main function
    (char * const *)&argv[0] // Void pointer to pass to the new task
    // (here the cmdline arguments).
);
```

OS-Specific Information

NuttX

[NuttX](#) is the primary RTOS for running PX4 on a flight-control board. It is open source (BSD license), light-weight, efficient and very stable.

Modules are executed as tasks: they have their own file descriptor lists, but they share a single address space. A task can still start one or more threads that share the file descriptor list.

Each task/thread has a fixed-size stack, and there is a periodic task which checks that all stacks have enough free space left (based on stack coloring).

Linux/macOS

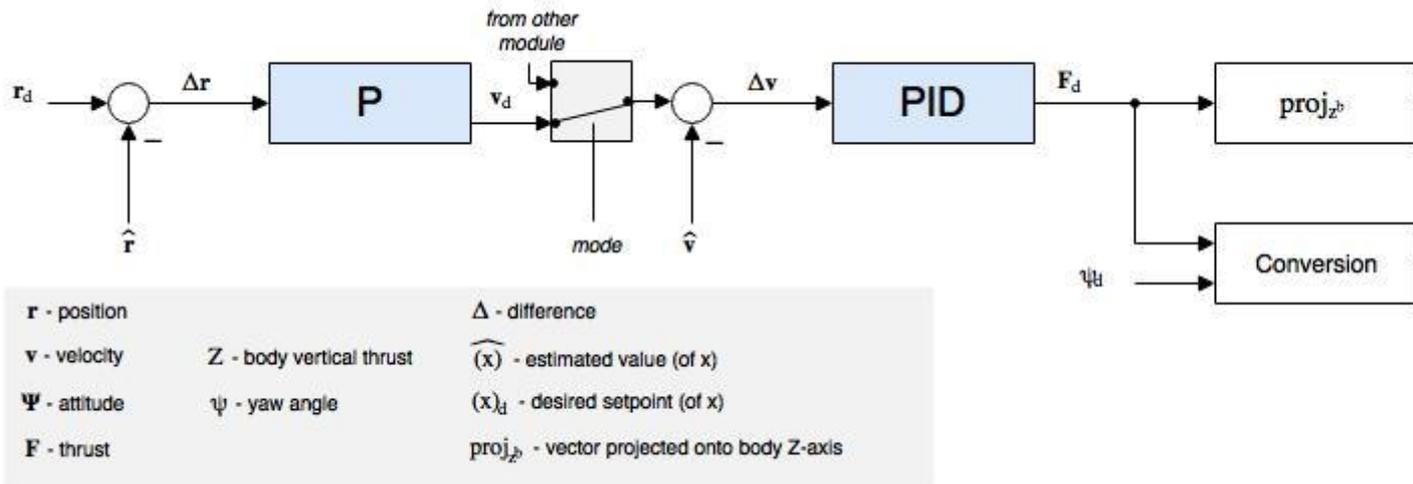
On Linux or macOS, PX4 runs in a single process, and the modules run in their own threads (there is no distinction between tasks and threads as on NuttX).

Controller Diagrams

This section contains diagrams for the main PX4 controllers.

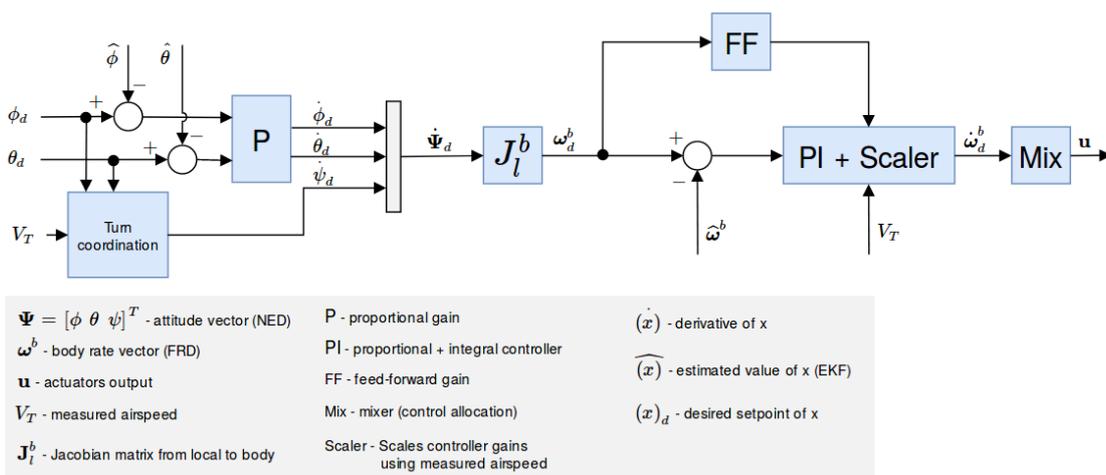
The diagrams use the standard [PX4 notation](#) (and each have an annotated legend).

Multicopter Position Controller



- Estimates come from [EKF2](#).
- This is a standard cascaded position-velocity loop.
- Depending on the mode, the outer (position) loop is bypassed (shown as a multiplexer after the outer loop). The position loop is only used when holding position or when the requested velocity in an axis is null.
- The integrator in the inner loop (velocity) controller includes an anti-reset windup (ARW) using a clamping method.

Fixed-Wing Attitude Controller



The attitude controller works using a cascaded loop method. The outer loop computes the error between the attitude setpoint and the estimated attitude that, multiplied by a gain (P controller), generates a rate setpoint. The inner loop then computes the error in

rates and uses a PI (proportional + integral) controller to generate the desired angular acceleration.

The angular position of the control effectors (ailerons, elevators, rudders, ...) is then computed using this desired angular acceleration and a priori knowledge of the system through control allocation (also known as mixing). Furthermore, since the control surfaces are more effective at high speed and less effective at low speed, the controller - tuned for cruise speed - is scaled using the airspeed measurements (if such a sensor is used).

If no airspeed sensor is used then gain scheduling for the FW attitude controller is disabled (it's open loop); no correction is/can be made in TECS using airspeed feedback.

The feedforward gain is used to compensate for aerodynamic damping. Basically, the two main components of body-axis moments on an aircraft are produced by the control surfaces (ailerons, elevators, rudders, ... - producing the motion) and the aerodynamic damping (proportional to the body rates - counteracting the motion). In order to keep a constant rate, this damping can be compensated using feedforward in the rate loop.

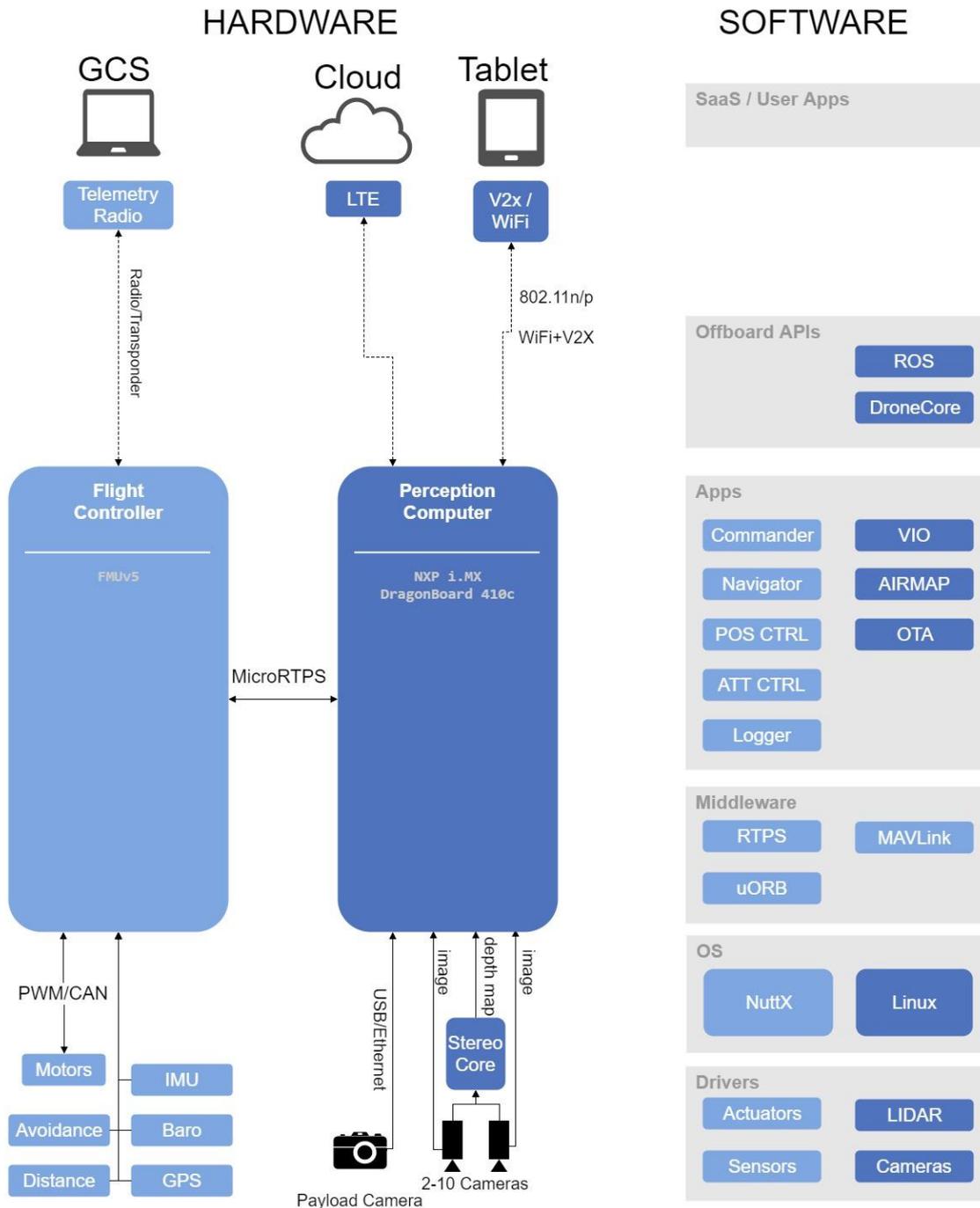
The roll and pitch controllers have the same structure and the longitudinal and lateral dynamics are assumed to be uncoupled enough to work independently. The yaw controller, however, generates its yaw rate setpoint using the turn coordination constraint in order to minimize lateral acceleration, generated when the aircraft is slipping. The yaw rate controller also helps to counteract adverse yaw effects (https://youtu.be/sNV_SDDxuWk) and to damp the [Dutch roll mode](#) by providing extra directional damping.

3.2 Dronecode Platform Hardware/Software Architecture

The diagram below provides a forward-looking high level overview of the [Dronecode Platform](#). The left hand side shows one possible hardware configuration with a *flight controller* (light blue) connected to a *perception computer* (dark blue) via [RTPS](#). The perception computer provides vision control and object avoidance using a camera sensor array, and has a separate payload camera.

The right hand side of the diagram shows the end-to-end software stack. The stack "approximately" aligns horizontally with the hardware parts of the diagram, and is colour coded to show which software is running on the flight controller and which on the companion computer.

The [PX4 Architectural Overview](#) provides information about the flight stack and middleware. Offboard APIs are covered in [ROS](#) and [DroneCore](#).



3.3 Flight Modes

Flight Modes define how the autopilot responds to user input and controls vehicle movement. They are loosely grouped into *manual*, *assisted* and *auto* modes, based on the level/type of control provided by the autopilot. The pilot transitions between flight modes using switches on the remote control or with a [ground control station](#).

Not all flight modes are available on all vehicle types, and some modes behave differently on different vehicle types (as described below). Finally, some flight modes

make sense only under specific pre-flight and in-flight conditions (e.g. GPS lock, airspeed sensor, vehicle attitude sensing along an axis). The system will not allow transitions to those modes until the right conditions are met.

The sections below provide an overview of the modes, followed by a [flight mode evaluation diagram](#) that shows the conditions under which PX4 will transition into a new mode.

Flight Mode Summary

Manual flight modes

"Manual" modes are those where the user has direct control over the vehicle via the RC control (or joystick). Vehicle movement always follows stick movement, but the level/type of response changes depending on the mode. For example, experienced fliers can use modes that provide direct passthrough of stick positions to actuators, while beginners will often choose modes that are less responsive to sudden stick-position changes.

- **Fixed wing aircraft/ rovers / boats:**
 - **MANUAL:** The pilot's control inputs (raw user inputs from RC transmitter) are passed directly to the output mixer.
 - **STABILIZED:** The pilot's inputs are passed as roll and pitch *angle* commands and a manual yaw command. If the RC roll and pitch sticks are centered, the autopilot regulates the roll and pitch angles to zero, hence stabilizing (leveling-out) the attitude against any wind disturbances. However, in this mode the position of the aircraft is not controlled by the autopilot, hence the position can drift due to wind.
- **Multirotors:**
 - **MANUAL:** The pilot's control inputs (raw user inputs from RC transmitter) are passed directly to the output mixer.
 - **ACRO:** The pilot's inputs are passed as roll, pitch, and yaw *rate* commands to the autopilot. The autopilot controls the angular rates, but not the attitude. Hence, if the RC sticks are centered the vehicle will not level-out. This allows the multirotor to become completely inverted. Throttle is passed directly to the output mixer.
 - **RATTITUDE** The pilot's inputs are passed as roll, pitch, and yaw *rate* commands to the autopilot if they are greater than the mode's threshold, i.e. if the RC sticks are a certain distance away from the center position. If not the inputs are passed as roll and pitch *angle* commands and a yaw *rate* command. Throttle is passed directly to the output mixer. In short, the autopilot acts as an angular rate controller when the RC sticks are away from center (like in the ACRO mode), whereas when the RC sticks are centered, the autopilot acts as an attitude controller (like in the Stabilized mode).
 - **STABILIZED** The pilot's inputs are passed as roll and pitch *angle* commands and a yaw *rate* command. Throttle is passed directly to the output mixer. The autopilot controls

the attitude, meaning it regulates the roll and pitch angles to zero when the RC sticks are centered, consequently leveling-out the attitude. However, in this mode the position of the vehicle is not controlled by the autopilot, hence the position can drift due to wind.

Assisted flight modes

"Assisted" modes are also user controlled but offer some level of "automatic" assistance - for example, automatically holding position/direction, against wind. Assisted modes often make it much easier to gain or restore controlled flight.

- **ALTCTL** (Altitude Control)
 - **Fixed wing aircraft:** When the roll, pitch and yaw (RPY) RC sticks are all centered (or less than some specified deadband range) the aircraft will return to straight and level flight and keep its current altitude. Its x and y position will drift with the wind.
 - **Multirotors:** Roll, pitch and yaw inputs are as in Stabilised mode. Throttle inputs indicate climb or sink at a predetermined maximum rate. Throttle has large deadzone. Centered Throttle holds altitude steady. The autopilot only controls altitude so the x,y position of the vehicle can drift due to wind.
- **POSCTL** (Position Control)
 - **Fixed wing aircraft:** Neutral inputs (centered RC sticks) give level flight and it will crab against the wind if needed to maintain a straight line.
 - **Multirotors** Roll controls left-right speed, pitch controls front-back speed over ground. Yaw controls yaw rate as in MANUAL mode. Throttle controls climb/descent rate as in ALTCTL mode. This means that the x, y, z position of the vehicle is held steady by the autopilot against any wind disturbances, when the roll, pitch and throttle sticks are centered.

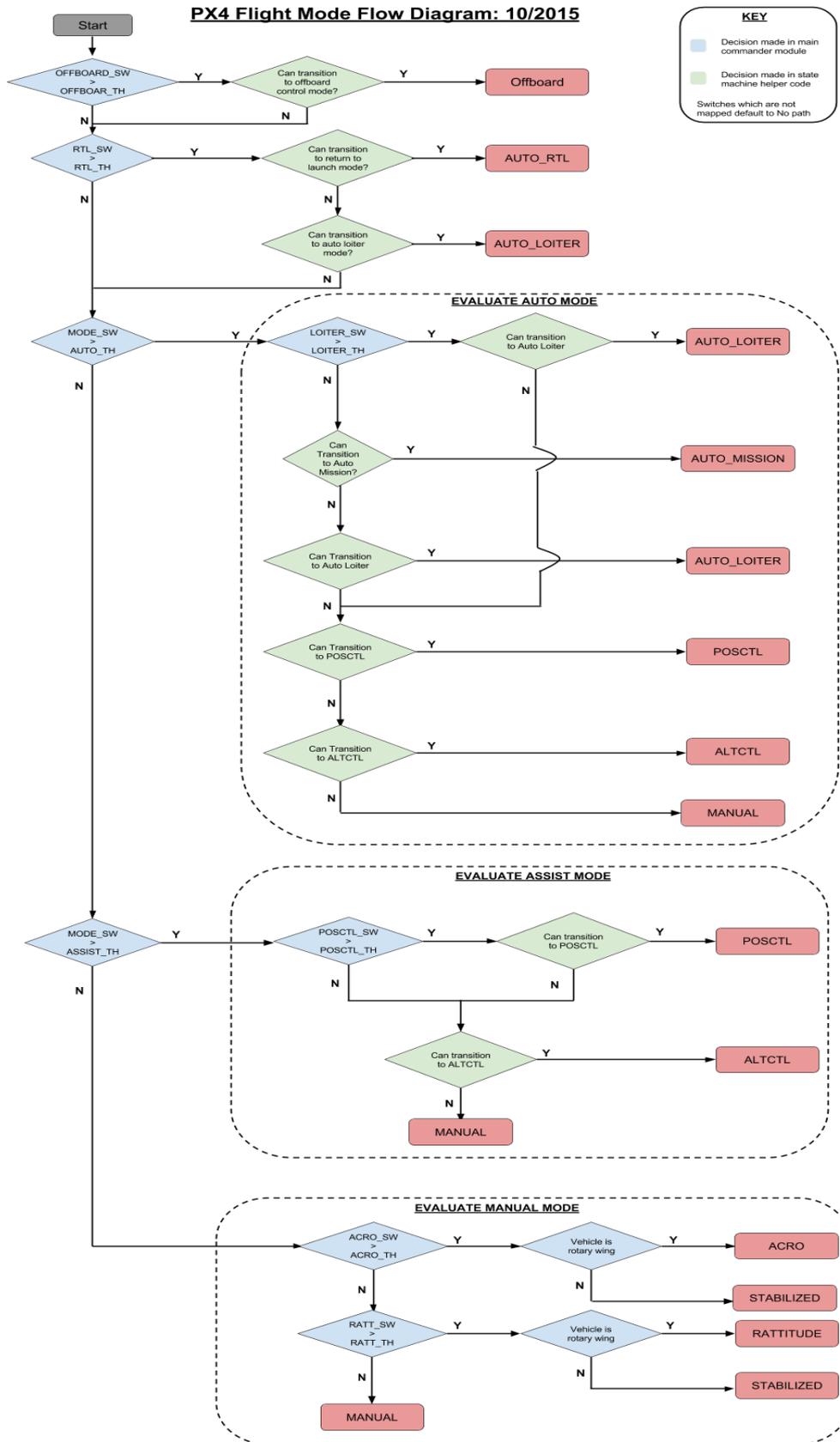
Auto flight modes

"Auto" modes are those where the controller requires little to no user input (e.g. to takeoff, land and fly missions).

- **AUTO_LOITER** (Loiter)
 - **Fixed wing aircraft:** The aircraft loiters around the current position at the current altitude (or possibly slightly above the current altitude, good for 'I'm losing it').
 - **Multirotors:** The multirotor hovers / loiters at the current position and altitude.
- **AUTO_RTL** (Return to Land)
 - **Fixed wing aircraft:** The aircraft returns to the home position and loiters in a circle above the home position.
 - **Multirotors:** The multirotor returns in a straight line on the current altitude (if the current altitude is higher than the home position + [RTL_RETURN_ALT](#)) or on the [RTL_RETURN_ALT](#) (if the [RTL_RETURN_ALT](#) is higher than the current altitude), then lands automatically.

-
- **AUTO_MISSION** (Mission)
 - **All system types:** The aircraft obeys the programmed mission sent by the ground control station (GCS). If no mission received, aircraft will LOITER at current position instead.
 - **OFFBOARD** (Offboard) In this mode the position, velocity or attitude reference / target / setpoint is provided by a companion computer connected via serial cable and MAVLink. The offboard setpoint can be provided by APIs like [MAVROS](#) or [Dronekit](#).

Flight Mode Evaluation Diagram



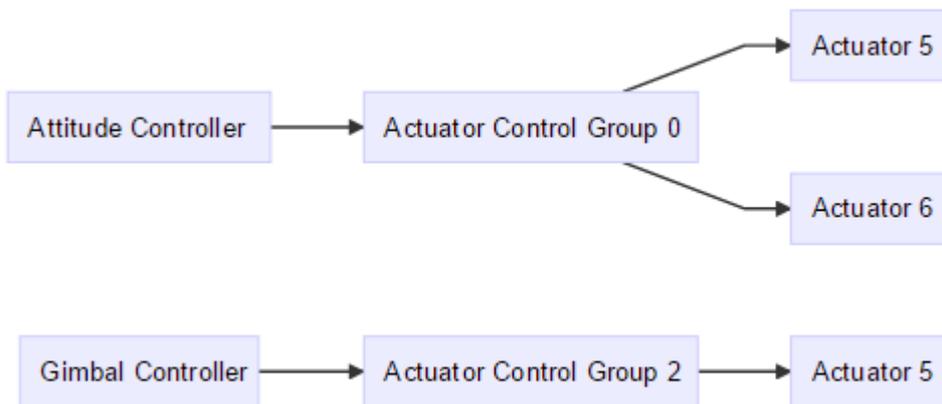
3.4 Mixing and Actuators

The PX4 architecture ensures that the airframe layout does not require special case handling in the core controllers.

Mixing means to take force commands (e.g. `turn right`) and translate them to actuator commands which control motors or servos. For a plane with one servo per aileron this means to command one of them high and the other low. The same applies for multicopters: Pitching forward requires changing the speed of all motors. Separating the mixer logic from the actual attitude controller greatly improves reusability.

Control Pipeline

A particular controller sends a particular normalized force or torque demand (scaled from -1..+1) to the mixer, which then sets individual actuators accordingly. The output driver (e.g. UART, UAVCAN or PWM) then scales it to the actuators native units, e.g. a PWM value of 1300.



Control Groups

PX4 uses control groups (inputs) and output groups. Conceptually they are very simple: A control group is e.g. `attitude`, for the core flight controls, or `gimbal` for payload. An output group is one physical bus, e.g. the first 8 PWM outputs for servos. Each of these groups has 8 normalized (-1..+1) command ports, which can be mapped and scaled through the mixer. A mixer defines how each of these 8 signals of the controls are connected to the 8 outputs.

For a simple plane control 0 (roll) is connected straight to output 0 (aileron). For a multicopter things are a bit different: control 0 (roll) is connected to all four motors and combined with throttle.

Control Group #0 (Flight Control)

- 0: roll (-1..1)
- 1: pitch (-1..1)
- 2: yaw (-1..1)
- 3: throttle (0..1 normal range, -1..1 for variable pitch / thrust reversers)
- 4: flaps (-1..1)
- 5: spoilers (-1..1)
- 6: airbrakes (-1..1)
- 7: landing gear (-1..1)

Control Group #1 (Flight Control VTOL/Alternate)

- 0: roll ALT (-1..1)
- 1: pitch ALT (-1..1)
- 2: yaw ALT (-1..1)
- 3: throttle ALT (0..1 normal range, -1..1 for variable pitch / thrust reversers)
- 4: reserved / aux0
- 5: reserved / aux1
- 6: reserved / aux2
- 7: reserved / aux3

Control Group #2 (Gimbal)

- 0: gimbal roll
- 1: gimbal pitch
- 2: gimbal yaw
- 3: gimbal shutter
- 4: reserved
- 5: reserved
- 6: reserved
- 7: reserved (parachute, -1..1)

Control Group #3 (Manual Passthrough)

- 0: RC roll
- 1: RC pitch
- 2: RC yaw
- 3: RC throttle
- 4: RC mode switch
- 5: RC aux1

-
- 6: RC aux2
 - 7: RC aux3

Control Group #6 (First Payload)

- 0: function 0 (default: parachute)
- 1: function 1
- 2: function 2
- 3: function 3
- 4: function 4
- 5: function 5
- 6: function 6
- 7: function 7

Virtual Control Groups

These groups are NOT mixer inputs, but serve as meta-channels to feed fixed wing and multicopter controller outputs into the VTOL governor module.

Control Group #4 (Flight Control MC VIRTUAL)

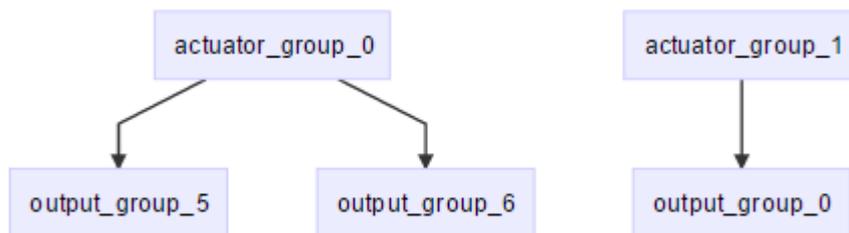
- 0: roll ALT (-1..1)
- 1: pitch ALT (-1..1)
- 2: yaw ALT (-1..1)
- 3: throttle ALT (0..1 normal range, -1..1 for variable pitch / thrust reversers)
- 4: reserved / aux0
- 5: reserved / aux1
- 6: reserved / aux2
- 7: reserved / aux3

Control Group #5 (Flight Control FW VIRTUAL)

- 0: roll ALT (-1..1)
- 1: pitch ALT (-1..1)
- 2: yaw ALT (-1..1)
- 3: throttle ALT (0..1 normal range, -1..1 for variable pitch / thrust reversers)
- 4: reserved / aux0
- 5: reserved / aux1
- 6: reserved / aux2
- 7: reserved / aux3

Mapping

Since there are multiple control groups (like flight controls, payload, etc.) and multiple output groups (first 8 PWM outputs, UAVCAN, etc.), one control group can send command to multiple output groups.



PX4 mixer definitions

Files in `ROMFS/px4fmu_common/mixers` implement mixers that are used for predefined airframes. They can be used as a basis for customisation, or for general testing purposes.

Syntax

Mixer definitions are text files; lines beginning with a single capital letter followed by a colon are significant. All other lines are ignored, meaning that explanatory text can be freely mixed with the definitions.

Each file may define more than one mixer; the allocation of mixers to actuators is specific to the device reading the mixer definition, and the number of actuator outputs generated by a mixer is specific to the mixer.

For example: each simple or null mixer is assigned to outputs 1 to x in the order they appear in the mixer file.

A mixer begins with a line of the form

```
<tag>: <mixer arguments>
```

The tag selects the mixer type; 'M' for a simple summing mixer, 'R' for a multirotor mixer, etc.

Null Mixer

A null mixer consumes no controls and generates a single actuator output whose value is always zero. Typically a null mixer is used as a placeholder in a collection of mixers in order to achieve a specific pattern of actuator outputs.

The null mixer definition has the form:

```
Z:
```

Simple Mixer

A simple mixer combines zero or more control inputs into a single actuator output. Inputs are scaled, and the mixing function sums the result before applying an output scaler.

A simple mixer definition begins with:

```
M: <control count>
```

```
O: <-ve scale> <+ve scale> <offset> <lower limit> <upper limit>
```

If <control count> is zero, the sum is effectively zero and the mixer will output a fixed value that is <offset> constrained by <lower limit> and <upper limit>.

The second line defines the output scaler with scaler parameters as discussed above. Whilst the calculations are performed as floating-point operations, the values stored in the definition file are scaled by a factor of 10000; i.e. an offset of -0.5 is encoded as -5000.

The definition continues with <control count> entries describing the control inputs and their scaling, in the form:

```
S: <group> <index> <-ve scale> <+ve scale> <offset> <lower limit> <upper limit>
```

The <group> value identifies the control group from which the scaler will read, and the <index> value an offset within that group. These values are specific to the device reading the mixer definition.

When used to mix vehicle controls, mixer group zero is the vehicle attitude control group, and index values zero through three are normally roll, pitch, yaw and thrust respectively.

The remaining fields on the line configure the control scaler with parameters as discussed above. Whilst the calculations are performed as floating-point operations, the values stored in the definition file are scaled by a factor of 10000; i.e. an offset of -0.5 is encoded as -5000.

An example of a typical mixer file is explained [here](#).

Multicopter Mixer

The multicopter mixer combines four control inputs (roll, pitch, yaw, thrust) into a set of actuator outputs intended to drive motor speed controllers.

The mixer definition is a single line of the form:

```
R: <geometry> <roll scale> <pitch scale> <yaw scale> <idlespeed>
```

The supported geometries include:

- 4x - quadrotor in X configuration
- 4+ - quadrotor in + configuration
- 6x - hexacopter in X configuration
- 6+ - hexacopter in + configuration
- 8x - octocopter in X configuration
- 8+ - octocopter in + configuration

Each of the roll, pitch and yaw scale values determine scaling of the roll, pitch and yaw controls relative to the thrust control. Whilst the calculations are performed as floating-point operations, the values stored in the definition file are scaled by a factor of 10000; i.e. an factor of 0.5 is encoded as 5000.

Roll, pitch and yaw inputs are expected to range from -1.0 to 1.0, whilst the thrust input ranges from 0.0 to 1.0. Output for each actuator is in the range -1.0 to 1.0.

Idlespeed can range from 0.0 to 1.0. Idlespeed is relative to the maximum speed of motors and it is the speed at which the motors are commanded to rotate when all control inputs are zero.

In the case where an actuator saturates, all actuator values are rescaled so that the saturating actuator is limited to 1.0.

3.5 PWM_limit State Machine

[PWM_limit State Machine] Controls PWM outputs as a function of pre-armed and armed inputs. Provides a delay between assertion of "armed" and a ramp-up of throttle on assertion of the armed signal.

Quick Summary

Inputs

- armed: asserted to enable dangerous behaviors such as spinning propellers
- pre-armed: asserted to enable benign behaviors such as moving control surfaces

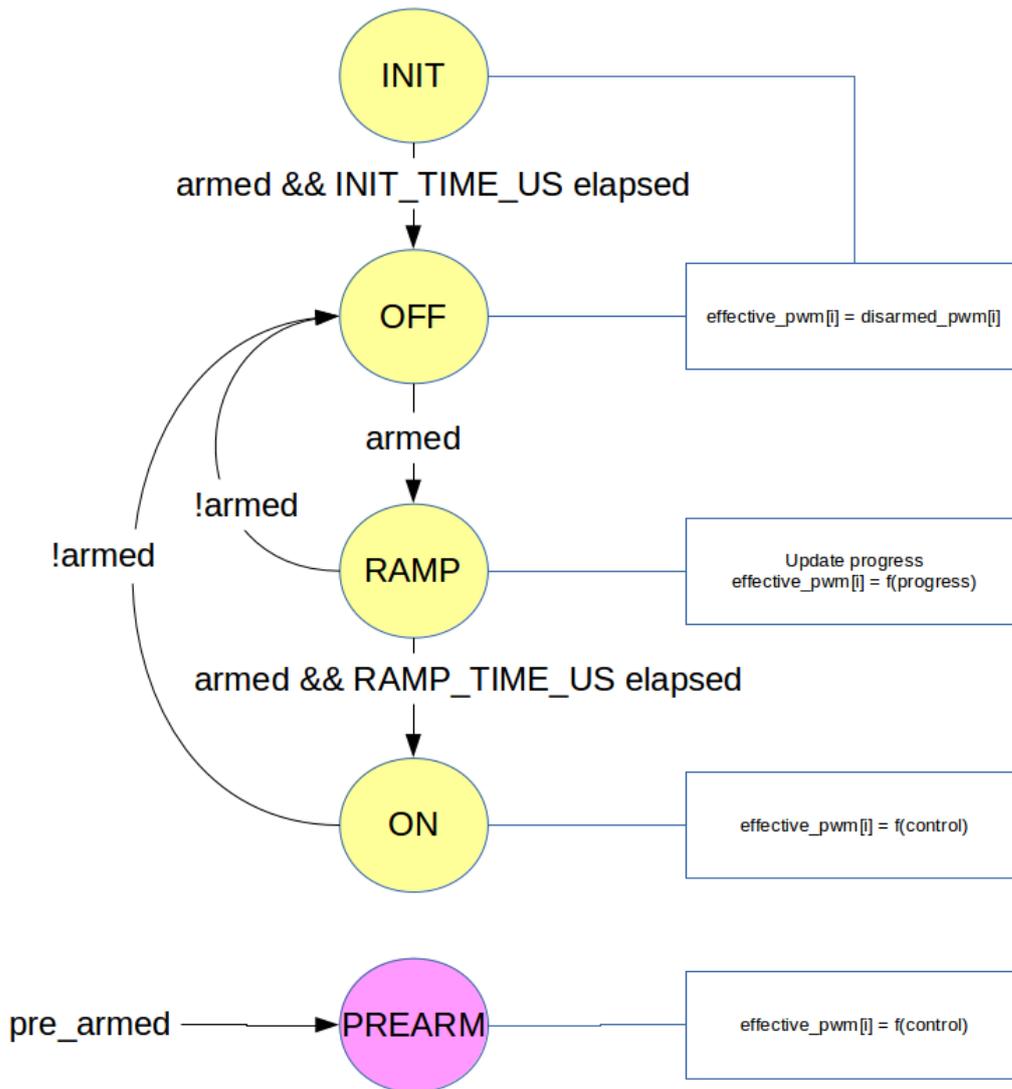
-
- this input overrides the current state
 - assertion of pre-armed immediately forces behavior of state ON, regardless of current state ** deassertion of pre-armed reverts behavior to current state

States

- INIT and OFF
 - pwm outputs set to disarmed values.
- RAMP
 - pwm outputs ramp from disarmed values to min values.
- ON
 - pwm outputs set according to control values.

State Transition Diagram

PWM_LIMIT_STATE_



Note: input `pre_armed` overrides the current state while asserted. When deasserted the state machine functions normally. Throttle controls must be set to NaN while `pre_armed` is asserted.

4. Simulation

Simulators allow PX4 flight code to control a computer modeled vehicle in a simulated "world". You can interact with this vehicle just as you might with a real vehicle, using *QGroundControl*, an offboard API, or a radio controller/gamepad.

Simulation is a quick, easy, and most importantly, *safe* way to test changes to PX4 code before attempting to fly in the real world. It is also a good way to start flying with PX4 when you haven't yet got a vehicle to experiment with.

PX4 supports both *Software In the Loop (SITL)* simulation, where the flight stack runs on computer (either the same computer or another computer on the same network) and *Hardware In the Loop (HITL)* simulation using a simulation firmware on a real flight controller board.

Information about available simulators and how to set them up are provided in the next section. The other sections provide general information about how the simulator works, and are not required to *use* the simulators.

Supported Simulators

The following simulators work with PX4 for HITL and/or SITL simulation.

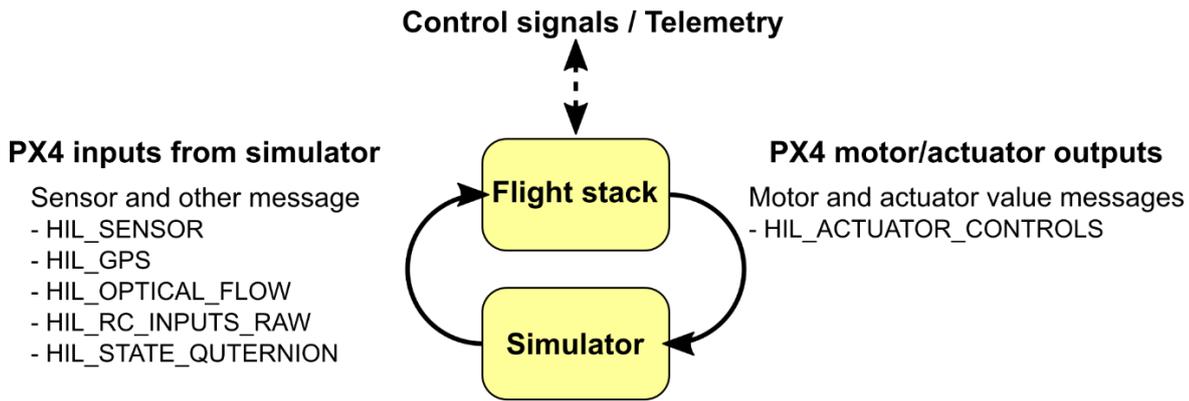
Simulator	Description
Gazebo	<p>This simulator is highly recommended.</p> <p>A powerful 3D simulation environment that is particularly suitable for testing object-avoidance and computer vision. It can also be used for multi-vehicle Simulation and is commonly used with ROS, a collection of tools for automating vehicle control.</p> <p>Supported Vehicles: Quad (Iris and Solo), Hex (Typhoon H480), Generic quad delta VTOL, Tailsitter, Plane, Rover, Submarine (coming soon!)</p>
jMAVSim	<p>A simple multirotor simulator that allows you to fly <i>copter</i> type vehicles around a simulated world.</p> <p>It is easy to set up and can be used to test that your vehicle can take off, fly, land, and responds appropriately to various fail conditions (e.g. GPS failure).</p> <p>Supported Vehicles: Quad</p>
AirSim	<p>A cross platform simulator that provides physically and visually realistic simulations. This simulator is resource intensive, and requires a very significantly more powerful computer than the other simulators described here.</p> <p>Supported Vehicles: Iris (MultiRotor model and a configuration for PX4 QuadRotor in the X configuration).</p>
XPlane (HITL only)	<p>A comprehensive and powerful fixed-wing flight simulator that offers very realistic flight models.</p> <p>Supported Vehicles: Plane</p>

Instructions for how to setup and use the simulators are in the topics linked above.

The remainder of this topic is a "somewhat generic" description of how the simulation infrastructure works. It is not required to *use* the simulators.

Simulator MAVLink API

All simulators communicate with PX4 using the Simulator MAVLink API. This API defines a set of MAVLink messages that supply sensor data from the simulated world to PX4 and return motor and actuator values from the flight code that will be applied to the simulated vehicle. The image below shows the message flow.



A SITL build of PX4 uses [simulator_mavlink.cpp](#) to handle these messages while a hardware build in HIL mode uses [mavlink_receiver.cpp](#). Sensor data from the simulator is written to PX4 uORB topics. All motors / actuators are blocked, but internal software is fully operational.

The messages are described below (see links for specific detail).

Message	Direction	Description
MAV_MODE:MAV_MODE_FLAG_HIL_ENABLED	NA	Mode flag when using simulation. All motors/actuators are blocked, but internal software is fully operational.
HIL_ACTUATOR_CONTROLS	PX4 to Sim	PX4 control outputs (to motors, actuators).
HIL_SENSOR	Sim to PX4	Simulated IMU readings in SI units in NED body frame.
HIL_GPS	Sim to PX4	The simulated GPS RAW sensor value.
HIL_OPTICAL_FLOW	Sim to PX4	Simulated optical flow from a flow sensor (e.g. PX4FLOW or optical mouse sensor)
HIL_STATE_QUATERNION	Sim to PX4	Contains the actual "simulated" vehicle position, attitude, speed etc. This can be logged and compared to PX4's estimates for analysis and debugging (for example, checking how well an estimator works for noisy (simulated) sensor inputs).
HIL_RC_INPUTS_RAW	Sim to PX4	The RAW values of the RC channels received.

Default PX4 MAVLink UDP Ports

By default, PX4 uses commonly established UDP ports for MAVLink communication with ground control stations (e.g. *QGroundControl*), Offboard APIs (e.g. DroneCore, MAVROS) and simulator APIs (e.g. Gazebo). These ports are:

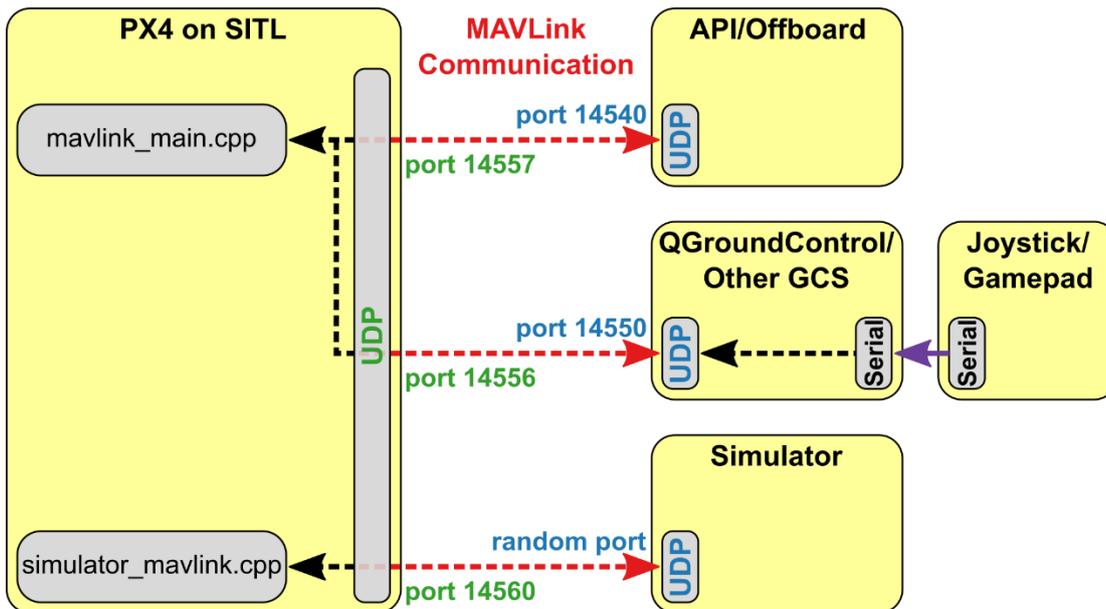
- Port **14540** is used for communication with offboard APIs. Offboard APIs are expected to listen for connections on this port.
- Port **14550** is used for communication with ground control stations. GCS are expected to listen for connections on this port. *QGroundControl* listens to this port by default.
- Port **14560** is used for communication with simulators. PX4 listens to this port, and simulators are expected to initiate the communication by broadcasting data to this port.

The ports for the GCS and offboard APIs are set in configuration files, while the simulator broadcast port is hard-coded in the simulation MAVLink module.

SITL Simulation Environment

The diagram below shows a typical SITL simulation environment for any of the supported simulators. The different parts of the system connect via UDP, and can be run on either the same computer or another computer on the same network.

- PX4 uses a simulation-specific module to listen on UDP port 14560. Simulators connect to this port, then exchange information using the [Simulator MAVLink API](#) described above. PX4 on SITL and the simulator can run on either the same computer or different computers on the same network.
- PX4 uses the normal MAVLink module to connect to GroundStations (which listen on port 14550) and external developer APIs like DroneCore or ROS (which listen on port 14540).
- A serial connection is used to connect Joystick/Gamepad hardware via *QGroundControl*.



If you use the normal build system SITL `make` configuration targets (see next section) then both SITL and the Simulator will be launched on the same computer and the ports above will automatically be configured. You can configure additional MAVLink UDP connections and otherwise modify the simulation environment in the build configuration and initialisation files.

Starting/Building SITL Simulation

The build system makes it very easy to build and start PX4 on SITL, launch a simulator, and connect them. For example, you can launch a SITL version of PX4 that uses the EKF2 estimator and simulate a plane in Gazebo with just the following command (provided all the build and gazebo dependencies are present!):

```
make posix_sitl_ekf2 gazebo_plane
```

It is also possible to separately build and start SITL and the various simulators, but this is nowhere near as "turnkey".

The syntax to call `make` with a particular configuration and initialisation file is:

```
make [CONFIGURATION_TARGET] [VIEWER_MODEL_DEBUGGER]
```

where:

- **CONFIGURATION_TARGET:** has the format `[OS][_PLATFORM][_FEATURE]`
 - **OS:** posix, nuttx, qurt
 - **PLATFORM:** SITL (or in principle any platform supported among the different OS: bebop, eagle, excelsior, etc.)
 - **FEATURE:** A particular high level feature - for example which estimator to use (ekf2, lpe) or to run tests or simulate using a replay.

You can get a list of all available configuration targets using the command:

```
make list_config_targets
```

- **VIEWER_MODEL_DEBUGGER:** has the format [SIMULATOR]_[MODEL]_[_DEBUGGER]
- **SIMULATOR:** This is the simulator ("viewer") to launch and connect: gazebo, jmavsim
- **MODEL:** The vehicle model to use (e.g. iris, rover, tailsitter, etc). This corresponds to a specific [initialisation file](#) that will be used to configure PX4. This might define the start up for a particular vehicle, or allow simulation of multiple vehicles (we explain how to find available init files in the next section).
- **DEBUGGER:** Debugger to (optionally) use: none, ide, gdb, lldb, ddd, valgrind, callgrind. For more information see [Simulation Debugging](#).

You can get a list of all available VIEWER_MODEL_DEBUGGER options using the command:

```
make posix list_vmd_make_targets
```

Notes:

- Most of the values in the CONFIGURATION_TARGET and VIEWER_MODEL_DEBUGGER have defaults, and are hence optional. For example, gazebo is equivalent to gazebo_iris OR gazebo_iris_none.
- You can use three underscores if you want to specify a default value between two other settings. For example, gazebo___gdb is equivalent to gazebo_iris_gdb.
- You can use a none value for VIEWER_MODEL_DEBUGGER to start PX4 and wait for a simulator. For example start PX4 using `make posix_sitl_default none` and jMAVSim using `./Tools/jmavsim_run.sh`.

Init File Location

The settings for each configuration target are defined in appropriately named files in [/Firmware/cmake/configs](#). Within each file there is a setting `config_sitl_rcS_dir` that defines the location of the folder where the configuration stores its init files.

In the cmake config file for [posix_sitl_ekf2](#) you can see that the init file will be stored in the folder: **Firmware/posix-configs/SITL/init/ekf2/**.

```
set(config_sitl_rcS_dir
    posix-configs/SITL/init/ekf2
)
```

Generally the init files are located using a consistent folder naming convention. For example, `make posix_sitl_ekf2 gazebo_iris` corresponds to the following folder structure:

```
Firmware/
  posix-configs/ (os=posix)
    SITL/        (platform=sitl)
      init/
        ekf2/   (feature=ekf2)
          iris   (init file name)
```

Example Startup File

A slightly reduced version of the startup file for `make posix_sitl_ekf2 gazebo_iris` (</Firmware/posix-configs/SITL/init/ekf2/iris>) is shown below.

```
uorb start
param load
dataman start
param set BAT_N_CELLS 3
param set CAL_ACC0_ID 1376264
param set CAL_ACC0_XOFF 0.01
...
...
param set SYS_MC_EST_GROUP 2
param set SYS_RESTART_TYPE 2
replay tryapplyparams
simulator start -s
tone_alarmstart
gyrosim start
accelsim start
barosim start
adcsim start
gpssim start
pwm_out_sim mode_pwm
sensors start
commander start
land_detector start multicopter
navigator start
ekf2 start
mc_pos_control start
mc_att_control start
mixer load /dev/pwm_output0 ROMFS/px4fmu_common/mixers/quad_dc.main.mix
mavlink start -u 14556 -r 4000000
mavlink start -u 14557 -r 4000000 -m onboard -o 14540
mavlink stream -r 50 -s POSITION_TARGET_LOCAL_NED -u 14556
mavlink stream -r 50 -s LOCAL_POSITION_NED -u 14556
mavlink stream -r 50 -s GLOBAL_POSITION_INT -u 14556
mavlink stream -r 50 -s ATTITUDE -u 14556
mavlink stream -r 50 -s ATTITUDE_QUATERNION -u 14556
mavlink stream -r 50 -s ATTITUDE_TARGET -u 14556
mavlink stream -r 50 -s SERVO_OUTPUT_RAW_0 -u 14556
mavlink stream -r 20 -s RC_CHANNELS -u 14556
mavlink stream -r 250 -s HIGHRES_IMU -u 14556
mavlink stream -r 10 -s OPTICAL_FLOW_RAD -u 14556
logger start -e -t
```

```
mavlink boot_complete
replay trystart
```

Note the sections that set parameters, start simulator drivers and other modules. A few of the more relevant lines for simulation are highlighted below.

1. Simulator being started:

```
simulator start -s
```

2. PWM out mode being set for simulator:

```
pwm_out_sim mode_pwm
```

3. Set MAVLink ports:

- This line starts the MAVLink instance for connecting to offboard APIs. It broadcasts on 14540 and listens for responses on 14557. The `-m onboard` flag specifies a set of messages that will be streamed over the interface.

```
mavlink start -u 14557 -r 4000000 -m onboard -o 14540
```

- This line starts MAVLink instance for connecting to *QGroundControl*/GCSs. PX4 listens for messages on port 14556.

```
mavlink start -u 14556 -r 4000000
```

- The broadcast port is not explicitly set (the default is used: 14550).
- The messages that are streamed over this interface are specified using `mavlink stream` as shown below:

```
mavlink stream -r 50 -s POSITION_TARGET_LOCAL_NED -u 14556
mavlink stream -r 50 -s LOCAL_POSITION_NED -u 14556
...
```

For more information about using the MAVLink module see [Modules Reference: Communication > MAVLink](#).

HITL Simulation Environment

With Hardware-in-the-Loop (HITL) simulation the normal PX4 firmware is run on real hardware. The HITL Simulation Environment is documented in: [HITL Simulation](#).

Joystick/Gamepad Integration

QGroundControl desktop versions can connect to a USB Joystick/Gamepad and send its movement commands and button presses to PX4 over MAVLink. This works on both

SITL and HITL simulations, and allows you to directly control the simulated vehicle. If you don't have a joystick you can alternatively control the vehicle using QGroundControl's onscreen virtual thumbsticks.

For setup information see the *QGroundControl User Guide*:

- [Joystick Setup](#)
- [Virtual Joystick](#)

Camera Simulation

PX4 supports capture of both still images and video from within the [Gazebo](#) simulated environment. This can be enabled/set up as described in [Gazebo > Video Streaming](#).

The simulated camera is a gazebo plugin that implements the [MAVLink Camera Protocol](#). PX4 connects/integrates with this camera in *exactly the same way* as it would with any other MAVLink camera:

[TRIG INTERFACE](#) must be set to 3 to configure the camera trigger driver for use with a MAVLink camera

In this mode the driver just sends a [CAMERA_TRIGGER](#) message whenever an image capture is requested. For more information see [Camera Trigger](#).

PX4 must forward all camera commands between the GCS and the (simulator) MAVLink Camera. You can do this by starting [mavlink](#) with the `-f` flag as shown, specifying the UDP ports for the new connection.

```
mavlink start -u 14558 -o 14530 -r 4000 -f -m camera
```

More than just the camera MAVLink messages will be forwarded, but the camera will ignore those that it doesn't consider relevant.

The same approach can be used by other simulators to implement camera support.

4.1 jMAVSim with SITL

jMAVSim is a simple multirotor/Quad simulator that allows you to fly *copter* type vehicles running PX4 around a simulated world. It is easy to set up and can be used to test that your vehicle can take off, fly, land, and responds appropriately to various fail conditions (e.g. GPS failure).

Supported Vehicles:

- Quad

This topic shows how to set up jMAVSim to connect with a SITL version of PX4.

jMAVSim can also be used for HITL Simulation ([as shown here](#)).

Simulation Environment

Software in the Loop Simulation runs the complete system on the host machine and simulates the autopilot. It connects via local network to the simulator. The setup looks like this:

SimulatorMAVLinkSITL

Running SITL

After ensuring that the [simulation prerequisites](#) are installed on the system, just launch: The convenience make target will compile the POSIX host build and run the simulation.

```
make posix_sitl_default jmavsim
```

This will bring up the PX4 shell:

```
[init] shell id: 140735313310464
```

```
[init] task name: px4
```

```
____ _  _  _  _  
|  _ \ \ \ \ / /  / |  
| |_/ / \ v /  / / |  
|  _/  /  \ / /_ |  
| |    / / ^ \ \  _ |  
\_|    \ \  \  |_/
```

Ready to fly.

```
pxh>
```

Important Files

- The startup script is in the [posix-configs/SITL/init](#) folder and named `rcS_SIM_AIRFRAME`, the default is `rcS_jmavsim_iris`.

-
- The root file system (the equivalent of / as seen by the) is located inside the build directory: `build/posix_sitl_default/src/firmware/posix/rootfs/`

Taking it to the Sky

And a window with the 3D view of the [jMAVSim](#) simulator:



The system will print the home position once it finished initializing (`telem> home:`
`55.7533950, 37.6254270, -0.00`). You can bring it into the air by typing:
`pxh> commander takeoff`

Joystick or thumb-joystick support is available through QGroundControl (QGC). To use manual input, put the system in a manual flight mode (e.g. POSCTL, position control). Enable the thumb joystick from the QGC preferences menu.

Simulating a Wifi Drone

There is a special target to simulate a drone connected via Wifi on the local network:

```
make broadcast jmavsim
```

The simulator broadcasts its address on the local network as a real drone would do.

Extending and Customizing

To extend or customize the simulation interface, edit the files in the **Tools/jMAVSim** folder. The code can be accessed through the [jMAVSim repository](#) on Github.

The build system enforces the correct submodule to be checked out for all dependencies, including the simulator. It will not overwrite changes in files in the directory, however, when these changes are committed the submodule needs to be registered in the Firmware repo with the new commit hash. To do so, `git add Tools/jMAVSim` and commit the change. This will update the GIT hash of the simulator.

Interfacing to ROS

The simulation can be [interfaced to ROS](#) the same way as onboard a real vehicle.

4.2 Gazebo Simulation

[Gazebo](#) is a powerful 3D simulation environment for autonomous robots that is particularly suitable for testing object-avoidance and computer vision. This page describes its use with SITL and a single vehicle. Gazebo can also be used with [HITL](#) and for [multi-vehicle simulation](#).

Supported Vehicles: Quad ([Iris](#) and [Solo](#)), Hex (Typhoon H480), [Generic quad delta VTOL](#), Tailsitter, Plane, Rover, Submarine (coming soon!)

Gazebo is often used with [ROS](#), a toolkit/offboard API for automating vehicle control. If you plan to use PX4 with ROS you should instead [follow the instructions here](#) to install Gazebo as part of ROS!



See [Simulation](#) for general information about simulators, the simulation environment and available simulation configuration (e.g. supported vehicles).

Installation

Gazebo 8 setup is included in our standard build instructions:

- **macOS:** [Development Environment on Mac](#)
- **Linux:** [Development Environment on Linux > jMAVSim/Gazebo Simulation](#)
- **Windows:** Not supported.

Additional installation instructions can be found on gazebo.org.

Running the Simulation

You can run a simulation by starting PX4 SITL and gazebo with the airframe configuration to load (multicopters, planes, VTOL, optical flow and multi-vehicle simulations are supported).

The easiest way to do this is to open a terminal in the root directory of the PX4 *Firmware* repository and call `make` for the targets as shown in the following sections.

You can use the [instructions below](#) to keep Gazebo running and only re-launch PX4. This is quicker than restarting both.

For the full list of build targets run `make posix list_vmd_make_targets` (and filter on those that start with `gazebo_`).

Quadrotor

```
cd ~/src/Firmware
make posix_sitl_default gazebo
```

Quadrotor with Optical Flow

```
make posix gazebo_iris_opt_flow
```

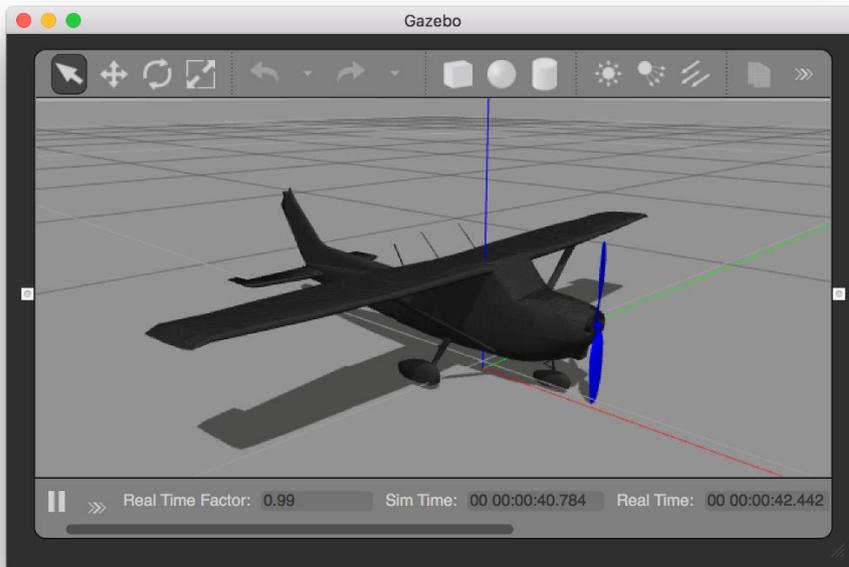
3DR Solo

```
make posix gazebo_solo
```



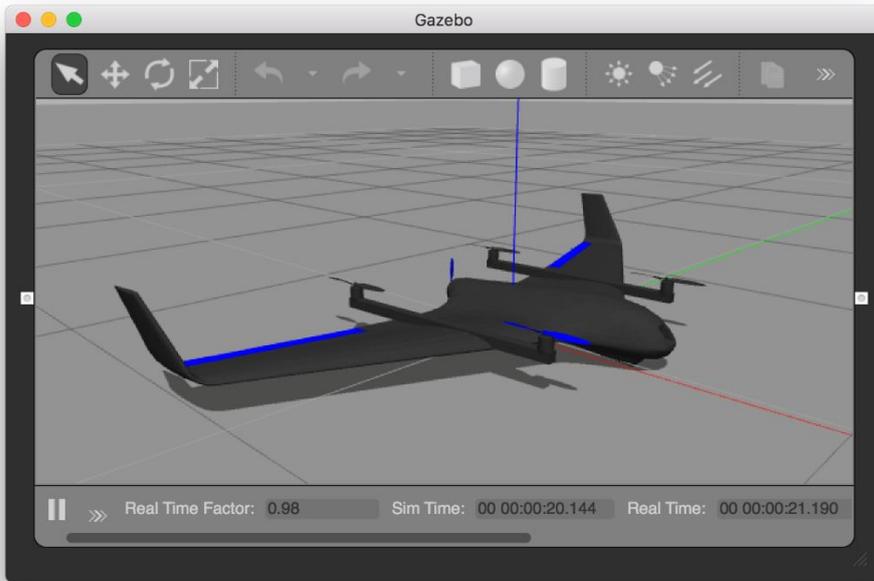
Standard Plane

```
make posix gazebo_plane
```



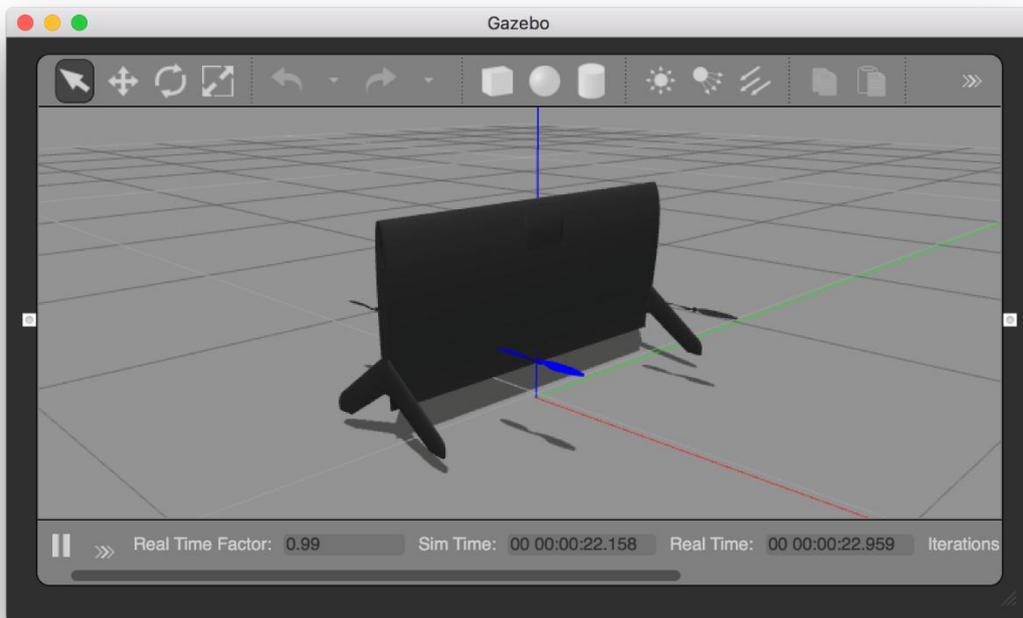
Standard VTOL

```
make posix_sitl_default gazebo_standard_vtol
```



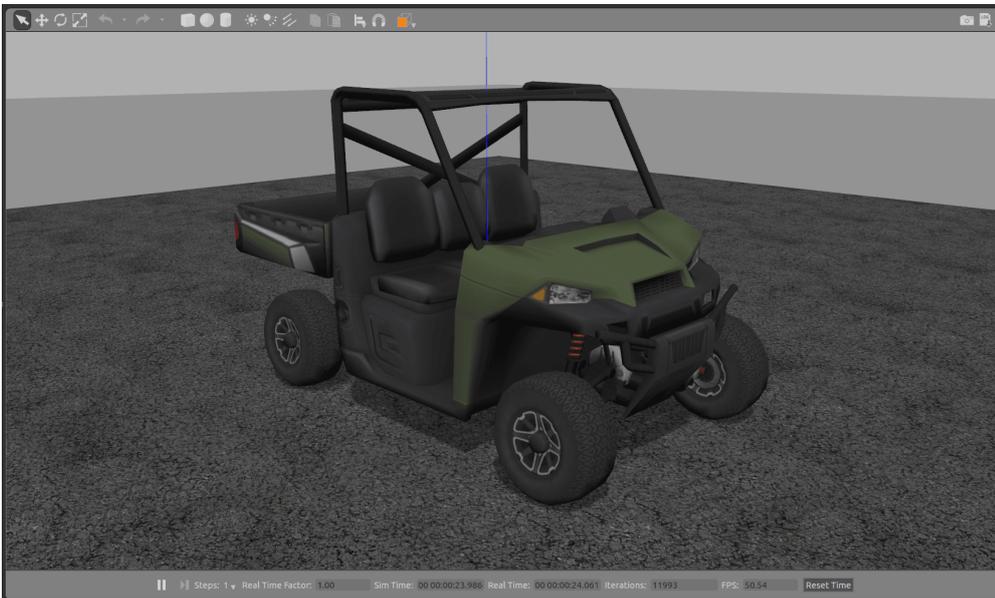
Tailsitter VTOL

```
make posix_sitl_default gazebo_tailsitter
```



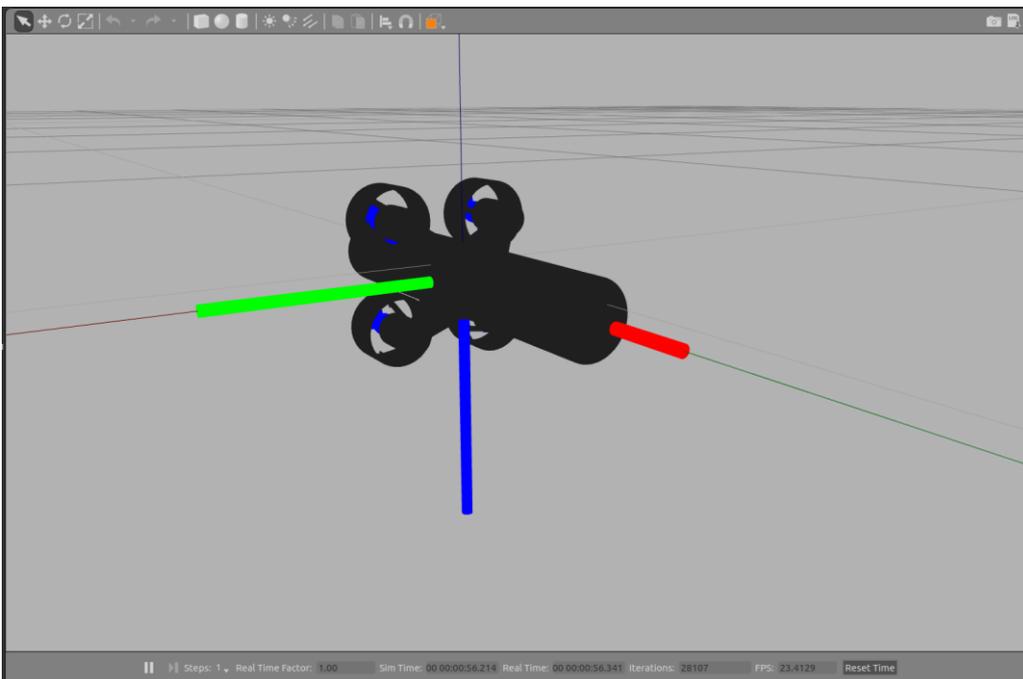
Ackerman vehicle (UGV/Rover)

```
make posix gazebo_rover
```



HippoCampus TUHH (UUV: Unmanned Underwater Vehicle)

```
make posix_sitl_default gazebo_hippocampus
```



Change World

The current default world is the **iris.world** located in the directory [worlds](#). The default surrounding in the **iris.world** uses a heightmap as ground. This ground can cause difficulty when using a distance sensor. If there are unexpected results with that heightmap, we recommend you change the model in **iris.model** from `uneven_ground` to `asphalt_plane`.

Taking it to the Sky

Please refer to the [Installing Files and Code](#) guide if you run into any errors.

This will bring up the PX4 shell:

```
[init] shell id: 140735313310464
[init] task name: px4
```

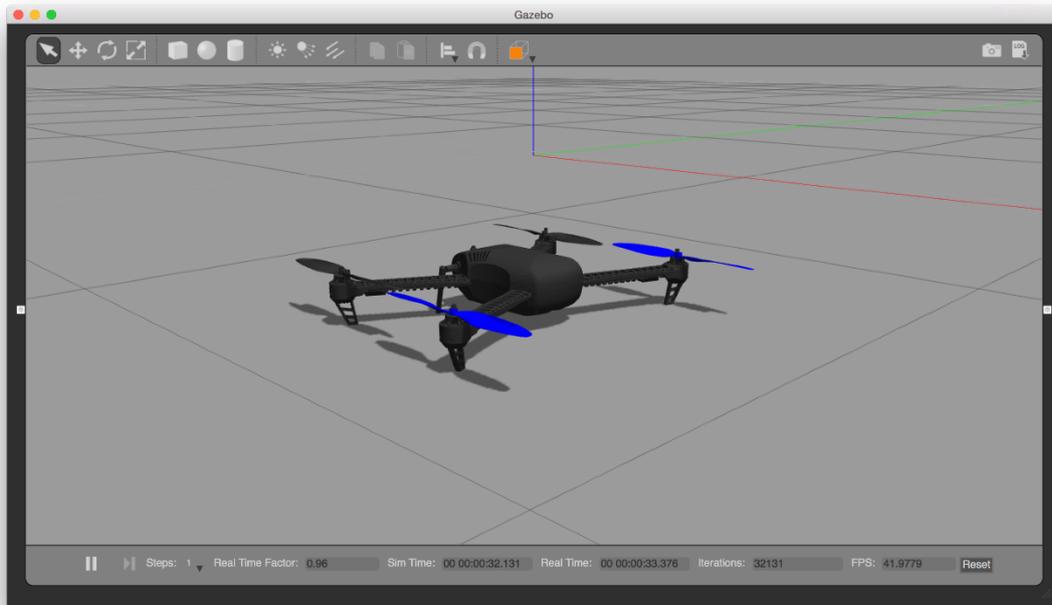
```

  _____
 |  _  \ \ \ / /  /  |
 | |_/ / \ v /  / / |
 |  _/  / \ / / _ |
 | |    / / ^ \ \ _ |
 \_|    \ \ \  |_/
```

```
px4 starting.
```

```
pxh>
```

Right-clicking the quadrotor model allows to enable follow mode from the context menu, which is handy to keep it in view.



The system will print the home position once it finished initializing (`telem> home:`
`55.7533950, 37.6254270, -0.00`). You can bring it into the air by typing:
`pxh> commander takeoff`

Usage/Configuration Options

Set Custom Takeoff Location

The default takeoff location in SITL Gazebo can be overridden using environment variables.

The variables to set are: `PX4_HOME_LAT`, `PX4_HOME_LON`, and `PX4_HOME_ALT`.
As an example:

```
export PX4_HOME_LAT=28.452386
export PX4_HOME_LON=-13.867138
export PX4_HOME_ALT=28.5
make posix gazebo
```

Using a Joystick

Joystick and thumb-joystick support are supported through *QGroundControl* ([setup instructions here](#)).

Simulating GPS Noise

Gazebo can simulate GPS noise that is similar to that typically found in real systems (otherwise reported GPS values will be noise-free/perfect). This is useful when working on applications that might be impacted by GPS noise - e.g. precision positioning.

GPS noise is enabled if the target vehicle's SDF file contains a value for the `gpsNoise` element (i.e. it has the line: `<gpsNoise>true</gpsNoise>`). It is enabled by default in many vehicle SDF

files: **solo.sdf**, **iris.sdf**, **standard_vtol.sdf**, **delta_wing.sdf**, **plane.sdf**, **typhoon_h480**, **tailsitter.sdf**.

To enable/disable GPS noise:

1. Build any gazebo target in order to generate SDF files (for all vehicles). For example:

```
make posix_sitl_default gazebo_iris
```

The SDF files are not overwritten on subsequent builds.

2. Open the SDF file for your target vehicle (e.g. `./Tools/sitl_gazebo/models/iris/iris.sdf`).
3. Search for the `gpsNoise` element:

```
<plugin name='gps_plugin' filename='libgazebo_gps_plugin.so'>
  <robotNamespace/>
  <gpsNoise>true</gpsNoise>
</plugin>
```

- If it is present, GPS is enabled. You can disable it by deleting the line: `<gpsNoise>true</gpsNoise>`
- If it is not preset GPS is disabled. You can enable it by adding the `gpsNoise` element to the `gps_plugin` section (as shown above).

The next time you build/restart Gazebo it will use the new GPS noise setting.

Starting Gazebo and PX4 Separately

For extended development sessions it might be more convenient to start Gazebo and PX4 separately or even from within an IDE.

In addition to the existing cmake targets that run `sitl_run.sh` with parameters for px4 to load the correct model it creates a launcher targets named `px4_<mode>` that is a thin wrapper around original sitl px4 app. This thin wrapper simply embeds app arguments like current working directories and the path to the model file.

To start Gazebo and PX4 separately:

-
- Run gazebo (or any other sim) server and client viewers via the terminal:

```
make posix_sitl_default gazebo_none_ide
```

- In your IDE select `px4_<mode>` target you want to debug (e.g. `px4_iris`)
- Start the debug session directly from IDE

This approach significantly reduces the debug cycle time because simulator (e.g. gazebo) is always running in background and you only re-run the px4 process which is very light.

Video Streaming

PX4 SITL for Gazebo supports UDP video streaming from a Gazebo camera sensor attached to a vehicle model. You can connect to this stream from *QGroundControl* (on UDP port 5600) and view video of the Gazebo environment from the simulated vehicle - just as you would from a real camera. The video is streamed using a *gststreamer* pipeline.

Video streaming from Gazebo and the Gazebo widget to turn streaming on/off are not enabled by default. This article explains how to enable them. In the near future we expect these features to be enabled by default.

Prerequisites

Install *Gstreamer 1.0* and its dependencies:

```
sudo apt-get install $(apt-cache --names-only search ^gstreamer1.0-* | awk '{ print $1 }' | grep -v gstreamer1.0-hybris) -y
```

Enable GStreamer Plugin

This step will not be required once video streaming is enabled by default.

Enable the *GStreamer Plugin* (if disabled) by changing the `BUILD_GSTREAMER_PLUGIN` option to "ON" in [<Firmware>/Tools/sitl_gazebo/CMakeLists.txt](#) (as shown below):

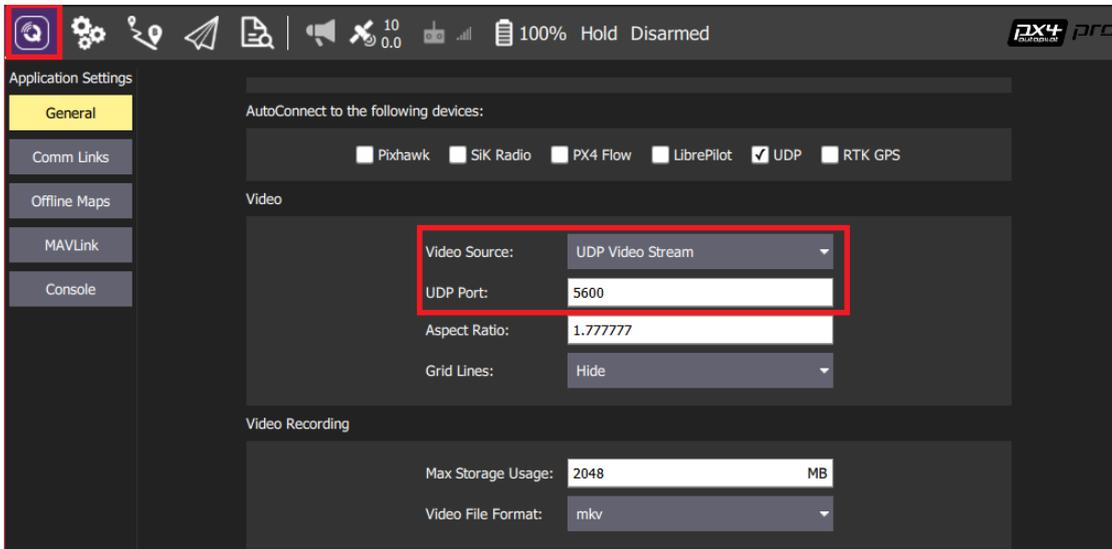
```
option(BUILD_GSTREAMER_PLUGIN "enable gstreamer plugin" "ON")
```

Once the plugin is enabled you can run SITL with Gazebo in the normal way:

```
make clean
make posix gazebo_typhoon_h480
```

How to View Gazebo Video

The easiest way to view the SITL/Gazebo camera video stream is in *QGroundControl*. Simply open **Settings > General** and set **Video Source** to *UDP Video Stream* and **UDP Port** to *5600*:



The video from Gazebo should then display in *QGroundControl* just as it would from a real camera.

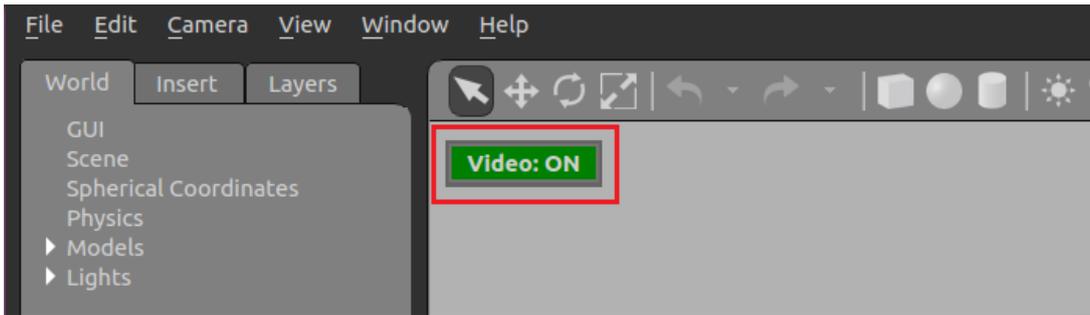
It is also possible to view the video using the *Gstreamer Pipeline*. Simply enter the following terminal command:

```
gst-launch-1.0 -v udpsrc port=5600 caps='application/x-rtp, media=(string)video,
clock-rate=(int)90000, encoding-name=(string)H264'
! rtph264depay ! avdec_h264 ! videoconvert ! autovideosink fps-update-interval=1000
sync=false
```

Gazebo GUI to Start/Stop Video Streaming

This feature is supported for Gazebo version 7.

Video streaming can be enabled/disabled using the Gazebo UI *Video ON/OFF* button.



To enable the button:

1. Open the "world" file to be modified (e.g. [<Firmware>/Tools/sitl_gazebo/worlds/typhoon_h480.world](#)).
2. Within the default world name="default" section, add the gui section for the libgazebo_video_stream_widget (as shown below):

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <world name="default">
    <gui>
      <plugin name="video_widget" filename="libgazebo_video_stream_widget.so"/>
    </gui>
  <!-- A global light source -->
  <include>
  ...
```

This section present in **typhoon_h480.world** - you just need to uncomment the section.

3. Rebuild SITL:

```
make clean
make posix gazebo_typhoon_h480
```

Extending and Customizing

To extend or customize the simulation interface, edit the files in the `Tools/sitl_gazebo` folder. The code is available on the [sitl_gazebo repository](#) on Github.

The build system enforces the correct GIT submodules, including the simulator. It will not overwrite changes in files in the directory.

Interfacing to ROS

The simulation can be [interfaced to ROS](#) the same way as onboard a real vehicle.

Further Information

- [ROS with Gazebo Simulation](#)
- [Gazebo Octomap](#)

4.3 AirSim Simulation

AirSim is a open-source, cross platform simulator for drones, built on Unreal Engine. It provides physically and visually realistic simulations of Pixhawk/PX4 using either Hardware-In-The-Loop (HITL) or Software-In-The-Loop (SITL).

The main entry point for the documentation is the [Github AirSim README](#).

The main entry point for documentation on working with PX4 is [PX4 Setup for AirSim](#) (covering both HITL and SITL).

4.4 Multi-Vehicle Simulation

This tutorial explains how to simulate multiple UAV vehicles using Gazebo and SITL.

It demonstrates an example setup that opens the Gazebo client GUI showing two Iris vehicles in an empty world. You can then control the vehicles with *QGroundControl* and MAVROS in a similar way to how you would manage a single vehicle.

This is tested only in Linux.

Required

- ROS indigo or higher
- [MAVROS package](#)
- Gazebo 7 (see [Gazebo Simulation](#))
- a clone of latest [PX4/Firmware](#)

Build and test

To build an example setup, follow the step below:

1. Clone the PX4/Firmware code, then build the SITL code

```
cd Firmware_clone
git submodule update --init --recursive
make posix_sitl_default sitl_gazebo
```

2. Source your environment:

```
source Tools/setup_gazebo.bash $(pwd) $(pwd)/build/posix_sitl_default
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd):$(pwd)/Tools/sitl_gazebo
```

3. Run launch file:

```
roslaunch px4 multi_uav_mavros_sitl.launch
```

You can specify `gui:=false` in the above `roslaunch` to launch Gazebo without its UI.

The tutorial example opens the Gazebo client GUI showing two Iris vehicles in an empty world.

You can control the vehicles with *QGroundControl* or MAVROS in a similar way to how you would manage a single vehicle:

- *QGroundControl* will have a drop-down to select the vehicle that is "in focus"
- MAVROS requires that you include the proper namespace before the topic/service path (e.g. for `<group ns="uav1">` you'll use `/uav1/mavros/mission/push`).

What's happening?

For each simulated vehicle, the following is required:

- **Gazebo model:** This is defined as `xacro` file in `Firmware/Tools/sitl_gazebo/models/rotors_description/urdf/<model>_base.xacro` see [here](#). Currently, the model `xacro` file is assumed to end with `base.xacro`. This model should have an argument called `mavlink_udp_port` which defines the UDP port on which gazebo will communicate with PX4 node. The model's `xacro` file will be used to generate an `urdf` model that contains UDP port that you select. To define the UDP port, set the `mavlink_udp_port` in the launch file for each vehicle, see [here](#) as an example. If you are using the same vehicle model, you don't need a separate `xacro` file for each vehicle. The same `xacro` file is adequate.
- **PX4 node:** This is the SITL PX4 app. It communicates with the simulator, Gazebo, through the same UDP port defined in the Gazebo vehicle model, i.e. `mavlink_udp_port`. To set the UDP port on the PX4 SITL app side, you need to set the `SITL_UDP_PRT` parameter in the startup file to match the `mavlink_udp_port` discussed previously, see [here](#). The path of the startup file in the launch file is generated based on the `vehicle` and `ID` arguments, see [here](#). The `MAV_SYS_ID` for each vehicle in the startup file, see [here](#), should match the `ID` for that vehicle in the launch file [here](#). This will help make sure you keep the configurations consistent between the launch file and the startup file.
- **MAVROS node** (optional): A separate MAVROS node can be run in the launch file, see [here](#), in order to connect to PX4 SITL app, if you want to control your vehicle through ROS. You need to start a MAVLink stream on a unique set of ports in the

startup file, see [here](#). Those unique set of ports need to match those in the launch file for the MAVROS node, see [here](#).

The launch file `multi_uav_mavros_sitl.launch` does the following,

- loads a world in gazebo,

```
<!-- Gazebo sim -->
<include file="$(find gazebo_ros)/launch/empty_world.launch">
  <arg name="gui" value="$(arg gui)"/>
  <arg name="world_name" value="$(arg world)"/>
  <arg name="debug" value="$(arg debug)"/>
  <arg name="verbose" value="$(arg verbose)"/>
  <arg name="paused" value="$(arg paused)"/>
</include>
```

- for each vehicle,
- creates urdf model from xacro, loads gazebo model and runs PX4 SITL app instance

```
<!-- PX4 SITL and vehicle spawn -->
<include file="$(find px4)/launch/single_vehicle_spawn.launch">
  <arg name="x" value="0"/>
  <arg name="y" value="0"/>
  <arg name="z" value="0"/>
  <arg name="R" value="0"/>
  <arg name="P" value="0"/>
  <arg name="Y" value="0"/>
  <arg name="vehicle" value="$(arg vehicle)"/>
  <arg name="rcS" value="$(find px4)/posix-configs/SITL/init/$(arg est)/$(arg
vehicle)_$(arg ID)"/>
  <arg name="mavlink_udp_port" value="14560"/>
  <arg name="ID" value="$(arg ID)"/>
</include>
```

- runs a mavros node

```
<!-- MAVROS -->
<include file="$(find mavros)/launch/px4.launch">
  <arg name="fcu_url" value="$(arg fcu_url)"/>
  <arg name="gcs_url" value=""/>
  <arg name="tgt_system" value="$(arg ID)"/>
  <arg name="tgt_component" value="1"/>
</include>
```

The complete block for each vehicle is enclosed in a set of `<group>` tags to separate the ROS namespaces of the vehicles.

To add a third iris to this simulation there are two main components to consider:

-
- add `UAV3` to `multi_uav_mavros_sitl.launch`
 - duplicate the group of either existing vehicle (`UAV1` or `UAV2`)
 - increment the `ID` arg to `3`
 - select a different port for `mavlink_udp_port` arg for communication with Gazebo
 - selects ports for MAVROS communication by modifying both port numbers in the `fcu_url` arg
 - create a startup file, and change the file as follows:
 - make a copy of an existing iris rcS startup file (`iris_1` or `iris_2`) and rename it `iris_3`
 - `MAV_SYS_ID` value to `3`
 - `SITL_UDP_PRT` value to match that of the `mavlink_udp_port` launch file arg
 - the first `mavlink` start port and the `mavlink` stream port values to the same values, which is to be used for QGC communication
 - the second `mavlink` start ports need to match those used in the launch file `fcu_url` arg
- Be aware of which port is `src` and `dst` for the different endpoints.

Additional Resources

- See [Simulation](#) for a description of the UDP port configuration.
- See [URDF in Gazebo](#) for more information about spawning the model with xacro.
- See [RotorS](#) for more xacro models.

4.5 Hardware in the Loop Simulation (HITL)

Hardware-in-the-Loop (HITL or HIL) is a simulation mode in which normal PX4 firmware is run on real flight controller hardware. This approach has the benefit of testing most of the actual flight code on the real hardware.

PX4 supports HITL for multicopters (using jMAVSim or Gazebo) and fixed wing (using Gazebo or X-Plane demo/full version).

HITL-Compatible Airframes

The current list of compatible airframes vs Simulators is:

Airframe	SYS_AUTOSTART	X-Plane	Gazebo	jMAVSim
HILStar (X-Plane)	1000	Y		
HIL Quadcopter X	1001		Y	Y
Standard planes	2100	Y		
Generic Quadrotor x copter	4001		Y	Y
DJI Flame Wheel f450	4011		Y	Y

HITL Simulation Environment

With Hardware-in-the-Loop (HITL) simulation the normal PX4 firmware is run on real hardware. The HITL configuration is slightly different for Gazebo, jMAVSim and X-Plane.

For more information see: [Simulation](#).

JMAVSim/Gazebo HITL Environment

JMAVSim or Gazebo (running on a development computer) are connected to the flight controller hardware via USB/UART. The simulator acts as gateway to share MAVLink data between PX4 and *QGroundControl*.

Gazebo can additionally share MAVLink data with an offboard API!

The diagram below shows the simulation environment:

- A HITL configuration is selected (via *QGroundControl*) that doesn't start any real sensors.
- *jMAVSim* or *Gazebo* are connected to the flight controller via USB.
- The simulator is connected to *QGroundControl* via UDP and bridges its MAVLink messages to PX4.
- (Optional) A serial connection can be used to connect Joystick/Gamepad hardware via *QGroundControl*.
- (Optional - Gazebo only) Gazebo can also connect to an offboard API and bridge MAVLink messages to PX4.

X-Plane HITL Environment

QGroundControl is connected to the flight controller hardware via USB, and acts as a gateway to forward data between the X-Plane simulator running on a development

computer, PX4, and any offboard API. The diagram below shows the simulation environment:

- A HITL configuration is selected (via *QGroundControl*) that doesn't start any real sensors.
- *QGroundControl* is connected to the flight controller via USB.
- *QGroundControl* is connected to the simulator and offboard API via UDP.
- A serial connection is used to connect Joystick/Gamepad hardware via *QGroundControl*.

HITL vs SITL

SITL runs on a development computer in a simulated environment, and uses firmware specifically generated for that environment. Other than simulation drivers to provide fake environmental data from the simulator the system behaves normally.

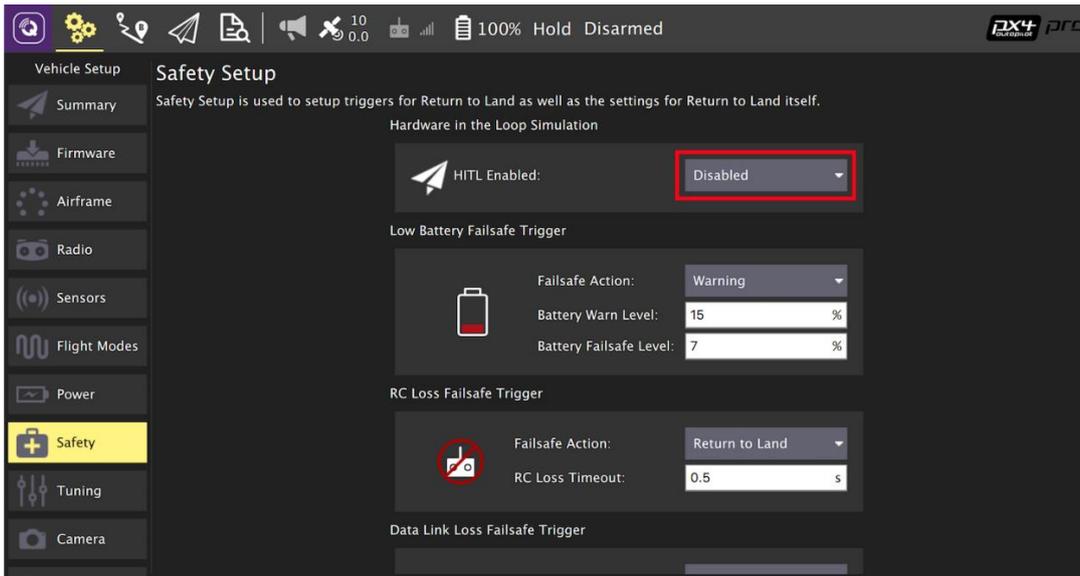
By contrast, HITL runs normal PX4 firmware in "HITL mode", on normal hardware. The simulation data enters the system at a different point than for SITL. Core modules like commander and sensors have HIL modes at startup that bypass some of the normal functionality.

In summary, HITL runs PX4 on the actual hardware using standard firmware, but SITL actually executes more of the standard system code.

Setting up HITL

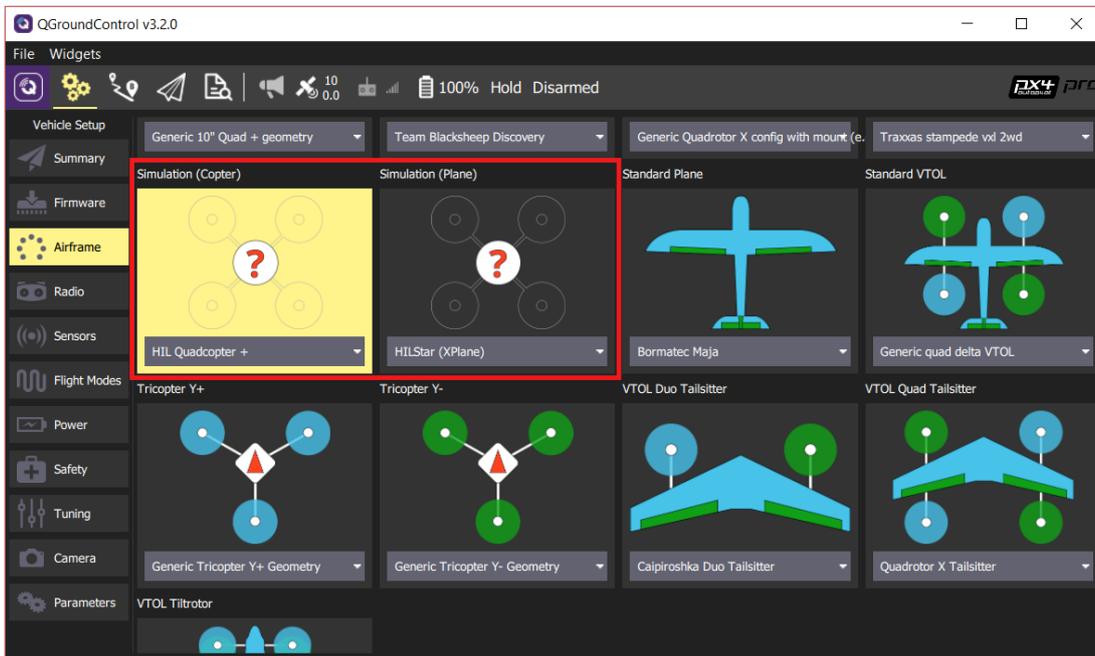
PX4 Configuration

1. Connect the autopilot directly to *QGroundControl* via USB.
2. Enable HITL Mode
 - i. Open **Setup > Safety** section.
 - ii. Enable HITL mode by selecting **Enabled** from the *HITL Enabled* list:



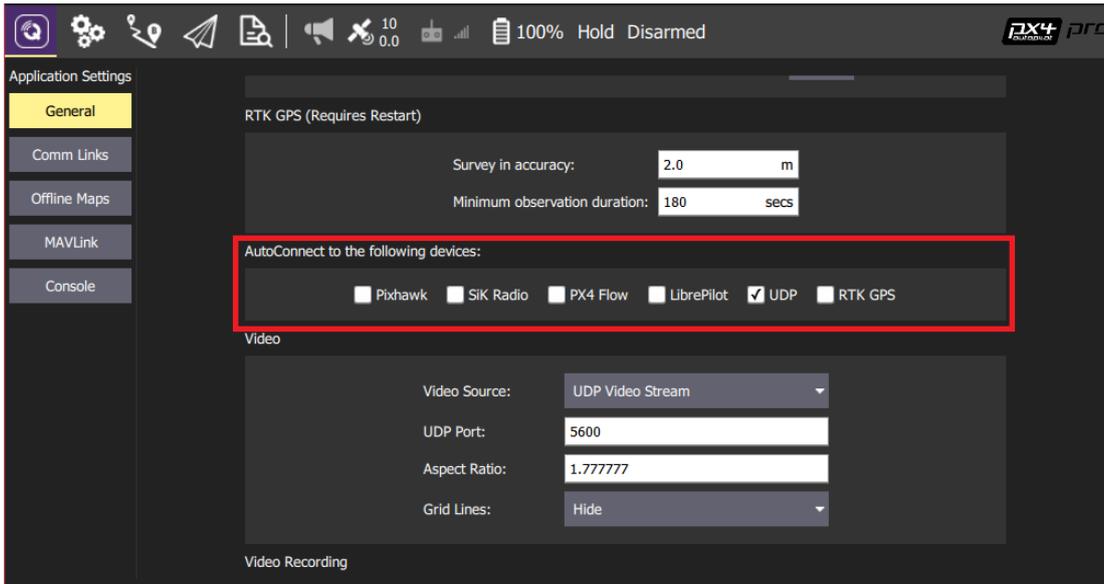
3. Select Airframe

- i. Open **Setup > Airframes**
- ii. Select a [compatible airframe](#) you want to test. Generally you'll select *HILStar* for Fixed Wing/X-Plane simulator and a *HIL QuadCopter* option for copters (and jMAVSIM or Gazebo).



4. Setup UDP

- i. Under the *General* tab of the settings menu, uncheck all *AutoConnect* boxes except for **UDP**.



5. (Gazebo only) Set the `SYS_COMPANION` parameter to `921600` (see [PX4 User Guide > Parameters](#) for instructions on how to change parameters).
6. (Optional) Configure Joystick and Failsafe. If you want to use RC instead of Joystick, configure Radio setup before step 4.

Set the following [parameters](#) in order to use a joystick instead of an RC remote control transmitter:

- [COM_RC_IN_MODE](#) to "Joystick/No RC Checks". This allows joystick input and disables RC input checks.
- [NAV_DLL_ACT](#) to "Disabled". This ensures that no RC failsafe actions interfere when not running HITL with a radio control.

The *QGroundControl User Guide* also has instructions on [Joystick](#) and [Virtual Joystick](#) setup.

Once configuration is complete, **close** *QGroundControl* and disconnect the flight controller hardware from the computer.

Simulator Setup

Follow the appropriate setup steps for your simulator in the following sections.

Gazebo

Make sure *QGroundControl* is not running!

Update the environment variables:

```
cd <Firmware_clone>
make posix_sitl_default gazebo
source Tools/setup_gazebo.bash $(pwd) $(pwd)/build/posix_sitl_default
```

Open the vehicle model's sdf file (e.g. **Tools/sitl_gazebo/models/iris/iris.sdf**).

Under the `mavlink_interface` plugin section, change the `serialEnabled` and `hil_mode` parameters to `true`.

```
<plugin name='mavlink_interface' filename='librotors_gazebo_mavlink_interface.so'>
  <robotNamespace/>
  <imuSubTopic>/imu</imuSubTopic>
  <imu_rate>170</imu_rate>
  <gpsSubTopic>/gps</gpsSubTopic>
  <mavlink_addr>INADDR_ANY</mavlink_addr>
  <mavlink_udp_port>14560</mavlink_udp_port>
  <serialEnabled>true</serialEnabled>
  <serialDevice>/dev/ttyACM0</serialDevice>
  <baudRate>921600</baudRate>
  <qgc_addr>INADDR_ANY</qgc_addr>
  <qgc_udp_port>14550</qgc_udp_port>
  <hil_mode>true</hil_mode>
  <hil_state_level>false</hil_state_level>
```

Replace the `serialDevice` parameter (`/dev/ttyACM0`) if necessary.

The serial device depends on what port is used to connect the vehicle to the computer (this is usually `/dev/ttyACM0`). An easy way to check on Ubuntu is to plug in the autopilot, open up a terminal, and type `dmesg | grep "tty"`. The correct device will be the last one shown.

Connect the flight controller to the computer and wait for it to boot.

Run Gazebo in HITL mode

```
gazebo Tools/sitl_gazebo/worlds/iris.world
```

Start *QGroundControl*. It should autoconnect to PX4 and Gazebo.

jMAVSim (Quadrotor only)

Make sure *QGroundControl* is not running!

Run jMAVSim in HITL mode (replace the serial port name `/dev/ttyACM0` if necessary - e.g. on Mac OS this would be `/dev/tty.usbmodem1`):

```
./Tools/jmavsim_run.sh -q -d /dev/ttyACM0 -b 921600 -r 250
```

Start *QGroundControl*. It should autoconnect to PX4 and jMAVSim.

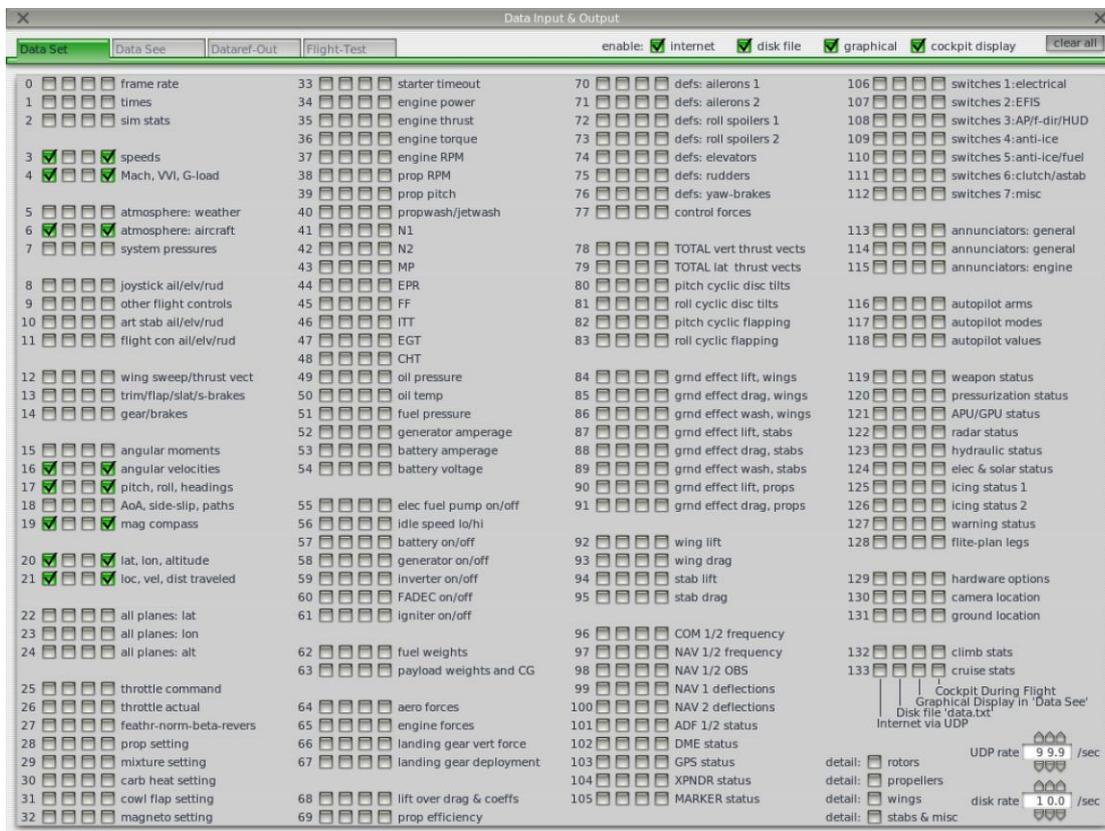
Using X-Plane (Fixed Wing only)

X-Plane is currently not recommended. Among other issues, the frame update rate is too slow to run the system realistically.

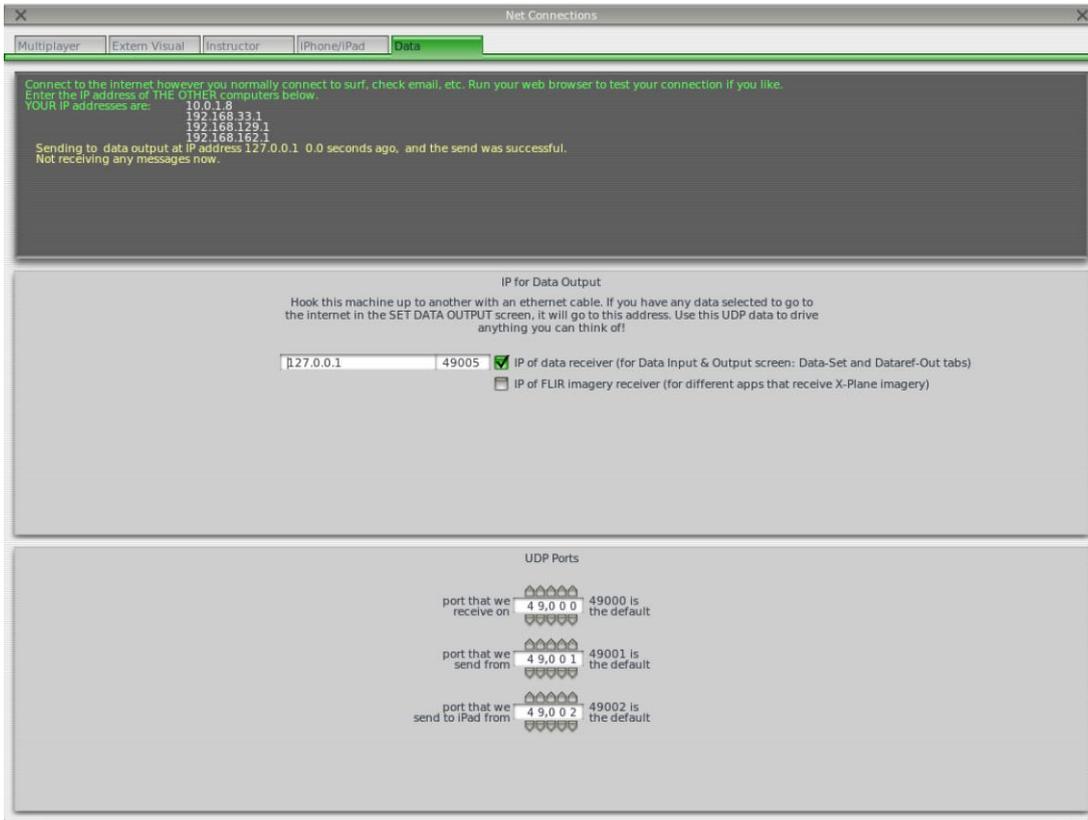
To set up X-Plane:

Open X-Plane

In **Settings > Data Input and Output**, set these checkboxes:



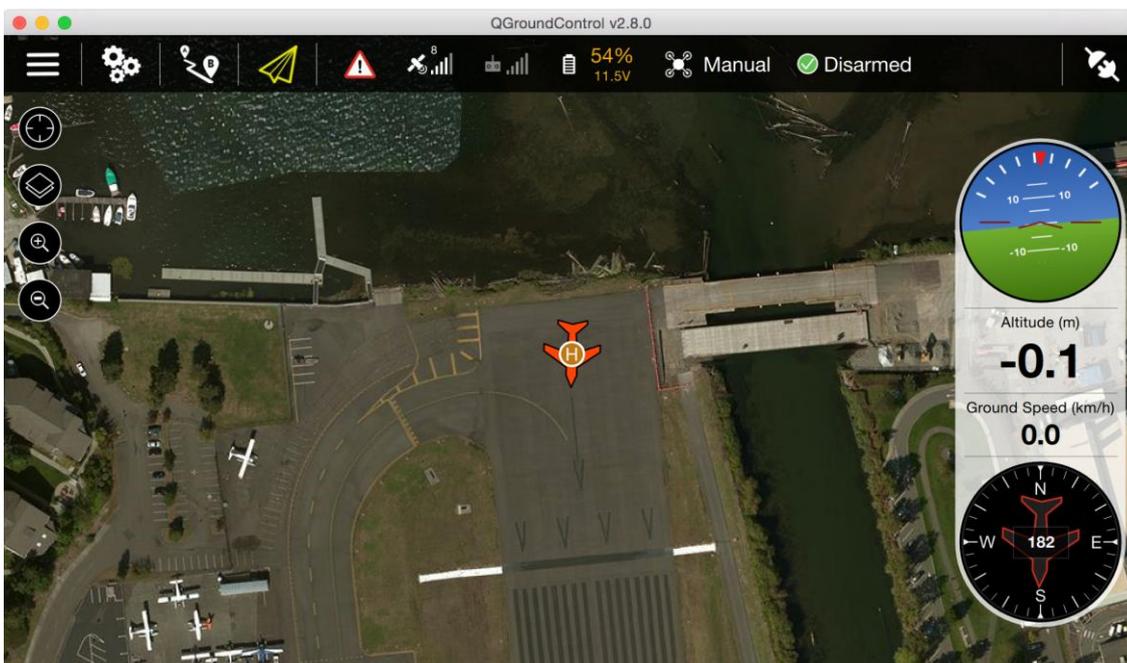
In **Settings > Net Connections**, in the *Data* tab, set localhost and port 49005 as IP address, as shown below:



Enable X-Plane HITL in *QGroundControl*:

Open *QGroundControl*

Open **Widgets > HITL Config**. Select X-Plane 10 in the drop-down and hit connect. Once the system is connected, battery status, GPS status and aircraft position should all become valid:



Fly an Autonomous Mission in HITL

You should not be able to use *QGroundControl* to [run missions](#) and otherwise control the vehicle.

5. Hardware

This section contains topics about:

- Modifying PX4 to work with *new* [flight controller hardware](#) and [vehicles/airframes](#)
- Integrating new [sensors and actuators](#), [telemetry radio](#), and other peripherals.

The [PX4 User Guide](#) contains topics about using and configuring *existing/supported* hardware.

5.1 PX4 Reference Flight Controller Design

The PX4 reference design is the [Pixhawk series](#) of flight controllers. First released in 2011, this design is now in its 5th [generation](#) (with the 6th generation board design in progress).

Binary Compatibility

All boards manufactured to a particular design are expected to be binary compatible (i.e. can run the same firmware). From 2018 we will offer a binary compatibility test suite that will allow us to verify and certify this compatibility.

FMU generations 1-3 were designed as open hardware, while FMU generations 4 and 5 provided only pinout and power supply specifications (schematics were created by individual manufacturers). In order to better ensure compatibility, FMUv6 and onward will return to a complete reference design model.

Reference Design Generations

- FMUv1: Development board (STM32F407, 128 KB RAM, 1MB flash, [schematics](#)) (no longer supported by PX4)
- FMUv2: Pixhawk (STM32F427, 168 MHz, 192 KB RAM, 1MB flash, [schematics](#))
- FMUv3: Pixhawk variants with 2MB flash (3DR Pixhawk 2 (Solo), Hex Pixhawk 2.1, Holybro Pixfalcon, 3DR Pixhawk Mini, STM32F427, 168 MHz, 192 KB RAM, 2 MB flash, [schematics](#))
- FMUv4: Pixracer (STM32F427, 168 MHz, 192 KB RAM, 2 MB flash, [pinout](#))
- FMUv4 PRO: Drotek Pixhawk 3 PRO (STM32F467, 168 MHz, 384 KB RAM, 2 MB flash, [pinout](#))
- FMUv5: final name TBD (STM32F7675, 200 MHz, 512 KB RAM, 2 MB flash, [pinout](#))
- FMUv6: work in progress, final name TBD, variant 6s (STM32H7, 400 MHz, 2 MB RAM, 2 MB flash) and variant 6i (i.MX RT1050, 600 MHz, 512 KB RAM, external flash)

5.2 Flight Controller Porting Guide

This topic is for developers who want to port PX4 to work with *new* flight controller hardware.

Architecture

PX4 consists of two main layers: The board support and middleware layer on top of the host OS (NuttX, Linux or any other POSIX platform like Mac OS). And the applications (Flight Stack in [src/modules](#)).

This guide is focused only on the middleware as the applications will run on any board target.

NuttX Boards

The location of the main files for a NuttX board are:

- Board startup and PX4 board configuration: [src/drivers/boards](#). It contains bus and GPIO mappings and the board initialization code.
- FMUv5 example: [src/drivers/boards/px4fmu-v5](#).
- NuttX board config: [nuttx-configs](#). The OS gets loaded as part of the application build.
- FMUv5 example: [nuttx-configs/px4fmu-v5/include/board.h](#).
- NuttX OS config (created with menuconfig): [nuttx-configs/px4fmu-v5/nsh/defconfig](#).
- Linker scripts and other required settings: [nuttx-configs](#).
- FMUv5 example: [nuttx-configs/px4fmu-v5](#).
- Boot file system (startup script): [ROMFS/px4fmu_common](#)
- Board build configuration: [cmake/configs/nuttx_px4fmu-v5_default.cmake](#).
- Drivers: [src/drivers](#).
- Reference config: Running `make px4fmu-v4_default` builds the FMUv4 config, which is the current NuttX reference configuration.

NuttX Menuconfig

If you need to modify the NuttX [menuconfig](#) you can do this using the PX4 shortcuts:

```
make px4fmu-v2_default menuconfig
make px4fmu-v2_default qconfig
```

QuRT / Hexagon

- The start script is located in [posix-configs/](#).

-
- The OS configuration is part of the default Linux image (TODO: Provide location of LINUX IMAGE and flash instructions).
 - The PX4 middleware configuration is located in [src/drivers/boards](#). TODO: ADD BUS CONFIG
 - Drivers: [DriverFramework](#).
 - Reference config: Running `make qurt_eagle_release` builds the Snapdragon Flight reference config.

Linux Boards

Linux boards do not include the OS and kernel configuration. These are already provided by the Linux image available for the board (which needs to support the inertial sensors out of the box).

- [cmake/configs/posix_rpi_cross.cmake](#) - RPI cross-compilation.

Related Information

- [Device Drivers](#) - How to support new peripheral hardware (device drivers)
- [Building the Code](#) - How to build source and upload firmware
- Supported Flight Controllers:
 - [Autopilot Hardware](#) (PX4 User Guide)
 - [Supported boards list](#) (Github)
- [Supported Peripherals](#) (PX4 User Guide)

5.3 Airframes

PX4 has a flexible [mixing system](#) that allows it to support almost any imaginable vehicle type/frame through a single codebase:

- **Planes:** Normal planes, flying wings, inverted V-tail planes, etc.
- **Multicopters:** Helicopters, tricopters, quadcopters, hexarotors, dodecarotors in many different geometries.
- **VTOL Airframes:** VTOL configurations including: Tailsitters, Tiltrotors, and QuadPlanes (plane + quad).
- **UGVs/Rovers:** Basic support has been added for Unmanned Ground Vehicles, enabling both manual and mission-based control.

You can find a list of all supported frame types and motor outputs in the [Airframes Reference](#).

This section provides information that is relevant to developers who want to add support for new vehicles or vehicle types to PX4, including build logs for vehicles that are still being developed.

PX4 is also well-suited for use in other vehicle types and general robots, ranging from submarine, boats, or amphibious vehicles, through to experimental aircraft and rockets. *Let us know* if you have a new vehicle or frame-type you want to help support in PX4.

Build logs for some of the supported airframes can be found in [PX4 User Guide > Airframes](#).

Airframes Reference

This list is auto-generated from the source code.

The **AUX** channels are only available on Pixhawk Boards (labeled with **AUX OUT**).

This page lists all supported airframes and types including the motor assignment and numbering. The motors in **green** rotate clockwise, the ones in **blue** counterclockwise.

Copter

Coaxial Helicopter

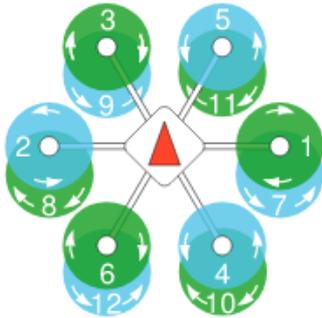


Common Outputs

- **MAIN1**: Left swashplate servomotor, pitch axis
- **MAIN2**: Right swashplate servomotor, roll axis
- **MAIN3**: Upper rotor (CCW)
- **MAIN4**: Lower rotor (CW)

Name	
Esky (Big) Lama v4	Maintainer: Emmanuel Roussel <code>SYS_AUTOSTART</code> = 15001

Dodecarotor cox



Common Outputs

- **MAIN1**: motor 1
- **MAIN2**: motor 2
- **MAIN3**: motor 3
- **MAIN4**: motor 4
- **MAIN5**: motor 5
- **MAIN6**: motor 6
- **AUX1**: motor 7
- **AUX2**: motor 8
- **AUX3**: motor 9
- **AUX4**: motor 10
- **AUX5**: motor 11
- **AUX6**: motor 12

Name	
Generic Dodecarotor cox geometry	Maintainer: William Peale <code>SYS_AUTOSTART</code> = 24001

Helicopter

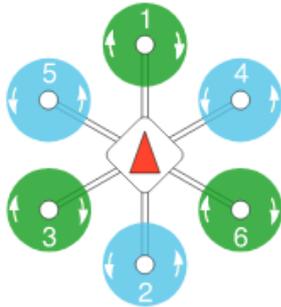


Common Outputs

- **MAIN1**: main motor
- **MAIN2**: front swashplate servo
- **MAIN3**: right swashplate servo
- **MAIN4**: left swashplate servo
- **MAIN5**: tail-rotor servo

Name	
Blade 130X	Maintainer: Bart Slinger <code>SYS_AUTOSTART</code> = 16001

Hexarotor +

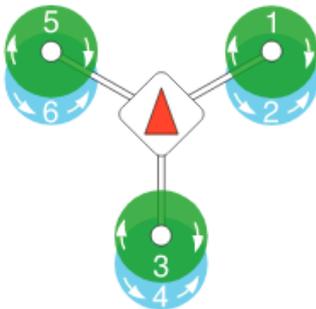


Common Outputs

- **MAIN1**: motor1
- **MAIN2**: motor2
- **MAIN3**: motor3
- **MAIN4**: motor4
- **MAIN5**: motor5
- **MAIN6**: motor6
- **AUX1**: feed-through of RC AUX1 channel
- **AUX2**: feed-through of RC AUX2 channel
- **AUX3**: feed-through of RC AUX3 channel

Name	
Generic Hexarotor + geometry	Maintainer: Lorenz Meier <code>SYS_AUTOSTART = 7001</code>

Hexarotor Coaxial

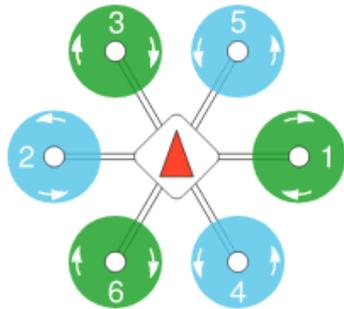


Common Outputs

- **MAIN1**: front right top, CW; angle:60; direction:CCW
- **MAIN2**: front right bottom, CCW; angle:60; direction:CCW
- **MAIN3**: back top, CW; angle:180; direction:CCW
- **MAIN4**: back bottom, CCW; angle:180; direction:CCW
- **MAIN5**: front left top, CW; angle:-60; direction:CCW
- **MAIN6**: front left bottom, CCW; angle:-60; direction:CCW
- **AUX1**: feed-through of RC AUX1 channel
- **AUX2**: feed-through of RC AUX2 channel
- **AUX3**: feed-through of RC AUX3 channel

Name	
Generic Hexarotor coaxial geometry	Maintainer: Lorenz Meier <code>SYS_AUTOSTART = 11001</code>

Hexarotor x

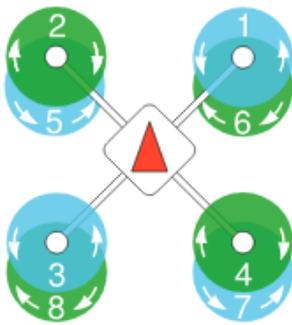


Common Outputs

- **MAIN1**: motor 1
- **MAIN2**: motor 2
- **MAIN3**: motor 3
- **MAIN4**: motor 4
- **MAIN5**: motor 5
- **MAIN6**: motor 6
- **AUX1**: feed-through of RC AUX1 channel
- **AUX2**: feed-through of RC AUX2 channel
- **AUX3**: feed-through of RC AUX3 channel

Name	
Generic Hexarotor x geometry	Maintainer: Lorenz Meier SYS_AUTOSTART = 6001

Octo Coax Wide

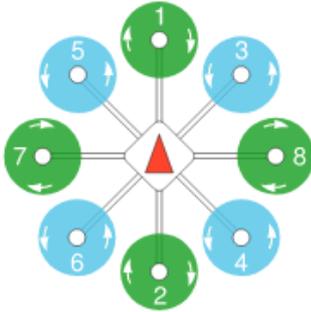


Common Outputs

- **MAIN1**: motor 1
- **MAIN2**: motor 2
- **MAIN3**: motor 3
- **MAIN4**: motor 4
- **MAIN5**: motor 5
- **MAIN6**: motor 6
- **MAIN7**: motor 7
- **MAIN8**: motor 8

Name	
Steadidrone MAVRIK	Maintainer: Simon Wilks SYS_AUTOSTART = 12002

Octorotor +

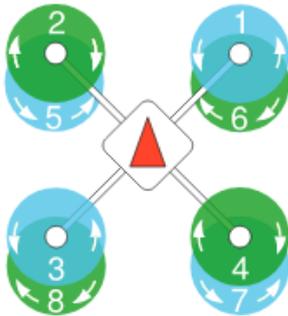


Common Outputs

- **MAIN1**: motor 1
- **MAIN2**: motor 2
- **MAIN3**: motor 3
- **MAIN4**: motor 4
- **MAIN5**: motor 5
- **MAIN6**: motor 6
- **MAIN7**: motor 7
- **MAIN8**: motor 8
- **AUX1**: feed-through of RC AUX1 channel
- **AUX2**: feed-through of RC AUX2 channel
- **AUX3**: feed-through of RC AUX3 channel

Name	
Generic Octocopter + geometry	Maintainer: Lorenz Meier <code>SYS_AUTOSTART</code> = 9001

Octorotor Coaxial

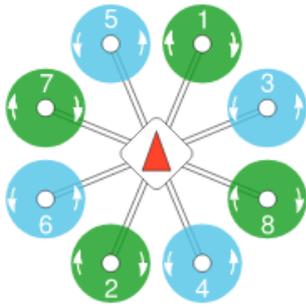


Common Outputs

- **MAIN1**: motor 1
- **MAIN2**: motor 2
- **MAIN3**: motor 3
- **MAIN4**: motor 4
- **MAIN5**: motor 5
- **MAIN6**: motor 6
- **MAIN7**: motor 7
- **MAIN8**: motor 8

Name	
Generic 10" Octo coaxial geometry	Maintainer: Lorenz Meier <code>SYS_AUTOSTART</code> = 12001

Octorotor x

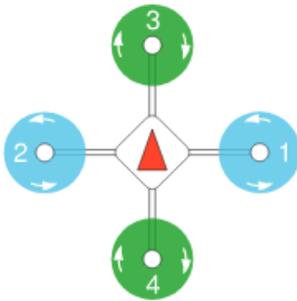


Common Outputs

- **MAIN1**: motor 1
- **MAIN2**: motor 2
- **MAIN3**: motor 3
- **MAIN4**: motor 4
- **MAIN5**: motor 5
- **MAIN6**: motor 6
- **MAIN7**: motor 7
- **MAIN8**: motor 8
- **AUX1**: feed-through of RC AUX1 channel
- **AUX2**: feed-through of RC AUX2 channel
- **AUX3**: feed-through of RC AUX3 channel

Name	
Generic Octocopter X geometry	Maintainer: Lorenz Meier <code>SYS_AUTOSTART</code> = 8001

Quadrotor +

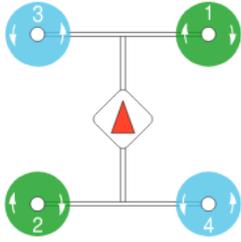


Common Outputs

- **MAIN1**: motor 1
- **MAIN2**: motor 2
- **MAIN3**: motor 3
- **MAIN4**: motor 4
- **MAIN5**: feed-through of RC AUX1 channel
- **MAIN6**: feed-through of RC AUX2 channel
- **AUX1**: feed-through of RC AUX1 channel
- **AUX2**: feed-through of RC AUX2 channel
- **AUX3**: feed-through of RC AUX3 channel
- **AUX4**: feed-through of RC FLAPS channel

Name	
Generic 10" Quad + geometry	Maintainer: Lorenz Meier <code>SYS_AUTOSTART</code> = 5001

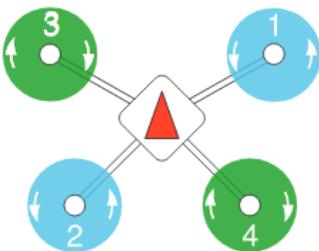
Quadrotor H



Common Outputs	
•	MAIN1 : motor 1
•	MAIN2 : motor 2
•	MAIN3 : motor 3
•	MAIN4 : motor 4
•	MAIN5 : feed-through of RC AUX1 channel
•	MAIN6 : feed-through of RC AUX2 channel

Name	
Reaper 500 Quad	Maintainer: Blankered <code>SYS_AUTOSTART = 4040</code>

Quadrotor Wide



Common Outputs	
•	AUX1 : feed-through of RC AUX1 channel
•	AUX2 : feed-through of RC AUX2 channel
•	AUX3 : feed-through of RC AUX3 channel
•	AUX4 : feed-through of RC FLAPS channel

Name	
Team Blacksheep Discovery	Maintainer: Lorenz Meier <code>SYS_AUTOSTART = 10015</code> Specific Outputs: <ul style="list-style-type: none"> • MAIN1: motor 1 • MAIN2: motor 2 • MAIN3: motor 3 • MAIN4: motor 4 • MAIN5: feed-through of RC AUX1 channel • MAIN6: feed-through of RC AUX2 channel
3DR Iris Quadrotor	Maintainer: Lorenz Meier <code>SYS_AUTOSTART = 10016</code> Specific Outputs: <ul style="list-style-type: none"> • MAIN1: motor 1 • MAIN2: motor 2 • MAIN3: motor 3 • MAIN4: motor 4

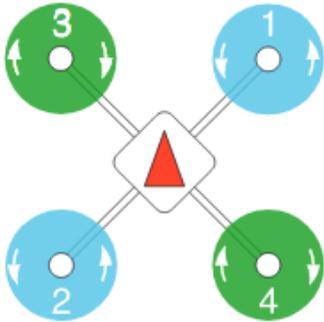
Steadidrone QU4D	Maintainer: Lorenz Meier SYS_AUTOSTART = 10017 Specific Outputs: <ul style="list-style-type: none"> • MAIN1: motor 1 • MAIN2: motor 2 • MAIN3: motor 3 • MAIN4: motor 4 • MAIN5: feed-through of RC AUX1 channel • MAIN6: feed-through of RC AUX2 channel
Team Blacksheep Discovery Endurance	Maintainer: Simon Wilks SYS_AUTOSTART = 10018 Specific Outputs: <ul style="list-style-type: none"> • MAIN1: motor 1 • MAIN2: motor 2 • MAIN3: motor 3 • MAIN4: motor 4 • MAIN5: feed-through of RC AUX1 channel • MAIN6: feed-through of RC AUX2 channel

Quadrotor asymmetric

Common Outputs	
<ul style="list-style-type: none"> • MAIN1: motor1 (front right: CCW) • MAIN2: motor2 (back left: CCW) • MAIN3: motor3 (front left: CW) • MAIN4: motor4 (back right: CW) • MAIN5: feed-through of RC AUX1 channel • MAIN6: feed-through of RC AUX2 channel 	

Name	
Spedix S250AQ	Maintainer: Lorenz Meier SYS_AUTOSTART = 4051

Quadrotor x



Common Outputs

- **MAIN1**: motor 1
- **MAIN2**: motor 2
- **MAIN3**: motor 3
- **MAIN4**: motor 4
- **MAIN5**: feed-through of RC AUX1 channel
- **MAIN6**: feed-through of RC AUX2 channel

Name	
Generic Quadrotor x	<p>Maintainer: Lorenz Meier</p> <p><code>SYS_AUTOSTART</code> = 4001</p> <p>Specific Outputs:</p> <ul style="list-style-type: none"> • AUX1: feed-through of RC AUX1 channel • AUX2: feed-through of RC AUX2 channel • AUX3: feed-through of RC AUX3 channel • AUX4: feed-through of RC FLAPS channel
Generic Quadrotor x with mount (e.g. gimbal)	<p>Maintainer: Leon Mueller</p> <p><code>SYS_AUTOSTART</code> = 4002</p> <p>Specific Outputs:</p> <ul style="list-style-type: none"> • AUX1: Mount pitch • AUX2: Mount roll • AUX3: Mount yaw • AUX4: Mount retract
Lumenier QAV-R (raceblade) 5" arms	<p>Maintainer: James Goppert</p> <p><code>SYS_AUTOSTART</code> = 4003</p>
H4 680mm with Z1 Tiny2 Gimbal	<p>Maintainer: Leon Mueller</p> <p><code>SYS_AUTOSTART</code> = 4004</p>

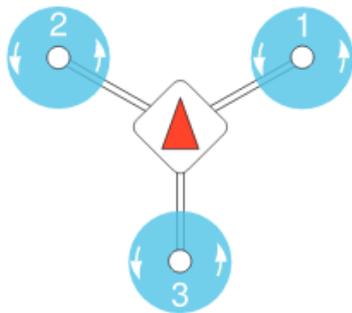
Lumenier QAV250	Maintainer: Lorenz Meier SYS_AUTOSTART = 4009
DJI Flame Wheel F330	Maintainer: Lorenz Meier SYS_AUTOSTART = 4010
DJI Flame Wheel F450	Maintainer: Lorenz Meier SYS_AUTOSTART = 4011
F450-sized quadrotor with CAN	Maintainer: Pavel Kirienko SYS_AUTOSTART = 4012
Parrot Bebop Frame	Maintainer: Michael Schaeuble SYS_AUTOSTART = 4013
Hobbyking Micro PCB	Maintainer: Thomas Gubler SYS_AUTOSTART = 4020
3DR Solo	Maintainer: Andreas Antener SYS_AUTOSTART = 4030
3DR DIY Quad	Maintainer: Lorenz Meier SYS_AUTOSTART = 4031

Generic 250 Racer	Maintainer: Lorenz Meier SYS_AUTOSTART = 4050
DJI Matrice 100	Maintainer: James Goppert SYS_AUTOSTART = 4060
Intel Aero Ready to Fly Drone	Maintainer: Beat Kueng SYS_AUTOSTART = 4070
ZMR250 Racer	Maintainer: Anton Matosov SYS_AUTOSTART = 4080
NanoMind 110 Quad	Maintainer: Henry Zhang SYS_AUTOSTART = 4090
Crazyflie 2.0	Maintainer: Dennis Shtatov SYS_AUTOSTART = 4900

Simulation (Copter)

Name	
HIL Quadcopter X	Maintainer: Lorenz Meier <code>SYS_AUTOSTART = 1001</code>

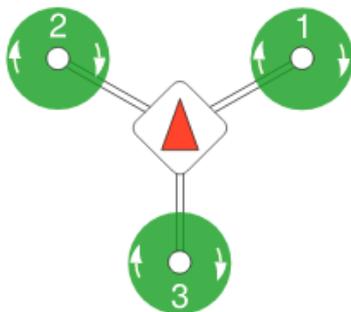
Tricopter Y+



Common Outputs
<ul style="list-style-type: none"> • MAIN1: motor 1 • MAIN2: motor 2 • MAIN3: motor 3 • MAIN4: yaw servo

Name	
Generic Tricopter Y+ Geometry	Maintainer: Trent Lukaczyk <code>SYS_AUTOSTART = 14001</code>

Tricopter Y-

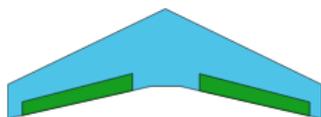


Common Outputs
<ul style="list-style-type: none"> • MAIN1: motor 1 • MAIN2: motor 2 • MAIN3: motor 3 • MAIN4: yaw servo

Name	
Generic Tricopter Y- Geometry	Maintainer: Trent Lukaczyk <code>SYS_AUTOSTART = 14002</code>

Plane

Flying Wing

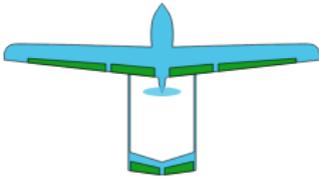


Common Outputs

- **MAIN1**: left aileron
- **MAIN2**: right aileron
- **MAIN4**: throttle
- **AUX1**: feed-through of RC AUX1 channel
- **AUX2**: feed-through of RC AUX2 channel
- **AUX3**: feed-through of RC AUX3 channel

Name	
Generic Flying Wing	<code>SYS_AUTOSTART</code> = 3000
IO Camflyer	Maintainer: Simon Wilks <code>SYS_AUTOSTART</code> = 3030
Phantom FPV Flying Wing	Maintainer: Simon Wilks <code>SYS_AUTOSTART</code> = 3031
Skywalker X5 Flying Wing	Maintainer: Julian Oes <code>SYS_AUTOSTART</code> = 3032
Wing Wing (aka Z-84) Flying Wing	Maintainer: Lorenz Meier <code>SYS_AUTOSTART</code> = 3033
FX-79 Buffalo Flying Wing	Maintainer: Simon Wilks <code>SYS_AUTOSTART</code> = 3034
Viper	Maintainer: Simon Wilks <code>SYS_AUTOSTART</code> = 3035
Sparkle Tech Pigeon	Maintainer: Simon Wilks <code>SYS_AUTOSTART</code> = 3036
Modified Parrot Disco	Maintainer: Jan Liphardt <code>SYS_AUTOSTART</code> = 3037
TBS Caipirinha	Maintainer: Lorenz Meier <code>SYS_AUTOSTART</code> = 3100

Plane A-Tail



Common Outputs	
	<ul style="list-style-type: none"> • MAIN1: aileron right • MAIN2: aileron left • MAIN3: v-tail right • MAIN4: v-tail left • MAIN5: throttle • MAIN6: wheel • MAIN7: flaps right • MAIN8: flaps left • AUX1: feed-through of RC AUX1 channel • AUX2: feed-through of RC AUX2 channel • AUX3: feed-through of RC AUX3 channel

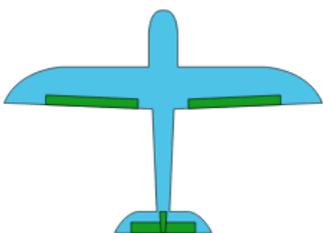
Name	
Applied Aeronautics Albatross	Maintainer: Andreas Antener SYS_AUTOSTART = 2106

Simulation (Plane)

Common Outputs	
	<ul style="list-style-type: none"> • MAIN1: aileron • MAIN2: elevator • MAIN3: rudder • MAIN4: throttle • MAIN5: flaps • MAIN6: gear

Name	
HILStar (XPlane)	Maintainer: Lorenz Meier SYS_AUTOSTART = 1000

Standard Plane

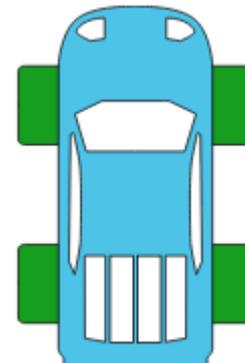


Common Outputs	
	<ul style="list-style-type: none"> • AUX1: feed-through of RC AUX1 channel • AUX2: feed-through of RC AUX2 channel • AUX3: feed-through of RC AUX3 channel

Name	
Standard Plane	Maintainer: Lorenz Meier SYS_AUTOSTART = 2100 Specific Outputs: <ul style="list-style-type: none"> • MAIN1: aileron • MAIN2: elevator • MAIN3: throttle • MAIN4: rudder • MAIN5: flaps • MAIN6: gear
Bormatec Maja	Maintainer: Andreas Antener SYS_AUTOSTART = 2105 Specific Outputs: <ul style="list-style-type: none"> • MAIN1: aileron • MAIN2: aileron • MAIN3: elevator • MAIN4: rudder • MAIN5: throttle • MAIN6: wheel • MAIN7: flaps

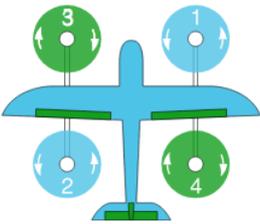
Rover

Name	
Generic Ground Vehicle	SYS_AUTOSTART = 50000 Specific Outputs: <ul style="list-style-type: none"> • MAIN2: steering • MAIN4: throttle
Axial Racing AX10	Maintainer: John Doe SYS_AUTOSTART = 50001 Specific Outputs: <ul style="list-style-type: none"> • MAIN1: pass-through of control group 0, channel 0 • MAIN2: pass-through of control group 0, channel 1 • MAIN3: pass-through of control group 0, channel 2 • MAIN4: pass-through of control group 0, channel 3 • MAIN5: pass-through of control group 0, channel 4 • MAIN6: pass-through of control group 0, channel 5 • MAIN7: pass-through of control group 0, channel 6 • MAIN8: pass-through of control group 0, channel 7
Traxxas stampede vxl 2wd	Maintainer: Marco Zorzi SYS_AUTOSTART = 50002 Specific Outputs: <ul style="list-style-type: none"> • MAIN2: steering • MAIN4: throttle



VTOL

Standard VTOL

Name	
<p>Generic Quadplane VTOL</p> 	<p><code>SYS_AUTOSTART = 13000</code></p> <p>Specific Outputs:</p> <ul style="list-style-type: none"> • MAIN1: motor 1 • MAIN2: motor 2 • MAIN3: motor 3 • MAIN4: motor 4 • AUX1: Aileron 1 • AUX2: Aileron 2 • AUX3: Elevator • AUX4: Rudder • AUX5: Throttle
<p>Fun Cub Quad VTOL</p>	<p>Maintainer: Simon Wilks</p> <p><code>SYS_AUTOSTART = 13005</code></p> <p>Specific Outputs:</p> <ul style="list-style-type: none"> • MAIN1: motor 1 • MAIN2: motor 2 • MAIN3: motor 3 • MAIN4: motor 4 • AUX1: Aileron 1 • AUX2: Aileron 2 • AUX3: Elevator • AUX4: Rudder • AUX5: Throttle
<p>Generic quad delta VTOL</p>	<p>Maintainer: Simon Wilks</p> <p><code>SYS_AUTOSTART = 13006</code></p> <p>Specific Outputs:</p> <ul style="list-style-type: none"> • MAIN1: motor 1 • MAIN2: motor 2 • MAIN3: motor 3 • MAIN4: motor 4 • AUX1: Right elevon • AUX2: Left elevon • AUX3: Motor
<p>Generic AAVT v-tail plane airframe with Quad VTOL.</p>	<p>Maintainer: Sander Smeets</p> <p><code>SYS_AUTOSTART = 13007</code></p>
<p>QuadRanger</p>	<p>Maintainer: Sander Smeets</p> <p><code>SYS_AUTOSTART = 13008</code></p>
<p>Sparkle Tech Ranger VTOL</p>	<p>Maintainer: Andreas Antener</p> <p><code>SYS_AUTOSTART = 13009</code></p>

DeltaQuad	Maintainer: Sander Smeets SYS_AUTOSTART = 13013 Specific Outputs: <ul style="list-style-type: none"> • MAIN1: motor 1 • MAIN2: motor 2 • MAIN3: motor 3 • MAIN4: motor 4 • MAIN5: Right elevon • MAIN6: Left elevon • MAIN7: Pusher motor • MAIN8: Pusher reverse channel
-----------	--

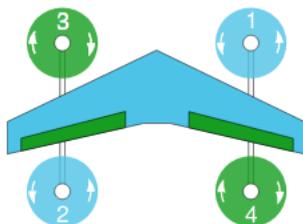
VTOL Duo Tailsitter



Common Outputs	
• MAIN1: motor right	
• MAIN2: motor left	
• MAIN5: elevon right	
• MAIN6: elevon left	

Name	
Caipiroshka Duo Tailsitter	Maintainer: Roman Bapst SYS_AUTOSTART = 13001

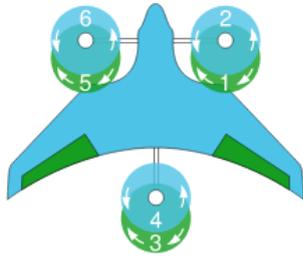
VTOL Quad Tailsitter



Common Outputs	
• MAIN1: motor 1	
• MAIN2: motor 2	
• MAIN3: motor 3	
• MAIN4: motor 4	
• MAIN5: elevon left	
• MAIN6: elevon right	
• MAIN7: canard surface	
• MAIN8: rudder	

Name	
Quadrotor X Tailsitter	Maintainer: Roman Bapst SYS_AUTOSTART = 13003
Quadrotor + Tailsitter	Maintainer: Roman Bapst SYS_AUTOSTART = 13004

VTOL Tiltrotor



Common Outputs
<ul style="list-style-type: none"> • AUX1: Tilt servo • AUX2: Elevon 1 • AUX3: Elevon 2 • AUX4: Gear

Name	
BirdsEyeView Aerobotics FireFly6	Maintainer: Roman Bapst <code>SYS_AUTOSTART = 13002</code> Specific Outputs: <ul style="list-style-type: none"> • MAIN1: Front right motor bottom • MAIN2: Front right motor top • MAIN3: Back motor bottom • MAIN4: Back motor top • MAIN5: Front left motor bottom • MAIN6: Front left motor top
CruiseAder Claire	Maintainer: Samay Siga <code>SYS_AUTOSTART = 13010</code>
E-flite Convergence	Maintainer: Andreas Antener <code>SYS_AUTOSTART = 13012</code> Specific Outputs: <ul style="list-style-type: none"> • MAIN1: Motor right • MAIN2: Motor left • MAIN3: Motor back • MAIN4: empty • MAIN5: Tilt servo right • MAIN6: Tilt servo left • MAIN7: Elevon right • MAIN8: Elevon left

Adding a New Airframe Configuration

PX4 uses canned airframe configurations as starting point for airframes. The configurations are defined in [config files](#) that are stored in the [ROMFS/px4fmu_common/init.d](#) folder. The config files reference [mixer files](#) that

describe the physical configuration of the system, and which are stored in the [ROMFS/px4fmu_common/mixers](#) folder.

Adding a configuration is straightforward: create a new config file in the [init.d folder](#) (prepend the filename with an unused autostart ID), then [build and upload](#) the software.

Developers who do not want to create their own configuration can instead customize existing configurations using text files on the microSD card, as detailed on the [custom system startup](#) page.

Configuration File Overview

The configuration in the config and mixer files consists of several main blocks:

- Airframe documentation (used in the [Airframes Reference](#) and *QGroundControl*).
- Vehicle-specific parameter settings, including [tuning gains](#).
- The controllers and apps it should start, e.g. multicopter or fixed wing controllers, land detectors etc.
- The physical configuration of the system (e.g. a plane, wing or multicopter). This is called a [mixer](#).

These aspects are mostly independent, which means that many configurations share the same physical layout of the airframe, start the same applications and differ most in their tuning gains.

New airframe files are only automatically added to the build system after a clean build (run `make clean`).

Config File

A typical configuration file is shown below ([original file here](#)) .

The first section is the airframe documentation. This is used in the [Airframes Reference](#) and *QGroundControl*.

```
#!nsh
#
# @name Wing Wing (aka Z-84) Flying Wing
#
# @url https://docs.px4.io/en/framebuild_plane/wing_wing_z84.html
#
# @type Flying Wing
# @class Plane
#
# @output MAIN1 left aileron
# @output MAIN2 right aileron
```

```
# @output MAIN4 throttle
#
# @output AUX1 feed-through of RC AUX1 channel
# @output AUX2 feed-through of RC AUX2 channel
# @output AUX3 feed-through of RC AUX3 channel
#
# @maintainer Lorenz Meier <lorenz@px4.io>
#
```

The next section specifies vehicle-specific parameters, including [tuning gains](#):

```
sh /etc/init.d/rc.fw_defaults

if [ $AUTOCONF == yes ]
then
    param set BAT_N_CELLS 2
    param set FW_AIRSPD_MAX 15
    param set FW_AIRSPD_MIN 10
    param set FW_AIRSPD_TRIM 13
    param set FW_R_TC 0.3
    param set FW_P_TC 0.3
    param set FW_L1_DAMPING 0.74
    param set FW_L1_PERIOD 16
    param set FW_LND_ANG 15
    param set FW_LND_FLALT 5
    param set FW_LND_HHDIST 15
    param set FW_LND_HVIRT 13
    param set FW_LND_TLALT 5
    param set FW_THR_LND_MAX 0
    param set FW_PR_FF 0.35
    param set FW_RR_FF 0.6
    param set FW_RR_P 0.04
fi
```

Set frame type ([MAV_TYPE](#)):

```
# Configure this as plane
set MAV_TYPE 1
```

Set the [mixer](#) to use:

```
# Set mixer
set MIXER wingwing
```

Configure PWM outputs (specify the outputs to drive/activate, and the levels).

```
# Provide ESC a constant 1000 us pulse
set PWM_OUT 4
set PWM_DISARMED 1000
```

If you want to reverse a channel, never do this on your RC transmitter or with e.g. `RC1_REV`. The channels are only reversed when flying in manual mode, when you switch in an autopilot flight mode, the channels output will still be wrong (it only inverts your RC signal). Thus for a correct channel assignment change either your PWM signals with `PWM_MAIN_REV1` (e.g. for channel one) or change the signs of the output scaling in the corresponding mixer (see below).

Mixer File

First read [Concepts > Mixing](#). This provides background information required to interpret this mixer file.

A typical mixer file is shown below.

The mixer file contains several blocks of code, each of which refers to one actuator or ESC. So if you have e.g. two servos and one ESC, the mixer file will contain three blocks of code.

The plugs of the servos / motors go in the order of the mixers in this file.

So MAIN1 would be the left aileron, MAIN2 the right aileron, MAIN3 is empty (note the Z: zero mixer) and MAIN4 is throttle (to keep throttle on output 4 for common fixed wing configurations).

A mixer is encoded in normalized units from -10000 to 10000, corresponding to -1..+1.

```
M: 2
O:   10000 10000   0 -10000 10000
S: 0 0  -6000  -6000   0 -10000 10000
S: 0 1   6500   6500   0 -10000 10000
```

Where each number from left to right means:

- M: Indicates two scalars for two control inputs. It indicates the number of control inputs the mixer will receive.
- O: Indicates the output scaling (*1 in negative*, 1 in positive), offset (zero here), and output range (-1..+1 here). If you want to invert your PWM signal, the signs of the output scalings have to be changed. (O: -10000 -10000 0 -10000 10000)
- S: Indicates the first input scaler: It takes input from control group #0 (Flight Control) and the first input (roll). It scales the roll control input * 0.6 and reverts the sign (-0.6 becomes -6000 in scaled units). It applies no offset (0) and outputs to the full range (-1..+1)
- S: Indicates the second input scaler: It takes input from control group #0 (Flight Control) and the second input (pitch). It scales the pitch control input * 0.65. It applies no offset (0) and outputs to the full range (-1..+1)

In short, the output of this mixer would be $SERVO = ((roll\ input\ -0.6 + 0) + (pitch\ input\ 0.65 + 0)) * 1 + 0$

Behind the scenes, both scalers are added, which for a flying wing means the control surface takes maximum 60% deflection from roll and 65% deflection from pitch.

The complete mixer looks like this:

```
Delta-wing mixer for PX4FMU
=====

Designed for Wing Wing Z-84

This file defines mixers suitable for controlling a delta wing aircraft using
PX4FMU. The configuration assumes the elevon servos are connected to PX4FMU
servo outputs 0 and 1 and the motor speed control to output 3. Output 2 is
assumed to be unused.

Inputs to the mixer come from channel group 0 (vehicle attitude), channels 0
(roll), 1 (pitch) and 3 (thrust).

See the README for more information on the scaler format.

Elevon mixers
-----
Three scalers total (output, roll, pitch).

The scaling factor for roll inputs is adjusted to implement differential travel
for the elevons.

This first block of code is for Servo 0...

M: 2
O: 10000 10000 0 -10000 10000
S: 0 0 -6000 -6000 0 -10000 10000
S: 0 1 6500 6500 0 -10000 10000

And this is for Servo 1...

M: 2
O: 10000 10000 0 -10000 10000
S: 0 0 -6000 -6000 0 -10000 10000
S: 0 1 -6500 -6500 0 -10000 10000
```

Note that `in` principle, you could implement left/right wing asymmetric mixing, but `in` general the two blocks of code will be numerically equal, and just differ by the sign of the third line (`S: 0 1`), since to roll the plane, the two ailerons must move `in` OPPOSITE directions. The signs of the second lines (`S: 0 0`) are identical, since to pitch the plane, both servos need to move `in` the SAME direction.

Output 2

This mixer is empty.

Z:

Motor speed mixer

Two scalars total (output, thrust).

This mixer generates a full-range output (-1 to 1) from an input `in` the (0 - 1) range. Inputs below zero are treated as zero.

M: 1

O: 10000 10000 0 -10000 10000

S: 0 3 0 20000 -10000 -10000 10000

Tuning Gains

The following *PX4 User Guide* topics explain how to tune the parameters that will be specified in the config file:

- [Multicopter PID Tuning Guide](#)
- [Fixed Wing PID Tuning Guide](#)
- [VTOL Configuration](#)

Getting a New Airframe to Show in QGroundControl

The airframe meta data is bundled in the `.px4` firmware file (which is a zipped JSON file).

Flash the resulting `.px4` file in *QGroundControl* (custom file option) to load the meta data into the application. The new airframe will then be available in the user interface once you restart *QGroundControl*.

5.4 Driver Development

NuttX device drivers are based on the [Device](#) framework.

Linux and QuRT drivers are based on [DriverFramework](#). PX4 is currently being updated so that they can use the same drivers as NuttX.

Currently (December 2017) a small number of Linux/QuRT I2C drivers have been migrated (primarily for airspeed sensors). We plan to migrate the remaining drivers in coming releases.

Creating a Driver

PX4 almost exclusively consumes data from [uORB](#). Drivers for common peripheral types must publish the correct uORB messages (for example: gyro, accelerometer, pressure sensors, etc.).

The best approach for creating a new driver is to start with a similar driver as a template (see [src/drivers](#)).

More detailed information about working with specific I/O busses and sensors may be available in [Sensor and Actuator Buses](#) section.

Publishing the correct uORB topics is the only pattern that drivers *must* follow.

Core Architecture

PX4 is a [reactive system](#) and uses [uORB](#) publish/subscribe to transport messages. File handles are not required or used for the core operation of the system. Two main APIs are used:

- The publish / subscribe system which has a file, network or shared memory backend depending on the system PX4 runs on.
- The global device registry, which can be used to enumerate devices and get/set their configuration. This can be as simple as a linked list or map to the file system.

Device IDs

PX4 uses device IDs to identify individual sensors consistently across the system. These IDs are stored in the configuration parameters and used to match sensor calibration values, as well as to determine which sensor is logged to which logfile entry.

The order of sensors (e.g. if there is a `/dev/mag0` and an alternate `/dev/mag1`) does not determine priority - the priority is instead stored as part of the published uORB topic.

Decoding example

For the example of three magnetometers on a system, use the flight log (.px4log) to dump the parameters. The three parameters encode the sensor IDs and `MAG_PRIME` identifies which magnetometer is selected as the primary sensor. Each `MAGx_ID` is a 24bit number and should be padded left with zeros for manual decoding.

```
CAL_MAG0_ID = 73225.0
CAL_MAG1_ID = 66826.0
CAL_MAG2_ID = 263178.0
CAL_MAG_PRIME = 73225.0
```

This is the external HMC5983 connected via I2C, bus 1 at address `0x1E`: It will show up in the log file as `IMU.MagX`.

```
# device ID 73225 in 24-bit binary:
00000001 00011110 00001 001
```

```
# decodes to:
```

```
HMC5883 0x1E bus 1 I2C
```

This is the internal HMC5983 connected via SPI, bus 1, slave select slot 5. It will show up in the log file as `IMU1.MagX`.

```
# device ID 66826 in 24-bit binary:
00000001 00000101 00001 010
```

```
# decodes to:
```

```
HMC5883 dev 5 bus 1 SPI
```

And this is the internal MPU9250 magnetometer connected via SPI, bus 1, slave select slot 4. It will show up in the log file as `IMU2.MagX`.

```
# device ID 263178 in 24-bit binary:
00000100 00000100 00001 010
```

```
#decodes to:
```

```
MPU9250 dev 4 bus 1 SPI
```

Device ID Encoding

The device ID is a 24bit number according to this format. Note that the first fields are the least significant bits in the decoding example above.

```
struct DeviceStructure {
    enum DeviceBusType bus_type : 3;
    uint8_t bus: 5; // which instance of the bus type
    uint8_t address; // address on the bus (eg. I2C address)
    uint8_t devtype; // device class specific device type
};
```

The `bus_type` is decoded according to:

```
enum DeviceBusType {
    DeviceBusType_UNKNOWN = 0,
    DeviceBusType_I2C     = 1,
    DeviceBusType_SPI     = 2,
    DeviceBusType_UAVCAN  = 3,
};
```

and `devtype` is decoded according to:

```
#define DRV_MAG_DEVTTYPE_HMC5883 0x01
#define DRV_MAG_DEVTTYPE_LSM303D 0x02
#define DRV_MAG_DEVTTYPE_ACCELSIM 0x03
#define DRV_MAG_DEVTTYPE_MPU9250 0x04
#define DRV_ACC_DEVTTYPE_LSM303D 0x11
#define DRV_ACC_DEVTTYPE_BMA180  0x12
#define DRV_ACC_DEVTTYPE_MPU6000 0x13
#define DRV_ACC_DEVTTYPE_ACCELSIM 0x14
#define DRV_ACC_DEVTTYPE_GYROSIM  0x15
#define DRV_ACC_DEVTTYPE_MPU9250  0x16
#define DRV_GYR_DEVTTYPE_MPU6000  0x21
#define DRV_GYR_DEVTTYPE_L3GD20   0x22
#define DRV_GYR_DEVTTYPE_GYROSIM  0x23
#define DRV_GYR_DEVTTYPE_MPU9250  0x24
#define DRV_RNG_DEVTTYPE_MB12XX    0x31
#define DRV_RNG_DEVTTYPE_LL40LS    0x32
```

5.5 Telemetry Radios/Modems

Telemetry Radios can (optionally) be used to provide a wireless MAVLink connection between a ground control station like *QGroundControl* and a vehicle running PX4. This section contains topics about advanced use of supported radios and integrating new telemetry systems into PX4.

Supported Radio Systems

[PX4 User Guide > Telemetry](#) contains information about telemetry radio systems already supported by PX4. This includes radios that use the *SiK Radio* firmware and *3DR WiFi Telemetry Radios*.

Integrating Telemetry Systems

PX4 enables MAVLink-based telemetry via the telemetry port of a Pixhawk-based flight controller. Provided that a telemetry radio supports MAVLink and provides a UART interface with compatible voltage levels/connector, no further integration is required.

Telemetry systems that communicate using some other protocol will need more extensive integration, potentially covering both software (e.g. device drivers) and hardware (connectors etc.). While this has been done for specific cases (e.g. [FrSky Telemetry](#) enables sending vehicle status to an RC controller via an FrSky receiver) providing general advice is difficult. We recommend you start by [discussing with the development team](#).

SiK Radio

[SiK radio](#) is a collection of firmware and tools for telemetry radios.

Information about *using* SiK Radio can be found in the *PX4 User Guide*: [Telemetry > SiK Radio](#)

The ("developer") information below explains how to build SiK firmware from source and configure it using AT commands.

Build Instructions

You will need to install the required 8051 compiler, as this is not included in the default PX4 Build toolchain.

Mac OS

Install the toolchain:

```
brew install sdcc
```

Build the image for the standard SiK Radio / 3DR Radio:

```
git clone https://github.com/LorenzMeier/SiK.git
```

```
cd SiK/Firmware
```

```
make install
```

Upload it to the radio (**change the serial port name**):

```
tools/uploader.py --port /dev/tty.usbserial-CHANGETHIS dst/radio~hm_trp.ihx
```

Configuration Instructions

The radio supports AT commands for configuration.

```
screen /dev/tty.usbserial-CHANGETHIS 57600 8N1
```

Then start command mode:

DO NOT TYPE ANYTHING ONE SECOND BEFORE AND AFTER

```
+++
```

List the current settings:

```
ATI5
```

Then set the net ID, write settings and reboot radio:

```
ATS3=55
```

```
AT&W
```

```
ATZ
```

You might have to power-cycle the radio to connect it to the 2nd radio.

5.6 Sensor and Actuator I/O

This section contains topics about integrating sensors and actuators into PX4. It covers both sensor busses ([I2C](#), [UAVCAN](#), [UART](#), [SPI](#)) and also the main PWM ports.

I2C Bus Overview

I2C is a packet-switched serial communication protocol that allows multiple master devices to connect to multiple slave devices using only 2 wires per connection. It is intended for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.

Pixhawk/PX4 support it for:

- Connecting off board components that require greater data rates than provided by a strict serial UART: e.g. rangefinders.
- Compatibility with peripheral devices that only support I2C.
- Allowing multiple devices to attach to a single bus (useful for conserving ports). For example, LEDs, Compass, rangefinders etc.

IMUs (accelerometers/gyroscopes) should not be attached via I2C (typically the [SPI](#) bus is used). The bus is not fast enough even with a single device attached to allow vibration filtering (for instance), and the performance degrades further with every additional device on the bus.

Integrating I2C Devices

Drivers should `#include <drivers/device/i2c.h>` and then provide an implementation of the abstract base class `I2C` defined in **I2C.hpp** for the target hardware (i.e. for NuttX [here](#)).

Drivers will also need to include headers for their type of device (`drv_*.h`) in [/src/drivers/](#) - e.g. [drv_baro.h](#).

To include a driver in firmware you must add the driver to the [cmake config file](#) that corresponds to the target you want to build for:

```
drivers/sf1xx
```

For example, you can see/search for this driver in the [px4fmu-v4 default](#) configuration.

I2C Driver Examples

To find I2C driver examples, search for `i2c.h` in [/src/drivers/](#).

Just a few examples are:

- [drivers/sf1xx](#) - I2C Driver for [Lightware SF1XX LIDAR](#).
- [drivers/ms5611](#) - I2C Driver for the MS5611 and MS6507 barometric pressure sensor connected via I2C (or SPI).

Further Information

- [I2C](#) (Wikipedia)
- [I2C Comparative Overview](#) (learn.sparkfun.com)
- [Driver Framework](#)

UAVCAN Introduction



[UAVCAN](#) is an onboard network which allows the autopilot to connect to avionics. It supports hardware like:

- Motor controllers
 - [Pixhawk ESC](#)

-
- [SV2740 ESC](#)
 - [Zubax Orel 20](#)
 - Airspeed sensors
 - [Thiemar airspeed sensor](#)
 - GNSS receivers for GPS and GLONASS
 - [Zubax GNSS](#)

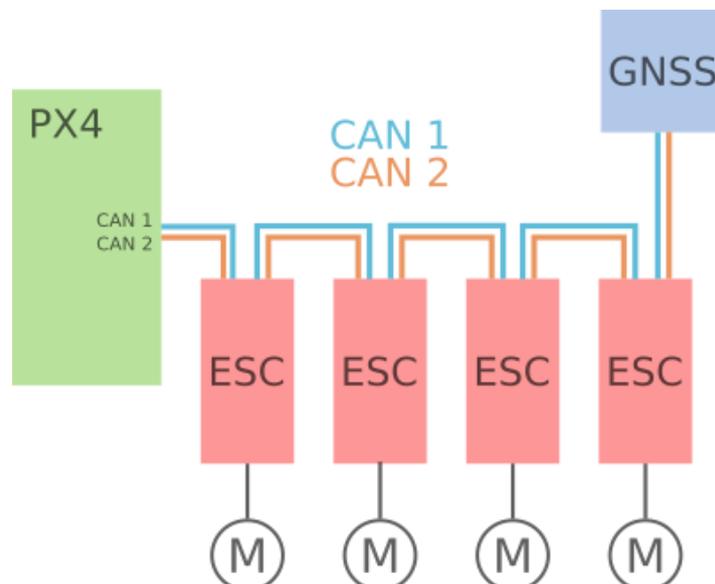
In contrast to hobby-grade devices it uses rugged, differential signalling and supports firmware upgrades over the bus. All motor controllers provide status feedback and implement field-oriented-control (FOC).

Initial Setup

The following instructions provide a step-by-step guide to connect and setup a quadcopter with ESCs and GPS connected via UAVCAN. The hardware of choice is a Pixhawk 2.1, Zubax Orel 20 ESCs and a Zubax GNSS GPS module.

Wiring

The first step is to connect all UAVCAN enabled devices with the flight controller. The following diagram displays how to wire all components. The used Zubax devices all support a redundant CAN interface in which the second bus is optional but increases the robustness of the connection.



It is important to mention that some devices require an external power supply (e.g. Zubax Orel 20) and others can be powered by the CAN connection (e.g. Zubax GNSS) itself. Please refer to the documentation of your hardware before continuing with the setup.

Firmware Setup

Next, follow the instructions in [UAVCAN Configuration](#) to activate the UAVCAN functionalities in the firmware. Disconnect your power supply and reconnect it. After the power cycle all UAVCAN devices should be detected which is confirmed by a beeping motor on the Orel 20 ESCs. You can now continue with the general setup and calibration.

Depending on the used hardware, it can be reasonable to perform an update of the firmware on the UAVCAN devices. This can be done via the UAVCAN itself and the PX4 firmware. For more details please refer to the instructions in [UAVCAN Firmware](#).

Upgrading Node Firmware

The PX4 middleware will automatically upgrade firmware on UAVCAN nodes if the matching firmware is supplied. The process and requirements are described on the [UAVCAN Firmware](#) page.

Enumerating and Configuring Motor Controllers

The ID and rotational direction of each motor controller can be assigned after installation in a simple setup routine: [UAVCAN Node Enumeration](#). The routine can be started by the user through QGroundControl.

Useful links

- [Homepage](#)
- [Specification](#)
- [Implementations and tutorials](#)

UAVCAN Bootloader Installation

UAVCAN devices typically ship with a bootloader pre-installed. Do not follow the instructions in this section unless you are developing UAVCAN devices.

Overview

The PX4 project includes a standard UAVCAN bootloader for STM32 devices.

The bootloader occupies the first 8–16 KB of flash, and is the first code executed on power-up. Typically, the bootloader performs low-level device initialization, automatically determines the CAN bus baud rate, acts as a UAVCAN dynamic node ID client to obtain a unique node ID, and waits for confirmation from the flight controller before proceeding with application boot.

This process ensures that a UAVCAN device can recover from invalid or corrupted application firmware without user intervention, and also permits automatic firmware updates.

Prerequisites

Installing or updating the UAVCAN bootloader requires:

- An SWD or JTAG interface (depending on device), for example the [BlackMagic Probe](#) or the [ST-Link v2](#);
- An adapter cable to connect your SWD or JTAG interface to the UAVCAN device's debugging port;
- A [supported ARM toolchain](#).

Device Preparation

If you are unable to connect to your device using the instructions below, it's possible that firmware already on the device has disabled the MCU's debug pins. To recover from this, you will need to connect your interface's NRST or nSRST pin (pin 15 on the standard ARM 20-pin connector) to your MCU's NRST pin. Obtain your device schematics and PCB layout or contact the manufacturer for details.

Installation

After compiling or obtaining a bootloader image for your device (refer to device documentation for details), the bootloader must be copied to the beginning of the device's flash memory.

The process for doing this depends on the SWD or JTAG interface used.

BlackMagic Probe

Ensure your BlackMagic Probe [firmware is up to date](#).

Connect the probe to your UAVCAN device, and connect the probe to your computer.

Identify the probe's device name. This will typically be `/dev/ttyACM<x>` or `/dev/ttyUSB<x>`. Power up your UAVCAN device, and run:

```
arm-none-eabi-gdb /path/to/your/bootloader/image.elf
```

At the `gdb` prompt, run:

```
target extended /dev/ttyACM0
monitor connect_srst enable
monitor swdp_scan
attach 1
set mem inaccessible-by-default off
load
run
```

If `monitor swdp_scan` returns an error, ensure your wiring is correct, and that you have an up-to-date version of the BlackMagic firmware.

ST-Link v2

Ensure you have a recent version—at least 0.9.0—of [OpenOCD](#).

Connect the ST-Link to your UAVCAN device, and connect the ST-Link to your computer.

Power up your UAVCAN device, and run:

```
openocd -f /path/to/your/openocd.cfg &
arm-none-eabi-gdb /path/to/your/bootloader/image.elf
```

At the `gdb` prompt, run:

```
target extended-remote localhost:3333
monitor reset halt
set mem inaccessible-by-default off
load
run
```

Segger J-Link Debugger

Connect the JLink Debugger to your UAVCAN device, and connect the JLink Debugger to your computer.

Power up your UAVCAN device, and run:

```
JLinkGDBServer -select USB=0 -device STM32F446RE -if SWD-DP -speed 20000 -vd
Open a second terminal, navigate to the directory that includes the px4esc_1_6-
bootloader.elf for the esc and run:
```

```
arm-none-eabi-gdb px4esc_1_6-bootloader.elf
```

At the `gdb` prompt, run:

```
tar ext :2331 load
```

Erasing Flash with SEGGER JLink Debugger

As a recovery method it may be useful to erase flash to factory defaults such that the firmware is using the default parameters. Go to the directory of your SEGGER installation and launch JLinkExe, then run:

```
device <name-of-device>
erase
```

Replace <name-of-device> with the name of the microcontroller, e.g. STM32F446RE for the Pixhawk ESC 1.6 or STM32F302K8 for the SV2470VC ESC.

UAVCAN Firmware Upgrading

Vectorcontrol ESC Codebase (Pixhawk ESC 1.6 and S2740VC)

Download the ESC code:

```
git clone https://github.com/thiemar/vectorcontrol
cd vectorcontrol
```

Flashing the UAVCAN Bootloader

Before updating firmware via UAVCAN, the Pixhawk ESC 1.6 requires the UAVCAN bootloader be flashed. To build the bootloader, run:

```
make clean && BOARD=px4esc_1_6 make -j8
```

After building, the bootloader image is located at `firmware/px4esc_1_6-bootloader.bin`, and the OpenOCD configuration is located at `openocd_px4esc_1_6.cfg`. Follow [these instructions](#) to install the bootloader on the ESC.

Compiling the Main Binary

```
BOARD=s2740vc_1_0 make && BOARD=px4esc_1_6 make
```

This will build the UAVCAN node firmware for both supported ESCs. The firmware images will be located at `com.thiemar.s2740vc-v1-1.0-1.0.<git hash>.bin` and `org.pixhawk.px4esc-v1-1.6-1.0.<git hash>.binn`.

Sapog Codebase (Pixhawk ESC 1.4 and Zubax Orel 20)

Download the Sapog codebase:

```
git clone https://github.com/PX4/sapog
cd sapog
git submodule update --init --recursive
```

Flashing the UAVCAN Bootloader

Before updating firmware via UAVCAN, the ESC requires the UAVCAN bootloader to be flashed. The bootloader can be built as follows:

```
cd bootloader
make clean && make -j8
cd ..
```

The bootloader image is located at `bootloader/firmware/bootloader.bin`, and the OpenOCD configuration is located at `openocd.cfg`. Follow [these instructions](#) to install the bootloader on the ESC.

Compiling the Main Binary

```
cd firmware
make RELEASE=1 # RELEASE is optional; omit to build the debug version
```

Beware, some newer version of GCC lead to segfaults during linking. Version 4.9 did work at the time of writing. The firmware image will be located at `firmware/build/io.px4.sapog-1.1-1.7.<xxxxxxxx>.application.bin`, where `<xxxxxxxx>` is an arbitrary sequence of numbers and letters. There are two hardware version of the Zubax Orel 20 (1.0 and 1.1). Make sure you copy the binary to the correct folder in the subsequent description. The ESC firmware will check the hardware version and works on both products.¹

Zubax GNSS

Please refer to the [project page](#) to learn how to build and flash the firmware. Zubax GNSS comes with a UAVCAN-capable bootloader, so its firmware can be updated in a uniform fashion via UAVCAN as described below.

Firmware Installation on the Autopilot

The UAVCAN node file names follow a naming convention which allows the Pixhawk to update all UAVCAN devices on the network, regardless of manufacturer. The firmware files generated in the steps above must therefore be copied to the correct locations on an SD card or the PX4 ROMFS in order for the devices to be updated.

The convention for firmware image names is:

```
<uavcan name>-<hw version major>.<hw version minor>-<sw version major>.<sw version minor>.<version hash>.bin
```

e.g. `com.thiemar.s2740vc-v1-1.0-1.0.68e34de6.bin`

However, due to space/performance constraints (names may not exceed 28 characters), the UAVCAN firmware updater requires those filenames to be split and stored in a directory structure like the following:

```
/fs/microsd/fw/<node name>/<hw version major>.<hw version minor>/<hw name>-<sw version major>.<sw version minor>.<git hash>.bin
```

e.g.

```
s2740vc-v1-1.0.68e34de6.bin
```

```
/fs/microsd/fw/io.px4.sapog/1.1/sapog-1.7.87c7bc0.bin
```

The ROMFS-based updater follows that pattern, but prepends the file name with `_` so you add the firmware in:

```
/etc/uavcan/fw/<device name>/<hw version major>.<hw version minor>/_<hw name>-<sw version major>.<sw version minor>.<git hash>.bin
```

Placing the binaries in the PX4 ROMFS

The resulting final file locations are:

- S2740VC ESC: `ROMFS/px4fmu_common/uavcan/fw/com.thiemar.s2740vc-v1/1.0/_s2740vc-v1-1.0.<git hash>.bin`
- Pixhawk ESC 1.6: `ROMFS/px4fmu_common/uavcan/fw/org.pixhawk.px4esc-v1/1.6/_px4esc-v1-1.6.<git hash>.bin`
- Pixhawk ESC 1.4: `ROMFS/px4fmu_common/uavcan/fw/org.pixhawk.sapog-v1/1.4/_sapog-v1-1.4..bin```
- Zubax GNSS v1: `ROMFS/px4fmu_common/uavcan/fw/com.zubax.gnss/1.0/gnss-1.0.<git hash>.bin`
- Zubax GNSS v2: `ROMFS/px4fmu_common/uavcan/fw/com.zubax.gnss/2.0/gnss-2.0.<git hash>.bin`

Note that the `ROMFS/px4fmu_common` directory will be mounted to `/etc` on Pixhawk.

Starting the Firmware Upgrade process

When using the PX4 Flight Stack, enable UAVCAN in the 'Power Config' section and reboot the system before attempting an UAVCAN firmware upgrade.

Alternatively UAVCAN firmware upgrading can be started manually on NSH via:

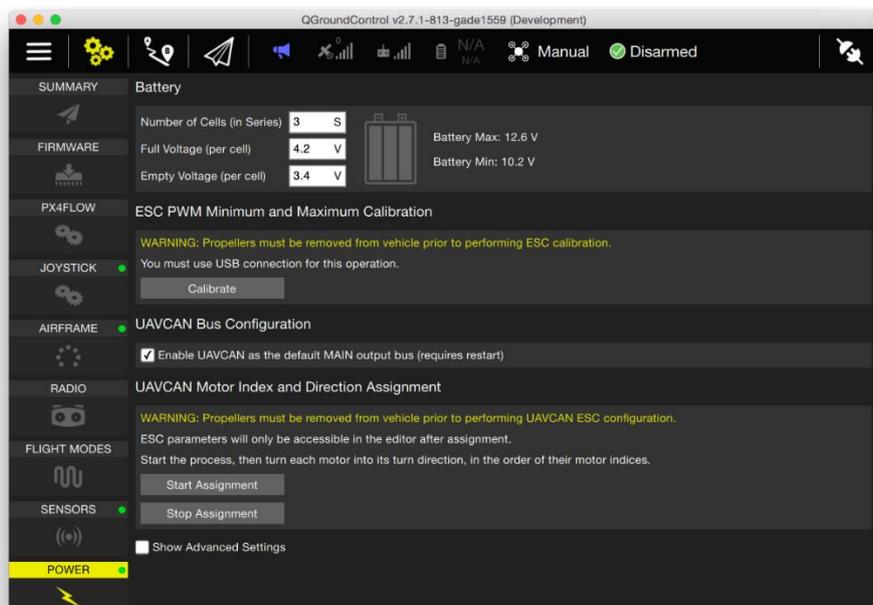
```
uavcan start
uavcan start fw
```

UAVCAN Enumeration and Configuration

Enable UAVCAN as the default motor output bus by ticking the 'Enable UAVCAN' checkbox as shown below. Alternatively the UAVCAN_ENABLE parameter can be set to '3' in the *QGroundControl* parameter editor. Set it to '2' to enable CAN, but leave motor outputs on PWM.

Use [QGroundControl](#) and switch to the Setup view. Select the Power Configuration on the left. Click on the 'start assignment' button.

After the first beep, turn the propeller on the first ESC swiftly into the correct turn direction. The ESCs will all beep each time one is enumerated. Repeat this step for all motor controllers in the order as shown on the [motor map](#). ESCs running the Sapog firmware will need to be rebooted after enumeration for the new enumeration ID to be applied. This step has to be performed only once and does not need to be repeated after firmware upgrades.



Various Notes

This is a collection of tips and tricks to solve issues when setting up or working with the UAVCAN.

Arm but motors not spinning

If the PX4 Firmware arms but the motors do not start to rotate, check the parameter **UAVCAN_ENABLE**. It should be set to 3 in order to use the ESCs connected via UAVCAN as output. Moreover, if the motors do not start spinning before thrust is increased, check **UAVCAN_ESC_IDLT** and set it to one.

Debugging with Zubax Babel

A great tool to debug the transmission on the UAVCAN bus is the [Zubax Babel](#) in combination with the [GUI tool](#). They can also be used independently from Pixhawk hardware in order to test a node or manually control UAVCAN enabled ESCs.

5.7 RTK GPS (Background)

[Real Time Kinematic](#) (RTK) provides centimeter-level GPS accuracy. This page explains how RTK is integrated into PX4.

Instructions for *using* RTK GPS are provided in the [PX4 User Guide](#).

Overview

RTK uses measurements of the phase of the signal's carrier wave, rather than the information content of the signal. It relies on a single reference station to provide real-time corrections, which can work with multiple mobile stations.

Two RTK GPS modules and a datalink are required to setup RTK with PX4. The fixed-position ground-based GPS unit is called the *Base* and the in-air unit is called the *Rover*. The Base unit connects to *QGroundControl* (via USB) and uses the datalink to stream RTCM corrections to the vehicle (using the MAVLink `GPS_RTCM_DATA` message). On the autopilot, the MAVLink packets are unpacked and sent to the Rover unit, where they are processed to get the RTK solution.

The datalink should typically be able to handle an uplink rate of 300 bytes per second (see the [Uplink Datarate](#) section below for more information).

Supported RTK GPS modules

PX4 currently only supports the single-frequency (L1) u-blox M8P based GNSS receivers for RTK.

A number of manufacturers have created products using this receiver. The list of devices that we have tested can be found [in the user guide](#).

u-blox has two variants of the M8P chip, the M8P-0 and the M8P-2. The M8P-0 can only be used as Rover, not as Base, whereas the M8P-2 can be used both as Rover or as Base.

Automatic Configuration

The PX4 GPS stack automatically sets up the u-blox M8P modules to send and receive the correct messages over the UART or USB, depending on where the module is connected (to *QGroundControl* or the autopilot).

As soon as the autopilot receives `GPS_RTCM_DATA` mavlink messages, it automatically forwards the RTCM data to the attached GPS module.

The U-Center RTK module configuration tool is not needed/used!

Both *QGroundControl* and the autopilot firmware share the same [PX4 GPS driver stack](#). In practice, this means that support for new protocols and/or messages only need to be added to one place.

RTCM messages

QGroundControl configures the RTK base station to output the following RTCM3.2 frames, each with 1 Hz:

- **1005** - Station coordinates XYZ for antenna reference point (Base position).
- **1077** - Full GPS pseudo-ranges, carrier phases, Doppler and signal strength (high resolution).
- **1087** - Full GLONASS pseudo-ranges, carrier phases, Doppler and signal strength (high resolution).

Uplink datarate

The raw RTCM messages from the base are packed into a MAVLink `GPS_RTCM_DATA` message and sent over the datalink. The maximum length of each MAVLink message is 182 bytes. Depending on the RTCM message, the MAVLink message is almost never completely filled.

The RTCM Base Position message (1005) is of length 22 bytes, while the others are all of variable length depending on the number of visible satellites and the number of signals from the satellite (only 1 for L1 units like M8P). Since at a given time, the *maximum* number of satellites visible from any single constellation is 12, under real-world conditions, theoretically an uplink rate of 300 B/s is sufficient.

If *MAVLink 1* is used, a 182-byte `GPS_RTCM_DATA` message is sent for every RTCM message, irrespective of its length. As a result the approximate uplink requirement is around 700+ bytes per second. This can lead to link saturation on low-bandwidth half-duplex telemetry modules (e.g. 3DR Telemetry Radios).

If *MAVLink 2* is used then any empty space in the `GPS_RTCM_DATA` message is removed. The resulting uplink requirement is about the same as the theoretical value (~300 bytes per second).

PX4 automatically switches to MAVLink 2 if the GCS and telemetry modules support it.

MAVLink 2 must be used on low-bandwidth links for good RTK performance. Care must be taken to make sure that the telemetry chain uses MAVLink 2 throughout. You can verify the protocol version by using the `mavlink status` command on the system console:

```
nsh> mavlink status
instance #0:
  GCS heartbeat: 593486 us ago
  mavlink chan: #0
  type:          3DR RADIO
  rssi:          219
  remote rssi:   219
  txbuf:         94
  noise:         61
  remote noise:  58
  rx errors:     0
  fixed:         0
  flow control:  ON
  rates:
  tx: 1.285 kB/s
  txerr: 0.000 kB/s
  rx: 0.021 kB/s
  rate mult: 0.366
  accepting commands: YES
```

```
MAVLink version: 2
transport protocol: serial (/dev/ttyS1 @57600)
```

5.8 Camera Trigger

The camera trigger driver allows the use of the AUX ports to send out pulses in order to trigger a camera. This can be used for multiple applications including timestamping photos for aerial surveying and reconstruction, synchronising a multi-camera system or visual-inertial navigation.

In addition to a pulse being sent out, a MAVLink message is published containing a sequence number (thus the current session's image sequence number) and the corresponding timestamp.

Trigger Modes/Configuration

Trigger Modes

Four different modes are supported, controlled by the [TRIG_MODE](#) parameter:

Value	Description
0	Camera triggering is disabled.
1	Camera triggered on time interval ("intervalometer"). Triggering can be enabled and disabled by using the MAV_CMD_DO_TRIGGER_CONTROL MAVLink command. See command interface for more details.
2	Camera triggered on time interval. Triggering always enabled.
3	Camera triggered based on distance. A shot is taken every time the set horizontal distance is exceeded. The minimum time interval between two shots is however limited by the set triggering interval.
4	Triggers automatically when flying a survey in Mission mode.

If it is your first time enabling the camera trigger app, remember to reboot after changing the `TRIG_MODE` parameter.

Trigger Hardware Configuration

You can choose which AUX pins to use for triggering using the [TRIG_PINS](#) parameter. The default is 56, which means that trigger is enabled on AUX 5 and AUX 6.

DO NOT CHANGE THE DEFAULT VALUE OF `TRIG_PINS` IF YOU NEED ACTUATOR OUTPUTS. With [TRIG_PINS](#) set to its **default** value of 56, you can use the AUX pins 1, 2, 3 and 4 as actuator outputs (for servos/ESCs). Due to the way the hardware timers are handled (1234 and 56 are 2 different groups handled by 2 timers), this is the **ONLY** combination which allows the simultaneous usage of camera trigger and FMU actuator outputs.

Trigger Interface Backends

The camera trigger driver supports several backends for triggering different types of camera interfaces. The type of camera can be specified using the [TRIG_INTERFACE](#) parameter:

Value	Interface	Description
1	<i>GPIO interface</i>	The AUX outputs are pulsed high or low (depending on the <code>TRIG_POLARITY</code> parameter) every <code>TRIG_INTERVAL</code> duration. This can be used to trigger most standard machine vision cameras directly. Note that on PX4FMU series hardware (Pixhawk, Pixracer, etc.), the signal level on the AUX pins is 3.3v.
2	<i>Seagull MAP2 interface</i>	This allows the use of the Seagull MAP2 to interface to a multitude of supported cameras. Pin 1 of the MAP2 should be connected to the lower AUX pin of <code>TRIG_PINS</code> (therefore, pin 1 to AUX 5 and pin 2 to AUX 6 by default). In this mode, PX4 also supports automatic power control and keep-alive functionalities of <i>Sony Multiport</i> cameras like the QX-1.
3	<i>MAVLink interface</i>	In this mode, no actual hardware output is used - the driver only sends the <code>CAMERA_TRIGGER</code> MAVLink message (by default, if the MAVLink application is in <code>onboard</code> mode. Otherwise, a custom stream will need to be enabled). PX4 must separately forward all MAVLink camera messages from a GCS to a connected MAVLink camera.
4	Generic PWM interface	Allows the use of infrared triggers or servos to trigger your camera.

Other Parameters

- [TRIG_POLARITY](#) - Relevant only while using the GPIO interface. Sets the polarity of the trigger pin. Active high means that the pin is pulled low normally and pulled high on a trigger event. Active low is vice-versa.
- [TRIG_INTERVAL](#) - Defines the time between two consecutive trigger events in milliseconds.

- [TRIG_ACTIVATION_TIME](#) - Defines the time in milliseconds the trigger pin is held in the "active" state before returning to neutral. In PWM modes, the minimum is limited to 40 ms to make sure we always fit an activate pulse into the 50Hz PWM signal.

The full list of parameters pertaining to the camera trigger module can be found on the [parameter reference](#) page.

Trigger Command Interface

The camera trigger driver supports the following commands.

MAV_CMD_DO_TRIGGER_CONTROL

[MAV_CMD_DO_TRIGGER_CONTROL](#) - Accepted in `TRIG_MODE 1` ("command controlled" mode).

Command Parameter	Description
Param #1	Trigger enable/disable (set to 0 for enable, 1 for start)
Param #2	Sequence reset (set to 1 to reset image sequence number, 0 to keep current sequence number)
Param #3	Trigger pause/restart (set to 0 for pause, 1 for restart). Pausing does not switch off/retract the camera.

MAV_CMD_DO_DIGICAM_CONTROL

[MAV_CMD_DO_DIGICAM_CONTROL](#) - Accepted in all modes. This is used by the GCS to test-shoot the camera from the user interface. The trigger driver does not yet support all camera control parameters defined by the MAVLink spec.

Command Parameter	Description
Param #5	Trigger one-shot command (set to 1 to trigger a single image frame)
Param #7	Trigger test shot - not logged or forwarded to GCS for geotagging (set to 1 to trigger a test shot)

MAV_CMD_DO_SET_CAM_TRIGG_DIST

[MAV_CMD_DO_SET_CAM_TRIGG_DIST](#) - Accepted in "mission controlled" mode (TRIG_MODE 4)

This command is auto-generated during missions to trigger the camera based on survey missions from the GCS.

Command Parameter	Description
Param #1	Trigger distance between shots (if >0). 0: triggering paused. -1: triggering disabled.
Param #2	Trigger activation time in ms - for GPIO mode only (TRIG_INTERFACE 1)
Param #2	Trigger single shot immediately (if >0)

MAV_CMD_DO_SET_CAM_TRIGG_INTERVAL

[MAV_CMD_DO_SET_CAM_TRIGG_INTERVAL](#) - Accepted in "mission controlled" mode (TRIG_MODE 4)

This command is auto-generated during missions to trigger the camera based on survey missions from the GCS.

Command Parameter	Description
Param #1	Trigger interval time.
Param #2	Shutter activation time in ms - for GPIO mode only (TRIG_INTERFACE 1)
Param #2	Trigger single shot immediately (if >0)



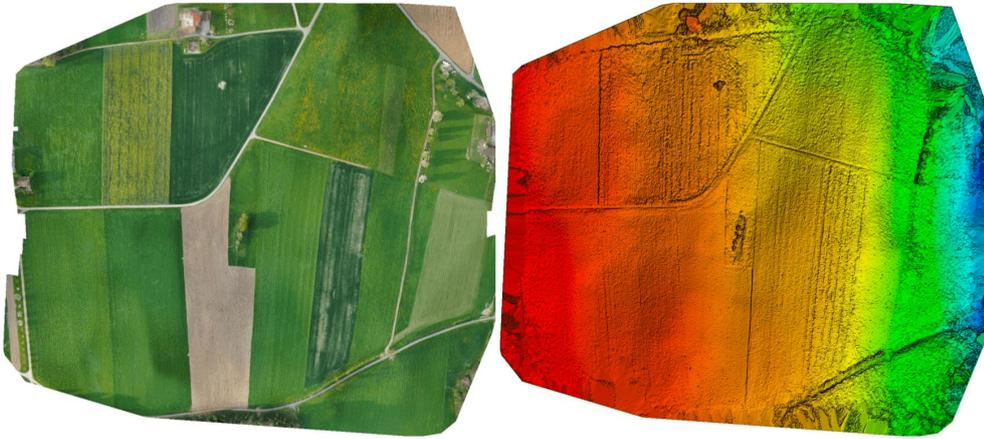
Testing Trigger Functionality

- On the PX4 console:
- `camera_trigger test`
- From *QGroundControl*:

Click on **Trigger Camera** in the main instrument panel. These shots are not logged or counted for geotagging.

Examples

Sony QX-1 example (Photogrammetry)



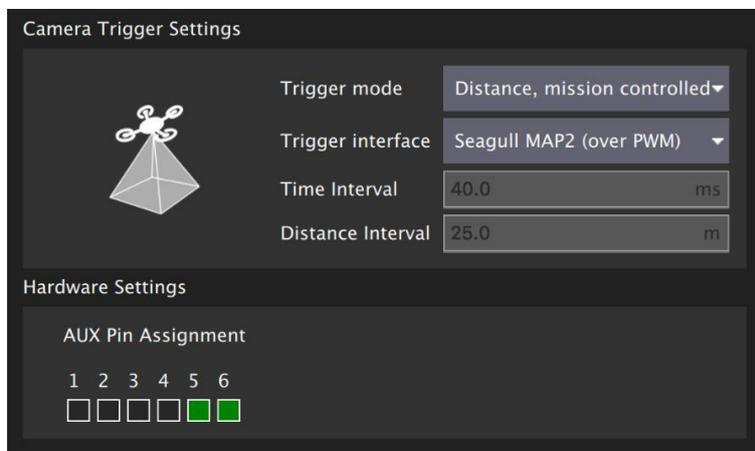
In this example, we will use a Seagull MAP2 trigger cable to interface to a Sony QX-1 and use the setup to create orthomosaics after flying a fully autonomous survey mission.

Trigger Settings

The camera trigger can be configured from the *QGroundControl* **Settings tab > Camera** page

- TRIG_INTERFACE: 2, Seagull MAP2.
- TRIG_MODE: 4, Mission controlled.

Leave the rest of the parameters at their defaults.



You will need to connect the Seagull MAP2 to the auxiliary/FMU pins on your autopilot. Pin 1 goes to AUX 5, and Pin 2 to AUX 6. The other end of the MAP2 cable will go into the QX-1's "MULTI" port.

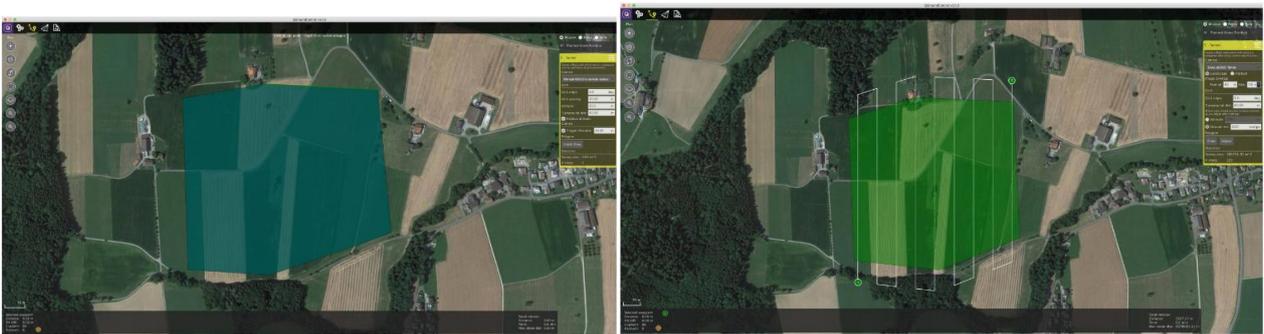
Camera Configuration

We use a Sony QX-1 with a 16-50mm f3.5-5.6 lens for this example.

To avoid autofocus and metering lag when the camera is triggered, the following guidelines should be followed :

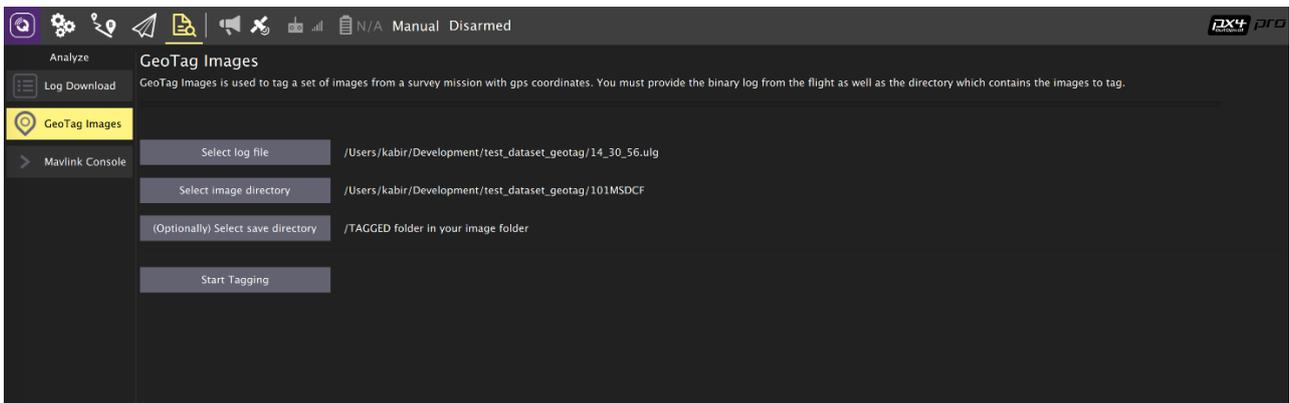
- Manual focus to infinity
- Set camera to continuous shooting mode
- Manually set exposure and aperture
- ISO should be set as low as possible
- Manual white balance suitable for scene

Mission Planning



Geotagging

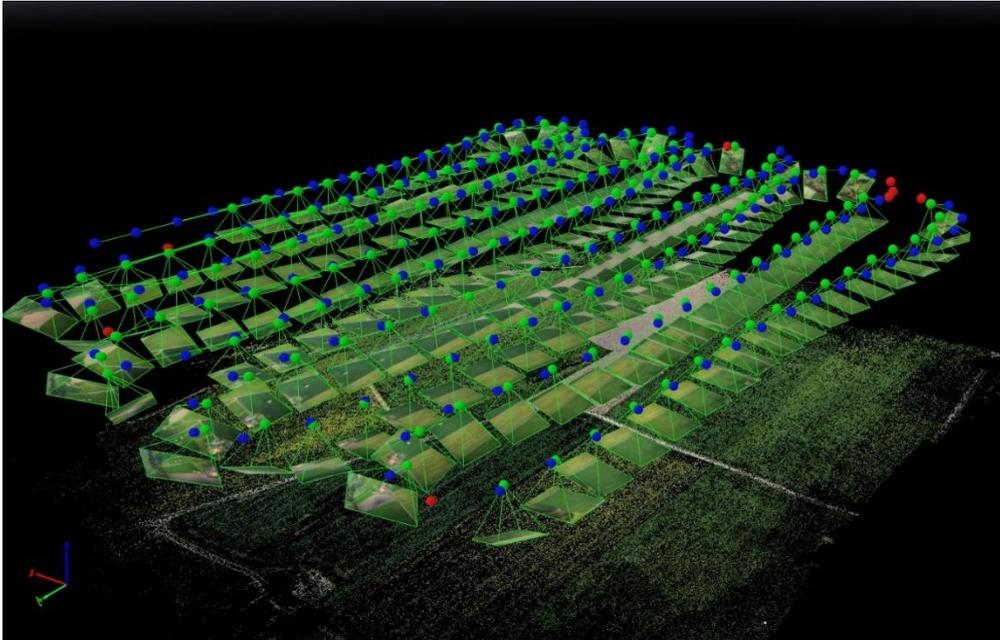
Download/copy the logfile and images from the flight and point *QGroundControl* to them. Then click **Start Tagging**.



You can verify the geotagging using a free online service like [Pic2Map](#). Note that *Pic2Map* is limited to only 40 images.

Reconstruction

We use [Pix4D](#) for 3D reconstruction.



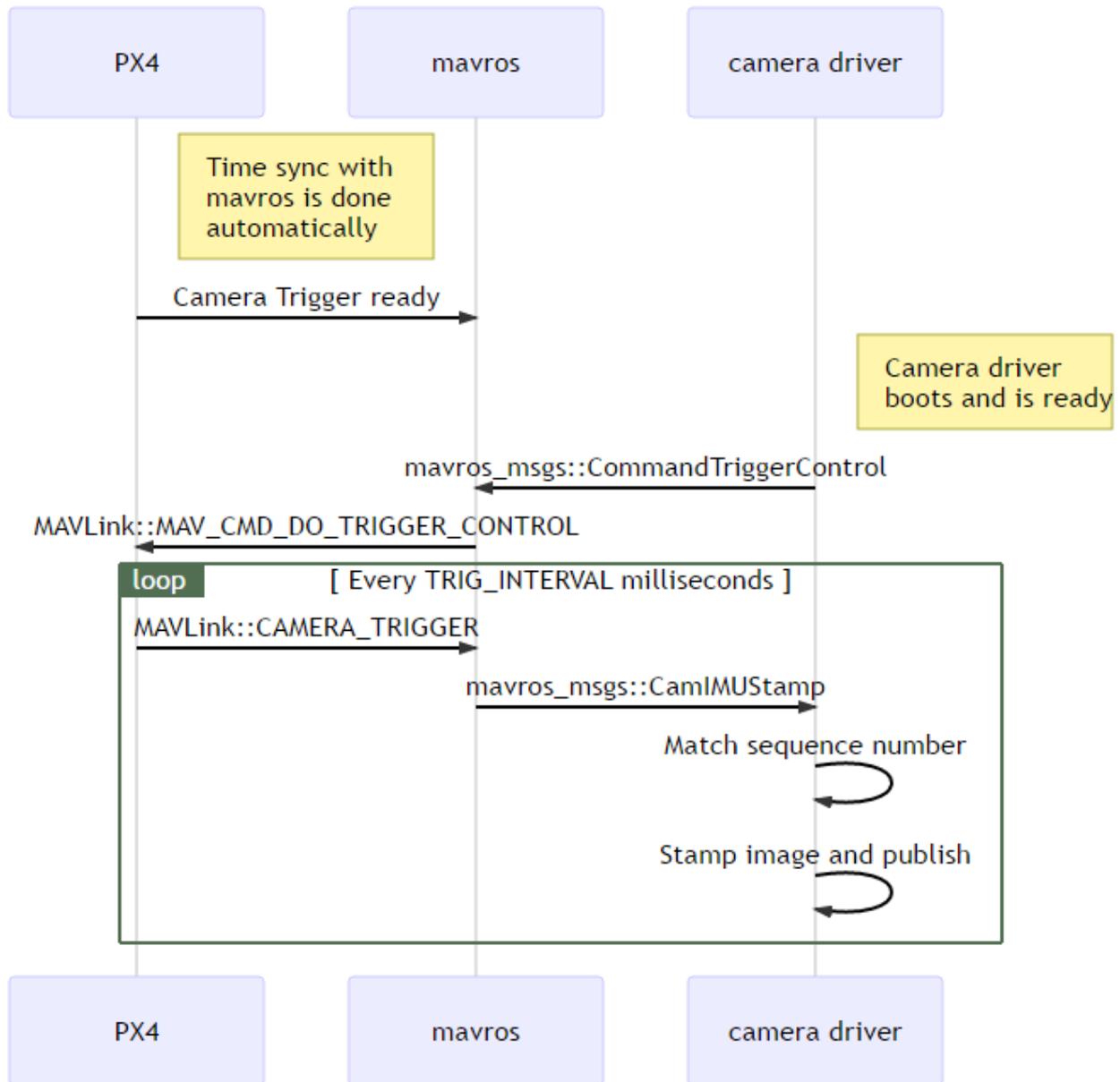
Camera-IMU Sync Example (VIO)

In this example, we will go over the basics of synchronising IMU measurements with visual data to build a stereo Visual-Inertial Navigation System (VINS). To be clear, the idea here isn't to take an IMU measurement exactly at the same time as we take a picture but rather to correctly time stamp our images so as to provide accurate data to our VIO algorithm.

The autopilot and companion have different clock bases (boot-time for the autopilot and UNIX epoch for companion), so instead of skewing either clock, we directly observe the time offset between the clocks. This offset is added or subtracted from the timestamps in the MAVLink messages (e.g [HIGHRES_IMU](#)) in the cross-middleware translator component (e.g Mavros on the companion and `mavlink_receiver` in PX4). The actual synchronisation algorithm is a modified version of the Network Time Protocol (NTP) algorithm and uses an exponential moving average to smooth the tracked time offset. This synchronisation is done automatically if Mavros is used with a high-bandwidth onboard link (MAVLink mode `onboard`).

For acquiring synchronised image frames and inertial measurements, we connect the trigger inputs of the two cameras to a GPIO pin on the autopilot. The timestamp of the inertial measurement from start of exposure and a image sequence number is recorded

and sent to the companion computer ([CAMERA_TRIGGER](#) message), which buffers these packets and the image frames acquired from the camera. They are matched based on the sequence number (first image frame is sequence 0), the images timestamped (with the timestamp from the `CAMERA_TRIGGER` message) and then published. The following diagram illustrates the sequence of events which must happen in order to correctly timestamp our images.



Step 1

First, set the `TRIG_MODE` to 1 to make the driver wait for the start command and reboot your FCU to obtain the remaining parameters.

Step 2

For the purposes of this example we will be configuring the trigger to operate in conjunction with a Point Grey Firefly MV camera running at 30 FPS.

- `TRIG_INTERVAL`: 33.33 ms
- `TRIG_POLARITY`: 0 (active low)
- `TRIG_ACT_TIME`: 0.5 ms. The manual specifies it only has to be a minimum of 1 microsecond.
- `TRIG_MODE`: 1, because we want our camera driver to be ready to receive images before starting to trigger. This is essential to properly process sequence numbers.
- `TRIG_PINS`: 56, Leave default.

Step 3

Wire up your cameras to your AUX port by connecting the ground and signal pins to the appropriate place.

Step 4

You will have to modify your driver to follow the sequence diagram above. Public reference implementations for [IDS Imaging UEye](#) cameras and for [IEEE1394 compliant](#) cameras are available.

5.9 Gimbal Control Setup

If you want to control a gimbal with a camera (or any other payload) attached to the vehicle, you need to configure how you want to control it and how PX4 can command it. This page explains the setup.

PX4 contains a generic mount/gimbal control driver with different input and output methods. The input defines how you control the gimbal: via RC or via MAVLink commands (for example in missions or surveys). The output defines how the gimbal is connected: some support MAVLink commands, others use PWM (described as AUX output in the following). Any input method can be selected to drive any output. Both have to be configured via parameters.

Parameters

[These parameters](#) are used to setup the mount driver. The most important ones are the input (`MNT_MODE_IN`) and the output (`MNT_MODE_OUT`) mode. By default, the input is disabled

and the driver does not run. After selecting the input mode, reboot the vehicle so that the mount driver starts.

If the input mode is set to `AUTO`, the mode will automatically be switched based on the latest input. To switch from mavlink to RC, a large stick motion is required.

AUX output

If the output mode is set to `AUX`, a mixer file is required to define the mapping for the output pins and the [mount mixer](#) is automatically selected (overriding any aux mixer provided by the airframe configuration).

The output assignment is as following:

- **AUX1:** Pitch
- **AUX2:** Roll
- **AUX3:** Yaw
- **AUX4:** Shutter/retract

Customizing the mixer configuration

Read [Mixing and Actuators](#) for an explanation of how mixers work and the format of the mixer file.

The outputs can be customized by [creating a mixer file](#) on the SD card with name `etc/mixers/mount.aux.mix`.

A basic basic mixer configuration for a mount is shown below.

```
# roll
M: 1
O:      10000  10000      0 -10000  10000
S: 2 0  10000  10000      0 -10000  10000

# pitch
M: 1
O:      10000  10000      0 -10000  10000
S: 2 1  10000  10000      0 -10000  10000

# yaw
M: 1
O:      10000  10000      0 -10000  10000
S: 2 2  10000  10000      0 -10000  10000
```

SITL

The Typhoon H480 model comes with a preconfigured simulated gimbal. To run it, use:

```
make posix gazebo_typhoon_h480
```

To just test the mount driver on other models or simulators, make sure the driver runs, using `vmount start`, then configure its parameters.

Testing

The driver provides a simple test command - it needs to be stopped first with `vmount stop`. The following describes testing in SITL, but the commands also work on a real device.

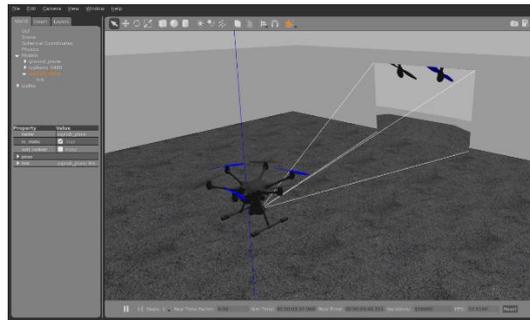
Start the simulation with (no parameter needs to be changed for that):

```
make posix gazebo_typhoon_h480
```

Make sure it's armed, eg. with `commander takeoff`, then use for example

```
vmount test yaw 30
```

to control the gimbal. Note that the simulated gimbal stabilizes itself, so if you send mavlink commands, set the `stabilize` flags to false.



5.10 Companion Computer for Pixhawk class

Interfacing a companion computer (Raspberry Pi, Odroid, Tegra K1) to Pixhawk-family boards always works the same way: They are interfaced using a serial port to `TELEM2`, the port intended for this purpose. The message format on this link is [MAVLink](#).

Pixhawk setup

Set the `SYS_COMPANION` parameter (in the System group) to one of these values.

Changing this parameter requires an autopilot reboot to become active.

- 0 to disable MAVLink output on TELEM2 (default)
- 921600 to enable MAVLink output at 921600 baud, 8N1 (recommended)
- 57600 to enable MAVLink output at 57600 baud, 8N1
- 157600 to enable MAVLink in *OSD* mode at 57600 baud
- 257600 to enable MAVLink in listen-only mode at 57600 baud

Companion computer setup

In order to receive MAVLink, the companion computer needs to run some software talking to the serial port. The most common options are:

- [MAVROS](#) to communicate to ROS nodes
- [C/C++ example code](#) to connect custom code
- [MAVProxy](#) to route MAVLink between serial and UDP

Hardware setup

Wire the serial port according to the instructions below. All Pixhawk serial ports operate at 3.3V and are 5V level compatible.

Many modern companion computers only support 1.8V levels on their hardware UART and can be damaged by 3.3V levels. Use a level shifter. In most cases the accessible hardware serial ports already have some function (modem or console) associated with them and need to be *reconfigured in Linux* before they can be used.

The safe bet is to use an FTDI Chip USB-to-serial adapter board and the wiring below. This always works and is easy to set up.

	TELEM2		FTDI	
	1	+5V (red)		DO NOT CONNECT!
	2	Tx (out)	5	FTDI RX (yellow) (in)
	3	Rx (in)	4	FTDI TX (orange) (out)
	4	CTS (in)	6	FTDI RTS (green) (out)
	5	RTS (out)	2	FTDI CTS (brown) (in)
	6	GND	1	FTDI GND (black)

Software setup on Linux

On Linux the default name of a USB FTDI would be like `\dev\ttyUSB0`. If you have a second FTDI linked on the USB or an Arduino, it will registered as `\dev\ttyUSB1`. To

avoid the confusion between the first plugged and the second plugged, we recommend you to create a symlink from `ttyUSBx` to a friendly name, depending on the Vendor and Product ID of the USB device.

Using `lsusb` we can get the vendor and product IDs.

```
$ lsusb
```

```
Bus 006 Device 002: ID 0bda:8153 Realtek Semiconductor Corp.
Bus 006 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 005 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 002: ID 05e3:0616 Genesys Logic, Inc.
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 004: ID 2341:0042 Arduino SA Mega 2560 R3 (CDC ACM)
Bus 003 Device 005: ID 26ac:0011
Bus 003 Device 002: ID 05e3:0610 Genesys Logic, Inc. 4-port hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 002: ID 0bda:8176 Realtek Semiconductor Corp. RTL8188CUS 802.11n WLAN Adapter
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

The Arduino is `Bus 003 Device 004: ID 2341:0042 Arduino SA Mega 2560 R3 (CDC ACM)`

The Pixhawk is `Bus 003 Device 005: ID 26ac:0011`

If you do not find your device, unplug it, execute `lsusb`, plug it, execute `lsusb` again and see the added device.

Therefore, we can create a new UDEV rule in a file called `/etc/udev/rules.d/99-pixhawk.rules` with the following content, changing the `idVendor` and `idProduct` to yours.

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="2341", ATTRS{idProduct}=="0042",
SYMLINK+="ttyArduino"
SUBSYSTEM=="tty", ATTRS{idVendor}=="26ac", ATTRS{idProduct}=="0011",
SYMLINK+="ttyPixhawk"
```

Finally, after a **reboot** you can be sure to know which device is what and put `/dev/ttyPixhawk` instead of `/dev/ttyUSB0` in your scripts.

Be sure to add yourself in the `tty` and `dialout` groups via `usermod` to avoid to have to execute scripts as root.

```
usermod -a -G tty ros-user
usermod -a -G dialout ros-user
```

6. Middleware

This section contains topics about PX4 middleware, including PX4 internal communication mechanisms ([uORB](#)), and between PX4 and offboard systems like companion computers and GCS (e.g. [MAVLink](#), [RTPS](#)).

For a detailed overview of the platform architecture see the [Architectural Overview](#).

6.1 uORB Messaging

Introduction

The uORB is an asynchronous `publish()` / `subscribe()` messaging API used for inter-thread/inter-process communication.

Look at the [tutorial](#) to learn how to use it in C++.

uORB is automatically started early on bootup as many applications depend on it. It is started with `uorb start`. Unit tests can be started with `uorb_tests`.

Adding a new topic

To add a new topic, you need to create a new `.msg` file in the `msg/` directory and add the file name to the `msg/CMakeLists.txt` list. From this, the needed C/C++ code is automatically generated.

Have a look at the existing `msg` files for supported types. A message can also be used nested in other messages.

To each generated C/C++ struct, a field `uint64_t timestamp` will be added. This is used for the logger, so make sure to fill it in when publishing the message.

To use the topic in the code, include the header:

```
#include <uORB/topics/topic_name.h>
```

By adding a line like the following in the `.msg` file, a single message definition can be used for multiple independent topic instances:

```
# TOPICS mission offboard_mission onboard_mission
```

Then in the code, use them as topic id: `ORB_ID(offboard_mission)`.

Publishing

Publishing a topic can be done from anywhere in the system, including interrupt context (functions called by the `hrt_call` API). However, advertising a topic is only possible outside of interrupt context. A topic has to be advertised in the same process as it's later published.

Listing Topics and Listening in

The `listener` command is only available on Pixracer (FMUv4) and Linux / OS X.

To list all topics, list the file handles:

```
ls /obj
```

To listen to the content of one topic for 5 messages, run the listener:

```
listener sensor_accel 5
```

The output is n-times the content of the topic:

```
TOPIC: sensor_accel #3
timestamp: 84978861
integral_dt: 4044
error_count: 0
x: -1
y: 2
z: 100
x_integral: -0
y_integral: 0
z_integral: 0
temperature: 46
range_m_s2: 78
scaling: 0
```

```
TOPIC: sensor_accel #4
timestamp: 85010833
integral_dt: 3980
error_count: 0
x: -1
y: 2
z: 100
x_integral: -0
y_integral: 0
z_integral: 0
temperature: 46
range_m_s2: 78
scaling: 0
```

On NuttX-based systems (Pixhawk, Pixracer, etc) the `listener` command can be called from within the *QGroundControl* MAVLink Console to inspect the values of sensors and other topics. This is a powerful debugging tool because it can be used even when QGC is connected over a wireless link (e.g. when the vehicle is flying). For more information see: [Sensor/Topic Debugging](#).

urb top Command

The command `uorb top` shows the publishing frequency of each topic in real-time:

update: 1s, num topics: 77

TOPIC NAME	INST	#SUB	#MSG	#LOST	#QSIZE
actuator_armed	0	6	4	0	1
actuator_controls_0	0	7	242	1044	1
battery_status	0	6	500	2694	1
commander_state	0	1	98	89	1
control_state	0	4	242	433	1
ekf2_innovations	0	1	242	223	1
ekf2_timestamps	0	1	242	23	1
estimator_status	0	3	242	488	1
mc_att_ctrl_status	0	0	242	0	1
sensor_accel	0	1	242	0	1
sensor_accel	1	1	249	43	1
sensor_baro	0	1	42	0	1
sensor_combined	0	6	242	636	1

The columns are: topic name, multi-instance index, number of subscribers, publishing frequency in Hz, number of lost messages (all subscribers combined), and queue size.

Multi-instance

uORB provides a mechanism to publish multiple independent instances of the same topic through `orb_advertise_multi`. It will return an instance index to the publisher. A subscriber will then have to choose to which instance to subscribe to using `orb_subscribe_multi` (`orb_subscribe` subscribes to the first instance). Having multiple instances is useful for example if the system has several sensors of the same type.

Make sure not to mix `orb_advertise_multi` and `orb_advertise` for the same topic.

The full API is documented in src/modules/uORB/uORBManager.hpp.

Troubleshooting and common Pitfalls

The following explains some common pitfalls and corner cases:

- The topic is not published: make sure the `ORB_ID()`'s of each call match. It is also important that `orb_subscribe` and `orb_unsubscribe` are **called from the same task** as `orb_check` and `orb_copy`. This applies to `px4_task_spawn_cmd()`, but also when using work queues (`work_queue()`).
- Make sure to clean up: use `orb_unsubscribe` and `orb_unadvertise`.

-
- A successful `orb_check()` or `px4_poll()` call requires an `orb_copy()`, otherwise the next poll will return immediately.
 - It is perfectly ok to call `orb_subscribe` before anyone advertised the topic.
 - `orb_check()` and `px4_poll()` will only return true for publications that are done after `orb_subscribe()`. This is important for topics that are not published regularly. If a subscriber needs the previous data, it should just do an unconditional `orb_copy()` right after `orb_subscribe()` (note that `orb_copy()` will fail if there is no advertiser yet).

6.2 uORB Publication/Subscription Graph

This page provides a uORB publication/subscription graph that shows the communication between modules. It is based on information that is extracted directly from the source code. Usage instructions are provided [below](#).

Search: Preset:

Graph Properties

The graph has the following properties:

- Modules are shown in gray with rounded corners while topics are displayed as coloured rectangular boxes.
- Associated modules and topics are connected by lines. Dashed lines indicate that the module publishes the topic, while solid lines indicate that the module subscribes to the topic.
- Some modules and topics are excluded:
 - Topics that are subscribed/published by many modules: `parameter_update`, `mavlink_log` and `log_message`.
 - The set of logged topics.
 - Topics that have no subscriber or no publisher.
 - Modules in **src/examples**.
- Hovering over a module/topic highlights all its connections.
- Double-clicking on a topic opens its message definition.
- Make sure your browser window is wide enough to display the full graph (the sidebar menu can be hidden with the icon in the top-left corner). You can also zoom the image.
- The *Preset* selection list allows you to refine the list of modules that are shown.
- The *Search* box can be used to find particular modules/topics (topics that are not selected by the search are greyed-out).

6.3 MAVLink Messaging

An overview of all messages can be found [here](#).

Create Custom MAVLink Messages

This tutorial assumes you have a [custom uORB](#) `ca_trajectory` message in `msg/ca_trajectory.msg` and a custom MAVLink `ca_trajectory` message in `mavlink/include/mavlink/v1.0/custom_messages/mavlink_msg_ca_trajectory.h` (see [here](#) how to create a custom MAVLink message and header).

Sending Custom MAVLink Messages

This section explains how to use a custom uORB message and send it as a MAVLink message.

Add the headers of the MAVLink and uORB messages to [mavlink_messages.cpp](#)

```
#include <uORB/topics/ca_trajectory.h>
#include <v1.0/custom_messages/mavlink_msg_ca_trajectory.h>
```

Create a new class in [mavlink_messages.cpp](#)

```
class MavlinkStreamCaTrajectory : public MavlinkStream
{
public:
    const char *get_name() const
    {
        return MavlinkStreamCaTrajectory::get_name_static();
    }
    static const char *get_name_static()
    {
        return "CA_TRAJECTORY";
    }
    uint8_t get_id()
    {
        return MAVLINK_MSG_ID_CA_TRAJECTORY;
    }
    static MavlinkStream *new_instance(Mavlink *mavlink)
    {
        return new MavlinkStreamCaTrajectory(mavlink);
    }
    unsigned get_size()
    {
```

```

        return MAVLINK_MSG_ID_CA_TRAJECTORY_LEN + MAVLINK_NUM_NON_PAYLOAD_BYTES;
    }

private:
    MavlinkOrbSubscription *_sub;
    uint64_t _ca_traj_time;

    /* do not allow top copying this class */
    MavlinkStreamCaTrajectory(MavlinkStreamCaTrajectory &);
    MavlinkStreamCaTrajectory& operator = (const MavlinkStreamCaTrajectory &);

protected:
    explicit MavlinkStreamCaTrajectory(Mavlink *mavlink) : MavlinkStream(mavlink),
        _sub(_mavlink->add_orb_subscription(ORB_ID(ca_trajectory))), // make sure
you enter the name of your uORB topic here
        _ca_traj_time(0)
    {}

    void send(const hrt_abstime t)
    {
        struct ca_traj_struct_s _ca_trajectory; //make sure ca_traj_struct_s is the
definition of your uORB topic

        if (_sub->update(&_ca_traj_time, &_ca_trajectory)) {
            mavlink_ca_trajectory_t _msg_ca_trajectory; //make sure
mavlink_ca_trajectory_t is the definition of your custom MAVLink message

            _msg_ca_trajectory.timestamp = _ca_trajectory.timestamp;
            _msg_ca_trajectory.time_start_usec = _ca_trajectory.time_start_usec;
            _msg_ca_trajectory.time_stop_usec = _ca_trajectory.time_stop_usec;
            _msg_ca_trajectory.coefficients = _ca_trajectory.coefficients;
            _msg_ca_trajectory.seq_id = _ca_trajectory.seq_id;

            _mavlink->send_message(MAVLINK_MSG_ID_CA_TRAJECTORY, &_msg_ca_trajectory);
        }
    }
};

```

Finally append the stream class to the `streams_list` at the bottom of `mavlink_messages.cpp`

```

StreamListItem *streams_list[] = {
    ...
    new StreamListItem(&MavlinkStreamCaTrajectory::new_instance,
&MavlinkStreamCaTrajectory::get_name_static),
    nullptr

```

```
};
```

Then make sure to enable the stream, for example by adding the following line to the startup script (-r configures the streaming rate, -u identifies the MAVLink channel on UDP port 14556):

```
mavlink stream -r 50 -s CA_TRAJECTORY -u 14556
```

Receiving Custom MAVLink Messages

This section explains how to receive a message over MAVLink and publish it to uORB.

Add a function that handles the incoming MAVLink message in [mavlink_receiver.h](#)

```
#include <uORB/topics/ca_trajectory.h>
#include <v1.0/custom_messages/mavlink_msg_ca_trajectory.h>
```

Add a function that handles the incoming MAVLink message in the `MavlinkReceiver` class in [mavlink_receiver.h](#)

```
void handle_message_ca_trajectory_msg(mavlink_message_t *msg);
```

Add an uORB publisher in the `MavlinkReceiver` class in [mavlink_receiver.h](#)

```
orb_advert_t _ca_traj_msg_pub;
```

Implement the `handle_message_ca_trajectory_msg` function in [mavlink_receiver.cpp](#)

```
void MavlinkReceiver::handle_message_ca_trajectory_msg(mavlink_message_t *msg)
```

```
{
    mavlink_ca_trajectory_t traj;
    mavlink_msg_ca_trajectory_decode(msg, &traj);

    struct ca_traj_struct_s f;
    memset(&f, 0, sizeof(f));

    f.timestamp = hrt_absolute_time();
    f.seq_id = traj.seq_id;
    f.time_start_usec = traj.time_start_usec;
    f.time_stop_usec = traj.time_stop_usec;
    for(int i=0;i<28;i++)
        f.coefficients[i] = traj.coefficients[i];

    if (_ca_traj_msg_pub == nullptr) {
        _ca_traj_msg_pub = orb_advertise(ORB_ID(ca_trajectory), &f);
    } else {
        orb_publish(ORB_ID(ca_trajectory), _ca_traj_msg_pub, &f);
    }
}
```

and finally make sure it is called in [MavlinkReceiver::handle_message\(\)](#)

```
MavlinkReceiver::handle_message(mavlink_message_t *msg)
```

```
{
  switch (msg->msgid) {
    ...
    case MAVLINK_MSG_ID_CA_TRAJECTORY:
      handle_message_ca_trajectory_msg(msg);
      break;
    ...
  }
}
```

Alternative to Creating Custom MAVLink Messages

Sometimes there is the need for a custom MAVLink message with content that is not fully defined.

For example when using MAVLink to interface PX4 with an embedded device, the messages that are exchanged between the autopilot and the device may go through several iterations before they are stabilized. In this case, it can be time-consuming and error-prone to regenerate the MAVLink headers, and make sure both devices use the same version of the protocol.

An alternative - and temporary - solution is to re-purpose debug messages. Instead of creating a custom MAVLink message `CA_TRAJECTORY`, you can send a message `DEBUG_VECT` with the string key `CA_TRAJ` and data in the `x`, `y` and `z` fields. See [this tutorial](#). for an example usage of debug messages.

This solution is not efficient as it sends character string over the network and involves comparison of strings. It should be used for development only!

General

Set streaming rate

Sometimes it is useful to increase the streaming rate of individual topics (e.g. for inspection in QGC). This can be achieved by typing the following line in the shell:

```
mavlink stream -u <port number> -s <mavlink topic name> -r <rate>
```

You can get the port number with `mavlink status` which will output (amongst others) `transport protocol: UDP (<port number>)`. An example would be

```
mavlink stream -u 14556 -s OPTICAL_FLOW_RAD -r 300
```

6.4 RTPS/ROS2 Interface: PX4-FastRTPS Bridge

The *PX4-FastRTPS Bridge* adds a Real Time Publish Subscribe (RTPS) interface to PX4, enabling the exchange of [uORB messages](#) between PX4 components and (offboard) *FastRTPS* applications.

RTPS is the underlying protocol of the Object Management Group's (OMG) Data Distribution Service (DDS) standard. It aims to enable scalable, real-time, dependable, high-performance and interoperable data communication using the publish/subscribe pattern. *FastRTPS* is a very lightweight cross-platform implementation of the latest version of the RTPS protocol and a minimum DDS API.

RTPS has been adopted as the middleware for the ROS2 (Robot Operating System). The bridge allows us to better integrate with ROS2, making it easy to share sensor values, commands, and other vehicle information.

This topic describes the bridge architecture, how it is compiled, and how to write a simple FastRTSP application to subscribe to PX4 changes.

When should RTPS be used?

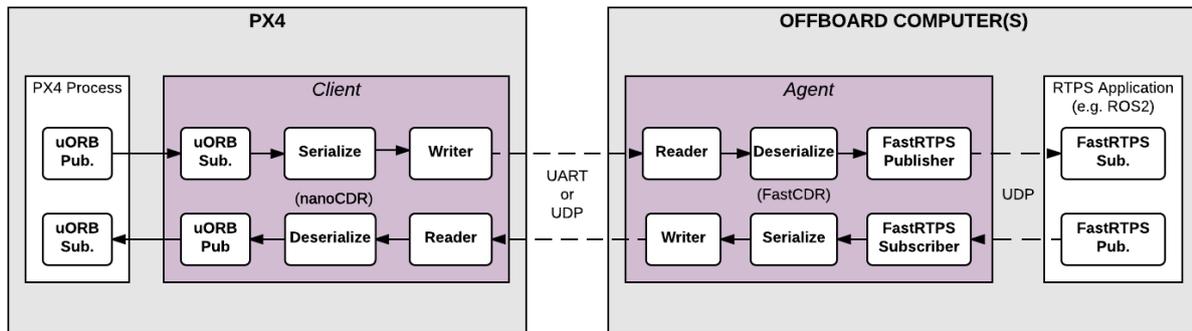
RTPS should be used in circumstances where there is a need to reliably share time-critical/real-time information between the flight controller and off board components. In particular it is useful in cases where off-board software needs to become a peer software component running in PX4 (by sending and receiving uORB topics).

Possible use cases include communicating with robotics libraries for computer vision, and other use cases where real time data to/from actuators and sensors is essential for vehicle control.

FastRTPS is not intended as a replacement for MAVLink. MAVLink remains the most appropriate protocol for communicating with ground stations, gimbals, cameras, etc. (although FastRTPS may open other opportunities for working with some peripherals).

RTPS can be used over slower links (e.g. like radio telemetry, but care should be taken not to overload the channel).

Architectural overview



The main elements of the architecture are the client and agent processes shown in the diagram above.

- The *Client* is PX4 middleware daemon process that runs on the flight controller. It subscribes to uORB topics published by other PX4 components and sends any updates to the *Agent* (via a UART or UDP port). It also receives messages from the *Agent* and publishes them as uORB message on PX4.
- The *Agent* runs as a daemon process on an offboard computer. It watches for uORB update messages from the *Client* and (re)publishes them over RTPS. It also subscribes to "uORB" RTPS messages from other RTPS applications and forwards them to the *Client*.
- The *Agent* and *Client* are connected via a serial link (UART) or UDP network. The uORB information is [CDR serialized](#) for sending (*CDR serialization* provides a common format for exchanging serial data between different platforms).
- The *Agent* and any *FastRTPS* applications are connected via UDP, and may be on the same or another device. In a typical configuration they will both be on the same system (e.g. a development computer, Linux companion computer or compute board), connected to the *Client* over a Wifi link or via USB.

Code generation

All the code needed to create, build and use the bridge is automatically generated when the PX4 Firmware is compiled.

The *Client* application is also compiled and built into the firmware as part of the normal build process. The *Agent* must be separately/manually compiled for the target computer.

[Fast RTPS must be installed](#) in order to generate the required code!

The bridge code can also be [manually generated](#). Most users will not need to do so, but the linked topic provides a more detailed overview of the build process and can be useful for troubleshooting.

Supported uORB messages

The generated bridge code will enable a specified subset of uORB topics to be published/subscribed via RTPS.

For *automatic code generation* (via the normal PX4 firmware build process) this set must be listed in the `/path/to/PX4/Firmware/src/modules/micrortps_bridge/CMakeLists.txt` file.

```
set(config_rtps_send_topics
  sensor_combined
  # Add new topic...
)

set(config_rtps_receive_topics
  sensor_baro
  # Add new topic...
)
```

At time of writing (August 2017), only the small set of uORB topics listed above are included in our cmake files: `posix_sitl_default.cmake`, `nuttx_px4fmu-v4_default.cmake`, `posix_sdfight_default.cmake`. It is likely you will need to edit your *cmake* file and add additional uORB topics. In future we hope to define a larger standard set.

For *manual code generation* the uORB topics that will be supported by the bridge are specified when you call `generate_microRTPS_bridge.py` (using the `-s/--send` and `-r/--receive` flags). See [Manual Generation of the Code](#) for more information.

Client (PX4 Firmware)

The *Client* source code is generated, compiled and built into the PX4 firmware as part of the normal build process.

To build and upload the firmware for NuttX/Pixhawk flight controllers:

```
make px4fmu-v4_default upload
```

To build and upload the firmware for Qualcomm Snapdragon Flight:

```
$ make eagle_default upload
```

The *Client* application can be launched from [NutShell/System Console](#). The command syntax is shown below (you can specify a variable number of arguments):

```
> micrortps_client start|stop [options]
-t <transport>          [UART|UDP] Default UART
-d <device>             UART device. Default /dev/ttyACM0
-u <update_time_ms>    Time in ms for uORB subscribed topics update. Default 0
-l <loops>              How many iterations will this program have. -1 for infinite.
Default -1.
-w <sleep_time_ms>     Time in ms for which each iteration sleep. Default 1ms
-b <baudrate>          UART device baudrate. Default 460800
-p <poll_ms>           Time in ms to poll over UART. Default 1ms
-r <reception port>    UDP port for receiving. Default 2019
-s <sending port>      UDP port for sending. Default 2020
```

By default the *Client* runs as a daemon, but you will need to start it manually. The PX4 Firmware initialisation code may in future automatically start the *Client* as a permanent daemon process.

Agent (Off Board FastRTPS Interface)

The *Agent* code is automatically *generated* when you build the associated PX4 firmware. You can find the source here: **build/src/modules/micrortps_bridge/micrortps_client/micrortps_agent/**.

To build the *Agent* application, compile the code:

```
cd build/<target-
platform>/src/modules/micrortps_bridge/micrortps_client/micrortps_agent
mkdir build && cd build
cmake ..
make
```

To cross-compile for the *Qualcomm Snapdragon Flight* platform see [this link](#).

The command syntax for the *Agent* is listed below:

```
$ ./micrortps_agent [options]
-t <transport>          [UART|UDP] Default UART.
-d <device>             UART device. Default /dev/ttyACM0.
-w <sleep_time_us>     Time in us for which each iteration sleep. Default 1ms.
-b <baudrate>          UART device baudrate. Default 460800.
-p <poll_ms>           Time in ms to poll over UART. Default 1ms.
-r <reception port>    UDP port for receiving. Default 2019.
-s <sending port>      UDP port for sending. Default 2020.
```

To launch the *Agent*, run `micrortps_agent` with appropriate options for specifying the connection to the *Client* (the default options connect from a Linux device to the *Client* over a UART port).

Creating a FastRTPS Listener application

Once the *Client* (on the flight controller) and the *Agent* (on an offboard computer) are running and connected, *FastRTPS* applications can publish and subscribe to uORB topics on PX4 using RTPS.

This example shows how to create a *FastRTPS* "listener" application that subscribes to the `sensor_combined` topic and prints out updates (from PX4). A connected RTPS application can run on any computer on the same network as the *Agent*. For this example the *Agent* and *Listener application* will be on the same computer. The `fastrtpsgen` script can be used to generate a simple RTPS application from an IDL message file.

RTPS messages are defined in IDL files and compiled to C++ using `fastrtpsgen`. As part of building the bridge code, IDL files are generated for the uORB message files that may be sent/received (see `build/BUILDPLATFORM/src/modules/micrortps_bridge/micrortps_agent/idl/*.idl`). These IDL files are needed when you create a FastRTPS application to communicate with PX4.

Enter the following commands to create the application:

```
cd /path/to/PX4/Firmware/src/modules/micrortps_bridge
mkdir micrortps_listener
cd micrortps_listener
fastrtpsgen -example x64Linux2.6gcc ../micrortps_agent/idl/sensor_combined.idl
```

This creates a basic subscriber and publisher, and a main-application to run them. To print out the data from the `sensor_combined` topic, modify the `onNewDataMessage()` method in `sensor_combined_Subscriber.cxx`:

```
void sensor_combined_Subscriber::SubListener::onNewDataMessage(Subscriber* sub)
{
    // Take data
    sensor_combined_st;

    if(sub->takeNextData(&st, &m_info))
    {
        if(m_info.sampleKind == ALIVE)
        {
            // Print your structure data here.
            ++n_msg;
            std::cout << "\n\n\n\n\n\n\n\n\n\n";
        }
    }
}
```

```

std::cout << "Sample received, count=" << n_msg << std::endl;
std::cout << "======" << std::endl;
std::cout << "gyro_rad: " << st.gyro_rad().at(0);
std::cout << ", " << st.gyro_rad().at(1);
std::cout << ", " << st.gyro_rad().at(2) << std::endl;
std::cout << "gyro_integral_dt: " << st.gyro_integral_dt() << std::endl;
std::cout << "accelerometer_timestamp_relative: " <<
st.accelerometer_timestamp_relative() << std::endl;
std::cout << "accelerometer_m_s2: " << st.accelerometer_m_s2().at(0);
std::cout << ", " << st.accelerometer_m_s2().at(1);
std::cout << ", " << st.accelerometer_m_s2().at(2) << std::endl;
std::cout << "accelerometer_integral_dt: " <<
st.accelerometer_integral_dt() << std::endl;
std::cout << "magnetometer_timestamp_relative: " <<
st.magnetometer_timestamp_relative() << std::endl;
std::cout << "magnetometer_ga: " << st.magnetometer_ga().at(0);
std::cout << ", " << st.magnetometer_ga().at(1);
std::cout << ", " << st.magnetometer_ga().at(2) << std::endl;
std::cout << "baro_timestamp_relative: " << st.baro_timestamp_relative()
<< std::endl;
std::cout << "baro_alt_meter: " << st.baro_alt_meter() << std::endl;
std::cout << "baro_temp_celcius: " << st.baro_temp_celcius() << std::endl;

    }
}
}

```

To build and run the application on Linux:

```

make -f makefile_x64Linux2.6gcc
bin/*/sensor_combined_PublisherSubscriber subscriber

```

Now you should see the sensor information being printed out:

```

Sample received, count=10119
Received sensor_combined data
====
gyro_rad: -0.0103228, 0.0140477, 0.000319406
gyro_integral_dt: 0.004
accelerometer_timestamp_relative: 0
accelerometer_m_s2: -2.82708, -6.34799, -7.41101
accelerometer_integral_dt: 0.004
magnetometer_timestamp_relative: -10210
magnetometer_ga: 0.60171, 0.0405879, -0.040995
baro_timestamp_relative: -17469
baro_alt_meter: 368.647
baro_temp_celcius: 43.93

```

If the *Listener application* does not print anything, make sure the *Client* is running.

Examples/tests

The following examples provide additional real-world demonstrations of how to use the features described in this topic.

- [Throughput test](#): A simple test to measure the throughput of the bridge.

Troubleshooting

Client reports that selected UART port is busy

If the selected UART port is busy, it's possible that the MAVLink application is already being used. If both MAVLink and RTPS connections are required you will have to either move the connection to use another port or configure the port so that it can be shared.

A quick/temporary fix to allow bridge testing during development is to stop MAVLink from *NuttShell*:

```
mavlink stop-all
```

Agent not built/fastrtpsgen is not found

The *Agent* code is generated using a *FastRTPS* tool called *fastrtpsgen*.

If you haven't installed Fast RTPS in the default path then you must specify its installation directory by setting the `FASTRTPSGEN_DIR` environment variable before executing *make*.

On Linux/Mac this is done as shown below:

```
export FASTRTPSGEN_DIR=/path/to/fastrtps/install/folder/bin
```

This should not be a problem if [Fast RTPS is installed in the default location](#).

Enable UART on Raspberry Pi

For UART transport on Raspberry Pi you will have to enable the serial port:

1. Make sure the `userid` (default is `pi`) is a member of the `dialout` group:

```
groups pi
```

```
sudo usermod -a -G dialout pi
```

2. You need to stop the GPIO serial console that is using the port:

```
sudo raspi-config
```

In the menu showed go to **Interfacing options > Serial**. Select **NO** for *Would you like a login shell to be accessible over serial?*. Valid and reboot.

3. Check UART in kernel:

```
sudo vi /boot/config.txt
```

And make sure that the `enable_uart` value is set to 1:

```
enable_uart=1
```

Additional information

- [FastRTPS Installation](#)
- [Manually Generate Client and Agent Code](#)
- [DDS and ROS middleware implementations](#)

6.5 Micro RTPS Throughput Test

This a simple test to measure the throughput of the [PX4-FastRTPS Bridge](#). It sends and receives 256-byte messages (simultaneously) at maximum rate, and then outputs the result.

This example requires that you [Manually Generate Client and Agent Code](#).

Create the uORB message

First create a new uORB message for this test in the folder **/Firmware/msg/**. The message file will be called **throughput_256.msg** and have the following content:

```
uint8[256] data
```

This can be done with the command line below:

```
cd /path/to/PX4/Firmware/msg
echo uint8[256] data > throughput_256.msg
```

Register the new message adding it to the list of messages in the file: **/Firmware/msg/CMakeLists.txt**:

```
...
wind_estimate.msg
throughput_256.msg
)
...
```

Give the message a topic id by adding a line in the **/Firmware/Tools/message_id.py** script:

```
...
    'wind_estimate': 94,
    'throughput_256': 95,
}
...
```

Disable automatic bridge code generation

Disable automatic generation of bridge code (as part of the PX4 build process) by setting the variable `GENERATE_RTSPS_BRIDGE` to `off` in the `.cmake` file for the target platform (`cmake/configs/`):

```
set(GENERATE_RTSPS_BRIDGE off)
```

Generate the bridge code

Manually generate bridge code using `generate_microRTSPS_bridge.py` (the code will send and receive "just" our `throughput_256` uORB topic):

```
cd /path/to/PX4/Firmware
python Tools/generate_microRTSPS_bridge.py --send msg/throughput_256.msg --receive
msg/throughput_256.msg
```

The *Client* source code is generated in `src/modules/micrortps_bridge/micrortps_client/` and the *Agent* in `src/modules/micrortps_bridge/micrortps_agent/`.

Modify the client code

Next we modify the *Client* to send a `throughput_256` message on every loop. This is required because the topic is not actually being published by PX4, and because we want to ensure that it is sent at the greatest possible rate.

Open the file `src/modules/micrortps_bridge/micrortps_client/microRTSPS_client.cpp`. Update the `while` loop in the `send()` function to look like this:

```
...
while (!_should_exit_task)
{
    //bool updated;
    //orb_check(fds[0], &updated);
    //if (updated)
    {
        // obtained data for the file descriptor
        struct throughput_256_s data = {};
        // copy raw data into local buffer
```

```

//orb_copy(ORB_ID(throughput_256), fds[0], &data);
data.data[0] = loop%256;
serialize_throughput_256(&data, data_buffer, &length, &microCDRWriter);
if (0 < (read = transport_node->write((char)95, data_buffer, length)))
{
    total_sent += read;
    ++sent;
}
}
usleep(_options.sleep_ms*1000);
++loop;
}
...

```

You may recall this is intended to be a *bidirectional* throughput test, where messages must also be sent from the *Agent* to the *Client*. You do not need to modify the *Agent* code to make this happen. As the *Agent* is an RTPS publisher and subscriber, it will automatically get notified of the RTPS messages it sends, and will then mirror these back to the client.

[Compile and launch](#) both the *Client* and the *Agent*.

Result

The test was executed with PX4 running on Pixracer, connected via a UART to an ordinary PC running Ubuntu 16.04. The default configuration was used for both the *Client/Agent*.

The throughput that was observed in the client shell window on completion is shown below:

```

SENT:      13255 messages in 13255 LOOPS, 3512575 bytes in 30.994 seconds - 113.33KB/s
RECEIVED: 13251 messages in 10000 LOOPS, 3511515 bytes in 30.994 seconds - 113.30KB/s

```

6.6 Manually Generate Client and Agent Code

This topic shows how to manually generate the code for the client and the agent (instead of [automatically generating](#) it when the PX4 Firmware is compiled).

The code is generated using the python script: `/Tools/generate_microRTPS_bridge.py`.

Disable automatic bridge code generation

First disable automatic generation of bridge code. Set the variable `GENERATE_RTPS_BRIDGE` to `off` in the `.cmake` file for the target platform:

```
set(GENERATE_RTPS_BRIDGE off)
```

Using `generate_microRTPS_bridge.py`

The `generate_microRTPS_bridge` tool's command syntax is shown below:

```
$ cd /path/to/PX4/Firmware/msg/tools
$ python generate_microRTPS_bridge.py -h
usage: generate_microRTPS_bridge.py [-h] [-s *.msg [*.msg ...]]
                                     [-r *.msg [*.msg ...]] [-a] [-c]
                                     [-t MSGDIR] [-o AGENTDIR] [-u CLIENTDIR]
                                     [-f FASTRTPSGEN]
```

optional arguments:

```
-h, --help            show this help message and exit
-s *.msg [*.msg ...], --send *.msg [*.msg ...]
                       Topics to be sent
-r *.msg [*.msg ...], --receive *.msg [*.msg ...]
                       Topics to be received
-a, --agent           Flag to generate the agent. Default is true.
-c, --client          Flag to generate the client. Default is true.
-t MSGDIR, --topic-msg-dir MSGDIR
                       Topics message dir. Default is: msg/
-o AGENTDIR, --agent-outdir AGENTDIR
                       Agent output dir. Default is:
                       src/modules/micrortps_bridge/micrortps_agent
-u CLIENTDIR, --client-outdir CLIENTDIR
                       Client output dir. Default is:
                       src/modules/micrortps_bridge/micrortps_client
-f FASTRTPSGEN, --fastrtpsgen-dir FASTRTPSGEN
                       fastrtpsgen installation dir. Default is: /bin
--delete-tree         Delete dir tree output dir(s)
```

Using with `--delete-tree` option erases the content of the `CLIENTDIR` and the `AGENTDIR` before creating new files and folders.

- The arguments `--send/-s` and `--receive/-r` specify the uORB topics that can be sent/received from PX4. Code will only be generated for specified messages.
- The output appears in `CLIENTDIR` (`-o src/modules/micrortps_bridge/micrortps_client`, by default) and in the `AGENTDIR` (`-u src/modules/micrortps_bridge/micrortps_agent`, by default).

-
- If no flag `-a` or `-c` is specified, both the client and the agent will be generated and installed.
 - The `-f` option may be needed if *Fast RTPS* was not installed in the default location (`-f /path/to/fastrtps/installation/bin`).

The example below shows how you can generate bridge code to publish/subscribe just the `sensor_baro` single uORB topic.

```
$ cd /path/to/PX4/Firmware
$ python Tools/generate_microRTPS_bridge.py -s msg/sensor_baro.msg -r
msg/sensor_combined.msg
```

Generated code

Code is generated for the *Client*, *Agent*, *CDR serialization/deserialization* of uORB messages, and the definition of the associated RTPS messages (IDL files).

Manually generated code for the bridge can be found here (by default):

- *Client*: `src/modules/micrortps_bridge/micrortps_client/`
- *Agent*: `src/modules/micrortps_bridge/micrortps_agent/`

uORB serialization code

Serialization functions are generated for all the uORB topics as part of the normal PX4 compilation process (and also for manual generation). For example, the following functions would be generated for the `sensor_combined.msg`:

```
void serialize_sensor_combined(const struct sensor_combined_s *input, char *output,
uint32_t *length, struct microCDR *microCDRWriter);
void deserialize_sensor_combined(struct sensor_combined_s *output, char *input,
struct microCDR *microCDRReader);
```

RTPS message IDL files

IDL files are generated from the uORB `.msg` files ([for selected uORB topics](#)) in the generation of the bridge. These can be found in: `src/modules/micrortps_bridge/micrortps_agent/idl/`

FastRTSP uses IDL files to define the structure of RTPS messages (in this case, RTPS messages that map to uORB topics). They are used to generate code for the *Agent*, and *FastRTSP* applications that need to publish/subscribe to uORB topics.

IDL files are compiled to C++ by the `fastrtpsgen` tool.

Verify code generation

You can verify successful code generation by checking that the output directories match the listing shown below (On Linux, the `tree` command can be used for listing the file structure).

Agent directory:

```
$ tree src/modules/micrortps_bridge/micrortps_agent
src/modules/micrortps_bridge/micrortps_agent
├── build
├── CMakeLists.txt
├── idl
│   ├── sensor_baro_.idl
│   └── sensor_combined_.idl
├── microRTPS_agent.cpp
├── microRTPS_transport.cpp
├── microRTPS_transport.h
├── RtpsTopics.cpp
├── RtpsTopics.h
├── sensor_baro_.cpp
├── sensor_baro_.h
├── sensor_baro_Publisher.cpp
├── sensor_baro_Publisher.h
├── sensor_baro_PubSubTypes.cpp
├── sensor_baro_PubSubTypes.h
├── sensor_combined_.cpp
├── sensor_combined_.h
├── sensor_combined_PubSubTypes.cpp
├── sensor_combined_PubSubTypes.h
├── sensor_combined_Subscriber.cpp
└── sensor_combined_Subscriber.h
2 directories, 20 files
```

Client directory:

```
$ tree src/modules/micrortps_bridge/micrortps_client
src/modules/micrortps_bridge/micrortps_client
├── CMakeLists.txt
├── microRTPS_client.cpp
├── microRTPS_client_dummy.cpp
├── microRTPS_client_main.cpp
├── microRTPS_transport.cpp
└── microRTPS_transport.h
0 directories, 4 files
```

Build and use the code

The manually generated *Client* code is built and used in *exactly* the same way as [automatically generated Client code](#).

Specifically, once manually generated, the *Client* source code is compiled and built into the PX4 firmware as part of the normal build process. For example, to compile the code and include it in firmware for NuttX/Pixhawk targets:

```
make px4fmu-v4_default upload
```

You must first [disable automatic bridge code generation](#) so that the toolchain uses the manually generated source code (and does not attempt to regenerate it).

The manually generated *Agent* code is also compiled and used in the same way as the [automatically generated code](#). The only difference is that the manually source code is created in **src/modules/micrortps_bridge/micrortps_agent** instead of **build/BUILDPLATFORM/src/modules/micrortps_bridge/micrortps_agent/**.

7. Modules & Commands Reference

The following pages document the PX4 modules, drivers and commands. They describe the provided functionality, high-level implementation overview and how to use the command-line interface.

This is auto-generated from the source code and contains the most recent modules documentation.

It is not a complete list and NuttX provides some additional commands as well (such as `free`). Use `help` on the console to get a list of all available commands, and in most cases `command help` will print the usage.

Since this is generated from source, errors must be reported/fixed in the [Firmware](#) repository. The documentation pages can be generated by running the following command from the root of the Firmware directory:

```
make module_documentation
```

The generated files will be written to the `modules` directory.

Categories

- [Command](#)
- [Communication](#)
- [Controller](#)
- [Driver](#)
- [Estimator](#)
- [System](#)

7.1 Modules Reference: Command

bl_update

Source: [systemcmds/bl_update](#)

Utility to flash the bootloader from a file

Usage

```
bl_update [arguments...]  
  setopt          Set option bits to unlock the FLASH (only needed if in locked  
                  state)  
  
  <file>          Bootloader bin file
```

config

Source: [systemcmds/config](#)

Configure a sensor driver (sampling & publication rate, range, etc.)

Usage

```
config <command> [arguments...]
```

Commands:

The <file:dev> argument is typically one of /dev/{gyro,accel,mag}i

`block` Block sensor topic publication

 <file:dev> Sensor device file

`unblock` Unblock sensor topic publication

 <file:dev> Sensor device file

`sampling` Set sensor sampling rate

 <file:dev> <rate> Sensor device file and sampling rate in Hz

`rate` Set sensor publication rate

 <file:dev> <rate> Sensor device file and publication rate in Hz

`range` Set sensor measurement range

 <file:dev> <rate> Sensor device file and range

`check` Perform sensor self-test (and print info)

 <file:dev> Sensor device file

dumpfile

Source: [systemcmds/dumpfile](#)

Dump file utility. Prints file size and contents in binary mode (don't replace LF with CR LF) to stdout.

Usage

```
dumpfile [arguments...]
```

 <file> File to dump

esc_calib

Source: [systemcmds/esc_calib](#)

Tool for ESC calibration

Calibration procedure (running the command will guide you through it):

- Remove props, power off the ESC's
- Stop attitude controllers: `mc_att_control stop`, `fw_att_control stop`
- Make sure safety is off
- Run this command

Usage

```
esc_calib [arguments...]  
  [-d <val>] Select PWM output device  
              values: <file:dev>, default: /dev/pwm_output0  
  [-l <val>] Low PWM value in us  
              default: 1000  
  [-h <val>] High PWM value in us  
              default: 2000  
  [-c <val>] select channels in the form: 1234 (1 digit per channel,  
              1=first)  
  [-m <val>] Select channels via bitmask (eg. 0xF, 3)  
              default: 0  
  [-a]       Select all channels
```

hardfault_log

Source: [systemcmds/hardfault_log](#)

Hardfault utility

Used in startup scripts to handle hardfaults

Usage

```
hardfault_log <command> [arguments...]  
Commands:  
  check          Check if there's an uncommitted hardfault  
  
  rearm          Drop an uncommitted hardfault  
  
  fault          Generate a hardfault (this command crashes the system :)  
  [0|1]         Hardfault type: 0=divide by 0, 1=Assertion (default=0)  
  
  commit         Write uncommitted hardfault to /fs/microsd/fault_%i.txt (and  
                rearm, but don't reset)  
  
  count          Read the reboot counter, counts the number of reboots of an
```

```
uncommitted hardfault (returned as the exit code of the program)
```

```
reset          Reset the reboot counter
```

led_control

Source: [systemcmds/led_control](#)

Description

Command-line tool to control & test the (external) LED's.

To use it make sure there's a driver running, which handles the led_control uorb topic.

There are different priorities, such that for example one module can set a color with low priority, and another module can blink N times with high priority, and the LED's automatically return to the lower priority state after the blinking. The `reset` command can also be used to return to a lower priority.

Examples

Blink the first LED 5 times in blue:

```
led_control blink -c blue -l 0 -n 5
```

Usage

```
led_control <command> [arguments...]
```

Commands:

```
test          Run a test pattern

on           Turn LED on

off          Turn LED off

reset        Reset LED priority

blink        Blink LED N times
  [-n <val>] Number of blinks
              default: 3
  [-s <val>] Set blinking speed
              values: fast|normal|slow, default: normal

breathe      Continuously fade LED in & out
```

```
flash          Two fast blinks and then off with frequency of 1Hz
```

The following arguments apply to all of the above commands except for 'test':

```
[-c <val>]  color
            values: red|blue|green|yellow|purple|amber|cyan|white, default:
            white
[-l <val>]  Which LED to control: 0, 1, 2, ... (default=all)
            default: -1
[-p <val>]  Priority
            default: 2
```

listener

Source: [systemcmds/topic_listener](#)

Utility to listen on uORB topics and print the data to the console.

Limitation: it can only listen to the first instance of a topic.

Usage

```
listener [arguments...]
        <topic_name> [<num_msgs>] uORB topic name and optionally number of messages
        (default=1)
```

mixer

Source: [systemcmds/mixer](#)

Description

Load or append mixer files to the ESC driver.

Note that the driver must support the used ioctl's, which is the case on NuttX, but for example not on RPi.

Usage

```
mixer <command> [arguments...]
Commands:
  load
    <file:dev> <file> Output device (eg. /dev/pwm_output0) and mixer file

  append
    <file:dev> <file> Output device (eg. /dev/pwm_output0) and mixer file
```

motor_ramp

Source: [systemcmds/motor_ramp](#)

Description

Application to test motor ramp up.

Before starting, make sure to stop any running attitude controller:

```
mc_att_control stop
fw_att_control stop
```

When starting, a background task is started, runs for several seconds (as specified), then exits.

Note: this command currently only supports the `/dev/pwm_output0` output.

Example

```
motor_ramp sine 1100 0.5
```

Usage

```
motor_ramp [arguments...]
    ramp|sine|square mode
    <min_pwm> <time> [<max_pwm>] pwm value in us, time in sec
```

WARNING: motors will ramp up to full speed!

motor_test

Source: [systemcmds/motor_test](#)

Utility to test motors.

Note: this can only be used for drivers which support the `motor_test` uorb topic (currently uavcan and tap_esc)

Usage

```
motor_test <command> [arguments...]
Commands:
  test          Set motor(s) to a specific output value
  [-m <val>]   Motor to test (0...7, all if not specified)
                default: -1
  [-p <val>]   Power (0...100)
                default: 0
```

stop	Stop all motors
iterate	Iterate all motors starting and stopping one after the other

mtd

Source: [systemcmds/mtd](#)

Utility to mount and test partitions (based on FRAM/EEPROM storage as defined by the board)

Usage

```
mtd <command> [arguments...]
```

Commands:

status	Print status information
start	Mount partitions
readtest	Perform read test
rwtest	Perform read-write test
erase	Erase partition(s)

The commands 'start', 'readtest', 'rwtest' and 'erase' have an optional parameter:

```
[<partition_name1> [<partition_name2> ...]] Partition names (eg. /fs/mtd_params), use system default if not provided
```

nshterm

Source: [systemcmds/nshterm](#)

Start an NSH shell on a given port.

This was previously used to start a shell on the USB serial port. Now there runs mavlink, and it is possible to use a shell over mavlink.

Usage

```
nshterm [arguments...]
```

```
<file:dev> Device on which to start the shell (eg. /dev/ttyACM0)
```

param

Source: [systemcmds/param](#)

Description

Command to access and manipulate parameters via shell or script.

This is used for example in the startup script to set airframe-specific parameters.

Parameters are automatically saved when changed, eg. with `param set`. They are typically stored to FRAM or to the SD card. `param select` can be used to change the storage location for subsequent saves (this will need to be (re-)configured on every boot).

Each parameter has a 'used' flag, which is set when it's read during boot. It is used to only show relevant parameters to a ground control station.

Examples

Change the airframe and make sure the airframe's default parameters are loaded:

```
param set SYS_AUTOSTART 4001
param set SYS_AUTOCONFIG 1
reboot
```

Usage

```
param <command> [arguments...]
```

Commands:

```
load          Load params from a file (overwrite all)
  [<file>]    File name (use default if not given)
```

```
import        Import params from a file
  [<file>]    File name (use default if not given)
```

```
save          Save params to a file
  [<file>]    File name (use default if not given)
```

```
select        Select default file
  [<file>]    File name (use <root>/eeprom/parameters if not given)
```

```
show          Show parameter values
  [-c]        Show only changed params
  [<filter>]  Filter by param name (wildcard at end allowed, eg. sys_*)
```

```

set          Set parameter to a value
  <param_name> <value> Parameter name and value to set
  [fail]      If provided, let the command fail if param is not found

compare     Compare a param with a value. Command will succeed if equal
  <param_name> <value> Parameter name and value to compare

greater     Compare a param with a value. Command will succeed if param is
            greater than the value
  <param_name> <value> Parameter name and value to compare

reset       Reset params to default
  [<exclude1> [<exclude2>]] Do not reset matching params (wildcard at end
            allowed)

reset_nostart Reset params to default, but keep SYS_AUTOSTART and
            SYS_AUTOCONFIG
  [<exclude1> [<exclude2>]] Do not reset matching params (wildcard at end
            allowed)

index       Show param for a given index
  <index>    Index: an integer >= 0

index_used  Show used param for a given index
  <index>    Index: an integer >= 0

find        Show index of a param
  <param>    param name

```

perf

Source: systemcmds/perf

Tool to print performance counters

Usage

```

perf [arguments...]
  reset          Reset all counters

  latency       Print HRT timer latency histogram

Prints all performance counters if no arguments given

```

pwm

Source: [systemcmds/pwm](#)

Description

This command is used to configure PWM outputs for servo and ESC control.

The default device `/dev/pwm_output0` are the Main channels, AUX channels are on `/dev/pwm_output1` (-d parameter).

It is used in the startup script to make sure the PWM parameters (`PWM_*`) are applied (or the ones provided by the airframe config if specified). `pwm info` shows the current settings (the trim value is an offset and configured with `PWM_MAIN_TRIMx` and `PWM_AUX_TRIMx`).

The disarmed value should be set such that the motors don't spin (it's also used for the kill switch), at the minimum value they should spin.

Channels are assigned to a group. Due to hardware limitations, the update rate can only be set per group. Use `pwm info` to display the groups. If the `-c` argument is used, all channels of any included group must be included.

The parameters `-p` and `-r` can be set to a parameter instead of specifying an integer: use `-p p:PWM_MIN` for example.

Note that in OneShot mode, the PWM range [1000, 2000] is automatically mapped to [125, 250].

Examples

Set the PWM rate for all channels to 400 Hz:

```
pwm rate -a -r 400
```

Test the outputs of eg. channels 1 and 3, and set the PWM value to 1200 us:

```
pwm arm
pwm test -c 13 -p 1200
```

Usage

```
pwm <command> [arguments...]
```

Commands:

<code>arm</code>	Arm output
<code>disarm</code>	Disarm output
<code>info</code>	Print current configuration of all channels
<code>forcefail</code>	Force Failsafe mode

```

on|off      Turn on or off

terminatefail Force Termination Failsafe mode
on|off      Turn on or off

rate        Configure PWM rates
-r <val>    PWM Rate in Hz (0 = Oneshot, otherwise 50 to 400Hz)

oneshot     Configure Oneshot125 (rate is set to 0)

failsafe    Set Failsafe PWM value

disarmed    Set Disarmed PWM value

min         Set Minimum PWM value

max         Set Maximum PWM value

test        Set Output to a specific value until 'q' or 'c' or 'ctrl-c'
            pressed

steps       Run 5 steps from 0 to 100%

```

The commands 'failsafe', 'disarmed', 'min', 'max' and 'test' require a PWM value:

```
-p <val>    PWM value (eg. 1100)
```

The commands 'rate', 'oneshot', 'failsafe', 'disarmed', 'min', 'max', 'test' and 'steps' additionally require to specify the channels with one of the following commands:

```

[-c <val>] select channels in the form: 1234 (1 digit per channel,
            1=first)
[-m <val>] Select channels via bitmask (eg. 0xF, 3)
            default: 0
[-g <val>] Select channels by group (eg. 0, 1, 2. use 'pwm info' to show
            groups)
            default: 0
[-a]       Select all channels

```

These parameters apply to all commands:

```

[-d <val>] Select PWM output device
            values: <file:dev>, default: /dev/pwm_output0
[-v]       Verbose output
[-e]       Exit with 1 instead of 0 on error

```

reboot

Source: [systemcmds/reboot](#)

Reboot the system

Usage

```
reboot [arguments...]  
  [-b]          Reboot into bootloader  
  [lock|unlock] Take/release the shutdown lock (for testing)
```

sd_bench

Source: [systemcmds/sd_bench](#)

Test the speed of an SD Card

Usage

```
sd_bench [arguments...]  
  [-b <val>] Block size for each read/write  
              default: 4096  
  [-r <val>] Number of runs  
              default: 5  
  [-d <val>] Duration of a run in ms  
              default: 2000  
  [-s]       Call fsync after each block (default=at end of each run)
```

top

Source: [systemcmds/top](#)

Monitor running processes and their CPU, stack usage, priority and state

Usage

```
top [arguments...]  
  once          print load only once
```

usb_connected

Source: [systemcmds/usb_connected](#)

Utility to check if USB is connected. Was previously used in startup scripts. A return value of 0 means USB is connected, 1 otherwise.

Usage

```
usb_connected [arguments...]
```

ver

Source: [systemcmds/ver](#)

Tool to print various version information

Usage

```
ver <command> [arguments...]
```

Commands:

hw	Hardware architecture
mcu	MCU info
git	git version information
bdate	Build date and time
gcc	Compiler info
bdate	Build date and time
uid	UUID
mfguid	Manufacturer UUID
uri	Build URI
all	Print all versions
hwcmp	Compare hardware version (returns 0 on match) <hw> [<hw2>] Hardware to compare against (eg. PX4FMU_V4). An OR comparison is used if multiple are specified
hwtypecmp	Compare hardware type (returns 0 on match) <hwtype> [<hwtype2>] Hardware type to compare against (eg. V2). An OR comparison is used if multiple are specified

7.2 Modules Reference: Communication

frsky_telemetry

Source: [drivers/telemetry/frsky_telemetry](#)

FrSky Telemetry support. Auto-detects D or S.PORT protocol.

Usage

```
frsky_telemetry <command> [arguments...]  
Commands:  
  start  
    [-d <val>] Select Serial Device  
                values: <file:dev>, default: /dev/ttyS6  
  
  stop  
  
  status
```

mavlink

Source: [modules/mavlink](#)

Description

This module implements the MAVLink protocol, which can be used on a Serial link or UDP network connection. It communicates with the system via uORB: some messages are directly handled in the module (eg. mission protocol), others are published via uORB (eg. vehicle_command).

Streams are used to send periodic messages with a specific rate, such as the vehicle attitude. When starting the mavlink instance, a mode can be specified, which defines the set of enabled streams with their rates. For a running instance, streams can be configured via `mavlink stream` command.

There can be multiple independent instances of the module, each connected to one serial device or network port.

Implementation

The implementation uses 2 threads, a sending and a receiving thread. The sender runs at a fixed rate and dynamically reduces the rates of the streams if the combined

bandwidth is higher than the configured rate (`-r`) or the physical link becomes saturated. This can be checked with `mavlink status`, see if `rate mult` is less than 1.

Careful: some of the data is accessed and modified from both threads, so when changing code or extend the functionality, this needs to be take into account, in order to avoid race conditions and corrupt data.

Examples

Start mavlink on `ttyS1` serial with baudrate 921600 and maximum sending rate of 80kB/s:

```
mavlink start -d /dev/ttyS1 -b 921600 -m onboard -r 80000
```

Start mavlink on UDP port 14556 and enable the `HIGHRES_IMU` message with 50Hz:

```
mavlink start -u 14556 -r 1000000
mavlink stream -u 14556 -s HIGHRES_IMU -r 50
```

Usage

```
mavlink <command> [arguments...]
```

Commands:

```
start          Start a new instance
  [-d <val>]   Select Serial Device
                values: <file:dev>, default: /dev/ttyS1
  [-b <val>]   Baudrate
                default: 57600
  [-r <val>]   Maximum sending data rate in B/s (if 0, use baudrate / 20)
                default: 0
  [-u <val>]   Select UDP Network Port (local)
                default: 14556
  [-o <val>]   Select UDP Network Port (remote)
                default: 14550
  [-t <val>]   Partner IP (broadcasting can be enabled via MAV_BROADCAST
                param)
                default: 127.0.0.1
  [-m <val>]   Mode: sets default streams and rates
                values: custom|camera|onboard|osd|magic|config|iridium,
                default: normal
  [-f]         Enable message forwarding to other Mavlink instances
  [-w]         Wait to send, until first message received
  [-x]         Enable FTP
  [on|off]    Enable/disable

stop-all      Stop all instances

status        Print status for all instances
```

```
stream      Configure the sending rate of a stream for a running instance
[-u <val>]  Select Mavlink instance via local Network Port
            default: 0
[-d <val>]  Select Mavlink instance via Serial Device
            values: <file:dev>
-s <val>    Mavlink stream to configure
-r <val>    Rate in Hz (0 = turn off)

boot_complete Enable sending of messages. (Must be) called as last step in
              startup script.
```

micrortps_client

Source: [modules/micrortps_bridge/micrortps_client](#)

Usage

```
micrortps_client <command> [arguments...]
```

Commands:

start

```
[-t <val>]  Transport protocol
            values: UART|UDP, default: UART
[-d <val>]  Select Serial Device
            values: <file:dev>, default: /dev/ttyACM0
[-b <val>]  Baudrate
            default: 460800
[-p <val>]  Poll timeout for UART in ms
            default: 1
[-u <val>]  Interval in ms to limit the update rate of all sent topics
            (0=unlimited)
            default: 0
[-l <val>]  Limit number of iterations until the program exits
            (-1=infinite)
            default: 10000
[-w <val>]  Time in ms for which each iteration sleeps
            default: 1
[-r <val>]  Select UDP Network Port for receiving (local)
            default: 2019
[-s <val>]  Select UDP Network Port for sending (remote)
            default: 2020
```

stop

status

uorb

Source: [modules/uORB](#)

Description

uORB is the internal pub-sub messaging system, used for communication between modules.

It is typically started as one of the very first modules and most other modules depend on it.

Implementation

No thread or work queue is needed, the module start only makes sure to initialize the shared global state. Communication is done via shared memory. The implementation is asynchronous and lock-free, ie. a publisher does not wait for a subscriber and vice versa. This is achieved by having a separate buffer between a publisher and a subscriber.

The code is optimized to minimize the memory footprint and the latency to exchange messages.

The interface is based on file descriptors: internally it uses `read`, `write` and `ioctl`. Except for the publications, which use `orb_advert_t` handles, so that they can be used from interrupts as well (on NuttX).

Messages are defined in the `/msg` directory. They are converted into C/C++ code at build-time.

If compiled with `ORB_USE_PUBLISHER_RULES`, a file with uORB publication rules can be used to configure which modules are allowed to publish which topics. This is used for system-wide replay.

Examples

Monitor topic publication rates. Besides `top`, this is an important command for general system inspection:

```
uorb top
```

Usage

```
uorb <command> [arguments...]
```

Commands:

```
start
```

```
status          Print topic statistics
```

```
top          Monitor topic publication rates
[-a]        print all instead of only currently publishing topics
[<filter1> [<filter2>]] topic(s) to match (implies -a)
```

7.3 Modules Reference: Driver

fmw

Source: [drivers/px4fmw](#)

Description

This module is responsible for driving the output and reading the input pins. For boards without a separate IO chip (eg. Pixracer), it uses the main channels. On boards with an IO chip (eg. Pixhawk), it uses the AUX channels, and the px4io driver is used for main ones.

It listens on the actuator_controls topics, does the mixing and writes the PWM outputs. In addition it does the RC input parsing and auto-selecting the method. Supported methods are:

- PPM
- SBUS
- DSM
- SUMD
- ST24

The module is configured via mode_* commands. This defines which of the first N pins the driver should occupy. By using mode_pwm4 for example, pins 5 and 6 can be used by the camera trigger driver or by a PWM rangefinder driver. Alternatively, the fmw can be started in one of the capture modes, and then drivers can register a capture callback with ioctl calls.

Implementation

By default the module runs on the work queue, to reduce RAM usage. It can also be run in its own thread, specified via start flag -t, to reduce latency. When running on the work queue, it schedules at a fixed frequency, and the pwm rate limits the update rate of the actuator_controls topics. In case of running in its own thread, the module polls on the actuator_controls topic. Additionally the pwm rate defines the lower-level IO timer rates.

Examples

It is typically started with:

```
fmu mode_pwm
```

To drive all available pins.

Capture input (rising and falling edges) and print on the console: start the fmu in one of the capture modes:

```
fmu mode_pwm3cap1
```

This will enable capturing on the 4th pin. Then do:

```
fmu test
```

Use the `pwm` command for further configurations (PWM rate, levels, ...), and the `mixer` command to load mixer files.

Usage

```
fmu <command> [arguments...]
```

Commands:

<code>start</code>	Start the task (without any mode set, use any of the mode_* cmds)
<code>[-t]</code>	Run as separate task instead of the work queue

All of the mode_* commands will start the fmu if not running already

```
mode_gpio
```

```
mode_rcin    Only do RC input, no PWM outputs
```

```
mode_pwm     Select all available pins as PWM
```

```
mode_pwm1
```

```
mode_pwm4
```

```
mode_pwm2
```

```
mode_pwm3
```

```
mode_pwm3cap1
```

```
mode_pwm2cap2
```

```
mode_pwm6
```

```
bind          Send a DSM bind command (module must be running)

sensor_reset  Do a sensor reset (SPI bus)
  [<ms>]      Delay time in ms between reset and re-enabling

peripheral_reset  Reset board peripherals
  [<ms>]      Delay time in ms between reset and re-enabling

i2c           Configure I2C clock rate
  <bus_id> <rate> Specify the bus id (>=0) and rate in Hz

test          Test inputs and outputs

fake          Arm and send an actuator controls command
  <roll> <pitch> <yaw> <thrust> Control values in range [-100, 100]

stop

status        print status info
```

gpio_led

Source: [modules/gpio_led](#)

Description

This module is responsible for driving a single LED on one of the FMU AUX pins.

It listens on the `vehicle_status` and `battery_status` topics and provides visual annunciation on the LED.

Implementation

The module runs on the work queue. It schedules at a fixed frequency or 5 Hz

Examples

It is started with:

```
gpio_led start
```

To drive an LED connected AUX1 pin.

OR with any of the available AUX pins

```
gpio_led start -p 5
```

To drive an LED connected AUX5 pin.

Usage

```
gpio_led <command> [arguments...]
```

Commands:

```
start          annunciation on AUX OUT pin
  [-p]         Use specified AUX OUT pin number (default: 1)

stop
```

gps

Source: [drivers/gps](#)

Description

GPS driver module that handles the communication with the device and publishes the position via uORB. It supports multiple protocols (device vendors) and by default automatically selects the correct one.

The module supports a secondary GPS device, specified via `-e` parameter. The position will be published on the second uORB topic instance, but it's currently not used by the rest of the system (however the data will be logged, so that it can be used for comparisons).

Implementation

There is a thread for each device polling for data. The GPS protocol classes are implemented with callbacks so that they can be used in other projects as well (eg. QGroundControl uses them too).

Examples

For testing it can be useful to fake a GPS signal (it will signal the system that it has a valid position):

```
gps stop
gps start -f
```

Usage

```
gps <command> [arguments...]
```

Commands:

```
start
```

```
[-d <val>] GPS device
           values: <file:dev>, default: /dev/ttyS3
[-e <val>] Optional secondary GPS device
           values: <file:dev>
[-f]      Fake a GPS signal (useful for testing)
[-s]      Enable publication of satellite info
[-i <val>] GPS interface
           values: spi|uart, default: uart
[-p <val>] GPS Protocol (default=auto select)
           values: ubx|mtk|ash
```

```
stop
```

```
status      print status info
```

vmount

Source: [drivers/vmount](#)

Description

Mount (Gimbal) control driver. It maps several different input methods (eg. RC or MAVLink) to a configured output (eg. AUX channels or MAVLink).

Documentation how to use it is on the [gimbal control](#) page.

Implementation

Each method is implemented in its own class, and there is a common base class for inputs and outputs. They are connected via an API, defined by the `ControlData` data structure. This makes sure that each input method can be used with each output method and new inputs/outputs can be added with minimal effort.

Examples

Test the output by setting a fixed yaw angle (and the other axes to 0):

```
vmount stop
vmount test yaw 30
```

Usage

```
vmount <command> [arguments...]
```

Commands:

```
start
```

```
test          Test the output: set a fixed angle for one axis (vmount must
              not be running)
              roll|pitch|yaw <angle> Specify an axis and an angle in degrees

stop

status       print status info
```

7.4 Modules Reference: Estimator

ekf2

Source: [modules/ekf2](#)

Description

Attitude and position estimator using an Extended Kalman Filter. It is used for Multirotors and Fixed-Wing.

The documentation can be found on the [tuning the ecl_ekf](#) page.

ekf2 can be started in replay mode (-r): in this mode it does not access the system time, but only uses the timestamps from the sensor topics.

Usage

```
ekf2 <command> [arguments...]
Commands:
  start
    [-r]          Enable replay mode

  stop

  status         print status info
```

7.5 Modules Reference: Controller

fw_att_control

Source: [modules/fw_att_control](#)

Description

fw_att_control is the fixed wing attitude controller.

Usage

```
fw_att_control <command> [arguments...]  
Commands:  
  
    stop  
  
    status          print status info
```

fw_pos_control_l1

Source: [modules/fw_pos_control_l1](#)

Description

fw_pos_control_l1 is the fixed wing position controller.

Usage

```
fw_pos_control_l1 <command> [arguments...]  
Commands:  
  
    stop  
  
    status          print status info
```

7.6 Modules Reference: System

dataman

Source: [modules/dataman](#)

Description

Module to provide persistent storage for the rest of the system in form of a simple database through a C API. Multiple backends are supported:

- a file (eg. on the SD card)
- FLASH (if the board supports it)
- FRAM
- RAM (this is obviously not persistent)

It is used to store structured data of different types: mission waypoints, mission state and geofence polygons. Each type has a specific type and a fixed maximum amount of storage items, so that fast random access is possible.

Implementation

Reading and writing a single item is always atomic. If multiple items need to be read/modified atomically, there is an additional lock per item type via `dm_lock`.

DM_KEY_FENCE_POINTS and **DM_KEY_SAFE_POINTS** items: the first data element is a `mission_stats_entry_s` struct, which stores the number of items for these types.

These items are always updated atomically in one transaction (from the mavlink mission manager). During that time, navigator will try to acquire the geofence item lock, fail, and will not check for geofence violations.

Usage

```
dataman <command> [arguments...]
```

Commands:

start

[-f <val>] Storage file

values: <file>

[-r] Use RAM backend (NOT persistent)

[-i] Use FLASH backend

The options `-f`, `-r` and `-i` are mutually exclusive. If nothing is specified, a

```
file 'dataman' is used

poweronrestart Restart dataman (on power on)

inflightrestart Restart dataman (in flight)

stop

status          print status info
```

land_detector

Source: [modules/land_detector](#)

Description

Module to detect the freefall and landed state of the vehicle, and publishing the `vehicle_land_detected` topic. Each vehicle type (multirotor, fixedwing, vtol, ...) provides its own algorithm, taking into account various states, such as commanded thrust, arming state and vehicle motion.

Implementation

Every type is implemented in its own class with a common base class. The base class maintains a state (landed, maybe_landed, ground_contact). Each possible state is implemented in the derived classes. A hysteresis and a fixed priority of each internal state determines the actual `land_detector` state.

Multicopter Land Detector

ground_contact: thrust setpoint and velocity in z-direction must be below a defined threshold for time `GROUND_CONTACT_TRIGGER_TIME_US`. When `ground_contact` is detected, the position controller turns off the thrust setpoint in body x and y.

maybe_landed: it requires `ground_contact` together with a tighter thrust setpoint threshold and no velocity in the horizontal direction. The trigger time is defined by `MAYBE_LAND_TRIGGER_TIME`. When `maybe_landed` is detected, the position controller sets the thrust setpoint to zero.

landed: it requires `maybe_landed` to be true for time `LAND_DETECTOR_TRIGGER_TIME_US`.

The module runs periodically on the HP work queue.

Usage

```
land_detector <command> [arguments...]  
Commands:  
  start          Start the background task  
    fixedwing|multicopter|vtol|ugv Select vehicle type  
  
  stop  
  
  status        print status info
```

load_mon

Source: [modules/load_mon](#)

Description

Background process running periodically with 1 Hz on the LP work queue to calculate the CPU load and RAM usage and publish the `cpuload` topic. On NuttX it also checks the stack usage of each process and if it falls below 300 bytes, a warning is output, which will also appear in the log file.

Usage

```
load_mon <command> [arguments...]  
Commands:  
  start          Start the background task  
  
  stop  
  
  status        print status info
```

logger

Source: [modules/logger](#)

Description

System logger which logs a configurable set of uORB topics and system printf messages (`PX4_WARN` and `PX4_ERR`) to ULog files. These can be used for system and flight performance evaluation, tuning, replay and crash analysis.

It supports 2 backends:

- Files: write ULog files to the file system (SD card)
- MAVLink: stream ULog data via MAVLink to a client (the client must support this)

Both backends can be enabled and used at the same time.

Implementation

The implementation uses two threads:

- The main thread, running at a fixed rate (or polling on a topic if started with `-p`) and checking for data updates
- The writer thread, writing data to the file

In between there is a write buffer with configurable size. It should be large to avoid dropouts.

Examples

Typical usage to start logging immediately:

```
logger start -e -t
```

Or if already running:

```
logger on
```

Usage

```
logger <command> [arguments...]
```

Commands:

start

[-m <val>] Backend mode

values: file|mavlink|all, default: all

[-e] Enable logging right after start until disarm (otherwise only when armed)

[-f] Log until shutdown (implies -e)

[-t] Use date/time for naming log directories and files

```
[-r <val>] Log rate in Hz, 0 means unlimited rate
            default: 280
[-b <val>] Log buffer size in KiB
            default: 12
[-q <val>] uORB queue size for mavlink mode
            default: 14
[-p <val>] Poll on a topic instead of running with fixed rate (Log rate
            and topic intervals are ignored if this is set)
            values: <topic_name>

on          start logging now, override arming (logger must be running)

off         stop logging now, override arming (logger must be running)

stop

status      print status info
```

module

Source: [templates/module](#)

Description

Section that describes the provided module functionality.

This is a template for a module running as a task in the background with start/stop/status functionality.

Implementation

Section describing the high-level implementation of this module.

Examples

CLI usage example:

```
module start -f -p 42
```

Usage

```
module <command> [arguments...]
```

```
Commands:
  start
    [-f]          Optional example flag
    [-p <val>]   Optional example parameter
                  default: 0

  stop

  status          print status info
```

replay

Source: [modules/replay](#)

Description

This module is used to replay ULog files.

There are 2 environment variables used for configuration: `replay`, which must be set to an ULog file name - it's the log file to be replayed. The second is the mode, specified via `replay_mode`:

- `replay_mode=ekf2`: specific EKF2 replay mode. It can only be used with the ekf2 module, but allows the replay to run as fast as possible.
- Generic otherwise: this can be used to replay any module(s), but the replay will be done with the same speed as the log was recorded.

The module is typically used together with uORB publisher rules, to specify which messages should be replayed. The replay module will just publish all messages that are found in the log. It also applies the parameters from the log.

The replay procedure is documented on the [System-wide Replay](#) page.

Usage

```
replay <command> [arguments...]
```

```
Commands:
  start          Start replay, using log file from ENV variable 'replay'

  trystart       Same as 'start', but silently exit if no log file given

  tryapplyparams Try to apply the parameters from the log file
```

```
stop
```

```
status          print status info
```

send_event

Source: [modules/events](#)

Description

Background process running periodically on the LP work queue to perform housekeeping tasks. It is currently only responsible for temperature calibration.

The tasks can be started via CLI or uORB topics (vehicle_command from MAVLink, etc.).

Usage

```
send_event <command> [arguments...]
```

Commands:

```
start          Start the background task
```

```
temperature_calibration Run temperature calibration process
```

```
[-g]          calibrate the gyro
```

```
[-a]          calibrate the accel
```

```
[-b]          calibrate the baro (if none of these is given, all will be  
              calibrated)
```

```
stop
```

```
status          print status info
```

sensors

Source: [modules/sensors](#)

Description

The sensors module is central to the whole system. It takes low-level output from drivers, turns it into a more usable form, and publishes it for the rest of the system.

The provided functionality includes:

- Read the output from the sensor drivers (`sensor_gyro`, etc.). If there are multiple of the same type, do voting and failover handling. Then apply the board rotation and temperature calibration (if enabled). And finally publish the data; one of the topics is `sensor_combined`, used by many parts of the system.
- Do RC channel mapping: read the raw input channels (`input_rc`), then apply the calibration, map the RC channels to the configured channels & mode switches, low-pass filter, and then publish as `rc_channels` and `manual_control_setpoint`.
- Read the output from the ADC driver (via `ioctl` interface) and publish `battery_status`.
- Make sure the sensor drivers get the updated calibration parameters (scale & offset) when the parameters change or on startup. The sensor drivers use the `ioctl` interface for parameter updates. For this to work properly, the sensor drivers must already be running when `sensors` is started.
- Do preflight sensor consistency checks and publish the `sensor_preflight` topic.

Implementation

It runs in its own thread and polls on the currently selected gyro topic.

Usage

```
sensors <command> [arguments...]  
Commands:  
  start  
    [-h]          Start in HIL mode  
  
  stop  
  
  status         print status info
```

8. Robotics

Robotics APIs allow you to control PX4 from outside the flight stack computing environment (flight controller) using a [companion computer](#) or other computing environment. The APIs communicate with PX4 using [MAVLink](#) or [RTPS](#).

PX4 can be used with robotics APIs including [DroneCore](#) and [ROS](#). [DroneKit](#) can also be used, but is not optimised for use with PX4.

8.1 Offboard Control

[Offboard control](#) is dangerous. It is the responsibility of the developer to ensure adequate preparation, testing and safety precautions are taken before offboard flights.

The idea behind off-board control is to be able to control the PX4 flight stack using software running outside of the autopilot. This is done through the Mavlink protocol, specifically the [SET_POSITION_TARGET_LOCAL_NED](#) and the [SET_ATTITUDE_TARGET](#) messages.

Offboard Control Firmware Setup

There are two things you want to setup on the firmware side before starting offboard development.

Map an RC switch to offboard mode activation

To do this, load up the parameters in *QGroundControl* and look for the `RC_MAP_OFFB_SW` parameter to which you can assign the RC channel you want to use to activate offboard mode. It can be useful to map things in such a way that when you fall out of offboard mode you go into position control.

Although this step isn't mandatory since you can activate offboard mode using a MAVLink message. We consider this method much safer.

Enable the companion computer interface

Look for the `SYS_COMPANION` parameter and set it to either 921600 (Recommended) or 57600. This parameter will activate a MAVLink stream on the Telem2 port with data streams specific to onboard mode with the appropriate baud rate (921600 8N1 or 57600 8N1).

For more information on these data streams, look for "MAVLINK_MODE_ONBOARD" in the [source code](#).

Hardware setup

Usually, there are three ways of setting up offboard communication.

Serial radios

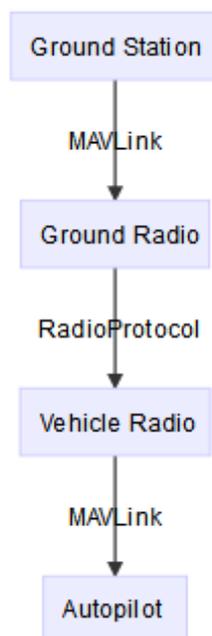
One connected to a UART port of the autopilot

One connected to a ground station computer

Example radios include:

[Lairdtech RM024](#)

[Digi International XBee Pro](#)



On-board processor

A small computer mounted onto the vehicle connected to the autopilot through a UART to USB adapter. There are many possibilities here and it will depend on what kind of additional on-board processing you want to do in addition to sending commands to the autopilot.

Small low power examples:

[Odroid C1+](#) or [Odroid XU4](#)

Raspberry Pi

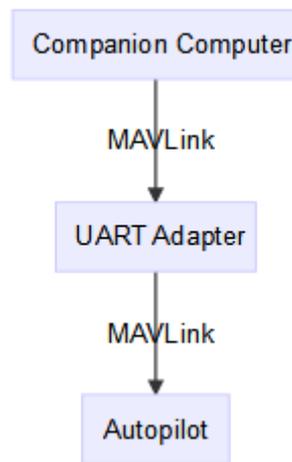
Intel Edison

Larger high power examples:

Intel NUC

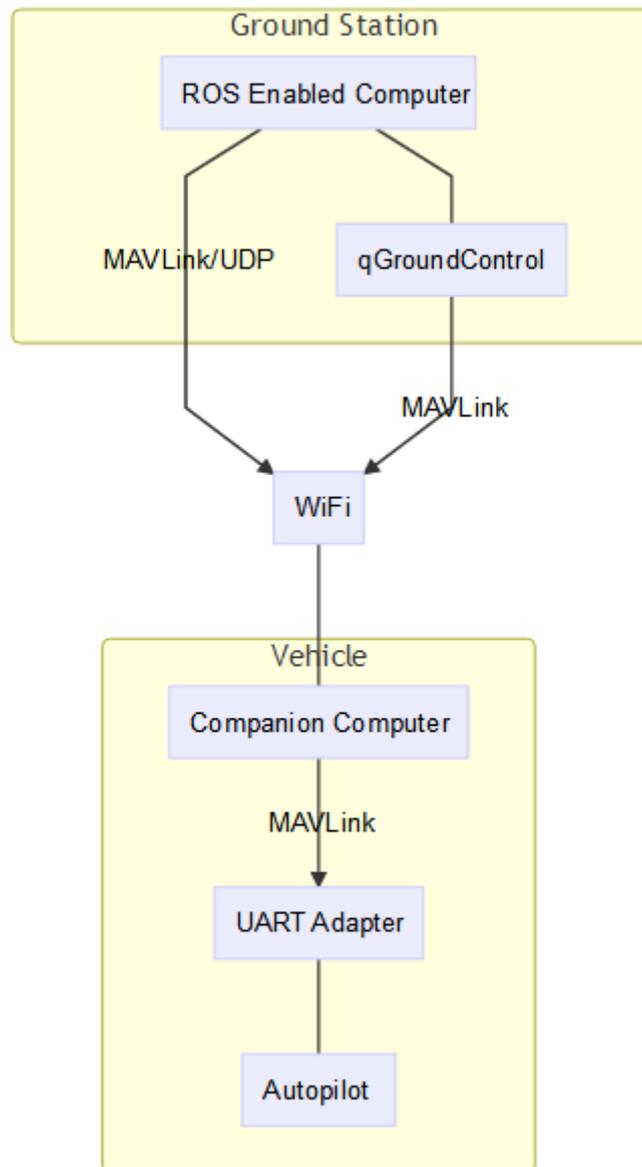
Gigabyte Brix

Nvidia Jetson TK1



On-board processor and wifi link to ROS (*Recommended*)

A small computer mounted onto the vehicle connected to the autopilot through a UART to USB adapter while also having a WiFi link to a ground station running ROS. This can be any of the computers from the above section coupled with a WiFi adapter. For example, the Intel NUC D34010WYB has a PCI Express Half-Mini connector which can accommodate an [Intel Wifi Link 5000](#) adapter.



8.2 Robotics using ROS

[ROS](#) (Robot Operating System) is a general purpose robotics library that can be used with PX4 for [offboard control](#). It uses the [MAVROS](#) node to communicate with PX4 running on hardware or using the [Gazebo Simulator](#).

This section contains topics about using ROS for offboard control with PX4.

ROS is only officially supported on Linux platforms.

8.2.1 MAVROS

The [mavros](#) ROS package enables MAVLink extendable communication between computers running ROS, MAVLink enabled autopilots, and MAVLink enabled GCS.

MAVROS is the "official" supported bridge between ROS and the MAVLink protocol. It is currently being extended to enable [fast-RTPS messaging](#), including a layer to translate PX4 [uORB messages](#) to common ROS idioms.

While MAVROS can be used to communicate with any MAVLink enabled autopilot this documentation will be in the context of enabling communication between the PX4 flight stack and a ROS enabled companion computer.

Installation

MAVROS can be installed either from source or binary. Developers working with ROS are advised to use the source installation.

Binary installation (Debian / Ubuntu)

Since v0.5 that programs available in precompiled debian packages for x86 and amd64 (x86_64). Also v0.9+ exists in ARMv7 repo for Ubuntu armhf. Just use `apt-get` for installation:

```
$ sudo apt-get install ros-indigo-mavros ros-indigo-mavros-extras
```

Source installation

Dependencies

This installation assumes you have a catkin workspace located at `~/catkin_ws` If you don't create one with:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws
$ catkin init
```

You will be using the ROS python tools `wstool`, `rosinstall`, and `catkin_tools` for this installation. While they may have been installed during your installation of ROS you can also install them with:

```
$ sudo apt-get install python-wstool python-rosinstall-generator python-catkin-tools
```

Note that while the package can be built using `catkin_make` the preferred method is using `catkin_tools` as it is a more versatile and "friendly" build tool.

If this is your first time using `wstool` you will need to initialize your source space with:

```
$ wstool init ~/catkin_ws/src
```

Now you are ready to do the build

```

# 1. get source (upstream - released)
$ roscpp_generator --upstream mavros | tee /tmp/mavros.rosinstall
# alternative: latest source
$ roscpp_generator --upstream-development mavros | tee /tmp/mavros.rosinstall

# 2. get latest released mavlink package
# you may run from this line to update ros-*-mavlink package
$ roscpp_generator mavlink | tee -a /tmp/mavros.rosinstall

# 3. Setup workspace & install deps
$ wstool merge -t src /tmp/mavros.rosinstall
$ wstool update -t src
$ rosdep install --from-paths src --ignore-src --rosdistro `echo $ROS_DISTRO` -y

# finally - build
$ catkin build

```

If you are installing mavros on a raspberry pi, you may get an error related to your os, when running "rosdep install ...". Add "--os=OS_NAME:OS_VERSION " to the rosdep command and replace OS_NAME with your OS name and OS_VERSION with your OS version (e.g. --os=debian:jessie).

8.2.2 MAVROS *Offboard* control example

Offboard control is dangerous. If you are operating on a real vehicle be sure to have a way of gaining back manual control in case something goes wrong.

The following tutorial will run through the basics of *Offboard* control through MAVROS as applied to an Iris quadcopter simulated in Gazebo with SITL running. At the end of the tutorial, you should see the same behaviour as in the video below, i.e. a slow takeoff to an altitude of 2 meters.

Code

Create the `offb_node.cpp` file in your ROS package (by also adding it to your `CMakeList.txt` so it is compiled), and paste the following inside it:

```

/**
 * @file offb_node.cpp
 * @brief Offboard control example node, written with MAVROS version 0.19.x, PX4 Pro Flight
 * Stack and tested in Gazebo SITL
 */

```

```

#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>

mavros_msgs::State current_state;
void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "offb_node");
    ros::NodeHandle nh;

    ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>
        ("mavros/state", 10, state_cb);
    ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>
        ("mavros/setpoint_position/local", 10);
    ros::ServiceClient arming_client = nh.serviceClient<mavros_msgs::CommandBool>
        ("mavros/cmd/arming");
    ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::SetMode>
        ("mavros/set_mode");

    //the setpoint publishing rate MUST be faster than 2Hz
    ros::Rate rate(20.0);

    // wait for FCU connection
    while(ros::ok() && !current_state.connected){
        ros::spinOnce();
        rate.sleep();
    }

    geometry_msgs::PoseStamped pose;
    pose.pose.position.x = 0;
    pose.pose.position.y = 0;
    pose.pose.position.z = 2;

    //send a few setpoints before starting
    for(int i = 100; ros::ok() && i > 0; --i){
        local_pos_pub.publish(pose);
        ros::spinOnce();
        rate.sleep();
    }
}

```

```

}

mavros_msgs::SetMode offb_set_mode;
offb_set_mode.request.custom_mode = "OFFBOARD";

mavros_msgs::CommandBool arm_cmd;
arm_cmd.request.value = true;

ros::Time last_request = ros::Time::now();

while(ros::ok()){
    if( current_state.mode != "OFFBOARD" &&
        (ros::Time::now() - last_request > ros::Duration(5.0))){
        if( set_mode_client.call(offb_set_mode) &&
            offb_set_mode.response.mode_sent){
            ROS_INFO("Offboard enabled");
        }
        last_request = ros::Time::now();
    } else {
        if( !current_state.armed &&
            (ros::Time::now() - last_request > ros::Duration(5.0))){
            if( arming_client.call(arm_cmd) &&
                arm_cmd.response.success){
                ROS_INFO("Vehicle armed");
            }
            last_request = ros::Time::now();
        }
    }

    local_pos_pub.publish(pose);

    ros::spinOnce();
    rate.sleep();
}

return 0;
}

```

Code explanation

```
#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>
```

The `mavros_msgs` package contains all of the custom messages required to operate services and topics provided by the MAVROS package. All services and topics as well as their corresponding message types are documented in the [mavros wiki](#).

```
mavros_msgs::State current_state;
void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}
```

We create a simple callback which will save the current state of the autopilot. This will allow us to check connection, arming and *Offboard* flags.

```
ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>("mavros/state", 10, state_cb);
ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>("mavros/setpoint_position/local",
10);
ros::ServiceClient arming_client = nh.serviceClient<mavros_msgs::CommandBool>("mavros/cmd/arming");
ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::SetMode>("mavros/set_mode");
```

We instantiate a publisher to publish the commanded local position and the appropriate clients to request arming and mode change. Note that for your own system, the "mavros" prefix might be different as it will depend on the name given to the node in its launch file.

```
//the setpoint publishing rate MUST be faster than 2Hz
ros::Rate rate(20.0);
```

The px4 flight stack has a timeout of 500ms between two *Offboard* commands. If this timeout is exceeded, the commander will fall back to the last mode the vehicle was in before entering *Offboard* mode. This is why the publishing rate **must** be faster than 2 Hz to also account for possible latencies. This is also the same reason why it is recommended to enter *Offboard* mode from *Position* mode, this way if the vehicle drops out of *Offboard* mode it will stop in its tracks and hover.

```
// wait for FCU connection
while(ros::ok() && !current_state.connected){
    ros::spinOnce();
    rate.sleep();
}
```

Before publishing anything, we wait for the connection to be established between MAVROS and the autopilot. This loop should exit as soon as a heartbeat message is received.

```
geometry_msgs::PoseStamped pose;
pose.pose.position.x = 0;
pose.pose.position.y = 0;
pose.pose.position.z = 2;
```

Even though the PX4 Pro Flight Stack operates in the aerospace NED coordinate frame, MAVROS translates these coordinates to the standard ENU frame and vice-versa. This is why we set *z* to positive 2.

```
//send a few setpoints before starting
for(int i = 100; ros::ok() && i > 0; --i){
    local_pos_pub.publish(pose);
    ros::spinOnce();
    rate.sleep();
}
```

Before entering *Offboard* mode, you must have already started streaming setpoints. Otherwise the mode switch will be rejected. Here, 100 was chosen as an arbitrary amount.

```
mavros_msgs::SetMode offb_set_mode;
offb_set_mode.request.custom_mode = "OFFBOARD";
```

We set the custom mode to OFFBOARD. A list of [supported modes](#) is available for reference.

```
mavros_msgs::CommandBool arm_cmd;
arm_cmd.request.value = true;

ros::Time last_request = ros::Time::now();

while(ros::ok()){
    if( current_state.mode != "OFFBOARD" &&
        (ros::Time::now() - last_request > ros::Duration(5.0))){
        if( set_mode_client.call(offb_set_mode) &&
            offb_set_mode.response.mode_sent){
            ROS_INFO("Offboard enabled");
        }
        last_request = ros::Time::now();
    } else {
        if( !current_state.armed &&
            (ros::Time::now() - last_request > ros::Duration(5.0))){
            if( arming_client.call(arm_cmd) &&
                arm_cmd.response.success){
                ROS_INFO("Vehicle armed");
            }
        }
    }
}
```

```

        }
        last_request = ros::Time::now();
    }
}

local_pos_pub.publish(pose);

ros::spinOnce();
rate.sleep();
}

```

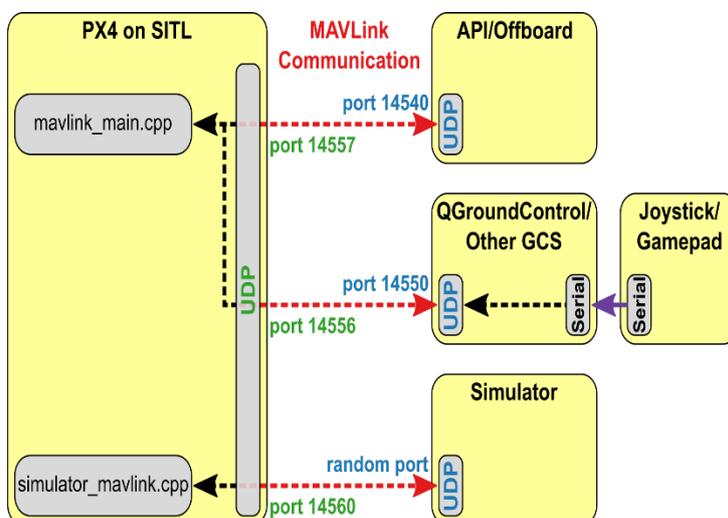
The rest of the code is pretty self explanatory. We attempt to switch to *Offboard* mode, after which we arm the quad to allow it to fly. We space out the service calls by 5 seconds so to not flood the autopilot with the requests. In the same loop, we continue sending the requested pose at the appropriate rate.

This code has been simplified to the bare minimum for illustration purposes. In larger systems, it is often useful to create a new thread which will be in charge of periodically publishing the setpoints.

8.2.3 ROS with Gazebo Simulation

ROS (Robot Operating System) can be used with PX4 and the [Gazebo simulator](#). It uses the [MAVROS](#) MAVLink node to communicate with PX4.

The ROS/Gazebo integration with PX4 follows the pattern in the diagram below (this shows the *generic PX4 simulation environment*). PX4 communicates with the simulator (e.g. Gazebo) to receive sensor data from the simulated world and send motor and actuator values. It communicates with the GCS and an Offboard API (e.g. ROS) to send telemetry from the simulated environment and receive commands.



The only *slight* difference to "normal behaviour" is that ROS initiates the connection on port 14557, while it is more typical for an offboard API to listen for connections on UDP port 14540.

Installing ROS and Gazebo

ROS is only supported on Linux (not macOS or Windows).

The easiest way to setup PX4 simulation with ROS on Ubuntu Linux is to use the standard installation script that can be found at [Development Environment on Linux > Gazebo with ROS](#). The script installs everything you need: PX4, ROS "Kinetic", the Gazebo 7 simulator, and MAVROS.

The script follows the [standard ROS "Kinetic" installation instructions](#), which include Gazebo 7. Installation of ROS Kinetic for other platforms is covered in the [ROS Wiki here](#).

Launching ROS/Simulation

The command below can be used to launch the simulation and connect ROS to it via MAVROS, where `fcu_url` is the IP / port of the computer running the simulation:

```
roslaunch mavros px4.launch fcu_url:="udp://:14540@192.168.1.36:14557"
```

To connect to localhost, use this URL:

```
roslaunch mavros px4.launch fcu_url:="udp://:14540@127.0.0.1:14557"
```

It can be useful to call `roslaunch` with the `-w` (warn) and/or `-v` (verbose) in order to get warnings about missing dependencies in your setup. For example:

```
roslaunch mavros px4.launch fcu_url:="udp://:14540@127.0.0.1:14557"
```

Launching Gazebo with ROS Wrappers

The Gazebo simulation can be modified to integrate sensors publishing directly to ROS topics e.g. the Gazebo ROS laser plugin. To support this feature, Gazebo must be launched with the appropriate ROS wrappers.

There are ROS launch scripts available to run the simulation wrapped in ROS:

- [posix_sitl.launch](#): plain SITL launch
- [mavros_posix_sitl.launch](#): SITL and MAVROS

To run SITL wrapped in ROS the ROS environment needs to be updated, then launch as usual:

(optional): only source the catkin workspace if you compiled MAVROS or other ROS packages from source:

```
cd <Firmware_clone>
```

```

make posix_sitl_default gazebo
source ~/catkin_ws/devel/setup.bash // (optional)
source Tools/setup_gazebo.bash $(pwd) $(pwd)/build/posix_sitl_default
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)/Tools/sitl_gazebo
roslaunch px4 posix_sitl.launch

```

Include one of the above mentioned launch files in your own launch file to run your ROS application in the simulation.

What's Happening Behind the Scenes

This section shows how the *roslaunch* instructions provided previously actually work (you can follow them to manually launch the simulation and ROS).

First start the simulator using the command below:

```
no_sim=1 make posix_sitl_default gazebo
```

The console will look like this:

```

[init] shell id: 46979166467136
[init] task name: px4

_____ _ _ _ _
|__\\ \\ // / |
||_// \V/ //||
| _// / \ //_|
|| ^\\ \__ |
\| V V |/

```

Ready to fly.

```

INFO LED::init
729 DevObj::init led
736 Added driver 0x2aba34001080 /dev/led0
INFO LED::init
742 DevObj::init led
INFO Not using /dev/ttyACM0 for radio control input. Assuming joystick input via MAVLink.
INFO Waiting for initial data on UDP. Please start the flight simulator to proceed..

```

Now in a new terminal make sure you will be able to insert the Iris model through the Gazebo menus, to do this set your environment variables to include the appropriate `sitl_gazebo` folders.

```

cd <Firmware_clone>
source Tools/setup_gazebo.bash $(pwd) $(pwd)/build/posix_sitl_default

```

Now start Gazebo like you would when working with ROS and insert the Iris quadcopter model. Once the Iris is loaded it will automatically connect to the px4 app.

```
roslaunch gazebo_ros empty_world.launch world_name:=$(pwd)/Tools/sitl_gazebo/worlds/iris.world
```

8.2.4 OctoMap 3D Models with ROS/Gazebo

The [OctoMap library](#) is an open source library for generating volumetric 3D environment models from sensor data. This model data can then be used by a drone for navigation and obstacle avoidance.

This guide covers how to use *OctoMap* with the Gazebo [Rotors Simulator](#) and ROS.

Installation

The installation requires ROS, Gazebo and the Rotors Simulator plugin. Follow the [Rotors Simulator instructions](#) to install.

Next, install the *OctoMap* library:

```
sudo apt-get install ros-indigo-octomap ros-indigo-octomap-mapping
rosdep install octomap_mapping
rosmake octomap_mapping
```

Now, open `~/catkin_ws/src/rotors_simulator/rotors_gazebo/CMakeLists.txt` and add the following lines to the bottom of the file

```
find_package(octomap REQUIRED)
include_directories(${OCTOMAP_INCLUDE_DIRS})
link_libraries(${OCTOMAP_LIBRARIES})
```

Open `~/catkin_ws/src/rotors_simulator/rotors_gazebo/package.xml` and add the following lines

```
<build_depend>octomap</build_depend>
<run_depend>octomap</run_depend>
```

Run the following two lines:

The first line changes your default shell editor to *gedit*. This is recommended for users who have little experience with *vim* (the default editor), but can otherwise be omitted.

```
export EDITOR='gedit'
roslaunch octomap_server octomap_tracking_server.launch
```

and change the two following lines:

```
<param name="frame_id" type="string" value="map" />
...
<!--remap from="cloud_in" to="/rgbdslam/batch_clouds" /-->
```

to:

```
<param name="frame_id" type="string" value="world" />
...
<remap from="cloud_in" to="/firefly/vi_sensor/camera_depth/depth/points" />
```

Running the Simulation

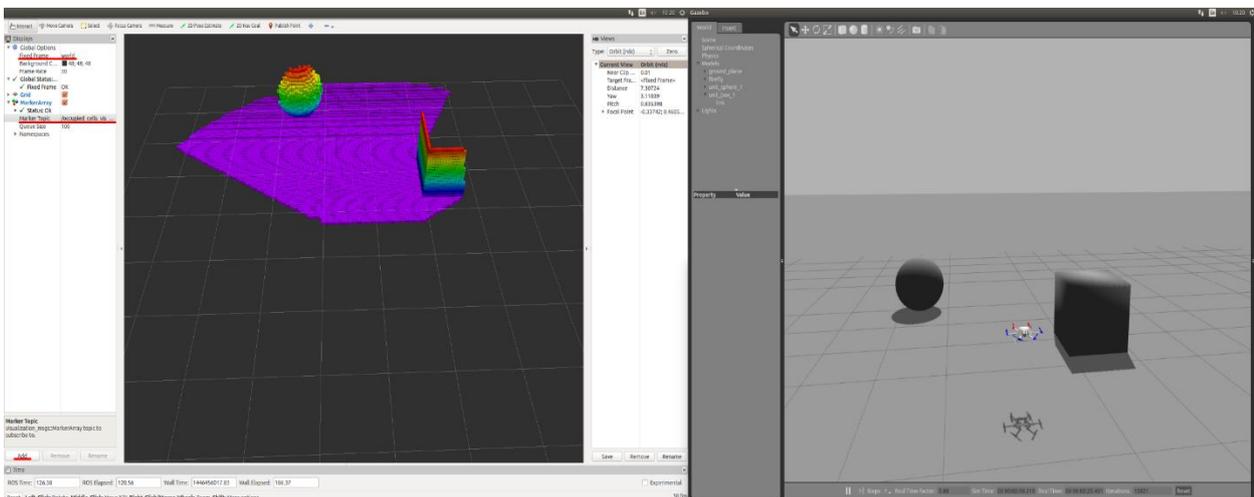
Run the following three lines in *separate* terminal windows. This opens up [Gazebo](#), [Rviz](#) and an octomap server.

```
roslaunch rotors_gazebo mav_hovering_example_with_vi_sensor.launch  mav_name:=firefly
rviz
roslaunch octomap_server octomap_tracking_server.launch
```

In *Rviz*, change the field 'Fixed Frame' from 'map' to 'world' in the top left of the window. Now click the add button in the bottom left and select MarkerArray. Then double click the MarkerArray and change 'Marker Topic' from '/free_cells_vis_array' to '/occupied_cells_vis_array'

Now you should see a part of the floor.

In the *Gazebo* window, insert a cube in front of the red rotors and you should see it in *Rviz*.



8.2.5 Raspberry Pi - ROS installation

This is a guide on how to install ROS-indigo on a Raspberry Pi 2 serving as a companion computer for Pixhawk.

Prerequisites

- A working Raspberry Pi with monitor, keyboard, or configured SSH connection
- This guide assumes that you have Raspbian "JESSIE" installed on your RPi. If not: [install it](#) or [upgrade](#) your Raspbian Wheezy to Jessie.

Installation

Follow [this guide](#) for the actual installation of ROS Indigo. Note: Install the "ROS-Comm" variant. The Desktop variant is too heavyweight.

Errors when installing packages

If you want to download packages (e.g. `sudo apt-get install ros-indigo-ros-tutorials`), you might get an error saying: "unable to locate package ros-indigo-ros-tutorials".

If so, proceed as follows: Go to your catkin workspace (e.g. `~/ros_catkin_ws`) and change the name of the packages.

```
$ cd ~/ros_catkin_ws
```

```
$ rosinstall_generator ros_tutorials --rostdistro indigo --deps --wet-only --exclude roslisp --tar > indigo-custom_ros.rosinstall
```

Next, update your workspace with wstool.

```
$ wstool merge -t src indigo-custom_ros.rosinstall
```

```
$ wstool update -t src
```

Next (still in your workspace folder), source and make your files.

```
$ source /opt/ros/indigo/setup.bash
```

```
$ source devel/setup.bash
```

```
$ catkin_make
```

8.2.6 Using Vision or Motion Capture Systems

Before following the instructions below, ensure that your autopilot has a firmware version with the LPE modules enabled. The LPE version of the PX4 firmware can be found inside the zip file of the latest PX4 release or it can be built from source using a build command such as `make px4fmv-v2_lpe`. See [Building the Code](#) for more details.

This page aims at getting a PX4 based system using position data from sources other than GPS (such as motion capture systems like VICON and Optitrack and vision based estimation systems like [ROVIO](#), [SVO](#) or [PTAM](#))

Position estimates can be sent both from an onboard computer as well as from offboard (example : VICON). This data is used to update its local position estimate relative to the local origin. Heading from the vision/motion capture system can also be optionally integrated by the attitude estimator.

The system can then be used for applications such as position hold indoors or waypoint navigation based on vision.

For vision, the MAVLink message used to send the pose data is [VISION_POSITION_ESTIMATE](#) and the message for all motion capture systems is [ATT_POS_MOCAP](#) messages.

The mavros ROS-MAVLink interface has default implementations to send these messages. They can also be sent using pure C/C++ code and direct use of the MAVLink library. The ROS topics are: `mocap_pose_estimate` for mocap systems and `vision_pose_estimate` for vision.

Check [mavros extras](#) for further info.

This feature has only been tested to work with the LPE estimator.

LPE Tuning for Vision or Mocap

Enabling external pose input

You need to set a few parameters (from QGroundControl or the NSH shell) to enable or disable vision/mocap usage in the system.

Set the system parameter `ATT_EXT_HDG_M` to 1 or 2 to enable external heading integration.

Setting it to 1 will cause vision to be used, while 2 enables mocap heading use.

Vision integration is enabled by default in LPE. You can control this using the `LPE_FUSION` parameter in QGroundControl. Make sure that "fuse vision position" is checked.

Disabling barometer fusion

If a highly accurate altitude is already available from vision or mocap information, it may be useful to disable the baro correction in LPE to reduce drift on the Z axis.

There is a bit field for this in the parameter `LPE_FUSION`, which you can set from QGroundControl. Just uncheck "fuse baro".

Tuning Noise Parameters

If your vision or mocap data is highly accurate, and you just want the estimator to track it tightly, you should reduce the standard deviation parameters, `LPE_VIS_XY` and `LPE_VIS_Z` (for vision) or `LPE_VIC_P` (for motion capture). Reducing them will cause the estimator to trust the incoming pose estimate more. You may need to set them lower than the allowed minimum and force-save.

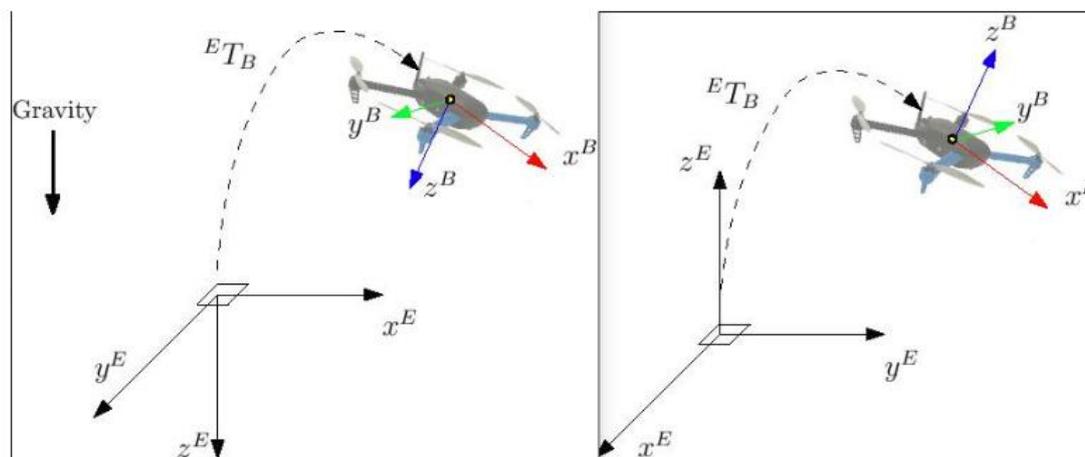
If performance is still poor, try increasing the `LPE_PN_V` parameter. This will cause the estimator to trust measurements more during velocity estimation..

Asserting on Reference Frames

This section shows how to setup the system with the proper reference frames. There are various representations but we will use two of them: ENU and NED.

- ENU has a ground-fixed frame where x axis points East, y points North and z up. Robot frame is x towards the front, z up and y accordingly.
- NED has x towards North, y East and z down. Robot frame is x towards the front, z down and y accordingly.

Frames are shown in the image below: NED on the left while ENU on the right.



With the external heading estimation, however, magnetic North is ignored and faked with a vector corresponding to world x axis (which can be placed freely at mocap calibration); yaw angle will be given with respect to local x .

When creating the rigid body in the mocap software, remember to first align the robot's local x axis with the world x axis otherwise yaw estimation will have an initial offset.

Using Mavros

With MAVROS this operation is straightforward. ROS uses ENU frames as convention, therefore position feedback must be provided in ENU. If you have an Optitrack system you can use [mocap_optitrack](#) node which streams the object pose on a ROS topic already in ENU. With a remapping you can directly publish it on `mocap_pose_estimate` as it is without any transformation and mavros will take care of NED conversions.

Without Mavros If you do not use MAVROS or ROS in general, you need to stream data over MAVLink with `ATT_POS_MOCAP` message. In this case you will need to apply a custom transformation depending on the system in order to obtain NED convention.

Let us take as an example the Optitrack framework; in this case the local frame has x and z on the horizontal plane (x front and z right) while y axis is vertical and pointing up. A simple trick is swapping x and z axis in order to obtain NED convention.

We call $x_{\{mav\}}$, $y_{\{mav\}}$ and $z_{\{mav\}}$ the coordinates that are sent through MAVLink as position feedback, then we obtain:

$$x_{\{mav\}} = x_{\{mocap\}} \quad y_{\{mav\}} = z_{\{mocap\}} \quad z_{\{mav\}} = -y_{\{mocap\}}$$

Regarding the orientation, keep the scalar part w of the quaternion the same and swap the vector part x , y and z in the same way. You can apply this trick with every system; you need to obtain a NED frame, look at your mocap output and swap axis accordingly.

First Flight

At this point, if you followed those steps, you are ready to test your setup.

Be sure to perform the following checks:

- **Before** creating the rigid body, align the robot with world x axis
- Stream over MAVLink and check the MAVLink inspector with QGroundControl, the local pose topic should be in NED
- Move the robot around by hand and see if the estimated local position is consistent (always in NED)
- Rotate the robot on the vertical axis and check the yaw with the MAVLink inspector

If those steps are consistent, you can try your first flight.

Put the robot on the ground and start streaming mocap feedback. Lower your left (throttle) stick and arm the motors.

At this point, with the left stick at the lowest position, switch to position control. You should have a green light. The green light tells you that position feedback is available and position control is now activated.

Put your left stick at the middle, this is the dead zone. With this stick value, the robot maintains its altitude; raising the stick will increase the reference altitude while lowering the value will decrease it. Same for right stick on x and y .

Increase the value of the left stick and the robot will take off, put it back to the middle right after. Check if it is able to keep its position.

If it works, you may want to set up an [offboard](#) experiment by sending position-setpoint from a remote ground station.

8.3 Using DroneKit to communicate with PX4

[DroneKit](#) helps you create powerful apps for UAVs. These apps run on a UAV's Companion Computer, and augment the autopilot by performing tasks that are both computationally intensive and require a low-latency link (e.g. computer vision).

DroneKit and PX4 are currently working on getting full compatibility. As of DroneKit-python 2.2.0 there is basic support for mission handling and vehicle monitoring.

Setting up DroneKit with PX4

Start by installing DroneKit-python from the current master.

```
git clone https://github.com/dronekit/dronekit-python.git
cd ./dronekit-python
sudo python setup.py build
sudo python setup.py install
```

Create a new python file and import DroneKit, pymavlink and basic modules

```
# Import DroneKit-Python
from dronekit import connect, Command, LocationGlobal
from pymavlink import mavutil
import time, sys, argparse, math
```

Connect to a MAVLink port of your drone or simulation (e.g. [JMavSim](#)).

```
# Connect to the Vehicle
print "Connecting"
connection_string = '127.0.0.1:14540'
vehicle = connect(connection_string, wait_ready=True)
```

Display some basic status information

```
# Display basic vehicle state
print " Type: %s" % vehicle._vehicle_type
print " Armed: %s" % vehicle.armed
print " System status: %s" % vehicle.system_status.state
print " GPS: %s" % vehicle.gps_0
print " Alt: %s" % vehicle.location.global_relative_frame.alt
```

Full mission example

The following python script shows a full mission example using DroneKit and PX4. Mode switching is not yet fully supported from DroneKit, we therefor send our own custom mode switching commands.

```
#####  
###  
# @File DroneKitPX4.py  
# Example usage of DroneKit with PX4  
#  
# @author Sander Smeets <sander@droneslab.com>  
#  
# Code partly based on DroneKit (c) Copyright 2015-2016, 3D Robotics.  
#####  
###  
  
# Import DroneKit-Python  
from dronekit import connect, Command, LocationGlobal  
from pymavlink import mavutil  
import time, sys, argparse, math  
  
#####  
###  
# Settings  
#####  
###  
  
connection_string      = '127.0.0.1:14540'  
MAV_MODE_AUTO         = 4  
# https://github.com/PX4/Firmware/blob/master/Tools/mavlink\_px4.py  
  
# Parse connection argument  
parser = argparse.ArgumentParser()  
parser.add_argument("-c", "--connect", help="connection string")  
args = parser.parse_args()  
  
if args.connect:  
    connection_string = args.connect
```

```
#####
###
# Init
#####
###

# Connect to the Vehicle
print "Connecting"
vehicle = connect(connection_string, wait_ready=True)

def PX4setMode(mavMode):
    vehicle._master.mav.command_long_send(vehicle._master.target_system, vehicle._master.target_component,
                                           mavutil.mavlink.MAV_CMD_DO_SET_MODE, 0,
                                           mavMode,
                                           0, 0, 0, 0, 0, 0)

def get_location_offset_meters(original_location, dNorth, dEast, alt):
    """
    Returns a LocationGlobal object containing the latitude/longitude `dNorth` and `dEast` metres from the
    specified `original_location`. The returned Location adds the entered `alt` value to the altitude of the
    `original_location`.

    The function is useful when you want to move the vehicle around specifying locations relative to
    the current vehicle position.

    The algorithm is relatively accurate over small distances (10m within 1km) except close to the poles.
    For more information see:
    http://gis.stackexchange.com/questions/2951/algorithm-for-offsetting-a-latitude-longitude-by-some-amount-
    of-meters
    """
    earth_radius=6378137.0 #Radius of "spherical" earth
    #Coordinate offsets in radians
    dLat = dNorth/earth_radius
    dLon = dEast/(earth_radius*math.cos(math.pi*original_location.lat/180))

    #New position in decimal degrees
    newlat = original_location.lat + (dLat * 180/math.pi)
    newlon = original_location.lon + (dLon * 180/math.pi)
    return LocationGlobal(newlat, newlon,original_location.alt+alt)

```

```

#####
###
# Listeners
#####
###

home_position_set = False

#Create a message listener for home position fix
@vehicle.on_message('HOME_POSITION')
def listener(self, name, home_position):
    global home_position_set
    home_position_set = True

#####
###
# Start mission example
#####
###

# wait for a home position lock
while not home_position_set:
    print "Waiting for home position..."
    time.sleep(1)

# Display basic vehicle state
print " Type: %s" % vehicle._vehicle_type
print " Armed: %s" % vehicle.armed
print " System status: %s" % vehicle.system_status.state
print " GPS: %s" % vehicle.gps_0
print " Alt: %s" % vehicle.location.global_relative_frame.alt

# Change to AUTO mode
PX4setMode(MAV_MODE_AUTO)
time.sleep(1)

# Load commands
cmds = vehicle.commands
cmds.clear()

home = vehicle.location.global_relative_frame

```

```

# takeoff to 10 meters
wp = get_location_offset_meters(home, 0, 0, 10);
cmd = Command(0,0,0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_TAKEOFF, 0, 1, 0, 0, 0, 0, wp.lat, wp.lon, wp.alt)
cmds.add(cmd)

# move 10 meters north
wp = get_location_offset_meters(wp, 10, 0, 0);
cmd = Command(0,0,0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 1, 0, 0, 0, 0, wp.lat, wp.lon, wp.alt)
cmds.add(cmd)

# move 10 meters east
wp = get_location_offset_meters(wp, 0, 10, 0);
cmd = Command(0,0,0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 1, 0, 0, 0, 0, wp.lat, wp.lon, wp.alt)
cmds.add(cmd)

# move 10 meters south
wp = get_location_offset_meters(wp, -10, 0, 0);
cmd = Command(0,0,0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 1, 0, 0, 0, 0, wp.lat, wp.lon, wp.alt)
cmds.add(cmd)

# move 10 meters west
wp = get_location_offset_meters(wp, 0, -10, 0);
cmd = Command(0,0,0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 1, 0, 0, 0, 0, wp.lat, wp.lon, wp.alt)
cmds.add(cmd)

# land
wp = get_location_offset_meters(home, 0, 0, 10);
cmd = Command(0,0,0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_LAND, 0, 1, 0, 0, 0, 0, wp.lat, wp.lon, wp.alt)
cmds.add(cmd)

# Upload mission
cmds.upload()
time.sleep(2)

# Arm vehicle
vehicle.armed = True

# monitor mission execution

```

```
nextwaypoint = vehicle.commands.next
while nextwaypoint < len(vehicle.commands):
    if vehicle.commands.next > nextwaypoint:
        display_seq = vehicle.commands.next+1
        print "Moving to waypoint %s" % display_seq
        nextwaypoint = vehicle.commands.next
    time.sleep(1)

# wait for the vehicle to land
while vehicle.commands.next > 0:
    time.sleep(1)

# Disarm vehicle
vehicle.armed = False
time.sleep(1)

# Close vehicle object before exiting script
vehicle.close()
time.sleep(1)
```

9 Debugging Topics

9.1 Frequently Asked Questions

Build Errors

Flash Overflow

Use the FMUv4 architecture to obtain double the flash. The first available board from this generation is the [Pixracer](#).

The amount of code that can be loaded onto a board is limited by the amount of flash memory it has. When adding additional modules or code its possible that the addition exceeds the flash memory. This will result in a "flash overflow". The upstream version will always build, but depending on what a developer adds it might overflow locally.

```
region `flash' overflowed by 12456 bytes
```

To remedy it, either use more recent hardware or remove modules from the build which are not essential to your use case. The configurations are stored [here](#). To remove a module, just comment it out:

```
#drivers/trone
```

USB Errors

The upload never succeeds

On Ubuntu, uninstall the modem manager:

```
sudo apt-get remove modemmanager
```

9.2 PX4 System Console

The system console allows low-level access to the system, debug output and analysis of the system boot process. The most convenient way to connect it is by using a [Dronecode probe](#), but a plain FTDI cable can be used as well.

System Console vs. Shell

There are multiple shells, but only one console: The system console is the location where all boot output (and applications auto-started on boot) is printed.

- System console (first shell): Hardware serial port
- Additional shells: Pixhawk on USB (e.g. lists as `/dev/tty.usbmodem1` on Mac OS)

USB shell: To just run a few quick commands or test an application connecting to the USB shell is sufficient. The Mavlink shell can be used for this, see below. The hardware serial console is only needed for boot debugging or when USB should be used for MAVLink to connect a [GCS](#).

Snapdragon Flight: Wiring the Console

The developer kit comes with a breakout board with three pins to access the console. Connect the bundled FTDI cable to the header and the breakout board to the expansion connector.

Pixracer / Pixhawk v3: Wiring the Console

Connect the 6-pos JST SH 1:1 cable to the Dronecode probe or connect the individual pins of the cable to a FTDI cable like this:

Pixracer / Pixhawk v3		FTDI	
1	+5V (red)		N/C
2	UART7 Tx	5	FTDI RX (yellow)
3	UART7 Rx	4	FTDI TX (orange)
4	SWDIO		N/C
5	SWCLK		N/C
6	GND	1	FTDI GND (black)

Pixhawk v1: Wiring the Console

The system console can be accessed through the Dronecode probe or an FTDI cable. Both options are explained in the section below.

Connecting via Dronecode Probe

Connect the 6-pos DF13 1:1 cable on the [Dronecode probe](#) to the SERIAL4/5 port of Pixhawk.

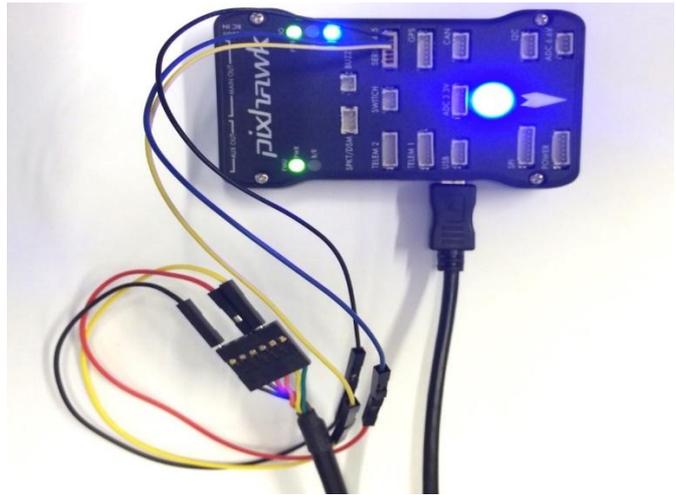
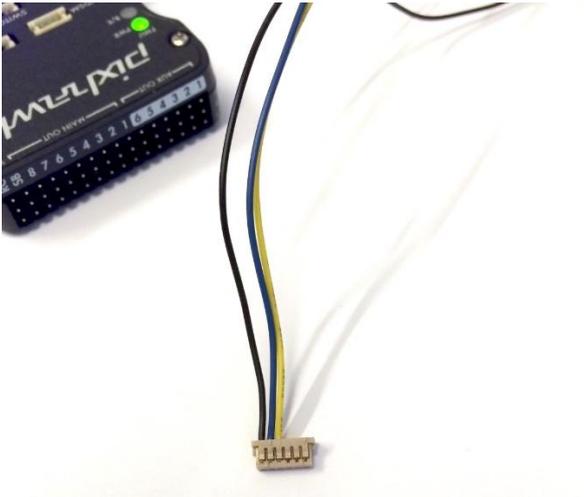


Connecting via FTDI 3.3V Cable

If no Dronecode probe is at hand an FTDI 3.3V (Digi-Key: [768-1015-ND](#)) will do as well.

Pixhawk 1/2		FTDI	
1	+5V (red)		N/C
2	S4 Tx		N/C
3	S4 Rx		N/C
4	S5 Tx	5	FTDI RX (yellow)
5	S5 Rx	4	FTDI TX (orange)
6	GND	1	FTDI GND (black)

The connector pinout is shown in the figure below.



The complete wiring is shown right.

Opening the Console

After the console connection is wired up, use the default serial port tool of your choice or the defaults described below:

Linux / Mac OS: Screen

Install screen on Ubuntu (Mac OS already has it installed):

```
sudo apt-get install screen
```

- Serial: Pixhawk v1 / Pixracer use 57600 baud
- Serial: Snapdragon Flight uses 115200 baud

Connect screen at BAUDRATE baud, 8 data bits, 1 stop bit to the right serial port (use `ls /dev/tty*` and watch what changes when unplugging / replugging the USB device). Common names are `/dev/ttyUSB0` and `/dev/ttyACM0` for Linux and `/dev/tty.usbserial-ABCBD` for Mac OS.

```
screen /dev/ttyXXX BAUDRATE 8N1
```

Windows: PuTTY

Download [PuTTY](#) and start it.

Then select 'serial connection' and set the port parameters to:

- 57600 baud
- 8 data bits
- 1 stop bit

Getting Started on the Console

Type `ls` to view the local file system, type `free` to see the remaining free RAM. The console will also display the system boot log when power-cycling the board.

```
nsh> ls
nsh> free
```

MAVLink Shell

For NuttX-based systems (Pixhawk, Pixracer, ...), the nsh console can also be accessed via mavlink. This works via serial link or WiFi (UDP/TCP). Make sure that QGC is not running, then start the shell with e.g. `./Tools/mavlink_shell.py /dev/ttyACM0` (in the Firmware source). Use `./Tools/mavlink_shell.py -h` to get a description of all available arguments which also displays the IP address of wifi connection. For e.g. `./Tools/mavlink_shell.py <IP address>` can be used to start nsh shell via wifi connection to the autopilot. You can also start nsh shell on QGC directly: **Analyze -> Mavlink Console**. You may first have to install the dependencies with `sudo pip install pymavlink pycserial`.

Snapdragon DSP Console

When you are connected to your Snapdragon board via usb you have access to the px4 shell on the posix side of things. The interaction with the DSP side (QuRT) is enabled with the `qshell` posix app and its QuRT companion.

With the Snapdragon connected via USB, open the mini-dm to see the output of the DSP:

```
/${HEXAGON_SDK_ROOT}/tools/debug/mini-dm/Linux_Debug/mini-dm
```

Note: Alternatively, especially on Mac, you can also use [nano-dm](#).

Run the main app on the linaro side:

```
cd /home/linaro
./px4 px4.config
```

You can now use all apps loaded on the DSP from the linaro shell with the following syntax:

```
pxh> qshell command [args ...]
```

For example, to see the available QuRT apps:

```
pxh> qshell list_tasks
```

The output of the executed command is displayed on the minidm.

9.3 Embedded Debugging

The autopilots running PX4 support debugging via GDB or LLDB.

Identifying large memory consumers

The command below will list the largest static allocations:

```
arm-none-eabi-nm --size-sort --print-size --radix=dec build/px4fmuv2_default/src/firmware/nuttx/firmware_nuttx | grep " [bBdD] "
```

This NSH command provides the remaining free memory:

```
free
```

And the top command shows the stack usage per application:

```
top
```

Stack usage is calculated with stack coloring and thus is not the current usage, but the maximum since the start of the task.

Heap allocations

Dynamic heap allocations can be traced on POSIX in SITL with [gperftools](#).

Install Instructions

Ubuntu:

```
sudo apt-get install google-perftools libgoogle-perftools-dev
```

Start heap profiling

First of all, build the firmware as follows:

```
make posix_sitl_default
```

Start jmavsim: `./Tools/jmavsim_run.sh`

In another terminal, type:

```
cd build/posix_sitl_default/tmp
export HEAPPROFILE=/tmp/heapprofile.hprof
export HEAP_PROFILE_TIME_INTERVAL=30
```

Enter this depending on your system:

Fedora:

```
env LD_PRELOAD=/lib64/libtcmalloc.so ../src/firmware/posix/px4 ../../posix-configs/SITL/init/lpe/iris
pprof --pdf ../src/firmware/posix/px4 /tmp/heapprofile.hprof.0001.heap > heap.pdf
```

Ubuntu:

```
env LD_PRELOAD=/usr/lib/libtcmalloc.so ../src/firmware/posix/px4 ../../posix-  
configs/SITL/init/lpe/iris  
google-pprof --pdf ../src/firmware/posix/px4 /tmp/heapprofile.hprof.0001.heap >  
heap.pdf
```

It will generate a pdf with a graph of the heap allocations. The numbers in the graph will all be zero, because they are in MB. Just look at the percentages instead. They show the live memory (of the node and the subtree), meaning the memory that was still in use at the end.

See the [gperftools docs](#) for more information.

Debugging Hard Faults in NuttX

A hard fault is a state when the operating system detects that it has no valid instructions to execute. This is typically the case when key areas in RAM have been corrupted. A typical scenario is when incorrect memory access smashed the stack and the processor sees that the address in memory is not a valid address for the microprocessors's RAM.

- NuttX maintains two stacks: The IRQ stack for interrupt processing and the user stack
- The stack grows downward. So the highest address in the example below is 0x20021060, the size is 0x11f4 (4596 bytes) and consequently the lowest address is 0x2001fe6c.

```
Assertion failed at file:armv7-m/up_hardfault.c line: 184 task: ekf_att_pos_estimator  
sp:      20003f90  
IRQ stack:  
  base: 20003fdc  
  size: 000002e8  
20003f80: 080d27c6 20003f90 20021060 0809b8d5 080d288c 000000b8 08097155 00000010  
20003fa0: 20003ce0 00000003 00000000 0809bb61 0809bb4d 080a6857 e000ed24 080a3879  
20003fc0: 00000000 2001f578 080ca038 000182b8 20017cc0 0809bad1 20020c14 00000000  
sp:      20020ce8  
User stack:  
  base: 20021060  
  size: 000011f4  
20020ce0: 60000010 2001f578 2001f578 080ca038 000182b8 0808439f 2001fb88 20020d4c  
20020d00: 20020d44 080a1073 666b655b 65686320 205d6b63 6f6c6576 79746963 76696420  
20020d20: 65747265 63202c64 6b636568 63636120 63206c65 69666e6f 08020067 0805c4eb  
20020d40: 080ca9d4 0805c21b 080ca1cc 080ca9d4 385833fb 38217db9 00000000 080ca964  
20020d60: 080ca980 080ca9a0 080ca9bc 080ca9d4 080ca9fc 080caa14 20022824 00000002  
20020d80: 2002218c 0806a30f 08069ab2 81000000 3f7fffec 00000000 3b4ae00c 3b12eaa6  
20020da0: 00000000 00000000 080ca010 4281fb70 20020f78 20017cc0 20020f98 20017cdc  
20020dc0: 2001ee0c 0808d7ff 080ca010 00000000 3f800000 00000000 080ca020 3aa35c4e
```

```

20020de0: 3834d331 00000000 01010101 00000000 01010001 000d4f89 000d4f89 000f9fda
20020e00: 3f7d8df4 3bac67ea 3ca594e6 be0b9299 40b643aa 41ebe4ed bcc04e1b 43e89c96
20020e20: 448f3bc9 c3c50317 b4c8d827 362d3366 b49d74cf ba966159 00000000 00000000
20020e40: 3eb4da7b 3b96b9b7 3eead66a 00000000 00000000 00000000 00000000 00000000
20020e60: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
20020e80: 00000016 00000000 00000000 00010000 00000000 3c23d70a 00000000 00000000
20020ea0: 00000000 20020f78 00000000 2001ed20 20020fa4 2001f498 2001f1a8 2001f500
20020ec0: 2001f520 00000003 2001f170 ffffffff9 3b831ad2 3c23d70a 00000000 00000000
20020ee0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
20020f00: 00000000 00000000 00000000 00000000 2001f4f0 2001f4a0 3d093964 00000001
20020f20: 00000000 0808ae91 20012d10 2001da40 0000260b 2001f577 2001da40 0000260b
20020f40: 2001f1a8 08087fd7 08087f9d 080cf448 0000260b 080afab1 080afa9d 00000003
20020f60: 2001f577 0809c577 2001ed20 2001f4d8 2001f498 0805e077 2001f568 20024540
20020f80: 00000000 00000000 00000000 0000260b 3d093a57 00000000 2001f540 2001f4f0
20020fa0: 0000260b 3ea5b000 3ddb5fa 00000000 3c23d70a 00000000 00000000 000f423f
20020fc0: 00000000 000182b8 20017cc0 2001ed20 2001f4e8 00000000 2001f120 0805ea0d
20020fe0: 2001f090 2001f120 2001eda8 ffffffff 000182b8 00000000 00000000 00000000
20021000: 00000000 00000000 00000009 00000000 08090001 2001f93c 0000000c 00000000
20021020: 00000101 2001f96c 00000000 00000000 00000000 00000000 00000000 00000000
20021040: 00000000 00000000 00000000 00000000 00000000 0809866d 00000000 00000000
R0: 20000f48 0a91ae0c 20020d00 20020d00 2001f578 080ca038 000182b8 20017cc0
R8: 2001ed20 2001f4e8 2001ed20 00000005 20020d20 20020ce8 0808439f 08087c4e
xPSR: 61000000 BASEPRI: 00000000 CONTROL: 00000000
EXC_RETURN: ffffffff9

```

To decode the hard fault, load the *exact* binary into the debugger:

```
arm-none-eabi-gdb build/px4fmu-v2_default/nuttx_px4fmu-v2_default.elf
```

Then in the GDB prompt, start with the last instructions in R8, with the first address in flash (recognizable because it starts with `0x080`, the first is `0x0808439f`). The execution is left to right. So one of the last steps before the hard fault was when `mavlink_log.c` tried to publish something,

```

(gdb) info line *0x0808439f
Line 77 of "../src/modules/systemlib/mavlink_log.c" starts at address 0x8084398
<mavlink_vasprintf+36>
    and ends at 0x80843a0 <mavlink_vasprintf+44>.
(gdb) info line *0x08087c4e
Line 311 of "../src/modules/uORB/uORBDevices_nuttx.cpp"
    starts at address 0x8087c4e <uORB::DeviceNode::publish(orb_metadata const*, void*,
void const*)+2>
    and ends at 0x8087c52 <uORB::DeviceNode::publish(orb_metadata const*, void*, void
const*)+6>.

```

9.4 Sensor/Topic Debugging using the Listener Command

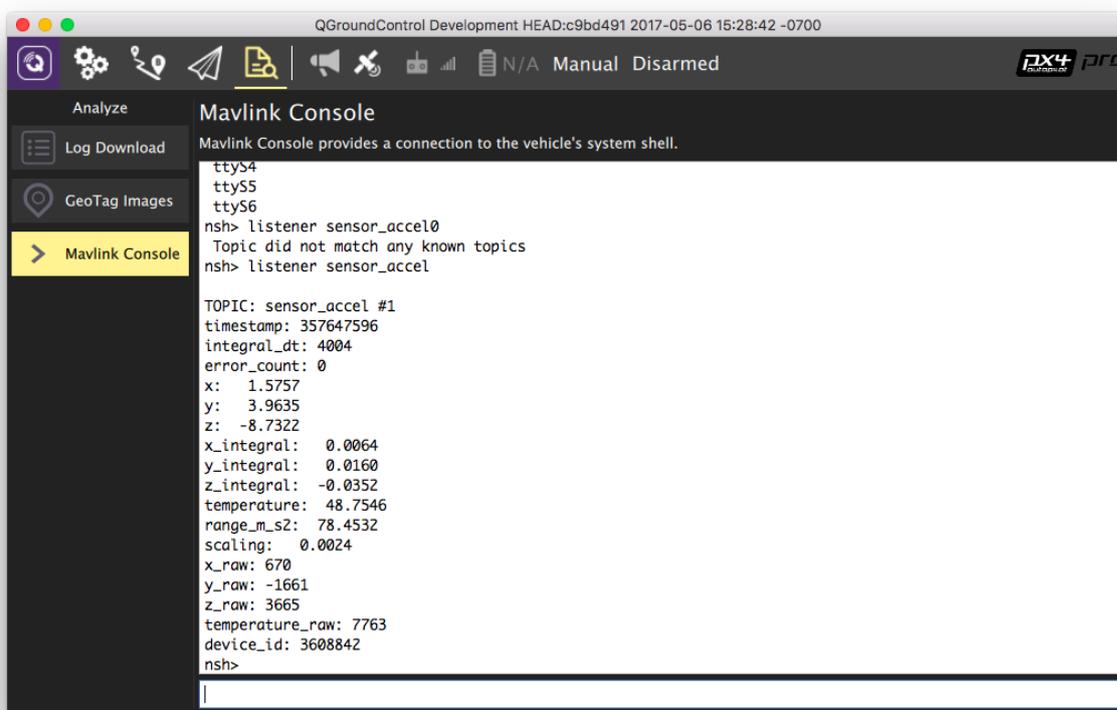
The uORB is an asynchronous `publish()` / `subscribe()` messaging API used for inter-thread/inter-process communication. The `listener` command can be used from the *QGroundControl MAVLink Console* to inspect topic (message) values, including the current values published by sensors.

This is a powerful debugging tool because it can be used even when QGC is connected over a wireless link (e.g. when the vehicle is flying).

The `listener` command is also available through the [System Console](#) and the [MAVLink Shell](#).

The `listener` command is only available on NuttX-based systems (Pixhawk, Pixracer, etc.) and Linux / OS X.

The image below demonstrates *QGroundControl* being used to get the value of the acceleration sensor.



```
QGroundControl Development HEAD:c9bd491 2017-05-06 15:28:42 -0700
Mavlink Console
Mavlink Console provides a connection to the vehicle's system shell.
ttyS4
ttyS5
ttyS6
nsh> listener sensor_accel0
Topic did not match any known topics
nsh> listener sensor_accel

TOPIC: sensor_accel #1
timestamp: 357647596
integral_dt: 4004
error_count: 0
x: 1.5757
y: 3.9635
z: -8.7322
x_integral: 0.0064
y_integral: 0.0160
z_integral: -0.0352
temperature: 48.7546
range_m_s2: 78.4532
scaling: 0.0024
x_raw: 670
y_raw: -1661
z_raw: 3665
temperature_raw: 7763
device_id: 3608842
nsh>
```

For more information about how to determine what topics are available and how to call `listener` see: [uORB Messaging > Listing Topics and Listening in](#).

9.5 Simulation Debugging

As the simulation is running on the host machine, all the desktop development tools are available.

CLANG Address Sanitizer (Mac OS, Linux)

The Clang address sanitizer can help to find alignment (bus) errors and other memory faults like segmentation faults. The command below sets the right compile options.

```
make clean # only required on first address sanitizer run after a normal build
PX4_ASAN=1 make posix jmavsim
```

Valgrind

```
brew install valgrind
```

OR

```
sudo apt-get install valgrind
```

Add instructions how to run Valgrind

Start combinations

SITL can be launched with and without debugger attached and with either jMAVSim or Gazebo as simulation backend. This results in the start options below:

```
make posix_sitl_default jmavsim
make posix_sitl_default jmavsim__gdb
make posix_sitl_default jmavsim__lldb

make posix_sitl_default gazebo
make posix_sitl_default gazebo__gdb
make posix_sitl_default gazebo__lldb

make posix_sitl_lpe jmavsim
make posix_sitl_lpe jmavsim__gdb
make posix_sitl_lpe jmavsim__lldb

make posix_sitl_lpe gazebo
make posix_sitl_lpe gazebo__gdb
make posix_sitl_lpe gazebo__lldb
```

where the last parameter is the <viewer_model_debugger> triplet (using three underscores implies the default 'iris' model). This will start the debugger and launch the SITL application. In order to break into the debugger shell and halt the execution, hit CTRL-C:

```
Process 16529 stopped
* thread #1: tid = 0x114e6d, 0x00007fff90f4430a
libsystem_kernel.dylib`__read_nocancel + 10, name = 'px4', queue = 'com.apple.main-
thread', stop reason = signal SIGSTOP
    frame #0: 0x00007fff90f4430a libsystem_kernel.dylib`__read_nocancel + 10
libsystem_kernel.dylib`__read_nocancel:
-> 0x7fff90f4430a <+10>: jae    0x7fff90f44314                ; <+20>
    0x7fff90f4430c <+12>: movq   %rax, %rdi
    0x7fff90f4430f <+15>: jmp    0x7fff90f3fc53                ; cerror_nocancel
    0x7fff90f44314 <+20>: retq
(lldb)
```

In order to not have the DriverFrameworks scheduling interfere with the debugging session SIGCONT should be masked in LLDB and GDB:

```
(lldb) process handle SIGCONT -n false -p false -s false
```

Or in the case of GDB:

```
(gdb) handle SIGCONT noprint nostop
```

After that the The lldb or gdb shells behave like normal sessions, please refer to the LLDB / GDB documentation.

The last parameter, the <viewer_model_debugger> triplet, is actually passed to make in the build directory, so

```
make posix_sitl_lpe jmavsim__gdb
```

is equivalent with

```
make posix_sitl_lpe    # Configure with cmake
make -C build/posix_sitl_lpe jmavsim__gdb
```

A full list of the available make targets in the build directory can be obtained with:

```
make help
```

but for your convenience, a list with just the <viewer_model_debugger> triplets is printed with the command

```
make list_vmd_make_targets
```

9.6 Compiler optimization

It is possible to suppress compiler optimization for given executables and/or modules (as added by cmake with `add_executable` or `add_library`) when configuring for `posix_sitl_*`. This can be handy when it is necessary to step through code with a debugger or print variables that would otherwise be optimized out.

To do so, set the environment variable `PX4_NO_OPTIMIZATION` to be a semi-colon separated list of regular expressions that match the targets that need to be compiled without optimization. This environment variable is ignored when the configuration isn't `posix_sitl_*`.

For example,

```
export PX4_NO_OPTIMIZATION='px4;^modules__uORB;^modules__systemlib$'
```

would suppress optimization of the targets: `platforms__posix__px4_layer`, `modules__systemlib`, `modules__uORB`, `examples__px4_simple_app`, `modules__uORB__uORB_tests` and `px4`.

The targets that can be matched with these regular expressions can be printed with the command:

```
make -C build/posix_sitl_* list_cmake_targets
```

9.7 Send and Receive Debug Values

It is often necessary during software development to output individual important numbers. This is where the generic `NAMED_VALUE_FLOAT`, `DEBUG` and `DEBUG_VECT` packets of MAVLink come in.

Mapping between MAVLink Debug Messages and uORB Topics

MAVLink debug messages are translated to/from uORB topics. In order to send or receive a MAVLink debug message, you have to respectively publish or subscribe to the corresponding topic. Here is a table that summarizes the mapping between MAVLink debug messages and uORB topics:

MAVLink message	uORB topic
<code>NAMED_VALUE_FLOAT</code>	<code>debug_key_value</code>
<code>DEBUG</code>	<code>debug_value</code>
<code>DEBUG_VECT</code>	<code>debug_vect</code>

Tutorial: Send String / Float Pairs

This tutorial shows how to send the MAVLink message `NAMED_VALUE_FLOAT` using the associated uORB topic `debug_key_value`.

The code for this tutorial is available here:

- [Debug Tutorial Code](#)
- [Enable the tutorial app](#) by uncommenting / enabling the mavlink debug app in the config of your board

All required to set up a debug publication is this code snippet. First add the header file:

```
#include <uORB/uORB.h>
#include <uORB/topics/debug_key_value.h>
```

Then advertise the debug value topic (one advertisement for different published names is sufficient). Put this in front of your main loop:

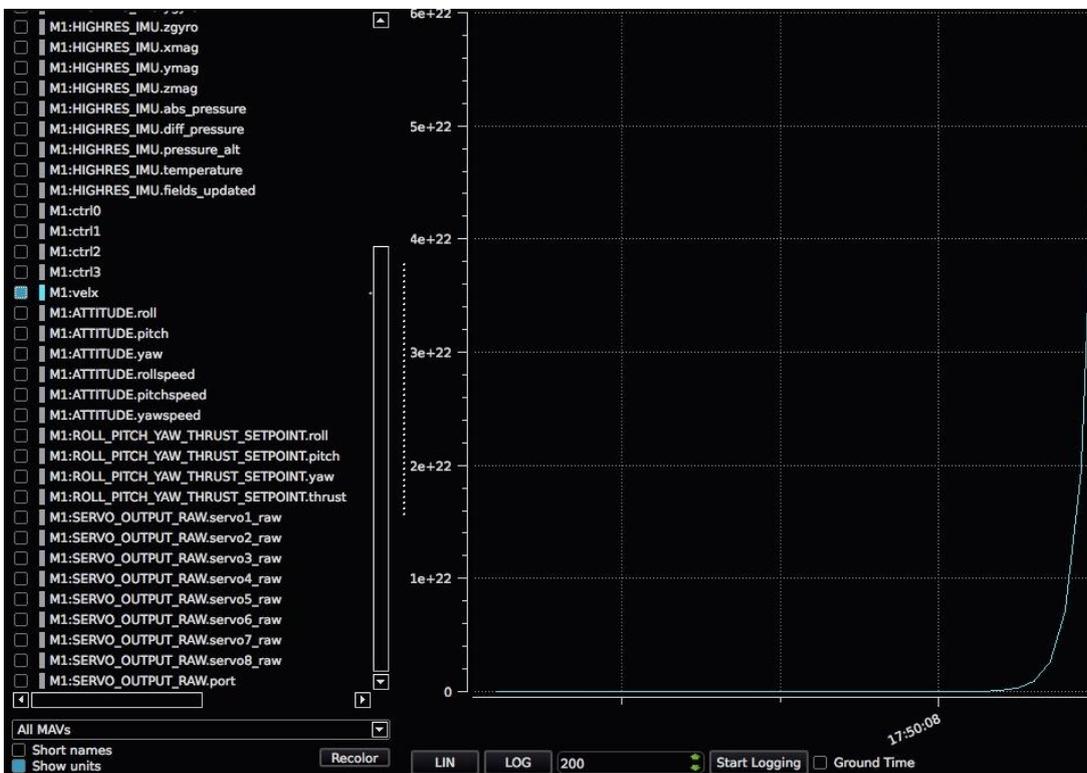
```
/* advertise debug value */
struct debug_key_value_s dbg = { .key = "velx", .value = 0.0f };
orb_advert_t pub_dbg = orb_advertise(ORB_ID(debug_key_value), &dbg);
```

And sending in the main loop is even simpler:

```
dbg.value = position[0];
orb_publish(ORB_ID(debug_key_value), pub_dbg, &dbg);
```

Multiple debug messages must have enough time between their respective publishings for Mavlink to process them. This means that either the code must wait between publishing multiple debug messages, or alternate the messages on each function call iteration.

The result in QGroundControl then looks like this on the real-time plot:



Tutorial: Receive String / Float Pairs

The following code snippets show how to receive the `velx` debug variable that was sent in the previous tutorial.

First, subscribe to the topic `debug_key_value`:

```
#include <poll.h>
#include <uORB/topics/debug_key_value.h>

int debug_sub_fd = orb_subscribe(ORB_ID(debug_key_value));
[...]
```

Then poll on the topic:

```
[...]
/* one could wait for multiple topics with this technique, just using one here */
px4_pollfd_struct_t fds[] = {
    { .fd = debug_sub_fd, .events = POLLIN },
};

while (true) {
    /* wait for debug_key_value for 1000 ms (1 second) */
    int poll_ret = px4_poll(fds, 1, 1000);

    [...]
```

When a new message is available on the `debug_key_value` topic, do not forget to filter it based on its key attribute in order to discard the messages with key different than `velx`:

```
[...]
if (fds[0].revents & POLLIN) {
    /* obtained data for the first file descriptor */
    struct debug_key_value_s dbg;

    /* copy data into local buffer */
    orb_copy(ORB_ID(debug_key_value), debug_sub_fd, &dbg);

    /* filter message based on its key attribute */
    if (strcmp(_sub_debug_vect.get().key, "velx") == 0) {
        PX4_INFO("velx:\t%8.4f", dbg.value);
    }
}
}
```

9.8 System-wide Replay

Based on ORB messages, it's possible to record and replay arbitrary parts of the system. For this to work, the new logger needs to be enabled (`SYS_LOGGER` set to 1). Replay is useful to test the effect of different parameter values based on real data, compare different estimators, etc.

Prerequisites

The first thing that needs to be done is to identify the module or modules that should be replayed. Then, identify all the inputs to these modules, i.e. subscribed ORB topics. For system-wide replay, this consists of all hardware input: sensors, RC input, mavlink commands and file system.

All identified topics need to be logged at full rate (see [logging](#)). For `ekf2` this is already the case with the default set of logged topics.

It is important that all replayed topics contain only a single absolute timestamp, which is the automatically generated field `timestamp`. Should there be more timestamps, then they must be relative with respect to the main timestamp. For an example, see [sensor_combined.msg](#). Reasons for this are given below.

Usage

- First, choose the file to replay, and build the target (from within the Firmware directory):

```
export replay=<absolute_path_to_log_file.ulg>
make posix_sitl_default
```

This will create the output in a separate build directory `build/posix_sitl_default_replay` (so that the parameters don't interfere with normal builds). It's possible to choose any posix SITL build target for replay, the build system knows through the `replay` environment variable that it's in replay mode.

- Add ORB publisher rules file

in `build/posix_sitl_default_replay/tmp/rootfs/orb_publisher.rules`. This file defines which module is allowed to publish which messages. It has the following format:

```
restrict_topics: <topic1>, <topic2>, ..., <topicN>
module: <module>
ignore_others: <true/false>
```

It means that the given list of topics should only be published by `<module>` (which is the command name). Publications to any of these topics from another module are silently ignored. If `ignore_others` is `true`, then publications to other topics from `<module>` are ignored.

For replay, we only want the `replay` module to be able to publish the previously identified list of topics. So for replaying `ekf2`, the rules file looks like this:

```
restrict_topics: sensor_combined, vehicle_gps_position, vehicle_land_detected
module: replay
ignore_others: true
```

This allows that the modules, which usually publish these topics, don't need to be disabled for replay.

- Optional: setup parameter overrides in the `filebuild/posix_sitl_default_replay/tmp/rootfs/replay_params.txt`. This file should contain a list of `<param_name> <value>`, like:

```
EKF2_GB_NOISE 0.001
```

By default, all parameters from the log file are applied. When a parameter changed during recording, it will be changed as well at the right time during replay. A parameter in the `replay_params.txt` will override the value and changes to it from the log file will not be applied.

- Optional: copy `dataman` missions file from the SD card to the build directory. Only necessary if a mission should be replayed.
- Start the replay:

```
make posix_sitl_default jmavsim
```

This will automatically open the log file, apply the parameters and start to replay. Once done, it will be reported and the process can be exited. Then the newly generated log file can be analyzed, it has `_replayed` appended to its file name.

Note that the above command will show the simulator as well, but depending on what is being replayed, it will not show what's actually going on. It's possible to connect via QGC and e.g. view the changing attitude during replay.

- Finally, unset the environment variable, so that the normal build targets are used again:

```
unset replay
```

Important Notes

- During replay, all dropouts in the log file are reported. These have a negative effect on replay and thus it should be taken care that dropouts are avoided during recording.
- It is currently only possible to replay in 'real-time', meaning as fast as the recording was done. This is planned to be extended in the future.
- A message that has a timestamp of 0 will be considered invalid and not be replayed.

EKF2 Replay

This is a specialization of the system-wide replay for fast EKF2 replay. It will automatically create the ORB publisher rules and works as following:

- Optionally set `SDLOG_MODE` to 1 to start logging from boot

-
- Record the log
 - To replay:

```
export replay_mode=ekf2
export replay=<abs_path_to_log.ulg>
make posix none
```

You can stop it after there's an output like:

```
INFO [replay] Replay done (published 9917 msgs, 2.136 s)
```

The parameters can be adjusted as well. They can be extracted from the log with (install `pyulog` with `sudo pip install pyulog` first):

```
ulog_params -i $replay -d ' ' | grep -e '^EKF2' >
build/posix_sitl_default_replay/tmp/rootfs/replay_params.txt
```

Then edit the parameters in the file as needed and restart the replay process with `make posix none`. This will create a new log file.

The location of the generated log is printed with a message like this:

```
INFO [logger] Opened log file: rootfs/fs/microsd/log/2017-03-
01/13_30_51_replayed.ulg
```

When finished, use `unset replay; unset replay_mode` to exit the replay mode.

Behind the Scenes

Replay is split into 3 components:

- a replay module
- ORB publisher rules
- time handling

The replay module reads the log and publishes the messages with the same speed as they were recorded. A constant offset is added to the timestamp of each message to match the current system time (this is the reason why all other timestamps need to be relative). The command `replay tryapplyparams` is executed before all other modules are loaded and applies the parameters from the log and user-set parameters. Then as the last command, `replay trystart` will again apply the parameters and start the actual replay. Both commands do nothing if the environment variable `replay` is not set. The ORB publisher rules allow to select which part of the system is replayed, as described above. They are only compiled for the posix SITL targets.

The **time handling** is still an **open point**, and needs to be implemented.

9.9 Poor Man's Sampling Profiler

This section describes how to assess performance of the PX4 system by means of profiling.

Credits for the idea belong to [Mark Callaghan and Domas Mituzas](#).

Approach

PMSP is a shell script that operates by interrupting execution of the firmware periodically in order to sample the current stack trace. Sampled stack traces are appended into a text file. Once sampling is finished (which normally takes about an hour or more), the collected stack traces are *folded*. The result of *folding* is another text file that contains the same stack traces, except that all similar stack traces (i.e. those that were obtained at the same point in the program) are joined together, and the number of their occurrences is recorded. The folded stacks are then fed into the visualization script, for which purpose we employ [FlameGraph - an open source stack trace visualizer](#).

Implementation

The script is located at `Debug/poor-mans-profiler.sh`. Once launched, it will perform the specified number of samples with the specified time interval. Collected samples will be stored in a text file in the system temp directory (typically `/tmp`). Once sampling is finished, the script will automatically invoke the stack folder, the output of which will be stored in an adjacent file in the temp directory. If the stacks were folded successfully, the script will invoke the FlameGraph script and store the result in an interactive SVG file. Please note that not all image viewers support interactive images; it is recommended to open the resulting SVG in a web browser.

The FlameGraph script must reside in the `PATH`, otherwise PMSP will refuse to launch. PMSP uses GDB to collect the stack traces. Currently it uses `arm-none-eabi-gdb`, other toolchains may be added in the future.

In order to be able to map memory locations to symbols, the script needs to be referred to the executable file that is currently running on the target. This is done with the help of the option `--elf=<file>`, which expects a path (relative to the root of the repository) pointing to the location of the currently executing ELF.

Usage example:

```
./poor-mans-profiler.sh --elf=build/px4fmu-  
v4_default/src/firmware/nuttx/firmware_nuttx --nsamples=30000
```

Note that every launch of the script will overwrite the old stacks. Should you want to append to the old stacks rather than overwrite them, use the option `--append`:

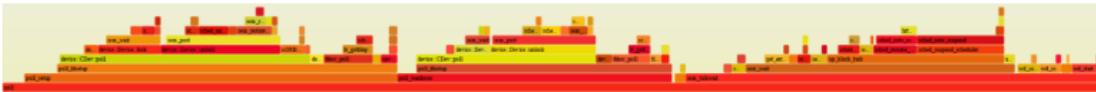
```
./poor-mans-profiler.sh --elf=build/px4fmu-  
v4_default/src/firmware/nutttx/firmware_nutttx --nsamples=30000 --append
```

As one might suspect, `--append` with `--nsamples=0` will instruct the script to only regenerate the SVG without accessing the target at all.

Please read the script for a more in depth understanding of how it works.

Understanding the Output

A screenshot of an example output is provided below (note that it is not interactive here):



On the flame graph, the horizontal levels represent stack frames, whereas the width of each frame is proportional to the number of times it was sampled. In turn, the number of times a function ended up being sampled is proportional to the duration times frequency of its execution.

Possible Issues

The script was developed as an ad-hoc solution, so it has some issues. Please watch out for them while using it:

- If GDB is malfunctioning, the script may fail to detect that, and continue running. In this case, obviously, no usable stacks will be produced. In order to avoid that, the user should periodically check the file `/tmp/pmpn-gdberr.log`, which contains the stderr output of the most recent invocation of GDB. In the future the script should be modified to invoke GDB in quiet mode, where it will indicate issues via its exit code.
- Sometimes GDB just sticks forever while sampling the stack trace. During this failure, the target will be halted indefinitely. The solution is to manually abort the script and re-launch it again with the `--append` option. In the future the script should be modified to enforce a timeout for every GDB invocation.
- Multithreaded environments are not supported. This does not affect single core embedded targets, since they always execute in one thread, but this limitation makes the profiler incompatible with many other applications. In the future the stack folder should be modified to support multiple stack traces per sample.

9.10 Logging

This describes the new logger, `SYS_LOGGER` set to 1.

The logger is able to log any ORB topic with all included fields. Everything necessary is generated from the `.msg` file, so that only the topic name needs to be specified. An

optional interval parameter specifies the maximum logging rate of a certain topic. All existing instances of a topic are logged.

The output log format is [ULog](#).

Usage

By default, logging is automatically started when arming, and stopped when disarming. A new log file is created for each arming session on the SD card. To display the current state, use `logger status` on the console. If you want to start logging immediately, use `logger on`. This overrides the arming state, as if the system was armed. `logger off` undoes this.

Use

```
logger help
```

for a list of all supported logger commands and parameters.

Configuration

The list of logged topics can be customized with a file on the SD card. Create a file `etc/logging/logger_topics.txt` on the card with a list of topics (For SITL, it's `build/posix_sitl_default/tmp/rootfs/fs/microsd/etc/logging/logger_topics.txt`):

```
<topic_name>, <interval>
```

The `<interval>` is optional, and if specified, defines the minimum interval in ms between two logged messages of this topic. If not specified, the topic is logged at full rate.

The topics in this file replace all of the default logged topics.

Scripts

There are several scripts to analyze and convert logging files in the [pyulog](#) repository.

Dropouts

Logging dropouts are undesired and there are a few factors that influence the amount of dropouts:

- Most SD cards we tested exhibit multiple pauses per minute. This shows itself as a several 100 ms delay during a write command. It causes a dropout if the write buffer fills up during this time. This effect depends on the SD card (see below).
- Formatting an SD card can help to prevent dropouts.
- Increasing the log buffer helps.
- Decrease the logging rate of selected topics or remove unneeded topics from being logged (`info.py <file>` is useful for this).

SD Cards

The following provides performance results for different SD cards. Tests were done on a Pixracer; the results are applicable to Pixhawk as well.

SD Card	Mean Seq. Write Speed [KB/s]	Max Write Time / Block (average) [ms]
SanDisk Extreme U3 32GB	461	15
Sandisk Ultra Class 10 8GB	348	40
Sandisk Class 4 8GB	212	60
SanDisk Class 10 32 GB (High Endurance Video Monitoring Card)	331	220
Lexar U1 (Class 10), 16GB High-Performance	209	150
Sandisk Ultra PLUS Class 10 16GB	196	500
Sandisk Pixtor Class 10 16GB	334	250
Sandisk Extreme PLUS Class 10 32GB	332	150

More important than the mean write speed is the maximum write time per block (of 4 KB). This defines the minimum buffer size: the larger this maximum, the larger the log buffer needs to be to avoid dropouts. Logging bandwidth with the default topics is around 50 KB/s, which all of the SD cards satisfy.

By far the best card we know so far is the **SanDisk Extreme U3 32GB**. This card is recommended, because it does not exhibit write time spikes (and thus virtually no dropouts). Different card sizes might work equally well, but the performance is usually different.

You can test your own SD card with `sd_bench -r 50`, and report the results to <https://github.com/PX4/Firmware/issues/4634>.

Log Streaming

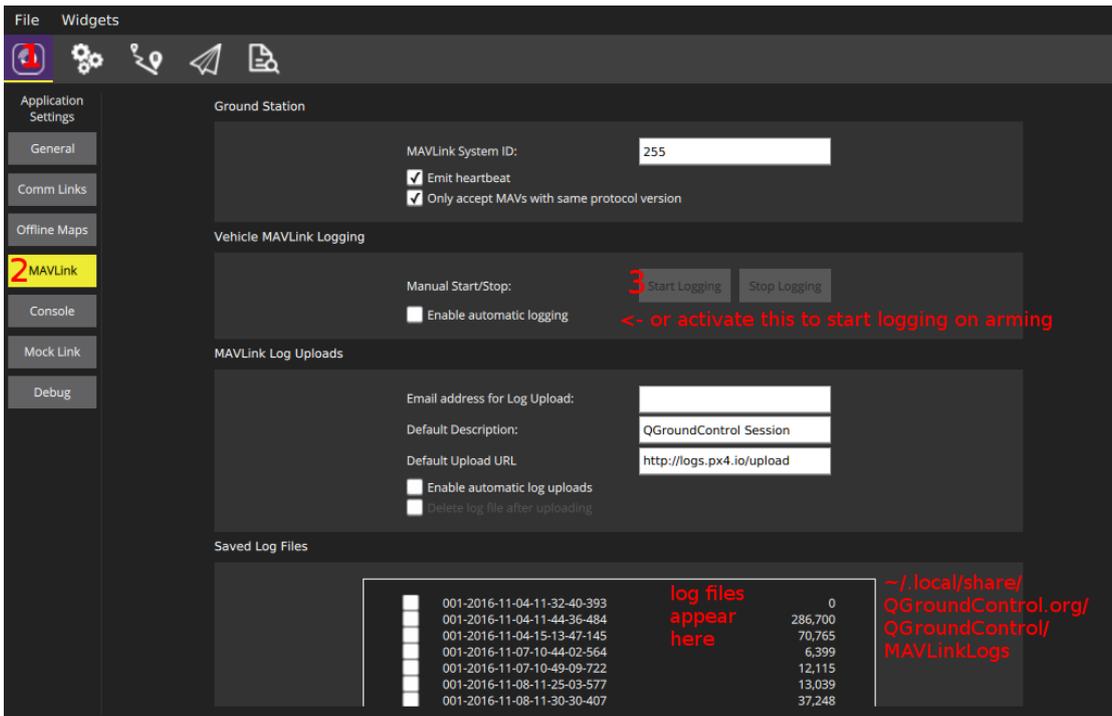
The traditional and still fully supported way to do logging is using an SD card on the FMU. However there is an alternative, log streaming, which sends the same logging data via MAVLink. This method can be used for example in cases where the FMU does not have an SD card slot (e.g. Intel® Aero Ready to Fly Drone) or simply to avoid having to deal with SD cards. Both methods can be used independently and at the same time.

The requirement is that the link provides at least ~50KB/s, so for example a WiFi link. And only one client can request log streaming at the same time. The connection does not need to be reliable, the protocol is designed to handle drops.

There are different clients that support ulog streaming:

`mavlink_ulo_streaming.py` script in Firmware/Tools.

- QGroundControl:



- [MAVGCL](#)

Diagnostics

- If log streaming does not start, make sure the `logger` is running (see above), and inspect the console output while starting.
- If it still does not work, make sure that Mavlink 2 is used. Enforce it by setting `MAV_PROTO_VER` to 2.
- Log streaming uses a maximum of 70% of the configured mavlink rate (`-r` parameter). If more is needed, messages are dropped. The currently used percentage can be inspected with `mavlink status` (1.8% is used in this example):

```
instance #0:
  GCS heartbeat: 160955 us ago
  mavlink chan: #0
  type:          GENERIC LINK OR RADIO
  flow control:  OFF
```

```
rates:
tx: 95.781 kB/s
txerr: 0.000 kB/s
rx: 0.021 kB/s
rate mult: 1.000
ULog rate: 1.8% of max 70.0%
accepting commands: YES
MAVLink version: 2
transport protocol: UDP (14556)
```

Also make sure `txerr` stays at 0. If this goes up, either the NuttX sending buffer is too small, the physical link is saturated or the hardware is too slow to handle the data.

9.11 Flight Log Analysis

This topic outlines approaches and software packages that can be used to analyze PX4 flight logs.

Reporting Flights

[Flight Reporting](#) (PX4 User Guide) explains how to download a log and report/discuss issues with a flight.

Structured Analysis

Before analyzing a flight log it is important to establish its context:

- If the analysis is done after a malfunction, did the log capture the crash or did it stop mid-air?
- Did all controllers track their references? The easiest way to establish this is to compare attitude roll and pitch rates to their set points.
- Does the sensor data look valid? Was there very strong vibration (a reasonable threshold for strong vibration is anything with a peak-to-peak of more than 2-3 m/s/s).
- If the root cause is not specific to the vehicle make sure to report it with a link to the log file (and video if one exists) on the [PX4 issue tracker](#).

Ruling Out Power Failures

If a log file ends mid-air, two main causes are possible: a power failure *or* a hard fault of the operating system.

On autopilots based on the [STM32 series](#), hard faults of the operating system are logged to the SD card. These are located on the top level of the SD card and

named *fault_date.log*, e.g. **fault_2017_04_03_00_26_05.log**. Please always check for the presence of this file if a flight log ends abruptly.

Analysis Tools

Flight Review (Online Tool)

[Flight Review](#) is the successor of *Log Muncher*. It is used in combination with the new [ULog](#) logging format.

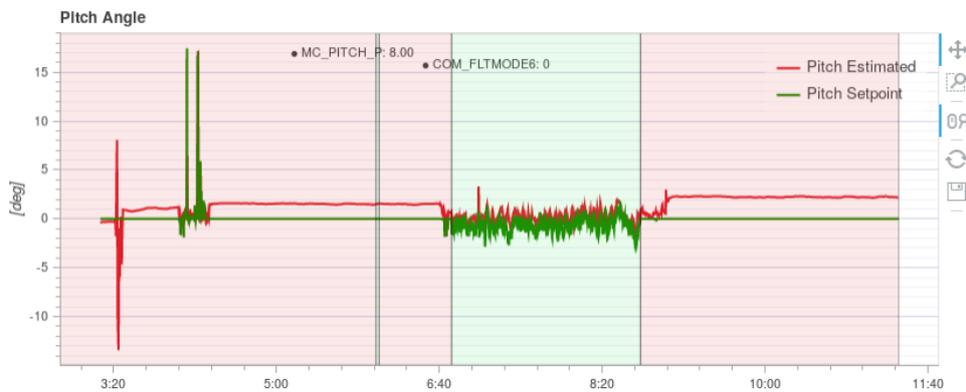
Key features:

- Web based, great for end-users.
- User can upload, load and then share report with others.
- Interactive plots.



Flight Review

UPLOAD DOWNLOAD ▾ NAVIGATION ▾ SIZE ▾ PLOT LEGEND ▾

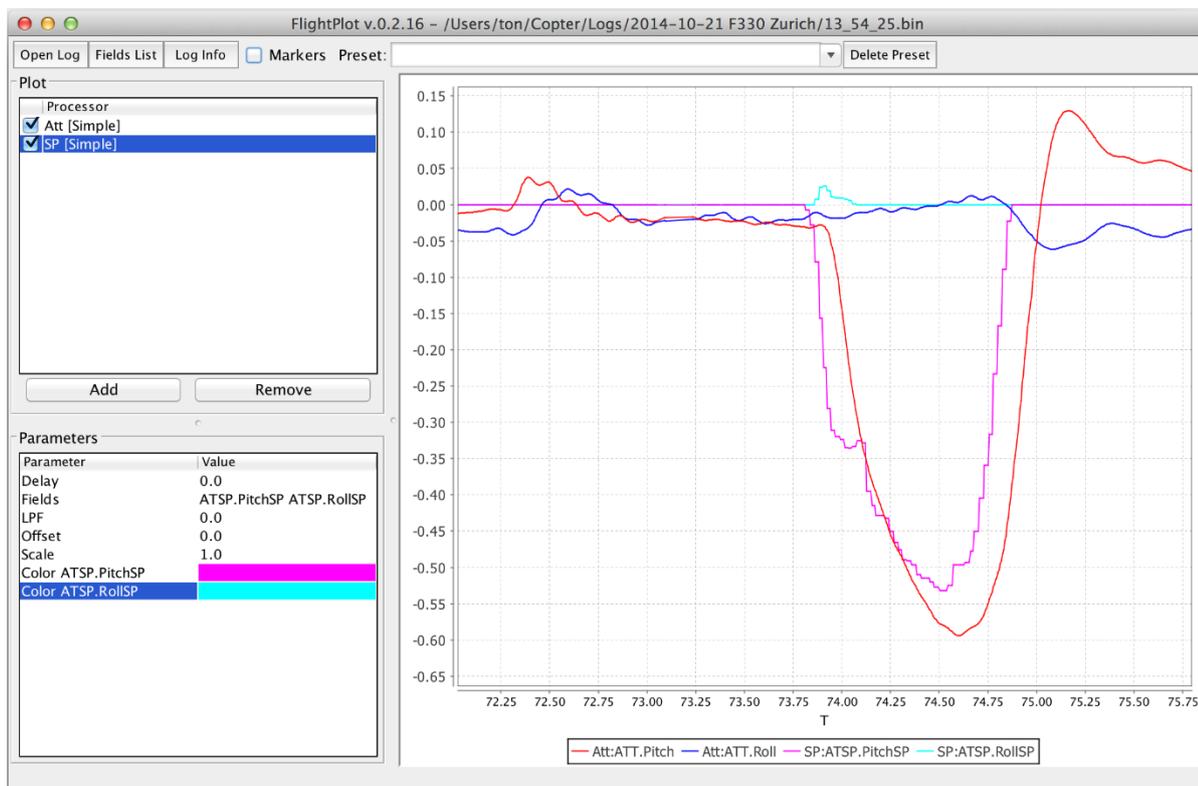


FlightPlot (Desktop)

FlightPlot is a desktop based tool for log analysis. It can be downloaded from [FlightPlot Downloads](#) (Linux, MacOS Windows).

Key features:

- Java based, cross-platform.
- Intuitive GUI, no programming knowledge required.
- Supports both new and old PX4 log formats (.ulg, .px4log, .bin)
- Allows saving plots as images.



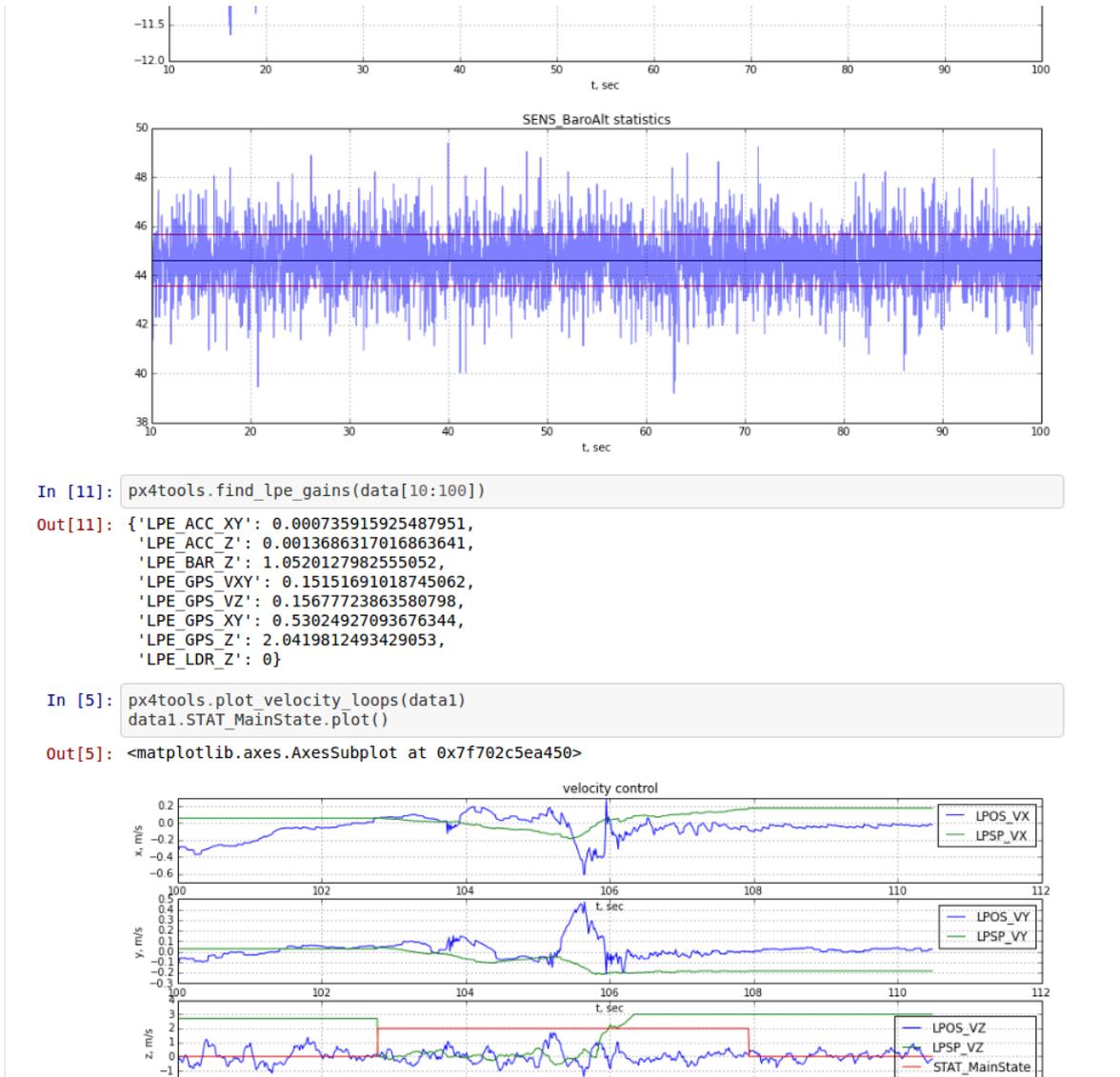
PX4Tools

PX4Tools is a log analysis toolbox for the PX4 autopilot written in Python. The recommended installation procedure is to use [anaconda3](#). See [px4tools github page](#) for details.

Key features:

- Easy to share, users can view notebooks on Github (e.g. <https://github.com/jgoppert/lpe-analysis/blob/master/15-09-30%20Kabir%20Log.ipynb>)
- Python based, cross platform, works with anaconda 2 and anaconda3
- iPython/ jupyter notebooks can be used to share analysis easily

- Advanced plotting capabilities to allow detailed analysis



9.12 ULog File Format

ULog is the file format used for logging system data. The format is self-describing, i.e. it contains the format and message types that are logged.

It can be used for logging device inputs (sensors, etc.), internal states (cpu load, attitude, etc.) and printf log messages.

The format uses Little Endian for all binary types.

Data types

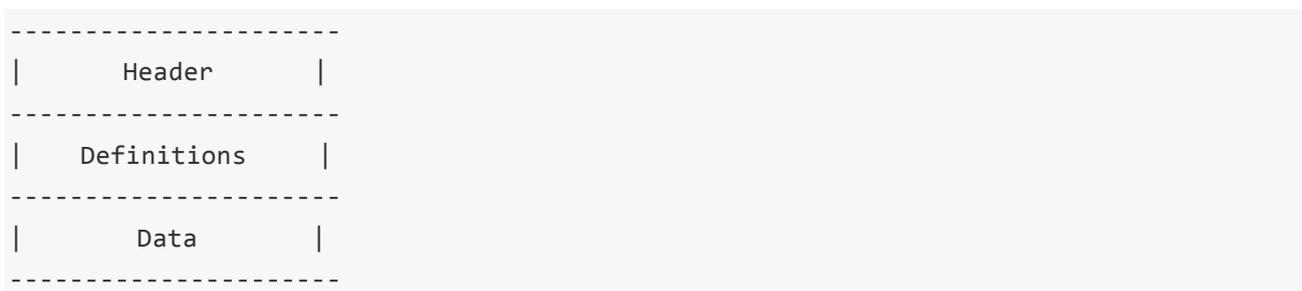
The following binary types are used. They all correspond to the types in C:

Type	Size in Bytes
int8_t, uint8_t	1
int16_t, uint16_t	2
int32_t, uint32_t	4
int64_t, uint64_t	8
float	4
double	8
bool, char	1

Additionally all can be used as an array, eg. `float[5]`. In general all strings (`char[length]`) do not contain a `'\0'` at the end. String comparisons are case sensitive.

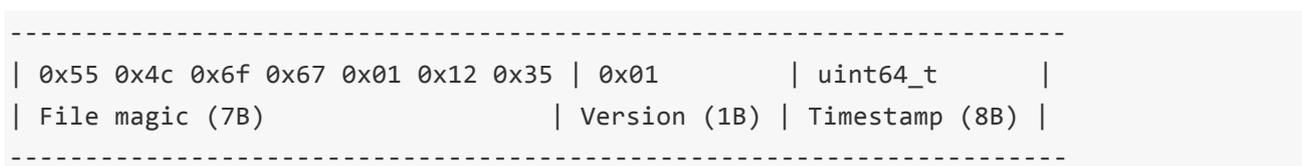
File structure

The file consists of three sections:



Header Section

The header is a fixed-size section and has the following format (16 bytes):



Version is the file format version, currently 1. Timestamp is an `uint64_t` integer, denotes the start of the logging in microseconds.

Definitions Section

Variable length section, contains version information, format definitions, and (initial) parameter values.

The Definitions and Data sections consist of a stream of messages. Each starts with this header:

```
struct message_header_s {
    uint16_t msg_size;
    uint8_t msg_type
};
```

`msg_size` is the size of the message in bytes without the header (`hdr_size= 3` bytes). `msg_type` defines the content and is one of the following:

- 'B': Flag bitset message.

```
struct ulog_message_flag_bits_s {
    uint8_t compat_flags[8];
    uint8_t incompat_flags[8];
    uint64_t appended_offsets[3]; ///< file offset(s) for appended data if appending
bit is set
};
```

This message **must** be the first message, right after the header section, so that it has a fixed constant offset.

- `compat_flags`: compatible flag bits. None of them is currently defined and all must be set to 0. These bits can be used for future ULog changes that are compatible with existing parsers. It means parsers can just ignore the bits if one of the unknown bits is set.
- `incompat_flags`: incompatible flag bits. The LSB bit of index 0 is set to one if the log contains appended data and at least one of the `appended_offsets` is non-zero. All other bits are undefined and must be set to 0. If a parser finds one of these bits set, it must refuse to parse the log. This can be used to introduce breaking changes that existing parsers cannot handle.
- `appended_offsets`: File offsets (0-based) for appended data. If no data is appended, all offsets must be zero. This can be used to reliably append data for logs that may stop in the middle of a message. A process appending data should do:
 - set the relevant `incompat_flags` bit,
 - set the first `appended_offsets` that is 0 to the length of the log file,
 - then append any type of messages that are valid for the Data section.

It is possible that there are more fields appended at the end of this message in future ULog specifications. This means a parser must not assume a fixed length of this message. If the message is longer than expected (currently 40 bytes), the exceeding bytes must just be ignored.

- 'F': format definition for a single (composite) type that can be logged or used in another definition as a nested type.

```
struct message_format_s {
    struct message_header_s header;
    char format[header.msg_size-hdr_size];
};
```

```
};
```

format: plain-text string with the following format: `message_name:field0;field1`; There can be an arbitrary amount of fields (at least 1), separated by `;`. A field has the format: `type field_name` or `type[array_length] field_name` for arrays (only fixed size arrays are supported). `type` is one of the basic binary types or a `message_name` of another format definition (nested usage). A type can be used before it's defined. There can be arbitrary nesting but no circular dependencies.

Some field names are special:

- `timestamp`: every logged message (`message_add_logged_s`) must include a timestamp field (does not need to be the first field). Its type can be: `uint64_t` (currently the only one used), `uint32_t`, `uint16_t` or `uint8_t`. The unit is always microseconds, except for `uint8_t` it's milliseconds. A log writer must make sure to log messages often enough to be able to detect wrap-arounds and a log reader must handle wrap-arounds (and take into account dropouts). The timestamp must always be monotonic increasing for a message serie with the same `msg_id`.
- `Padding`: field names that start with `_padding` should not be displayed and their data must be ignored by a reader. These fields can be inserted by a writer to ensure correct alignment.

If the padding field is the last field, then this field will not be logged, to avoid writing unnecessary data. This means the `message_data_s.data` will be shorter by the size of the padding. However the padding is still needed when the message is used in a nested definition.

- `'l'`: information message.

```
struct message_info_s {
    struct message_header_s header;
    uint8_t key_len;
    char key[key_len];
    char value[header.msg_size-hdr_size-1-key_len]
};
```

`key` is a plain string, as in the format message (can also be a custom type), but consists of only a single field without ending `;`, eg. `float[3] myvalues`. `value` contains the data as described by `key`.

Note that an information message with a certain key must occur at most once in the entire log. Parsers can store information messages as a dictionary.

Predefined information messages are:

key	Description	Example for value
char[value_len] sys_name	Name of the system	"PX4"
char[value_len] ver_hw	Hardware version	"PX4FMU_V4"
char[value_len] ver_sw	Software version (git tag)	"7f65e01"
char[value_len] ver_sw_branch	git branch	"master"
uint32_t ver_sw_release	Software version (see below)	0x010401ff
char[value_len] sys_os_name	Operating System Name	"Linux"
char[value_len] sys_os_ver	OS version (git tag)	"9f82919"
uint32_t ver_os_release	OS version (see below)	0x010401ff
char[value_len] sys_toolchain	Toolchain Name	"GNU GCC"
char[value_len] sys_toolchain_ver	Toolchain Version	"6.2.1"
char[value_len] sys_mcu	Chip name and revision	"STM32F42x, rev A"
char[value_len] sys_uuid	Unique identifier for vehicle (eg. MCU ID)	"392a93e32fa3"...
char[value_len] replay	File name of replayed log if in replay mode	"log001.ulg"
int32_t time_ref_utc	UTC Time offset in seconds	-3600

The format of `ver_sw_release` and `ver_os_release` is: 0xAABBCCTT, where AA is major, BB is minor, CC is patch and TT is the type. Type is defined as following: `>= 0`: development, `>= 64`: alpha version, `>= 128`: beta version, `>= 192`: RC version, `== 255`: release version. So for example 0x010402ff translates into the release version v1.4.2. This message can also be used in the Data section (this is however the preferred section).

- 'M': information message multi.

```
struct ulog_message_info_multiple_header_s {
    uint8_t is_continued; ///< can be used for arrays
    uint8_t key_len;
    char key[key_len];
    char value[header.msg_size-hdr_size-2-key_len]
};
```

The same as the information message, except that there can be multiple messages with the same key (parsers store them as a list). The `is_continued` can be used for split-up messages: if set to 1, it is part of the previous message with the same key. Parsers can store all information multi messages as a 2D list, using the same order as the messages occur in the log.

- 'P': parameter message. Same format as `message_info_s`. If a parameter dynamically changes during runtime, this message can also be used in the Data section. The data type is restricted to: `int32_t`, `float`.
This section ends before the start of the first `message_add_logged_s` OR `message_logging_s` message, whichever comes first.

Data Section

The following messages belong to this section:

- 'A': subscribe a message by name and give it an id that is used in `message_data_s`. This must come before the first corresponding `message_data_s`.

```
struct message_add_logged_s {
    struct message_header_s header;
    uint8_t multi_id;
    uint16_t msg_id;
    char message_name[header.msg_size-hdr_size-3];
};
```

`multi_id`: the same message format can have multiple instances, for example if the system has two sensors of the same type. The default and first instance must be 0. `msg_id`: unique id to match `message_data_s` data. The first use must set this to 0, then increase it. The same `msg_id` must not be used twice for different subscriptions, not even after unsubscribing. `message_name`: message name to subscribe to. Must match one of the `message_format_s` definitions.

- 'R': unsubscribe a message, to mark that it will not be logged anymore (not used currently).

```
struct message_remove_logged_s {
    struct message_header_s header;
    uint16_t msg_id;
};
```

- 'D': contains logged data.

```
struct message_data_s {
    struct message_header_s header;
    uint16_t msg_id;
    uint8_t data[header.msg_size-hdr_size];
};
```

`msg_id`: as defined by a `message_add_logged_s` message. `data` contains the logged binary message as defined by `message_format_s`. See above for special treatment of padding fields.

- 'L': Logged string message, i.e. `printf` output.

```
struct message_logging_s {
    struct message_header_s header;
```

```

uint8_t log_level;
uint64_t timestamp;
char message[header.msg_size-hdr_size-9]
};

```

timestamp: in microseconds, log_level: same as in the Linux kernel:

Name	Level value	Meaning
EMERG	'0'	System is unusable
ALERT	'1'	Action must be taken immediately
CRIT	'2'	Critical conditions
ERR	'3'	Error conditions
WARNING	'4'	Warning conditions
NOTICE	'5'	Normal but significant condition
INFO	'6'	Informational
DEBUG	'7'	Debug-level messages

- 'S': synchronization message so that a reader can recover from a corrupt message by searching for the next sync message (not used currently).

```

struct message_sync_s {
    struct message_header_s header;
    uint8_t sync_magic[8];
};

```

sync_magic: to be defined.

- 'O': mark a dropout (lost logging messages) of a given duration in ms. Dropouts can occur e.g. if the device is not fast enough.

```

struct message_dropout_s {
    struct message_header_s header;
    uint16_t duration;
};

```

- 'I': information message. See above.
- 'M': information message multi. See above.
- 'P': parameter message. See above.

Requirements for Parsers

A valid ULog parser must fulfill the following requirements:

- Must ignore unknown messages (but it can print a warning).
- Parse future/unknown file format versions as well (but it can print a warning).

-
- Must refuse to parse a log which contains unknown incompatibility bits set (`incompat_flags` of `ulog_message_flag_bits_s` message), meaning the log contains breaking changes that the parser cannot handle.
 - A parser must be able to correctly handle logs that end abruptly, in the middle of a message. The unfinished message should just be discarded.
 - For appended data: a parser can assume the Data section exists, i.e. the offset points to a place after the Definitions section.

Appended data must be treated as if it was part of the regular Data section.

Known Implementations

- PX4 Firmware: C++
 - [logger module](#)
 - [replay module](#)
 - [hardfault_log module](#): append hardfault crash data.
- [pyulog](#): python, ULog parser library with CLI scripts.
- [FlightPlot](#): Java, log plotter.
- [MAVLink](#): Messages for ULog streaming via MAVLink (note that appending data is not supported, at least not for cut off messages).
- [QGroundControl](#): C++, ULog streaming via MAVLink and minimal parsing for GeoTagging.
- [mavlink-router](#): C++, ULog streaming via MAVLink.
- [MAVGAnalysis](#): Java, ULog streaming via MAVLink and parser for plotting and analysis.

File Format Version History

Changes in version 2

Addition of `ulog_message_info_multiple_header_s` and `ulog_message_flag_bits_s` messages and the ability to append data to a log. This is used to add crash data to an existing log. If data is appended to a log that is cut in the middle of a message, it cannot be parsed with version 1 parsers. Other than that forward and backward compatibility is given if parsers ignore unknown messages.

10 Tutorials

10.1 QGroundControl

QGroundControl is an app to configure and fly a PX4 based autopilot. It is cross platform and supports all major operating systems:

- Mobile: Android and iOS (currently focused on tablet)
- Desktop: Windows, Linux, Mac OS

Planning Missions

To plan a new mission, switch to the planning tab, click on the + icon in the top left and click on the map to create waypoints. A context menu will open on the side to adjust the waypoints. Click on the highlight transmission icon to send them to the vehicle.



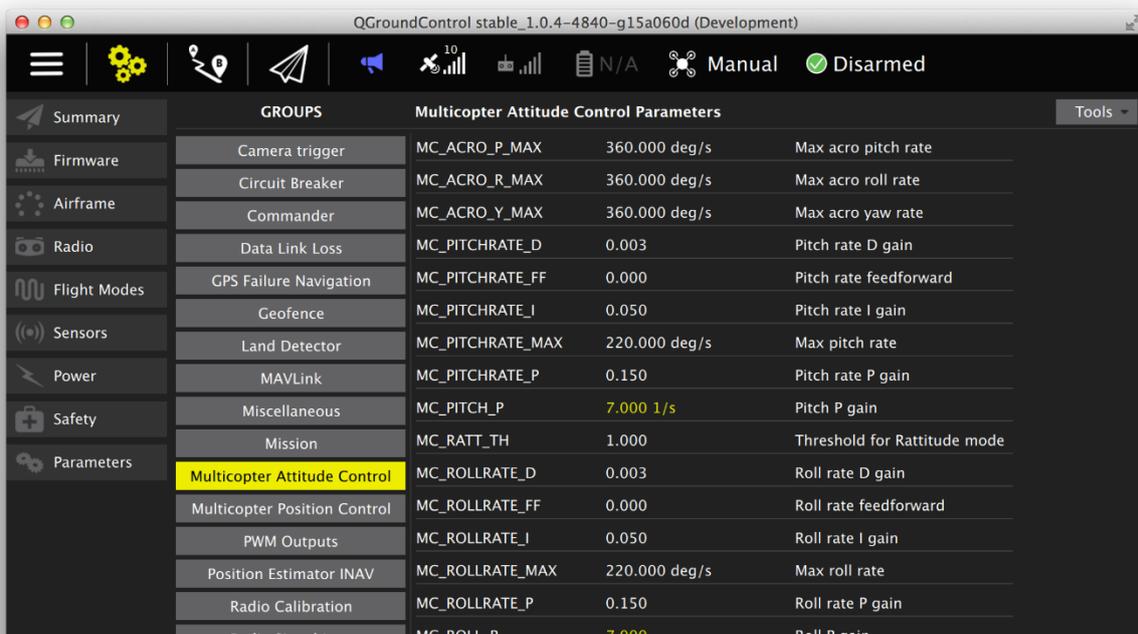
Flying Missions

Switch to the flying tab. The mission should be visible on the map. Click on the current flight mode to change it to MISSION and click on DISARMED to arm the vehicle. If the vehicle is already in flight it will fly to the first leg of the mission and then follow it.



Setting parameters

Switch to the setup tab. Scroll the menu on the left all the way to the bottom and click on the parameter icon. Parameters can be changed by double-clicking on them, which opens a context menu to edit, along with a more detailed description.



Installation

QGroundControl can be downloaded from its [website](#).

Developers are advised to use the latest daily build instead of the stable release.

Building from source

Firmware developers are encouraged to build from source in order to have a matching recent version to their flight code.

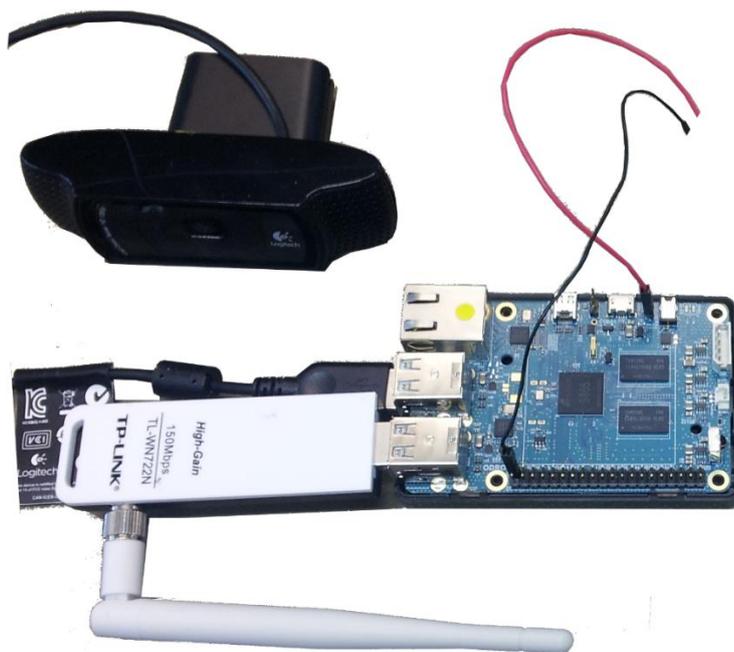
Follow the [QGroundControl build instructions](#) to install Qt and build the source code.

10.2 Video streaming in QGroundControl

This page shows how to set up a companion computer (Odroid C1) with a camera (Logitech C920) such that the video stream is transferred via the Odroid C1 to a network computer and displayed in the application QGroundControl that runs on this computer.

The whole hardware setup is shown in the figure below. It consists of the following parts:

- Odroid C1
- Logitech camera C920
- WiFi module TP-LINK TL-WN722N

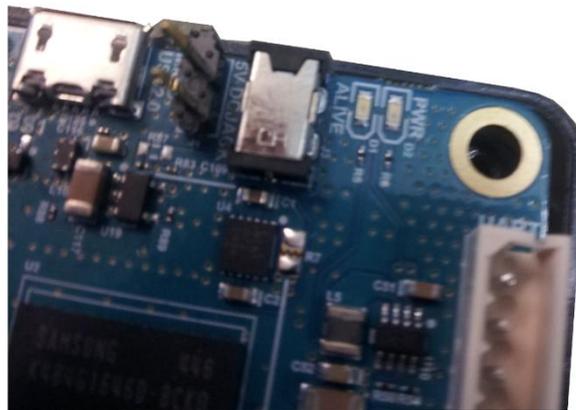


Install Linux environment in Odroid C1

To install the Linux environment (Ubuntu 14.04), follow the instruction given in the [Odroid C1 tutorial](#). In this tutorial it is also shown how to access the Odroid C1 with a UART cable and how to establish Ethernet connection.

Set up alternative power connection

The Odroid C1 can be powered via the 5V DC jack. If the Odroid is mounted on a drone, it is recommended to solder two pins next to the 5V DC jack by applying the through-hole soldering [method](#) as shown in the figure below. The power is delivered by connecting the DC voltage source (5 V) via a jumper cable (red in the image above) with the Odroid C1 and connect the ground of the circuit with a jumper cable (black in the image above) with a ground pin of the Odroid C1 in the example setup.



Enable WiFi connection for Odroid C1

In this this tutorial the WiFi module TP-LINK TL-WN722N is used. To enable WiFi connection for the Odroid C1, follow the steps described in the [Odroid C1 tutorial](#) in the section Establishing wifi connection with antenna.

Configure as WiFi Access Point

This sections shows how to set up the Odroid C1 such that it is an access point. The content is taken from this [tutorial](#) with some small adaptations. To enable to stream the video from the camera via the Odroid C1 to the QGroundControl that runs on a computer it is not required to follow this section. However, it is shown here because setting up the Odroid C1 as an access point allows to use the system in a stand-alone fashion. The TP-LINK TL-WN722N is used as a WiFi module. In the ensuing steps it is assumed that the Odroid C1 assigns the name wlan0 to your WiFi module. Change all occurrences of wlan0 to the appropriate interface if different (e.g. wlan1).

Onboard Computer as Access Point

For a more in depth explanation, you can look at [RPI-Wireless-Hotspot](#)

Install the necessary software

```
sudo apt-get install hostapd udhcpd
```

Configure DHCP. Edit the file `/etc/udhcpd.conf`

```
start 192.168.2.100 # This is the range of IPs that the hotspot will give to client
devices.
end 192.168.2.200
interface wlan0 # The device uDHCP listens on.
remaining yes
opt dns 8.8.8.8 4.2.2.2 # The DNS servers client devices will use (if routing through
the ethernet link).
opt subnet 255.255.255.0
opt router 192.168.2.1 # The Onboard Computer's IP address on wlan0 which we will set
up shortly.
opt lease 864000 # 10 day DHCP lease time in seconds
```

All other 'opt' entries should be disabled or configured properly if you know what you are doing.

Edit the file `/etc/default/udhcpd` and change the line:

```
DHCPD_ENABLED="no"
```

to

```
#DHCPD_ENABLED="no"
```

You will need to give the Onboard Computer a static IP address. Edit the file `/etc/network/interfaces` and replace the line `iface wlan0 inet dhcp` (or `iface wlan0 inet manual`) to:

```
auto wlan0
iface wlan0 inet static
address 192.168.2.1
netmask 255.255.255.0
network 192.168.2.0
broadcast 192.168.2.255
wireless-power off
```

Disable the original (WiFi Client) auto configuration. Change the lines (they probably will not be all next to each other or may not even be there at all):

```
allow-hotplug wlan0
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

to:

```
#allow-hotplug wlan0
```

```
#wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
#iface default inet dhcp
```

If you have followed the [Odroid C1 tutorial](#) to set up the WiFi connection, you might have created the file `/etc/network/interfaces.d/wlan0`. Please comment out all lines in that file such that those configurations have no effect anymore.

Configure HostAPD: To create a WPA-secured network, edit the file `/etc/hostapd/hostapd.conf` (create it if it does not exist) and add the following lines:

```
auth_algs=1
channel=6          # Channel to use
hw_mode=g
ieee80211n=1       # 802.11n assuming your device supports it
ignore_broadcast_ssid=0
interface=wlan0
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
# Change the to the proper driver
driver=nl80211
# Change these to something else if you want
ssid=OdroidC1
wpa_passphrase=QGroundControl
```

Change `ssid=`, `channel=`, and `wpa_passphrase=` to values of your choice. SSID is the hotspot's name which is broadcast to other devices, channel is what frequency the hotspot will run on, `wpa_passphrase` is the password for the wireless network. For many more options see the file `/usr/share/doc/hostapd/examples/hostapd.conf.gz`. Look for a channel that is not in use in the area. You can use tools such as `wavemon` for that.

Edit the file `/etc/default/hostapd` and change the line:

```
#DAEMON_CONF=""
```

to:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Your Onboard Computer should now be hosting a wireless hotspot. To get the hotspot to start on boot, run these additional commands:

```
sudo update-rc.d hostapd enable
sudo update-rc.d udhcpd enable
```

This is enough to have the Onboard Computer present itself as an Access Point and allow your ground station to connect. If you truly want to make it work as a real Access Point (routing the WiFi traffic to the Onboard Computer's ethernet connection), we need to configure the routing and network address translation (NAT). Enable IP forwarding in the kernel:

```
sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
```

To enable NAT in the kernel, run the following commands:

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED -j ACCEPT
```

```
sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
```

To make this permanent, run the following command:

```
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

Now edit the file `/etc/network/interfaces` and add the following line to the bottom of the file:

```
up iptables-restore < /etc/iptables.ipv4.nat
```

gstreamer Installation

To install gstreamer packages on the computer and on the Odroid C1 and start the stream, follow the instruction given in the [QGroundControl README](#).

If you cannot start the stream on the Odroid with the `uvch264s` plugin, you can also try to start it with the `v4l2src` plugin:

```
gst-launch-1.0 v4l2src device=/dev/video0 ! video/x-
h264,width=1920,height=1080,framerate=24/1 ! h264parse ! rtph264pay ! udpsink
host=xxx.xxx.xxx.xxx port=5000
```

Where `xxx.xxx.xxx.xxx` is the IP address where QGC is running. If you get the system error: `Permission denied`, you might need to prepend `sudo` to the command above.

If everything works, you should see the video stream on the bottom left corner in the flight-mode window of QGroundControl as shown in the screenshot below.



If you click on the video stream, the satellite map is shown in the left bottom corner and the video is shown in the whole background.

10.3 Long-distance Video Streaming in QGroundControl

This page shows how to set up a companion computer with a camera (Logitech C920 or RaspberryPi camera) such that the video stream is transferred from the UAV to a ground computer and displayed in *QGroundControl*. This setup uses WiFi in unconnected (broadcast) mode and software from the [Wifibroadcast project](#).

Wifibroadcast Overview

The *Wifibroadcast project* aims to mimic the advantageous properties of using an analog link to transmit HD video (and other) data when using WiFi radios. For example, it attempts to provide a video feed that degrades gracefully with signal degradation/distance.

Before using *Wifibroadcast* check regulators allow this kind of WiFi use in your country.

The high level benefits of *Wifibroadcast* include:

- Minimal latency by encoding every incoming RTP packet to a single WiFi (IEEE80211) packet and immediately sending (doesn't serialize to byte stream).
- Smart FEC support (immediately yield packet to video decoder if FEC pipeline without gaps).
- Stream encryption and authentication ([libsodium](#))
- Distributed operation. It can gather data from cards on different hosts, so that bandwidth is not limited to that of a single USB bus.
- Aggregation of MAVLink packets. It doesn't send WiFi packet for every MAVLink packet.
- [Enhanced OSD for Raspberry Pi](#) (consumes 10% CPU on Pi Zero).

Additional information is provided in the [FAQ](#) below.

Hardware Setup

The hardware setup consists of the following parts:

On TX (UAV) side:

- [NanoPI NEO2](#) (and/or Raspberry Pi if use Pi camera).
- [Logitech camera C920](#) or [Raspberry Pi camera](#).
- WiFi module [ALPHA AWUS051NH v2](#).

On RX (ground station side):

-
- Any computer with Linux (tested on Fedora 25 x86-64).
 - WiFi module with Ralink RT5572 chipset ([CSL 300Mbit Sticks](#) or [GWF-4M02](#)). OEM modules are cheap but you need to order them from China. CSL stick is expensive but available on ebay. See [wifibroadcast wiki > WiFi hardware](#) for more information on supported modules.

Hardware Modification

Alpha WUS051NH is a high power card that uses too much current while transmitting. If you power it from USB it will reset the port on most ARM boards. So you need to connect it to 5V BEC directly. You can do this two ways:

1. Make a custom USB cable. [You need to cut +5v wire from USB plug and connect it to BEC](#)
2. Cut a +5v wire on PCB near USB port and wire it to BEC. Don't do this if doubt. Use custom cable instead! Also I suggest to add 470uF low ESR capacitor (like ESC has) between power and ground to filter voltage spikes. Be aware of [ground loop](#) when using several ground wires.

Software Setup

1. Install **libpcap** and **libsodium** development libs.
2. Download [wifibroadcast sources](#).
3. [Patch](#) your kernel. You only need to patch the kernel on TX (except if you want to use a WiFi channel which is disabled in your region by CRDA).

Generate Encryption Keys

```
make
keygen
Copy rx.key to RX host and tx.key to TX host.
```

UAV Setup (TX)

1. Setup camera to output RTP stream:
 - a. Logitech camera C920 camera:

```
gst-launch-1.0 uvch264src device=/dev/video0 initial-bitrate=6000000 average-
bitrate=6000000 iframe-period=1000 name=src auto-start=true \
    src.vidsrc ! queue ! video/x-h264,width=1920,height=1080,framerate=30/1 !
h264parse ! rtph264pay ! udpsink host=localhost port=5600
```

b. RaspberryPi camera:

```
raspivid --nopreview --awb auto -ih -t 0 -w 1920 -h 1080 -fps 30 -b 4000000 -g 30
-pf high -o - | gst-launch-1.0 fdsrc ! h264parse ! rtph264pay ! udpsink
host=127.0.0.1 port=5600
```

2. Setup *Wifibroadcast* in TX mode:

```
git clone https://github.com/svpcom/wifibroadcast
cd wifibroadcast
make
./scripts/tx_standalone.sh wlan1 # where wlan1 is your WiFi TX interface
```

This will setup wifibroadcast using MCS #1: QPSK 1/2 40MHz Short GI modulation (30 Mbit/s) on 149 WiFi channel (in 5GHz band) and listening on UDP port 5600 for incoming data.

Ground Station Setup (RX)

1. Setup *Wifibroadcast* in RX mode:

```
git clone https://github.com/svpcom/wifibroadcast
cd wifibroadcast
make
./scripts/rx_standalone.sh wlan1 # your WiFi interface for RX
```

2. Run `qgroundcontrol` or

```
gst-launch-1.0 udpsrc port=5600 caps='application/x-rtp, media=(string)video, clock-
rate=(int)90000, encoding-name=(string)H264' \
! rtph264depay ! avdec_h264 ! clockoverlay valignment=bottom !
autovideosink fps-update-interval=1000 sync=false
to decode video.
```

Enhanced setup with RX antenna array, FPV goggles and OSD

See [wiki](#) article. Using RX setup above (and ALPHA AWUS051NH v2 as TX) I was able to receive stable 1080p video on 1-2km in any copter pitch/roll angles.

FAQ

What are the limitations of normal WiFi for long-distance video transfer?

Normal WiFi has the following problems when used for long distance video transfer:

-
- **Association:** The video transmitter and receiver need to be "associated". If one device loses association (for example due to weak signal strength) then video transmission stops instantly.
 - **Two-way communication:** Even if you are sending data only from source to sink a bi-directional data flow is required using WiFi. The reason for this is that a WiFi receiver needs to acknowledge the received packets. If the transmitter receives no acknowledgments it will drop the association. Therefore, you would need equally strong transmitters and antennas both on the aircraft and on the ground station. A setup with a strong transmitter in the air using an omni-directional antenna and a weak device on the ground using a high-gain antenna is not possible with normal WiFi.
 - **Rate control:** Normal WiFi connections switch automatically to a lower transmission rate if signal strength is too weak. This can result in an (automatically) selected rate that is too low to transfer video data. The result is that data can queue up and introduce an unpredictable latency of up to several seconds.
 - **One to one transfers:** Unless you use broadcast frames or similar techniques, a normal WiFi data flow is a one-to-one connection. A scenario where a bystander just locks onto your "channel" to watch your stream (as is possible in analog video transmission) is not easy to accomplish using traditional WiFi.
 - Limited diversity: Normal WiFi limits you to the number of diversity streams that your WiFi card offers.

How does Wifibroadcast overcome these limitations

Wifibroadcast puts the WiFi cards into monitor mode. This mode allows to send and receive arbitrary packets without association. This way a true unidirectional connection is established which mimics the advantageous properties of an analog link. Those are:

- The transmitter sends its data regardless of any associated receivers. Thus there is no risk of sudden video stall due to the loss of association
- The receiver receives video as long as it is in range of the transmitter. If it gets slowly out of range the video quality degrades but does not stall.
- The traditional scheme "single broadcaster – multiple receivers" works out of the box. If bystanders want to watch the video stream with their devices they just have to "switch to the right channel"
- *Wifibroadcast* allows you to use several low cost receivers in parallel and combine their data to increase probability of correct data reception. This so-called software diversity allows you to use identical receivers to improve reliability as well as complementary receivers (think of one receiver with an omnidirectional antenna covering 360° and several directional antennas for high distance all working in parallel)
- *Wifibroadcast* uses Forward Error Correction (FEC) to achieve a high reliability at low bandwidth requirements. It is able to repair lost or corrupted packets at the receiver.

How does the new Wifibroadcast differ from the original project?

The [original version of wifibroadcast](#) shares the same name as the [current project](#), but does not derive any code from it.

The original version used a byte-stream as input and split it to packets of fixed size (1024 by default). If a radio packet was lost and this was not corrected by FEC you'll get a hole at random (unexpected) place in the stream. This is especially bad if the data protocol is not resistant to (was not designed for) such random erasures.

The new version has been rewritten to use UDP as data source and pack one source UDP packet into one radio packet. Radio packets now have variable size depends on payload size. This reduces a video latency a lot.

What type of data can be transmitted using Wifibroadcast?

Any UDP with packet size ≤ 1466 . For example x264 inside RTP or MAVLink.

What are transmission guarantees?

Wifibroadcast uses Forward Error Correction (FEC) which can recover 4 lost packets from a 12 packet block with default settings. You can tune both TX and RX (simultaneously) to fit your needs.

What can cause multiple frame drops and messages XX packets

lost?

This can be due to:

1. Signal power is too low. Use high-power card or antennas with more gain. Use directed antenna on the RX side. Use additional RX card for diversity (add wlan2, wlan3, ... to RX program)
2. Signal power is too high. Especially if you use 30dBm TX indoors. Try to reduce TX power (for example hack CRDA database inside kernel and make several regions with power limit 10dBm and 20dBm).
3. Interference with other WiFi. Try to change WiFi channel and/or WiFi band.

Don't use band that the RC TX operates on! Or setup RTL properly to avoid model loss.

You can increase FEC block size (by default it is 8/12 - 8 data blocks and 4 FEC blocks), at the cost of increasing latency. Use additional RX card for diversity (add wlan2, wlan3, ... to RX program)

What ARM Boards are recommended for the UAV?

Board	Pros	Cons
Raspberry Pi Zero	<ul style="list-style-type: none">- Huge community- Camera support- HW video encoder/decoder with OMX API.	<ul style="list-style-type: none">- Hard to buy outside US (shipping costs >> its price)- Slow CPU- Only one USB bus- 512MB SDRAM
Odroid C0	<ul style="list-style-type: none">- Fast CPU- EMMC- 1GB SDRAM	<ul style="list-style-type: none">- Very sensitive to radio interference- Doesn't supported by mainline kernel- High cost- HW video encoder is broken- Bad PCB quality (too thin, ground pins without thermal relief)
NanoPi NEO2	<ul style="list-style-type: none">- ARM 64-bit CPU- Very cheap- Supported by mainline kernel- 3 independent USB busses- 1Gbps Ethernet port- 3 UARTs- Very small form-factor- Resistant to radio interference	<ul style="list-style-type: none">- Small community- 512MB SDRAM- No camera interface

This article chose to use Pi Zero as camera board (encode video) and NEO2 as main UAV board (wifibroadcast, MAVLink telemetry, etc.)

TODO

1. Make prebuilt packages. Pull requests are welcome.
2. Do a flight test with different cards/antennas.
3. Investigate how to set TX power without CRDA hacks.
4. Tune FEC for optimal latency/redundancy.
5. Inject packets with radio link RSSI to MAVLink stream

10.4 Optical Flow

Optical Flow uses a downward facing camera and a downward facing distance sensor for position estimation. Optical Flow based navigation is supported by all three estimators: EKF2, LPE and INAV (see below).

Setup

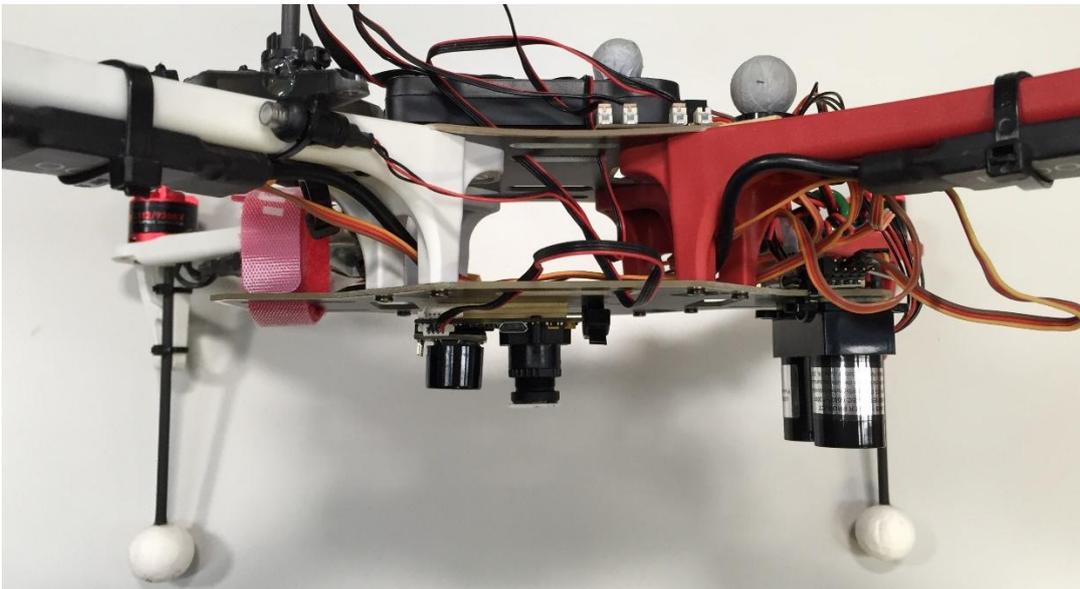
As mentioned above, an Optical Flow setup requires a downward facing camera which publishes to the `OPTICAL_FLOW_RAD` topic and a distance sensor (preferably a LiDAR) publishing messages to the `DISANCE_SENSOR` topic.

The output of the flow has to be as follows

Moving direction of the MAV	Integrated flow
Forwards	+ Y
Backwards	- Y
Right	- X
Left	+ X

And for pure rotations, the `integrated_xgyro` and `integrated_x` (respectively `integrated_ygyro` and `integrated_y`) have to be the same.

An exemplary setup is the PX4Flow and LIDAR-Lite (see picture).

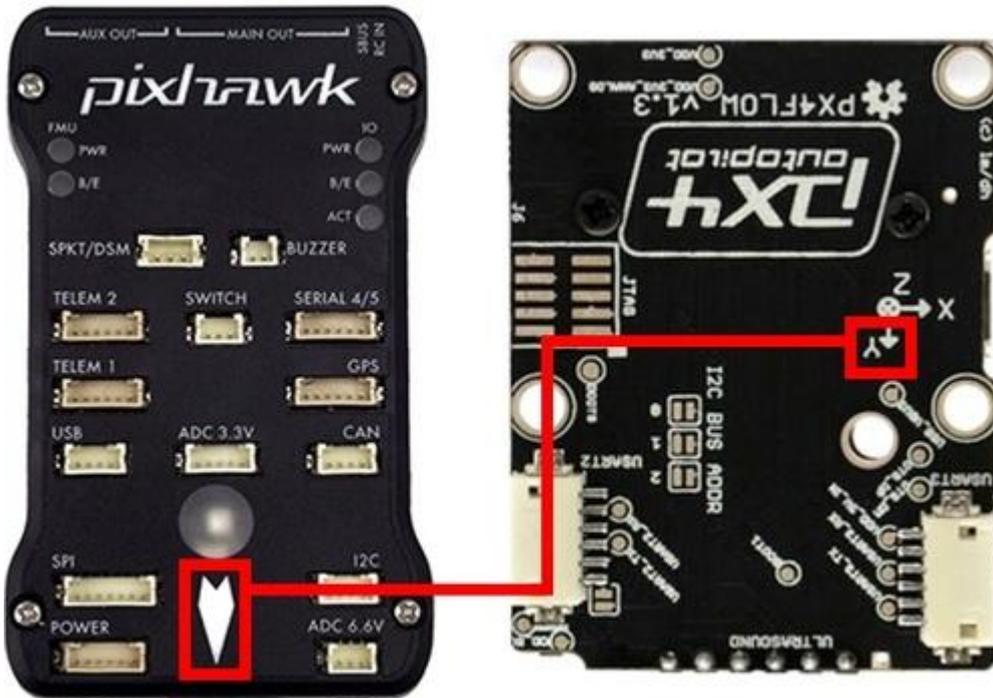


Cameras

PX4Flow

The easiest way to calculate the optical flow is to use the PX4Flow board. In order to use the PX4Flow board, just connect it with I2C. The recommended way of mounting it

is with the Sonar side facing forwards (see image). In this configuration the parameter `SENS_FLOW_ROT` should be 270 degrees (which is the default). Make sure the the PX4Flow board is well dampened.



Custom I2C Address

The default I2C address of the PX4Flow is 0x42, but it can be incremented using the three solder jumpers labeled "I2C BUS ADDR" on the picture above. This is useful if another device has the same address. The address increment is equal to the 3-bit value encoded by the jumpers. For example if jumper 0 and 1 are soldered and jumper 2 is unsoldered, the address is incremented by $1*1 + 1*2 + 0*4 = 3$, which gives address 0x45. If all jumpers are unsoldered, the camera will be automatically discovered by the autopilot firmware. If you modify the I2C address of the PX4Flow, make sure to start the PX4 driver with the correct address:

```
px4flow start          # address=0x42 (default)
px4flow stop
px4flow start -a 0x45  # address=0x45
```

Focusing the Lens

In order to ensure good optical flow quality, it is important to focus the camera on the PX4Flow to the desired height of flight. To focus the camera, put an object with text on (e. g. a book) and plug in the PX4Flow into USB and run QGroundControl. Under the settings menu, select the PX4Flow and you should see a camera image. Focus the lens

by unscrewing the set screw and loosening and tightening the lens to find where it is in focus.

If you fly above 3m, the camera will be focused at infinity and won't need to be changed for higher flight.



Figure: Use a text book to focus the flow camera at the height you want to fly, typically 1-3 meters. Above 3 meters the camera should be focused at infinity and work for all higher altitudes.

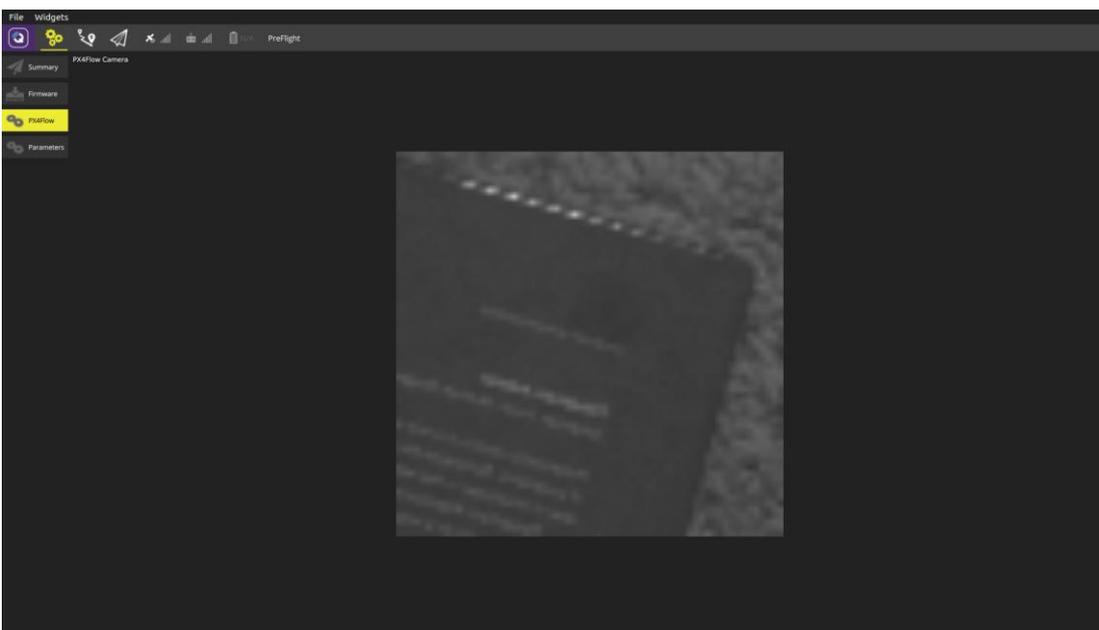


Figure: The px4flow interface in QGroundControl that can be used for focusing the camera

Other Cameras

It is also possible to use a board/quad that has an integrated camera (Bebop2, Snapdragon Flight). For this the [Optical Flow repo](#) can be used (see also [snap_cam](#)).

Range Finder

We recommend using a LIDAR over a Sonar, because of robustness and accuracy. One possibility is the [LIDAR-Lite](#).

Estimators

Extended Kalman Filter (EKF2)

In order to use the EKF2 estimator, make sure the parameter `SYS_MC_EST_GROUP` is set to 2 and reboot. For Optical Flow fusion, the parameter `EKF2_AID_MASK` has to be set accordingly.

Local Position Estimator (LPE)

10.5 Using the ecl EKF

This tutorial answers common questions about use of the ECL EKF algorithm.

What is the ecl EKF?

The Estimation and Control Library (ECL) uses an Extended Kalman Filter (EKF) algorithm to process sensor measurements and provide an estimate of the following states:

- Quaternion defining the rotation from North, East, Down local earth frame to X,Y,Z body frame
- Velocity at the IMU - North,East,Down (m/s)
- Position at the IMU - North,East,Down (m)
- IMU delta angle bias estimates - X,Y,Z (rad)
- IMU delta velocity bias estimates - X,Y,Z(m/s)
- Earth Magnetic field components - North,East,Down (gauss)
- Vehicle body frame magnetic field bias - X,Y,Z (gauss)
- Wind velocity - North,East (m/s)

The EKF runs on a delayed 'fusion time horizon' to allow for different time delays on each measurement relative to the IMU. Data for each sensor is FIFO buffered and retrieved from the buffer by the EKF to be used at the correct time. The delay compensation for each sensor is controlled by the [EKF2 * DELAY](#) parameters.

A complementary filter is used to propagate the states forward from the 'fusion time horizon' to current time using the buffered IMU data. The time constant for this filter is controlled by the [EKF2_TAU_VEL](#) and [EKF2_TAU_POS](#) parameters.

The 'fusion time horizon' delay and length of the buffers is determined by the largest of the [EKF2_*_DELAY](#) parameters. If a sensor is not being used, it is recommended to set its time delay to zero. Reducing the 'fusion time horizon' delay reduces errors in the complementary filter used to propagate states forward to current time.

The position and velocity states are adjusted to account for the offset between the IMU and the body frame before they are output to the control loops. The position of the IMU relative to the body frame is set by the [EKF2*_IMU*_POS*_X,Y,Z](#) parameters.

The EKF uses the IMU data for state prediction only. IMU data is not used as an observation in the EKF derivation. The algebraic equations for the covariance prediction, state update and covariance update were derived using the Matlab symbolic toolbox and can be found here: [Matlab Symbolic Derivation](#).

What sensor measurements does it use?

The EKF has different modes of operation that allow for different combinations of sensor measurements. On start-up the filter checks for a minimum viable combination of sensors and after initial tilt, yaw and height alignment is completed, enters a mode that provides rotation, vertical velocity, vertical position, IMU delta angle bias and IMU delta velocity bias estimates.

This mode requires IMU data, a source of yaw (magnetometer or external vision) and a source of height data. This minimum data set is required for all EKF modes of operation. Other sensor data can then be used to estimate additional states.

IMU

- Three axis body fixed Inertial Measurement unit delta angle and delta velocity data at a minimum rate of 100Hz. Note: Coning corrections should be applied to the IMU delta angle data before it is used by the EKF.

Magnetometer

Three axis body fixed magnetometer data (or external vision system pose data) at a minimum rate of 5Hz is required. Magnetometer data can be used in two ways:

- Magnetometer measurements are converted to a yaw angle using the tilt estimate and magnetic declination. This yaw angle is then used as an observation by the EKF. This method is less accurate and does not allow for learning of body frame field offsets, however it is more robust to magnetic anomalies and large start-up gyro biases. It is the default method used during start-up and on ground.
- The XYZ magnetometer readings are used as separate observations. This method is more accurate and allows body frame offsets to be learned, but assumes the earth magnetic field environment only changes slowly and performs less well when there are significant external magnetic anomalies. It is the default method when the vehicle is airborne and has climbed past 1.5 m altitude.

The logic used to select the mode is set by the [EKF2_MAG_TYPE](#) parameter.

Height

A source of height data - either GPS, barometric pressure, range finder or external vision at a minimum rate of 5Hz is required. Note: The primary source of height data is controlled by the [EKF2_HGT_MODE](#) parameter.

If these measurements are not present, the EKF will not start. When these measurements have been detected, the EKF will initialise the states and complete the tilt and yaw alignment. When tilt and yaw alignment is complete, the EKF can then transition to other modes of operation enabling use of additional sensor data:

GPS

GPS measurements will be used for position and velocity if the following conditions are met:

- GPS use is enabled via setting of the [EKF2_AID_MASK](#) parameter.
- GPS quality checks have passed. These checks are controlled by the [EKF2_GPS_CHECK](#) and `EKF2_REQ_*` parameters.
- GPS height can be used directly by the EKF via setting of the [EKF2_HGT_MODE](#) parameter.

Range Finder

Range finder distance to ground is used by a single state filter to estimate the vertical position of the terrain relative to the height datum.

If operating over a flat surface that can be used as a zero height datum, the range finder data can also be used directly by the EKF to estimate height by setting the [EKF2_HGT_MODE](#) parameter to 2.

Airspeed

Equivalent Airspeed (EAS) data can be used to estimate wind velocity and reduce drift when GPS is lost by setting [EKF2_ARSP_THR](#) to a positive value. Airspeed data will be used when it exceeds the threshold set by a positive value for [EKF2_ARSP_THR](#) and the vehicle type is not rotary wing.

Synthetic Sideslip

Fixed wing platforms can take advantage of an assumed sideslip observation of zero to improve wind speed estimation and also enable wind speed estimation without an airspeed sensor. This is enabled by setting the [EKF2_FUSE_BETA](#) parameter to 1.

Drag Specific Forces

Multi-rotor platforms can take advantage of the relationship between airspeed and drag force along the X and Y body axes to estimate North/East components of wind velocity. This is enabled by setting bit position 5 in the [EKF2_AID_MASK](#) parameter to true. The relationship between airspeed and specific force (IMU acceleration) along the X and Y body axes is controlled by the [EKF2_BCOEF_X](#) and [EKF2_BCOEF_Y](#) parameters which set the ballistic coefficients for flight in the X and Y directions respectively. The amount of specific force observation noise is set by the [EKF2_DRAG_NOISE](#) parameter.

Optical Flow

Optical flow data will be used if the following conditions are met:

- Valid range finder data is available.
- Bit position 1 in the [EKF2_AID_MASK](#) parameter is true.
- The quality metric returned by the flow sensor is greater than the minimum requirement set by the [EKF2_OF_QMIN](#) parameter.

External Vision System

Position and Pose Measurements from an external vision system, e.g. Vicon, can be used:

- External vision system horizontal position data will be used if bit position 3 in the [EKF2_AID_MASK](#) parameter is true.
- External vision system vertical position data will be used if the [EKF2_HGT_MODE](#) parameter is set to 3.
- External vision system pose data will be used for yaw estimation if bit position 4 in the [EKF2_AID_MASK](#) parameter is true.

How do I use the 'ecl' library EKF?

Set the [SYS_MC_EST_GROUP](#) parameter to 2 to use the ecl EKF.

What are the advantages and disadvantages of the ecl EKF over other estimators?

Like all estimators, much of the performance comes from the tuning to match sensor characteristics. Tuning is a compromise between accuracy and robustness and although we have attempted to provide a tune that meets the needs of most users, there will be applications where tuning changes are required.

For this reason, no claims for accuracy relative to the legacy combination of `attitude_estimator_q` + `local_position_estimator` have been made and the best choice of estimator will depend on the application and tuning.

Disadvantages

- The ecl EKF is a complex algorithm that requires a good understanding of extended Kalman filter theory and its application to navigation problems to tune successfully. It is therefore more difficult for users that are not achieving good results to know what to change.
- The ecl EKF uses more RAM and flash space
- The ecl EKF uses more logging space.
- The ecl EKF has had less flight time

Advantages

- The ecl EKF is able to fuse data from sensors with different time delays and data rates in a mathematically consistent way which improves accuracy during dynamic manoeuvres once time delay parameters are set correctly.
- The ecl EKF is capable of fusing a large range of different sensor types.
- The ecl EKF detects and reports statistically significant inconsistencies in sensor data, assisting with diagnosis of sensor errors.
- For fixed wing operation, the ecl EKF estimates wind speed with or without an airspeed sensor and is able to use the estimated wind in combination with airspeed measurements and sideslip assumptions to extend the dead-reckoning time available if GPS is lost in flight.
- The ecl EKF estimates 3-axis accelerometer bias which improves accuracy for tailsitters and other vehicles that experience large attitude changes between flight phases.
- The federated architecture (combined attitude and position/velocity estimation) means that attitude estimation benefits from all sensor measurements. This should provide the potential for improved attitude estimation if tuned correctly.

How do I check the EKF performance?

EKF outputs, states and status data are published to a number of uORB topics which are logged to the SD card during flight. The following guide assumes that data has been logged using the .ulog file format. To use the .ulog format, set the SYS_LOGGER parameter to 1.

The .ulog format data can be parsed in python by using the [PX4 pyulog library](#).

Most of the EKF data is found in the [ekf2 innovations](#) and [estimator status](#) uORB messages that are logged to the .ulog file.

A python script that automatically generates analysis plots and metadata can be found [here](#). To use this script file, cd to the `Tools/ec1_ekf` directory and enter `python process_logdata_ekf.py <log_file>.ulg`. This saves performance metadata in a csv file named `<log_file>.mdat.csv` and plots in a pdf file named `<log_file>.pdf`.

Multiple log files in a directory can be analysed using the [batch_process_logdata_ekf.py](#) script. When this has been done, the performance metadata files can be processed to provide a statistical assessment of the estimator performance across the population of logs using the [batch_process_metadata_ekf.py](#) script.

Output Data

- Attitude output data is found in the [vehicle_attitude](#) message.
- Local position output data is found in the [vehicle_local_position](#) message.
- Global (WGS-84) output data is found in the [vehicle_global_position](#) message.
- Wind velocity output data is found in the [wind_estimate](#) message.

States

Refer to states[32] in [estimator_status](#). The index map for states[32] is as follows:

- [0 ... 3] Quaternions
- [4 ... 6] Velocity NED (m/s)
- [7 ... 9] Position NED (m)
- [10 ... 12] IMU delta angle bias XYZ (rad)
- [13 ... 15] IMU delta velocity bias XYZ (m/s)
- [16 ... 18] Earth magnetic field NED (gauss)
- [19 ... 21] Body magnetic field XYZ (gauss)
- [22 ... 23] Wind velocity NE (m/s)
- [24 ... 32] Not Used

State Variances

Refer to covariances[28] in [estimator_status](#). The index map for covariances[28] is as follows:

- [0 ... 3] Quaternions
- [4 ... 6] Velocity NED (m/s)²
- [7 ... 9] Position NED (m²)
- [10 ... 12] IMU delta angle bias XYZ (rad²)
- [13 ... 15] IMU delta velocity bias XYZ (m/s)²
- [16 ... 18] Earth magnetic field NED (gauss²)
- [19 ... 21] Body magnetic field XYZ (gauss²)
- [22 ... 23] Wind velocity NE (m/s)²
- [24 ... 28] Not Used

Observation Innovations

- Magnetometer XYZ (gauss) : Refer to mag_innov[3] in [ekf2_innovations](#).
- Yaw angle (rad) : Refer to heading_innov in [ekf2_innovations](#).

-
- Velocity and position innovations : Refer to vel_pos_innov[6] in [ekf2_innovations](#). The index map for vel_pos_innov[6] is as follows:
 - [0 ... 2] Velocity NED (m/s)
 - [3 ... 5] Position NED (m)
 - True Airspeed (m/s) : Refer to airspeed_innov in [ekf2_innovations](#).
 - Synthetic sideslip (rad) : Refer to beta_innov in [ekf2_innovations](#).
 - Optical flow XY (rad/sec) : Refer to flow_innov in [ekf2_innovations](#).
 - Height above ground (m) : Refer to hagl_innov in [ekf2_innovations](#).

Observation Innovation Variances

- Magnetometer XYZ (gauss²) : Refer to mag_innov_var[3] in [ekf2_innovations](#).
- Yaw angle (rad²) : Refer to heading_innov_var in the ekf2_innovations message.
- Velocity and position innovations : Refer to vel_pos_innov_var[6] in [ekf2_innovations](#). The index map for vel_pos_innov_var[6] is as follows:
 - [0 ... 2] Velocity NED (m/s)²
 - [3 ... 5] Position NED (m²)
- True Airspeed (m/s)² : Refer to airspeed_innov_var in [ekf2_innovations](#).
- Synthetic sideslip (rad²) : Refer to beta_innov_var in [ekf2_innovations](#).
- Optical flow XY (rad/sec)² : Refer to flow_innov_var in [ekf2_innovations](#).
- Height above ground (m²) : Refer to hagl_innov_var in [ekf2_innovations](#).

Output Complementary Filter

The output complementary filter is used to propagate states forward from the fusion time horizon to current time. To check the magnitude of the angular, velocity and position tracking errors measured at the fusion time horizon, refer to output_tracking_error[3] in the ekf2_innovations message. The index map is as follows:

- [0] Angular tracking error magnitude (rad)
- [1] Velocity tracking error magnitude (m/s). The velocity tracking time constant can be adjusted using the [EKF2_TAU_VEL](#) parameter. Reducing this parameter reduces steady state errors but increases the amount of observation noise on the NED velocity outputs.
- [2] Position tracking error magnitude (m). The position tracking time constant can be adjusted using the [EKF2_TAU_POS](#) parameter. Reducing this parameter reduces steady state errors but increases the amount of observation noise on the NED position outputs.

EKF Errors

The EKF contains internal error checking for badly conditioned state and covariance updates. Refer to the filter_fault_flags in [estimator_status](#).

Observation Errors

There are two categories of observation faults:

- Loss of data. An example of this is a range finder failing to provide a return.
- The innovation, which is the difference between the state prediction and sensor observation is excessive. An example of this is excessive vibration causing a large vertical position error, resulting in the barometer height measurement being rejected.

Both of these can result in observation data being rejected for long enough to cause the EKF to attempt a reset of the states using the sensor observations. All observations have a statistical confidence checks applied to the innovations. The number of standard deviations for the check are controlled by the EKF2_*_GATE parameter for each observation type.

Test levels are available in [estimator_status](#) as follows:

- mag_test_ratio : ratio of the largest magnetometer innovation component to the innovation test limit
- vel_test_ratio : ratio of the largest velocity innovation component to the innovation test limit
- pos_test_ratio : ratio of the largest horizontal position innovation component to the innovation test limit
- hgt_test_ratio : ratio of the vertical position innovation to the innovation test limit
- tas_test_ratio : ratio of the true airspeed innovation to the innovation test limit
- hagl_test_ratio : ratio of the height above ground innovation to the innovation test limit

For a binary pass/fail summary for each sensor, refer to innovation_check_flags in [estimator_status](#).

GPS Quality Checks

The EKF applies a number of GPS quality checks before commencing GPS aiding. These checks are controlled by the [EKF2_GPS_CHECK](#) and EKF2_REQ_* parameters. The pass/fail status for these checks is logged in the [estimator_status.gps_check_fail_flags](#) message. This integer will be zero when all required GPS checks have passed. If the EKF is not commencing GPS alignment,

check the value of the integer against the bitmask definition `gps_check_fail_flags` in [estimator_status](#).

EKF Numerical Errors

The EKF uses single precision floating point operations for all of its computations and first order approximations for derivation of the covariance prediction and update equations in order to reduce processing requirements. This means that it is possible when re-tuning the EKF to encounter conditions where the covariance matrix operations become badly conditioned enough to cause divergence or significant errors in the state estimates.

To prevent this, every covariance and state update step contains the following error detection and correction steps:

- If the innovation variance is less than the observation variance (this requires a negative state variance which is impossible) or the covariance update will produce a negative variance for any of the states, then:
 - The state and covariance update is skipped
 - The corresponding rows and columns in the covariance matrix are reset
 - The failure is recorded in the [estimator_status](#) `filter_fault_flags` message
- State variances (diagonals in the covariance matrix) are constrained to be non-negative.
- An upper limit is applied to state variances.
- Symmetry is forced on the covariance matrix.

After re-tuning the filter, particularly re-tuning that involve reducing the noise variables, the value of `estimator_status.gps_check_fail_flags` should be checked to ensure that it remains zero.

What should I do if the height estimate is diverging?

The most common cause of EKF height diverging away from GPS and altimeter measurements during flight is clipping and/or aliasing of the IMU measurements caused by vibration. If this is occurring, then the following signs should be evident in the data

- [ekf2_innovations](#).`vel_pos_innov[3]` and [ekf2_innovations](#).`vel_pos_innov[5]` will both have the same sign.
- [estimator_status](#).`hgt_test_ratio` will be greater than 1.0

The recommended first step is to ensure that the autopilot is isolated from the airframe using an effective isolation mounting system. An isolation mount has 6 degrees of freedom, and therefore 6 resonant frequencies. As a general rule, the 6 resonant

frequencies of the autopilot on the isolation mount should be above 25Hz to avoid interaction with the autopilot dynamics and below the frequency of the motors.

An isolation mount can make vibration worse if the resonant frequencies coincide with motor or propeller blade passage frequencies.

The EKF can be made more resistant to vibration induced height divergence by making the following parameter changes:

- Double the value of the innovation gate for the primary height sensor. If using barometric height this is [EKF2_BARO_GATE](#).
- Increase the value of [EKF2_ACC_NOISE](#) to 0.5 initially. If divergence is still occurring, increase in further increments of 0.1 but do not go above 1.0

Note that the effect of these changes will make the EKF more sensitive to errors in GPS vertical velocity and barometric pressure.

What should I do if the position estimate is diverging?

The most common causes of position divergence are:

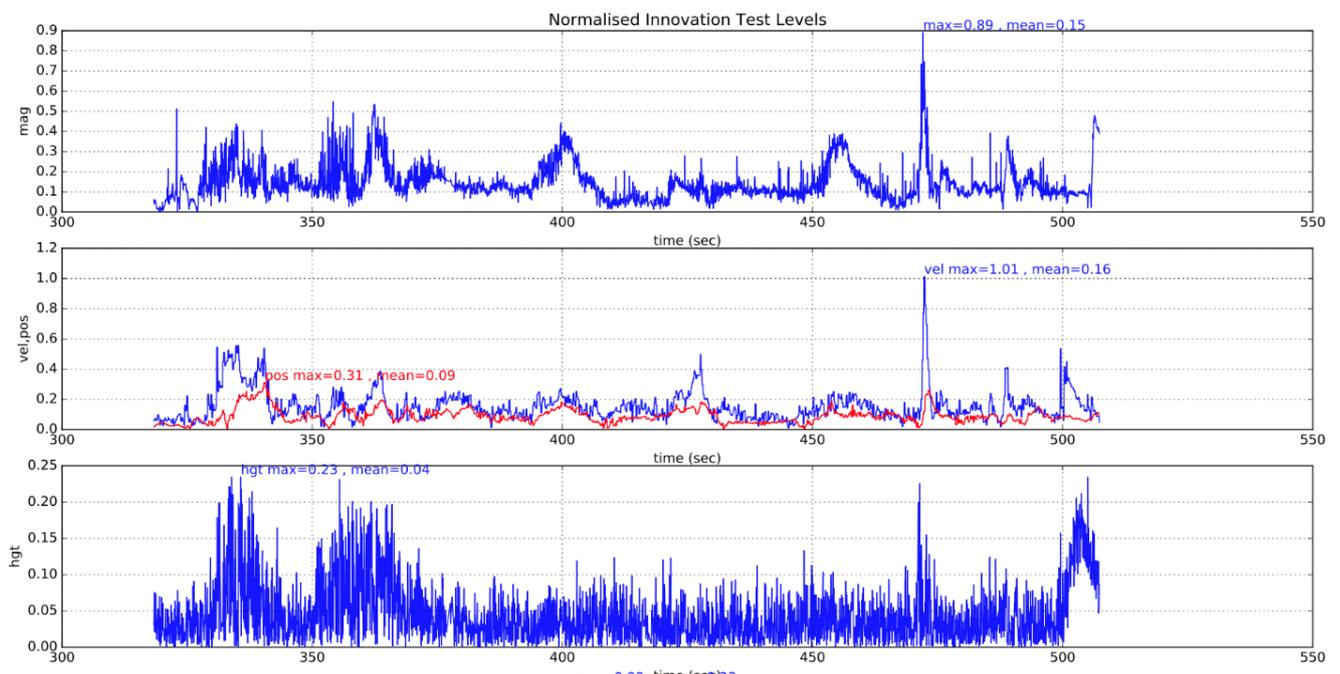
- High vibration levels.
 - Fix by improving mechanical isolation of the autopilot.
 - Increasing the value of [EKF2_ACC_NOISE](#) and [EKF2_GYR_NOISE](#) can help, but does make the EKF more vulnerable to GPS glitches.
- Large gyro bias offsets.
 - Fix by re-calibrating the gyro. Check for excessive temperature sensitivity (> 3 deg/sec bias change during warm-up from a cold start and replace the sensor if affected of insulate to to slow the rate of temperature change.
- Bad yaw alignment
 - Check the magnetometer calibration and alignment.
 - Check the heading shown QGC is within within 15 deg truth
- Poor GPS accuracy
 - Check for interference
 - Improve separation and shielding
 - Check flying location for GPS signal obstructions and reflectors (nearby tall buildings)
- Loss of GPS

Determining which of these is the primary cause requires a methodical approach to analysis of the EKF log data:

- Plot the velocity innovation test ratio - [estimator_status.vel_test_ratio](#)
- Plot the horizontal position innovation test ratio - [estimator_status.pos_test_ratio](#)

- Plot the height innovation test ratio - `estimator_status.hgt_test_ratio`
- Plot the magnetometer innovation test ratio - `estimator_status.mag_test_ratio`
- Plot the GPS receiver reported speed accuracy - `vehicle_gps_position.s_variance_m_s`
- Plot the IMU delta angle state estimates - `estimator_status.states[10]`, `states[11]` and `states[12]`
- Plot the EKF internal high frequency vibration metrics:
 - Delta angle coning vibration - `estimator_status.vibe[0]`
 - High frequency delta angle vibration - `estimator_status.vibe[1]`
 - High frequency delta velocity vibration - `estimator_status.vibe[2]`

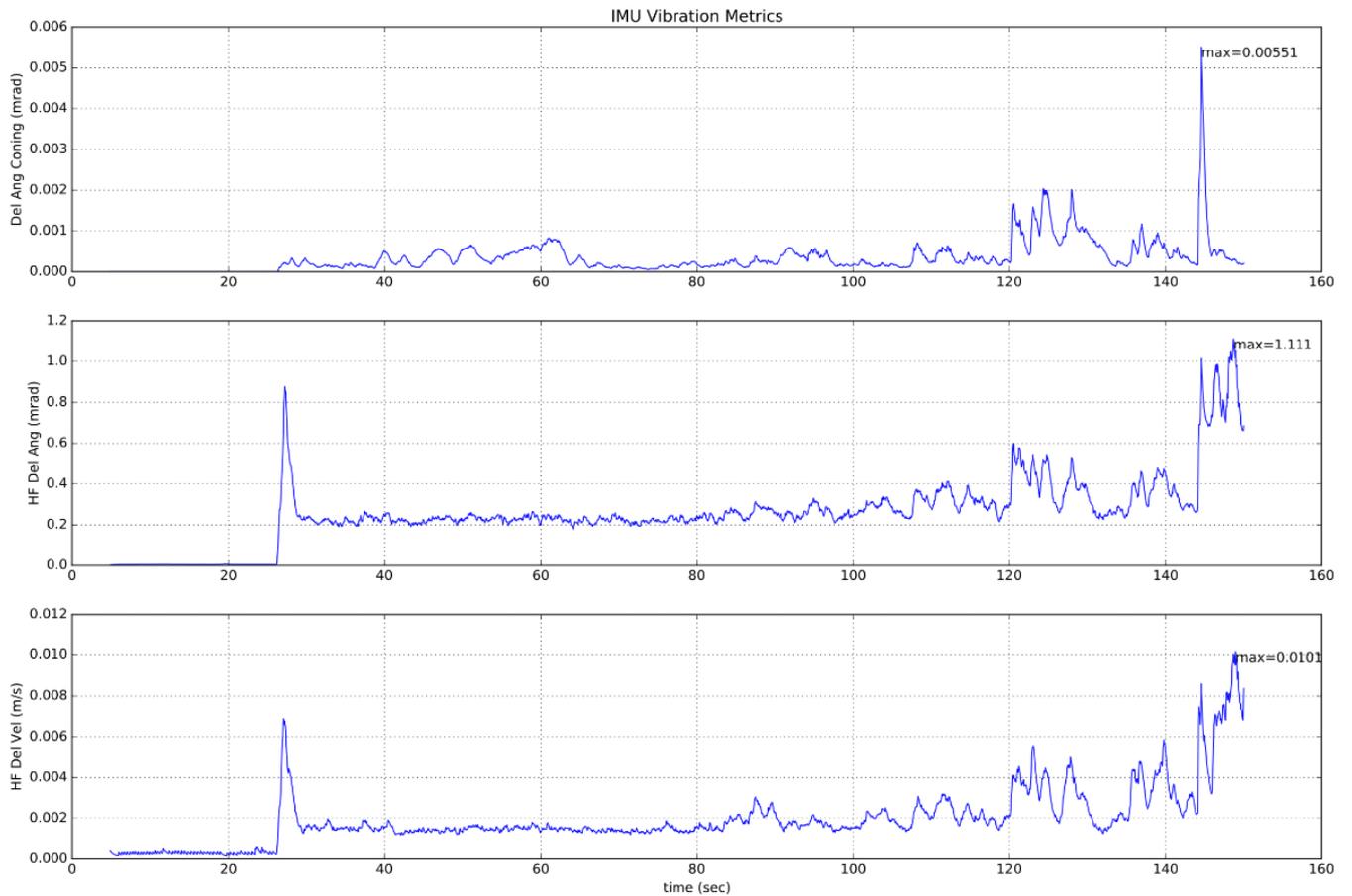
During normal operation, all the test ratios should remain below 0.5 with only occasional spikes above this as shown in the example below from a successful flight:



The following plot shows the EKF vibration metrics for a multirotor with good isolation. The landing shock and the increased vibration during takeoff and landing can be seen. Insufficient data has been gathered with these metrics to provide specific advice on maximum thresholds.

The below vibration metrics are of limited value as the presence of vibration at a frequency close to the IMU sampling frequency (1 kHz for most boards) will cause offsets to appear in the data that do not show up in the high frequency vibration metrics. The only way to detect aliasing errors is in their effect on inertial navigation accuracy and the rise in innovation levels.

In addition to generating large position and velocity test ratios of > 1.0 , the different error mechanisms affect the other test ratios in different ways:



Determination of Excessive Vibration

High vibration levels normally affect vertical position and velocity innovations as well as the horizontal components. Magnetometer test levels are only affected to a small extent.

(insert example plots showing bad vibration here)

Determination of Excessive Gyro Bias

Large gyro bias offsets are normally characterised by a change in the value of delta angle bias greater than $5E-4$ during flight (equivalent to ~ 3 deg/sec) and can also cause a large increase in the magnetometer test ratio if the yaw axis is affected. Height is normally unaffected other than extreme cases. Switch on bias value of up to 5 deg/sec can be tolerated provided the filter is given time to settle before flying. Pre-flight checks performed by the commander should prevent arming if the position is diverging.

(insert example plots showing bad gyro bias here)

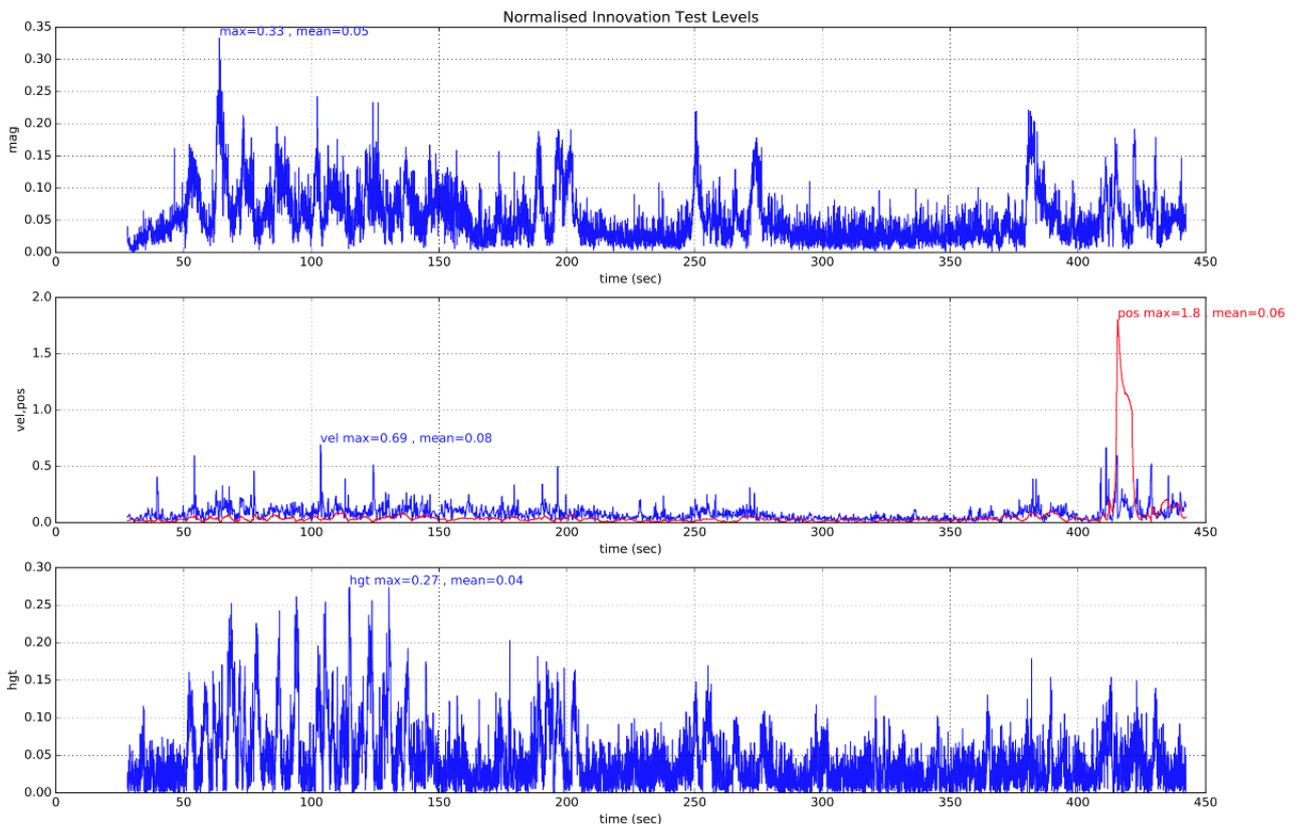
Determination of Poor Yaw Accuracy

Bad yaw alignment causes a velocity test ratio that increases rapidly when the vehicle starts moving due inconsistency in the direction of velocity calculated by the inertial nav and the GPS measurement. Magnetometer innovations are slightly affected. Height is normally unaffected.

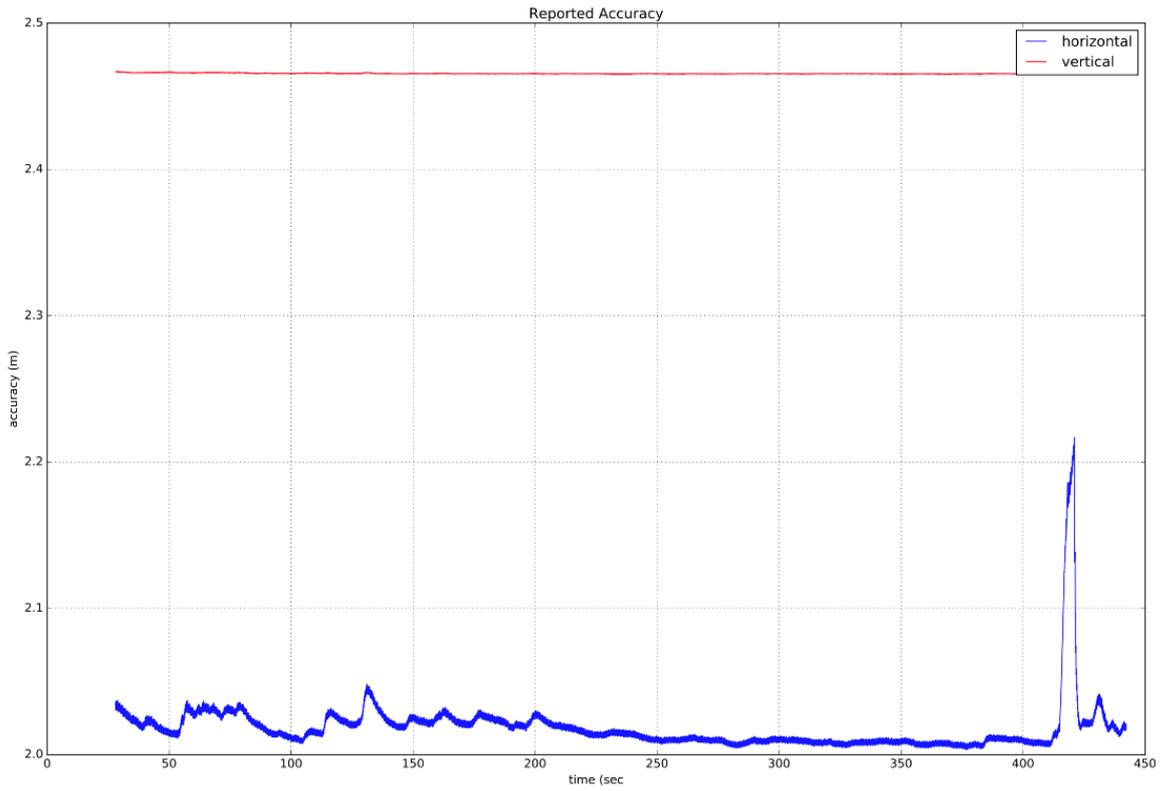
(insert example plots showing bad yaw alignment here)

Determination of Poor GPS Accuracy

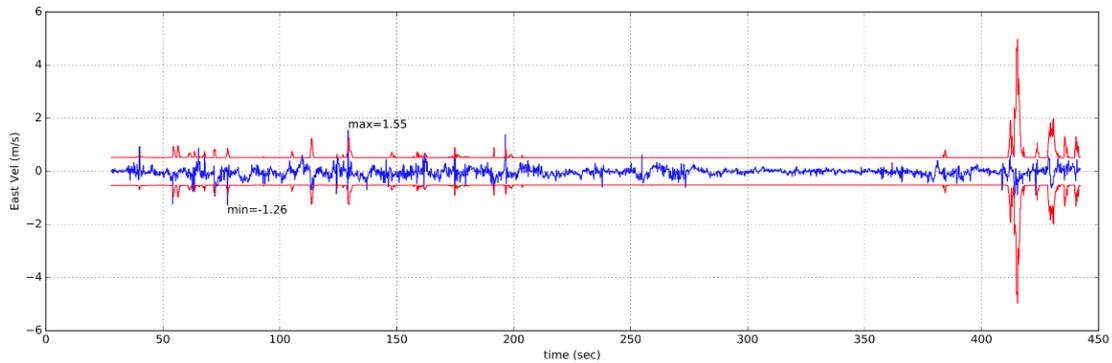
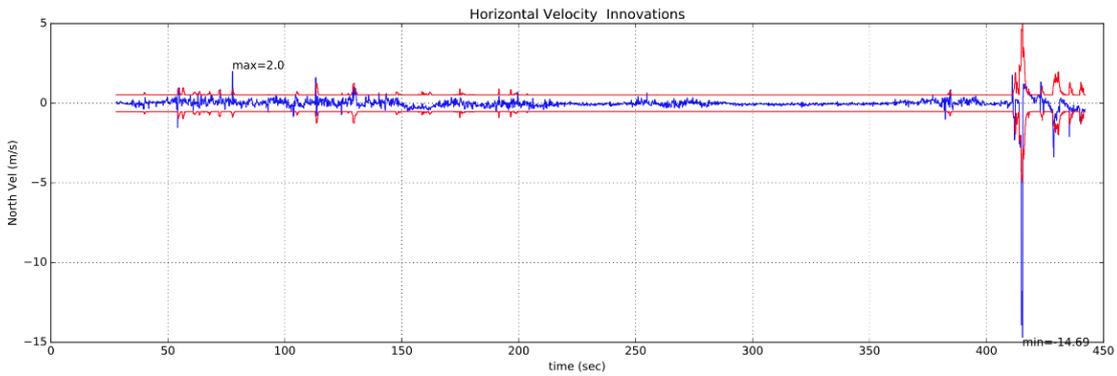
Poor GPS accuracy is normally accompanied by a rise in the reported velocity error of the receiver in conjunction with a rise in innovations. Transient errors due to multipath, obscuration and interference are more common causes. Here is an example of a temporary loss of GPS accuracy where the multi-rotor started drifting away from its loiter location and had to be corrected using the sticks. The rise in [estimator_status.vel_test_ratio](#) to greater than 1 indicates the GPs velocity was inconsistent with other measurements and has been rejected.



This is accompanied with rise in the GPS receivers reported velocity accuracy which indicates that it was likely a GPS error.



If we also look at the GPS horizontal velocity innovations and innovation variances, we can see the large spike in North velocity innovation that accompanies this GPS 'glitch' event.



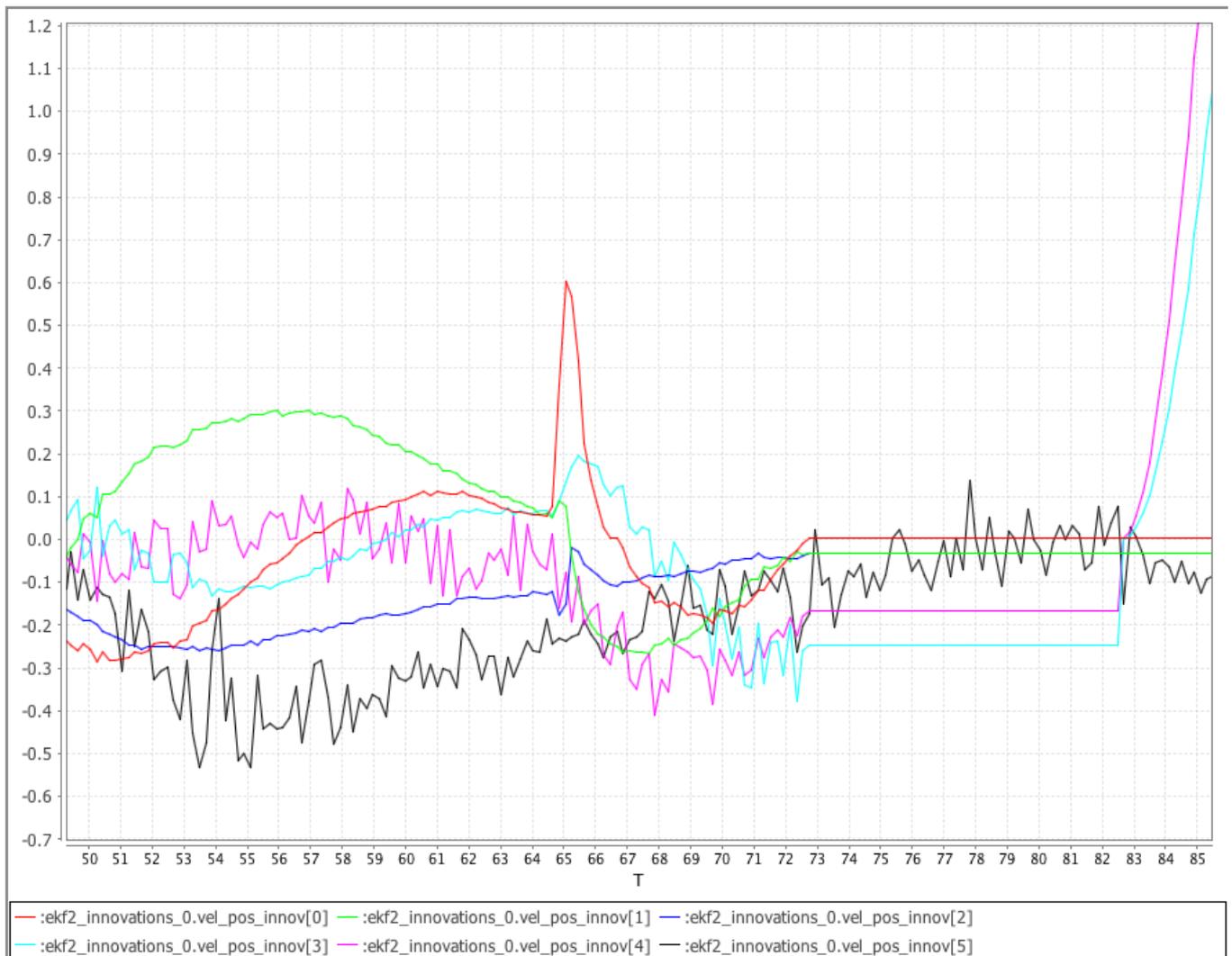
Determination of GPS Data Loss

Loss of GPS data will be shown by the velocity and position innovation test ratios 'flat-lining'. If this occurs, check the other GPS status data in `vehicle_gps_position` for further information.

The following plot shows the NED GPS velocity

innovations `ekf2_innovations_0.vel_pos_innov[0 ... 2]`, the GPS NE position innovations `ekf2_innovations_0.vel_pos_innov[3 ... 4]` and the Baro vertical position innovation `ekf2_innovations_0.vel_pos_innov[5]` generated from a simulated VTOL flight using SITL Gazebo.

The simulated GPS was made to lose lock at 73 seconds. Note the NED velocity innovations and NE position innovations 'flat-line' after GPS is lost. Note that after 10 seconds without GPS data, the EKF reverts back to a static position mode using the last known position and the NE position innovations start to change again.



10.6 Preflight Sensor and EKF Checks

The commander module performs a number of preflight sensor quality and EKF checks which are controlled by the COM_ARM<> parameters. If these checks fail, the motors are prevented from arming and the following error messages are produced:

- **PREFLIGHT FAIL: EKF HGT ERROR**
 - This error is produced when the IMU and height measurement data are inconsistent.
 - Perform an accel and gyro calibration and restart the vehicle. If the error persists, check the height sensor data for problems.
 - The check is controlled by the COM_ARM_EKF_HGT parameter.
- **PREFLIGHT FAIL: EKF VEL ERROR**
 - This error is produced when the IMU and GPS velocity measurement data are inconsistent.
 - Check the GPS velocity data for un-realistic data jumps. If GPS quality looks OK, perform an accel and gyro calibration and restart the vehicle.
 - The check is controlled by the COM_ARM_EKF_VEL parameter.
- **PREFLIGHT FAIL: EKF HORIZ POS ERROR**
 - This error is produced when the IMU and position measurement data (either GPS or external vision) are inconsistent.
 - Check the position sensor data for un-realistic data jumps. If data quality looks OK, perform an accel and gyro calibration and restart the vehicle.
 - The check is controlled by the COM_ARM_EKF_POS parameter.
- **PREFLIGHT FAIL: EKF YAW ERROR**
 - This error is produced when the yaw angle estimated using gyro data and the yaw angle from the magnetometer or external vision system are inconsistent.
 - Check the IMU data for large yaw rate offsets and check the magnetometer alignment and calibration.
 - The check is controlled by the COM_ARM_EKF_POS parameter
- **PREFLIGHT FAIL: EKF HIGH IMU ACCEL BIAS**
 - This error is produced when the IMU accelerometer bias estimated by the EKF is excessive.
 - The check is controlled by the COM_ARM_EKF_AB parameter.
- **PREFLIGHT FAIL: EKF HIGH IMU GYRO BIAS**
 - This error is produced when the IMU gyro bias estimated by the EKF is excessive.
 - The check is controlled by the COM_ARM_EKF_GB parameter.
- **PREFLIGHT FAIL: ACCEL SENSORS INCONSISTENT - CHECK CALIBRATION**
 - This error message is produced when the acceleration measurements from different IMU units are inconsistent.
 - This check only applies to boards with more than one IMU.
 - The check is controlled by the COM_ARM_IMU_ACC parameter.
- **PREFLIGHT FAIL: GYRO SENSORS INCONSISTENT - CHECK CALIBRATION**
 - This error message is produced when the angular rate measurements from different IMU units are inconsistent.

-
- This check only applies to boards with more than one IMU.
 - The check is controlled by the COM_ARM_IMU_GYR parameter.

COM_ARM_WO_GPS

The COM_ARM_WO_GPS parameter controls whether arming is allowed without a GPS signal. This parameter must be set to 0 to allow arming when there is no GPS signal present. Arming without GPS is only allowed if the flight mode selected does not require GPS.

COM_ARM_EKF_POS

The COM_ARM_EKF_POS parameter controls the maximum allowed inconsistency between the EKF inertial measurements and position reference (GPS or external vision). The default value of 0.5 allows the differences to be no more than 50% of the maximum tolerated by the EKF and provides some margin for error increase when flight commences.

COM_ARM_EKF_VEL

The COM_ARM_EKF_VEL parameter controls the maximum allowed inconsistency between the EKF inertial measurements and GPS velocity measurements. The default value of 0.5 allows the differences to be no more than 50% of the maximum tolerated by the EKF and provides some margin for error increase when flight commences.

COM_ARM_EKF_HGT

The COM_ARM_EKF_HGT parameter controls the maximum allowed inconsistency between the EKF inertial measurements and height measurement (Baro, GPS, range finder or external vision). The default value of 0.5 allows the differences to be no more than 50% of the maximum tolerated by the EKF and provides some margin for error increase when flight commences.

COM_ARM_EKF_YAW

The COM_ARM_EKF_YAW parameter controls the maximum allowed inconsistency between the EKF inertial measurements and yaw measurement (magnetometer or external vision). The default value of 0.5 allows the differences to be no more than 50% of the maximum tolerated by the EKF and provides some margin for error increase when flight commences.

COM_ARM_EKF_AB

The COM_ARM_EKF_AB parameter controls the maximum allowed EKF estimated IMU accelerometer bias. The default value of 0.005 allows for up to 0.5 m/s/s of accelerometer bias.

COM_ARM_EKF_GB

The COM_ARM_EKF_GB parameter controls the maximum allowed EKF estimated IMU gyro bias. The default value of 0.00087 allows for up to 5 deg/sec of switch on gyro bias.

COM_ARM_IMU_ACC

The COM_ARM_IMU_ACC parameter controls the maximum allowed inconsistency in acceleration measurements between the default IMU used for flight control and other IMU units if fitted.

COM_ARM_IMU_GYR

The COM_ARM_IMU_GYR parameter controls the maximum allowed inconsistency in angular rate measurements between the default IMU used for flight control and other IMU units if fitted.

10.7 Land Detector Configuration

The land detector is a dynamic vehicle model representing key vehicle states such as landed and ground contact.

Configuring Auto-Disarming

By default the land detector does detect landing, but does not auto-disarm. If the hysteresis parameter [COM_DISARM_LAND](#) is set to a non-zero value the system will auto-disarm after N seconds (the value it is set to).

Multicopter Land Detector Configuration

The complete set of parameters is available in the *QGroundControl* parameter editor under the `LNDMCP` prefix. The key parameters that might differ per airframe are these:

-
- [MPC_THR_HOVER](#) - the hover throttle of the system (in percent, default is 50%). It is important to set this correctly as not only does it make the altitude control more accurate, but it also ensures correct land detection. A racer or a big camera drone without payload mounted might need a much lower setting (e.g. 35%).
 - [MPC_THR_MIN](#) - the overall minimum throttle of the system. This should be set to enable a controlled descent.
 - [LNDMC_THR_RANGE](#) - this is a scaling factor to define the range between min and hover throttle that gets accepted as landed. Example: If the minimum throttle is 0.1, the hover throttle is 0.5 and the range is 0.2 (20%), then the highest throttle value that counts as landed is: $0.1 + (0.5 - 0.1) * 0.2 = 0.18$.

Land detector states

In order to detect landing, the multicopter first has to go through three different states, where each state contains the conditions from the previous states plus tighter constraints. If a condition cannot be reached because of missing sensors, then the condition is true by default. For instance, in Acro mode and no sensor is active except for the gyro sensor, then the detection solely relies on thrust output and time.

In order to proceed to the next state, each condition has to be true for some predefined time. If one condition fails, the land detector drops out of the current state immediately.

Ground contact

This state is reached if following conditions are true for 0.35 seconds:

- no vertical movement ([LNDMC_Z_VEL_MAX](#))
- no horizontal movement ([LNDMC_XY_VEL_MAX](#))
- low thrust $MPC_THR_MIN + (MPC_THR_HOVER - MPC_THR_MIN) * 0.3$ or velocity setpoint is 0.9 of land speed but vehicle has no vertical movement.

If the vehicle is in position- or velocity-control and ground contact was detected, the position controller will set the thrust vector along the body x-y-axis to zero.

Maybe landed

This state is reached if following conditions are true for 0.25 seconds:

- all conditions of ground contact are true
- is not rotating ([LNDMC_ROT_MAX](#))
- has low thrust $MPC_THR_MIN + (MPC_THR_HOVER - MPC_THR_MIN) * LNDMC_THR_RANGE$;

If the vehicle only has knowledge of thrust and angular rate, in order to proceed to the next state the vehicle has to have low thrust and no rotation for 8.0 seconds.

If the vehicle is in position or velocity control and maybe landed was detected, the position controller will set the thrust vector to zero.

Landed

This state is reached if following conditions are true for 0.3 seconds:

- all conditions of maybe landed are true

Fixed Wing Land Detector Configuration

The complete set of parameters is available under the `LNDFW` prefix. These two user parameters are sometimes worth tuning:

- [LNDFW_AIRSPD_MAX](#) - the maximum airspeed allowed for the system still to be considered landed. The default of 8 m/s is a reliable tradeoff between airspeed sensing accuracy and triggering fast enough. Better airspeed sensors should allow lower values of this parameter.
- [LNDFW_VELI_MAX](#) - the maximum velocity for the system to be still considered landed. This parameter can be adjusted to ensure a sooner or later land detection on throwing the airframe for hand-launches.

10.8 Flying with Motion Capture (VICON, Optitrack)

WORK IN PROGRESS

Indoor motion capture systems like VICON and Optitrack can be used to provide position and attitude data for vehicle state estimation, or to serve as ground-truth for analysis. The motion capture data can be used to update PX4's local position estimate relative to the local origin. Heading (yaw) from the motion capture system can also be optionally integrated by the attitude estimator.

Pose (position and orientation) data from the motion capture system is sent to the autopilot over MAVLink, using the [ATT_POS_MOCAP](#) message. See the section below on coordinate frames for data representation conventions. The [mavros](#) ROS-Mavlink interface has a default plugin to send this message. They can also be sent using pure C/C++ code and direct use of the MAVLink library.

Computing Architecture

It is **highly recommended** that you send motion capture data via an **onboard** computer (e.g Raspberry Pi, ODroid, etc.) for reliable communications. The onboard computer can be connected to the motion capture computer through WiFi, which offers reliable, high-bandwidth connection.

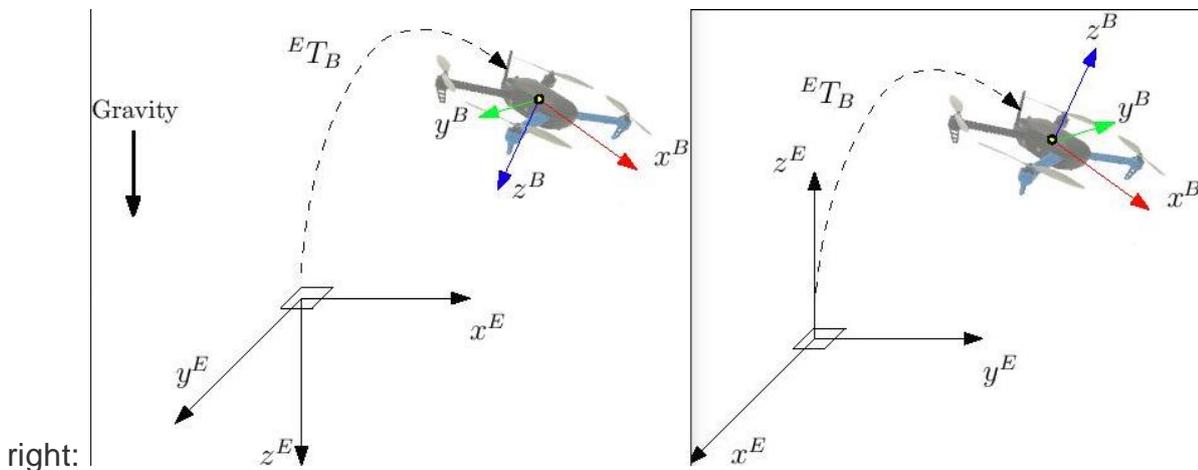
Most standard telemetry links like 3DR/SiK radios are **not** suitable for high-bandwidth motion capture applications.

Coordinate Frames

This section shows how to setup the system with the proper reference frames. There are various representations but we will use two of them: ENU and NED.

- ENU is a ground-fixed frame where **X** axis points East, **Y** points North and **Z** up. The robot/vehicle body frame is **X** towards the front, **Z** up and **Y** towards the left.
- NED has **X** towards North, **Y** East and **Z** down. The robot/vehicle body frame has **X** towards the front, **Z** down and **Y** accordingly.

Frames are shown in the image below. NED on the left, ENU on the



With the external heading estimation, however, magnetic North is ignored and faked with a vector corresponding to world x axis (which can be placed freely at mocap calibration); yaw angle will be given respect to local x .

When creating the rigid body in the motion capture software, remember to first align the robot with the world **X** axis otherwise yaw estimation will have an initial offset.

Estimator choice

LPE and Attitude Estimator Q

EKF2

The ROS topic for motion capture is `mocap_pose_estimate` for mocap systems and `vision_pose_estimate` for vision. Check [mavros extras](#) for further info.

10.9 S.Bus Driver for Linux

The *S.Bus Driver for Linux* allows a Linux-based autopilot to access up to 16 channels from a *Futaba S.Bus receiver* via a serial port. The driver should also work with other receivers that use the S.Bus protocol, including as FrSky, RadioLink, and even S.Bus encoders.

A signal inverter circuit is required (described below) to enable the device serial port to read data from the receiver.

The driver has been tested on Raspberry Pi running Rasbian Linux, when connected to the receiver through the onboard serial port or via a USB to TTY serial cable. It is expected to work on all Linux versions, and through all serial ports.

Signal inverter circuit

S.Bus is an *inverted* UART communication signal. As many serial ports/flight controllers cannot read an inverted UART signal, a signal inverter circuit is required between the receiver and serial port un-invert the signal. This section shows how to create an appropriate circuit.

This circuit is required for Raspberry Pi to read S.Bus remote control signals through the serial port or USB-to-TTY serial converter. It will also be required for many other flight controllers.

Required components

- 1x NPN transistor (e.g. NPN S9014 TO92)
- 1x 10K resistor
- 1x 1K resistor

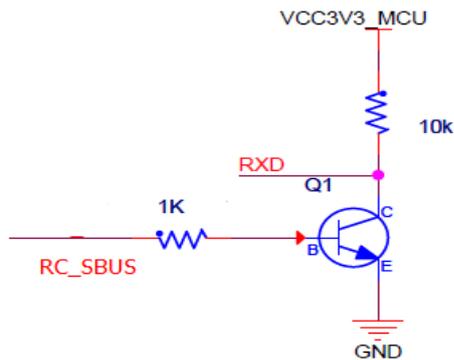
Any type/model of transistor can be used because the current drain is very low.

Raspberry Pi only has a single serial port. If this is already being used you can alternatively connect your S.Bus receiver to the RaPi USB port, via a USB to TTY serial cable (e.g. PL2302 USB to TTL serial converter)

Circuit diagram/Connections

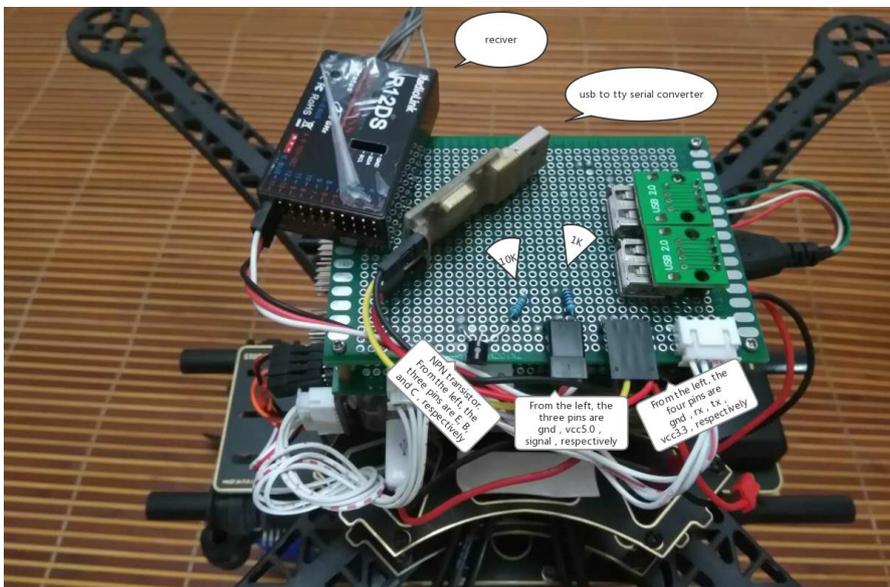
Connect the components as described below (and shown in the circuit diagram):

- S.Bus signal → 1K resistor → NPN transistor base
- NPN transistor emit → GND
- 3.3VCC → 10K resistor → NPN transistor collection → USB-to-TTY rxd
- 5.0VCC → S.Bus VCC
- GND → S.Bus GND



Breadboard image

The image below shows the connections on a breadboard.



Source code

- [Firmware/src/drivers/linux_sbush](#)

Usage

The command syntax is:

```
linux_sbus start|stop|status -d <device> -c <channel>
```

So for example, to automatically start the driver listening to 8 channels on device `/dev/ttyUSB0`, you would add the following line to the startup configuration file.

```
linux_sbus start -d /dev/ttyUSB0 -c 8
```

The original configuration files are located in **Firmware/posix-configs**. According to the official documentation, after you finish `make upload` related operations, all posix related configuration files will be placed in **/home/pi**. You can modify the file you want to use there.

11 Advanced Topics

11.1 System Startup

The PX4 boot is controlled by shell scripts in the [ROMFS/px4fmu_common/init.d](#) folder.

All files starting with a number and underscore (e.g. `10000_airplane`) are canned airframe configurations. They are exported at build-time into an `airframes.xml` file which is parsed by [QGroundControl](#) for the airframe selection UI. Adding a new configuration is covered [here](#).

The remaining files are part of the general startup logic, and the first executed file is the [rcS](#) script, which calls all other scripts.

Debugging the System Boot

A failure of a driver or software component can lead to an aborted boot.

An incomplete boot often materializes as missing parameters in the ground control stations, because the non-started applications did not initialize their parameters.

The right approach to debug the boot sequence is to connect the [system console](#) and power-cycle the board. The resulting boot log has detailed information about the boot sequence and should contain hints why the boot aborted.

Common boot failure causes

- A required sensor failed to start
- For custom applications: The system was out of RAM. Run the `free` command to see the amount of free RAM.
- A software fault or assertion resulting in a stack trace

Replacing the System Startup

In most cases customizing the default boot is the better approach, which is documented below. If the complete boot should be replaced, create a file `/fs/microsd/etc/rc.txt`, which is located in the `etc` folder on the microSD card. If this file is present nothing in the system will be auto-started.

Customizing the System Startup

The best way to customize the system startup is to introduce a [new airframe configuration](#). If only tweaks are wanted (like starting one more application or just using a different mixer) special hooks in the startup can be used.

The system boot files are UNIX FILES which require UNIX LINE ENDINGS. If editing on Windows use a suitable editor.

There are three main hooks. Note that the root folder of the microsd card is identified by the path `/fs/microsd`.

- `/fs/microsd/etc/config.txt`
- `/fs/microsd/etc/extras.txt`
- `/fs/microsd/etc/mixers/NAME_OF_MIXER`

Customizing the Configuration (config.txt)

The `config.txt` file is loaded after the main system has been configured and *before* it is booted and allows to modify shell variables.

Starting additional applications

The `extras.txt` can be used to start additional applications after the main system boot. Typically these would be payload controllers or similar optional custom components.

Calling an unknown command in system boot files may result in boot failure. Typically the system does not stream mavlink messages after boot failure, in this case check the error messages that are printed on the system console.

The following example shows how to start custom applications:

- Create a file on the SD card `etc/extras.txt` with this content:

```
custom_app start
```

- A command can be made optional by gating it with the `set +e` and `set -e` commands:

```
set +e
```

```
optional_app start      # Will not result in boot failure if optional_app is unknown  
or fails
```

```
set -e
```

```
mandatory_app start    # Will abort boot if mandatory_app is unknown or fails
```

Starting a custom mixer

By default the system loads the mixer from `/etc/mixers`. If a file with the same name exists in `/fs/microsd/etc/mixers` this file will be loaded instead. This allows to customize the mixer file without the need to recompile the Firmware.

Example

The following example shows how to add a custom aux mixer:

- Create a file on the SD card, `etc/mixers/gimbal.aux.mix` with your mixer content.
- Then to use it, create an additional file `etc/config.txt` with this content:

```
set MIXER_AUX gimbal
set PWM_AUX_OUT 1234
set PWM_AUX_DISARMED 1500
set PWM_AUX_MIN 1000
set PWM_AUX_MAX 2000
set PWM_AUX_RATE 50
```

11.2 Parameters & Configurations

The PX4 platform uses the param subsystem (a flat table of float and int32_t values) and text files (for mixers and startup scripts) to store its configuration.

The [system startup](#) and how [airframe configurations](#) work are detailed on other pages. This section discusses the param subsystem in detail

Command Line usage

The PX4 [system console](#) offers the `param` tool, which allows to set parameters, read their value, save them and export and restore to and from files.

Getting and Setting Parameters

The `param show` command lists all system parameters:

```
param show
```

To be more selective a partial parameter name with wildcard can be used:

```
nsh> param show RC_MAP_A*
Symbols: x = used, + = saved, * = unsaved
```

```
x RC_MAP_AUX1 [359,498] : 0
x RC_MAP_AUX2 [360,499] : 0
x RC_MAP_AUX3 [361,500] : 0
x RC_MAP_ACRO_SW [375,514] : 0
```

```
723 parameters total, 532 used.
```

Exporting and Loading Parameters

The standard save command will store the parameters in the current default file:

```
param save
```

If provided with an argument, it will store the parameters instead to this new location:

```
param save /fs/microsd/vtol_param_backup
```

There are two different commands to load parameters: `param load` will load a file and replace the current parameters with what is in this file, resulting in a 1:1 copy of the state that was present when these parameters were stored. `param import` is more subtle: It will only change parameter values which have been changed from the default. This is great to e.g. initially calibrate a board (without configuring it further), then importing the calibration data without overwriting the rest of the system configuration.

Overwrite the current parameters:

```
param load /fs/microsd/vtol_param_backup
```

Merge the current parameters with stored parameters (stored values which are non-default take precedence):

```
param import /fs/microsd/vtol_param_backup
```

C / C++ API

There is also a C and a separate C++ API which can be used to access parameter values.

Discuss param C / C++ API.

```
int32_t param = 0;
param_get(param_find("PARAM_NAME"), &param);
```

Parameter Meta Data

PX4 uses an extensive parameter meta data system to drive the user-facing presentation of parameters. Correct meta data is critical for good user experience in the ground station.

A typical parameter metadata section will look like this:

```
/**
 * Pitch P gain
 *
 * Pitch proportional gain, i.e. desired angular speed in rad/s for error 1 rad.
 *
 * @unit 1/s
 * @min 0.0
 * @max 10
 * @decimal 2
 * @increment 0.0005
 * @reboot_required true
 * @group Multicopter Attitude Control
 */
```

```
PARAM_DEFINE_FLOAT(MC_PITCH_P, 6.5f);
```

Where each line has this use:

```
/**
 * <title>
 *
 * <longer description, can be multi-line>
 *
 * @unit <the unit, e.g. m for meters>
 * @min <the minimum sane value. Can be overridden by the user>
 * @max <the maximum sane value. Can be overridden by the user>
 * @decimal <the minimum sane value. Can be overridden by the user>
 * @increment <the "ticks" in which this value will increment in the UI>
 * @reboot_required true <add this if changing the param requires a system restart>
 * @group <a title for parameters which form a group>
 */
```

```
PARAM_DEFINE_FLOAT(MC_PITCH_P, 6.5f);
```

11.3 Parameter Reference

This list is auto-generated from the source code and contains the most recent parameter documentation.

Attitude Q estimator

The module where these parameters are defined is: *modules/attitude_estimator_q*.

Name	Description	Min > Max (Incr.)	Default	Units
ATT_ACC_COMP (INT32)	Acceleration compensation based on GPS velocity		1	
ATT_BIAS_MAX (FLOAT)	Gyro bias limit	0 > 2	0.05	rad/s
ATT_EXT_HDG_M (INT32)	External heading usage mode (from Motion capture/Vision) Set to 1 to use heading estimate from vision. Set to 2 to use heading from motion capture Values: <ul style="list-style-type: none">• 0: None• 1: Vision• 2: Motion Capture	0 > 2	0	
ATT_MAG_DECL (FLOAT)	Magnetic declination, in degrees Comment: This parameter is not used in normal operation, as the declination is looked up based on the GPS coordinates of the vehicle.		0.0	deg
ATT_MAG_DECL_A (INT32)	Automatic GPS based declination compensation		1	
ATT_W_ACC (FLOAT)	Complimentary filter accelerometer weight	0 > 1	0.2	
ATT_W_EXT_HDG (FLOAT)	Complimentary filter external heading weight	0 > 1	0.1	
ATT_W_GYRO_BIAS (FLOAT)	Complimentary filter gyroscope bias weight	0 > 1	0.1	
ATT_W_MAG (FLOAT)	Complimentary filter magnetometer weight	0 > 1	0.1	

Battery Calibration

Name	Description	Min > Max (Incr.)	Default	Units
BAT_A_PER_V (FLOAT)	<p>Battery current per volt (A/V)</p> <p>Comment: The voltage seen by the 3.3V ADC multiplied by this factor will determine the battery current. A value of -1 means to use the board default.</p> <p>Module: modules/sensors</p>		-1.0	
BAT_CAPACITY (FLOAT)	<p>Battery capacity</p> <p>Comment: Defines the capacity of the attached battery.</p> <p>Reboot required: true</p> <p>Module: modules/systemlib</p>	-1.0 > 100000 (50)	-1.0	mAh
BAT_CNT_V_CURR (FLOAT)	<p>Scaling from ADC counts to volt on the ADC input (battery current)</p> <p>Comment: This is not the battery current, but the intermediate ADC voltage. A value of -1 signifies that the board defaults are used, which is highly recommended.</p> <p>Module: modules/sensors</p>		-1.0	

BAT_CNT_V_VOLT (FLOAT)	Scaling from ADC counts to volt on the ADC input (battery voltage) Comment: This is not the battery voltage, but the intermediate ADC voltage. A value of -1 signifies that the board defaults are used, which is highly recommended. Module: modules/sensors		-1.0	
BAT_CRIT_THR (FLOAT)	Critical threshold Comment: Sets the threshold when the battery will be reported as critically low. This has to be lower than the low threshold. This threshold commonly will trigger RTL. Reboot required: true Module: modules/systemlib	0.05 > 0.1 (0.01)	0.07	norm
BAT_EMERGEN_THR (FLOAT)	Emergency threshold Comment: Sets the threshold when the battery will be reported as dangerously low. This has to be lower than the critical threshold. This threshold commonly will trigger landing. Reboot required: true Module: modules/systemlib	0.03 > 0.07 (0.01)	0.05	norm
BAT_LOW_THR (FLOAT)	Low threshold Comment: Sets the threshold when the battery will be reported as low. This has to be higher than the critical threshold. Reboot required: true Module: modules/systemlib	0.12 > 0.4 (0.01)	0.15	norm

BAT_N_CELLS (INT32)	<p>Number of cells</p> <p>Comment: Defines the number of cells the attached battery consists of.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Unconfigured • 2: 2S Battery • 3: 3S Battery • 4: 4S Battery • 5: 5S Battery • 6: 6S Battery • 7: 7S Battery • 8: 8S Battery • 9: 9S Battery • 10: 10S Battery • 11: 11S Battery • 12: 12S Battery • 13: 13S Battery • 14: 14S Battery • 15: 15S Battery • 16: 16S Battery <p>Reboot required: true</p> <p>Module: modules/systemlib</p>		0	S
BAT_R_INTERNAL (FLOAT)	<p>Explicitly defines the per cell internal resistance</p> <p>Comment: If non-negative, then this will be used in place of BAT_V_LOAD_DROP for all calculations.</p> <p>Reboot required: true</p> <p>Module: modules/systemlib</p>	-1.0 > 0.2	-1.0	Ohms
BAT_SOURCE (INT32)	<p>Battery monitoring source</p> <p>Comment: This parameter controls the source of battery data. The value 'Power Module' means that measurements are expected to come from a power module. If the value is set to 'External' then the system expects to receive mavlink battery status messages.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Power Module • 1: External <p>Module: modules/sensors</p>	0 > 1	0	

BAT_V_CHARGED (FLOAT)	<p>Full cell voltage (5C load)</p> <p>Comment: Defines the voltage where a single cell of the battery is considered full under a mild load. This will never be the nominal voltage of 4.2V</p> <p>Reboot required: true</p> <p>Module: modules/systemlib</p>	(0.01)	4.05	V
BAT_V_DIV (FLOAT)	<p>Battery voltage divider (V divider)</p> <p>Comment: This is the divider from battery voltage to 3.3V ADC voltage. If using e.g. Mauch power modules the value from the datasheet can be applied straight here. A value of -1 means to use the board default.</p> <p>Module: modules/sensors</p>		-1.0	
BAT_V_EMPTY (FLOAT)	<p>Empty cell voltage (5C load)</p> <p>Comment: Defines the voltage where a single cell of the battery is considered empty. The voltage should be chosen before the steep dropoff to 2.8V. A typical lithium battery can only be discharged down to 10% before it drops off to a voltage level damaging the cells.</p> <p>Reboot required: true</p> <p>Module: modules/systemlib</p>	(0.01)	3.5	V
BAT_V_LOAD_DROP (FLOAT)	<p>Voltage drop per cell on full throttle</p> <p>Comment: This implicitly defines the internal resistance to maximum current ratio and assumes linearity. A good value to use is the difference between the 5C and 20-25C load. Not used if BAT_R_INTERNAL is set.</p> <p>Reboot required: true</p> <p>Module: modules/systemlib</p>	0.07 > 0.5 (0.01)	0.3	V
BAT_V_OFFS_CURR (FLOAT)	<p>Offset in volt as seen by the ADC input of the current sensor</p> <p>Comment: This offset will be subtracted before calculating the battery current based on the voltage.</p> <p>Module: modules/sensors</p>		0.0	

Camera Control

The module where these parameters are defined is: *modules/camera_feedback*.

Name	Description	Min > Max (Incr.)	Default	Units
CAM_FBACK_MODE (INT32)	Camera feedback mode Comment: Sets the camera feedback mode. Values: <ul style="list-style-type: none">• 0: Disabled• 1: Feedback on trigger	0 > 1	0	

Camera trigger

The module where these parameters are defined is: *drivers/camera_trigger*.

Name	Description	Min > Max (Incr.)	Default	Units
TRIG_ACT_TIME (FLOAT)	Camera trigger activation time Comment: This parameter sets the time the trigger needs to be pulled high or low.	0.1 > 3000	40.0	ms
TRIG_DISTANCE (FLOAT)	Camera trigger distance Comment: Sets the distance at which to trigger the camera.	0 > ? (1)	25.0	m
TRIG_INTERFACE (INT32)	Camera trigger Interface Comment: Selects the trigger interface Values: <ul style="list-style-type: none">• 1: GPIO• 2: Seagull MAP2 (over PWM)• 3: MAVLink (forward via MAV_CMD_IMAGE_START_CAPTURE)• 4: Generic PWM (IR trigger, servo) Reboot required: true		4	
TRIG_INTERVAL (FLOAT)	Camera trigger interval Comment: This parameter sets the time between two consecutive trigger events	4.0 > 10000.0	40.0	ms

TRIG_MODE (INT32)	<p>Camera trigger mode</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Disable • 1: Time based, on command • 2: Time based, always on • 3: Distance based, always on • 4: Distance based, on command (Survey mode) <p>Reboot required: true</p>	0 > 4	0	
TRIG_PINS (INT32)	<p>Camera trigger pin</p> <p>Comment: Selects which pin is used, ranges from 1 to 6 (AUX1-AUX6 on px4fmu-v2 and the rail pins on px4fmu-v4). The PWM interface takes two pins per camera, while relay triggers on every pin individually. Example: Value 56 would trigger on pins 5 and 6.</p> <p>Reboot required: true</p>	1 > 123456	56	
TRIG_POLARITY (INT32)	<p>Camera trigger polarity</p> <p>Comment: This parameter sets the polarity of the trigger (0 = active low, 1 = active high)</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Active low • 1: Active high 	0 > 1	0	

Circuit Breaker

The module where these parameters are defined is: *modules/systemlib*.

Name	Description	Min > Max (Incr.)	Default	Units
CBRK_AIRSPD_CHK (INT32)	Circuit breaker for airspeed sensor Comment: Setting this parameter to 162128 will disable the check for an airspeed sensor. WARNING: ENABLING THIS CIRCUIT BREAKER IS AT OWN RISK Reboot required: true	0 > 162128	0	
CBRK_BUZZER (INT32)	Circuit breaker for disabling buzzer Comment: Setting this parameter to 782097 will disable the buzzer audio notification. WARNING: ENABLING THIS CIRCUIT BREAKER IS AT OWN RISK Reboot required: true	0 > 782097	0	
CBRK_ENGINEFAIL (INT32)	Circuit breaker for engine failure detection Comment: Setting this parameter to 284953 will disable the engine failure detection. If the aircraft is in engine failure mode the engine failure flag will be set to healthy WARNING: ENABLING THIS CIRCUIT BREAKER IS AT OWN RISK Reboot required: true	0 > 284953	284953	

Name	Description	Min > Max (Incr.)	Default	Units
CBRK_AIRSPD_CHK (INT32)	Circuit breaker for airspeed sensor Comment: Setting this parameter to 162128 will disable the check for an airspeed sensor. WARNING: ENABLING THIS CIRCUIT BREAKER IS AT OWN RISK Reboot required: true	0 > 162128	0	
CBRK_BUZZER (INT32)	Circuit breaker for disabling buzzer Comment: Setting this parameter to 782097 will disable the buzzer audio notification. WARNING: ENABLING THIS CIRCUIT BREAKER IS AT OWN RISK Reboot required: true	0 > 782097	0	
CBRK_IO_SAFETY (INT32)	Circuit breaker for IO safety Comment: Setting this parameter to 22027 will disable IO safety. WARNING: ENABLING THIS CIRCUIT BREAKER IS AT OWN RISK Reboot required: true	0 > 22027	0	

CBRK_RATE_CTRL (INT32)	Circuit breaker for rate controller output Comment: Setting this parameter to 140253 will disable the rate controller uORB publication. WARNING: ENABLING THIS CIRCUIT BREAKER IS AT OWN RISK Reboot required: true	0 > 140253	0	
CBRK_SUPPLY_CHK (INT32)	Circuit breaker for power supply check Comment: Setting this parameter to 894281 will disable the power valid checks in the commander. WARNING: ENABLING THIS CIRCUIT BREAKER IS AT OWN RISK Reboot required: true	0 > 894281	0	
CBRK_USB_CHK (INT32)	Circuit breaker for USB link check Comment: Setting this parameter to 197848 will disable the USB connected checks in the commander. WARNING: ENABLING THIS CIRCUIT BREAKER IS AT OWN RISK Reboot required: true	0 > 197848	0	
CBRK_VELPO_SERR (INT32)	Circuit breaker for position error check Comment: Setting this parameter to 201607 will disable the position and velocity accuracy checks in the commander. WARNING: ENABLING THIS CIRCUIT BREAKER IS AT OWN RISK Reboot required: true	0 > 201607	0	

Commander

The module where these parameters are defined is: *modules/commander*.

Name	Description	Min > Max (Incr.)	Default	Units
COM_ARM_AUTH (INT32)	<p>Arm authorization parameters, this uint32_t will be splitted between starting from the LSB:</p> <ul style="list-style-type: none"> - 8bits to authorizer system id - 16bits to authentication method parameter, this will be used to store a timeout for the first 2 methods but can be used to another parameter for other new authentication methods. - 7bits to authentication method - one arm = 0 - two step arm = 1 <p>* the MSB bit is not used to avoid problems in the conversion between int and uint</p> <p>Comment: Default value: $(10 \ll 0 \mid 1000 \ll 8 \mid 0 \ll 24) = 256010$</p> <ul style="list-style-type: none"> - authorizer system id = 10 - authentication method parameter = 10000msec of timeout - authentication method = during arm 		256010	
COM_ARM_EKF_AB (FLOAT)	<p>Maximum value of EKF accelerometer delta velocity bias estimate that will allow arming.</p> <p>Note: ekf2 will limit the delta velocity bias estimate magnitude to be less than $EKF2_ABL_LIM * FILTER_UPDATE_PERIOD_MS * 0.001$ so this parameter must be less than that to be useful</p>	0.001 > 0.01 (0.0001)	2.4e-3	m/s

COM_ARM_EKF_GB (FLOAT)	Maximum value of EKF gyro delta angle bias estimate that will allow arming	0.0001 > 0.0017 (0.0001)	8.7e-4	rad
COM_ARM_EKF_HGT (FLOAT)	Maximum EKF height innovation test ratio that will allow arming	0.1 > 1.0 (0.05)	1.0	m
COM_ARM_EKF_POS (FLOAT)	Maximum EKF position innovation test ratio that will allow arming	0.1 > 1.0 (0.05)	0.5	m
COM_ARM_EKF_VEL (FLOAT)	Maximum EKF velocity innovation test ratio that will allow arming	0.1 > 1.0 (0.05)	0.5	m/s
COM_ARM_EKF_YAW (FLOAT)	Maximum EKF yaw innovation test ratio that will allow arming	0.1 > 1.0 (0.05)	0.5	rad
COM_ARM_IMU_ACC (FLOAT)	Maximum accelerometer inconsistency between IMU units that will allow arming	0.1 > 1.0 (0.05)	0.7	m/s/s
COM_ARM_IMU_GYR (FLOAT)	Maximum rate gyro inconsistency between IMU units that will allow arming	0.02 > 0.3 (0.01)	0.25	rad/s
COM_ARM_MAG (FLOAT)	Maximum magnetic field inconsistency between units that will allow arming	0.05 > 0.5 (0.05)	0.15	Gauss
COM_ARM_MIS_REQ (INT32)	Require valid mission to arm Comment: The default allows to arm the vehicle without a valid mission.		0	
COM_ARM_SWISBTN (INT32)	Arm switch is only a button Comment: The default uses the arm switch as real switch. If parameter set button gets handled like stick arming. Values: <ul style="list-style-type: none"> • 0: Arm switch is a switch that stays on when armed • 1: Arm switch is a button that only triggers arming and disarming 	0 > 1	0	
COM_ARM_WO_GPS (INT32)	Allow arming without GPS Comment: The default allows to arm the vehicle without GPS signal.		1	

COM_DISARM_LAND (INT32)	Time-out for auto disarm after landing Comment: A non-zero, positive value specifies the time-out period in seconds after which the vehicle will be automatically disarmed in case a landing situation has been detected during this period. The vehicle will also auto-disarm right after arming if it has not even flown, however the time will be longer by a factor of 5. A value of zero means that automatic disarming is disabled.	0 > 20 (1)	0	s
COM_DL_LOSS_T (INT32)	Datalink loss time threshold Comment: After this amount of seconds without datalink the data link lost mode triggers	5 > 300 (0.5)	10	s
COM_DL_REG_T (INT32)	Datalink regain time threshold Comment: After a data link loss: after this this amount of seconds with a healthy datalink the 'datalink loss' flag is set back to false	0 > 3 (0.5)	0	s
COM_EF_C2T (FLOAT)	Engine Failure Current/Throttle Threshold Comment: Engine failure triggers only below this current value	0.0 > 50.0 (1)	5.0	A/%
COM_EF_THROT (FLOAT)	Engine Failure Throttle Threshold Comment: Engine failure triggers only above this throttle value	0.0 > 1.0 (0.01)	0.5	norm
COM_EF_TIME (FLOAT)	Engine Failure Time Threshold Comment: Engine failure triggers only if the throttle threshold and the current to throttle threshold are violated for this time	0.0 > 60.0 (1)	10.0	s
COM_FLIGHT_UUID (INT32)	Next flight UUID Comment: This number is incremented automatically after every flight on disarming in	0 > ?	0	

<p>COM_FLTMODE1 (INT32)</p>	<p>First flightmode slot (1000-1160)</p> <p>Comment: If the main switch channel is in this range the selected flight mode will be applied.</p> <p>Values:</p> <ul style="list-style-type: none"> • -1: Unassigned • 0: Manual • 1: Altitude • 2: Position • 3: Mission • 4: Hold • 5: Return • 6: Acro • 7: Offboard • 8: Stabilized • 9: Rattitude • 10: Takeoff • 11: Land • 12: Follow Me 		-1	
<p>COM_FLTMODE2 (INT32)</p>	<p>Second flightmode slot (1160-1320)</p> <p>Comment: If the main switch channel is in this range the selected flight mode will be applied.</p> <p>Values:</p> <ul style="list-style-type: none"> • -1: Unassigned • 0: Manual • 1: Altitude • 2: Position • 3: Mission • 4: Hold • 5: Return • 6: Acro • 7: Offboard • 8: Stabilized • 9: Rattitude • 10: Takeoff • 11: Land • 12: Follow Me 		-1	

COM_FLTMODE3 (INT32)	Third flightmode slot (1320-1480) Comment: If the main switch channel is in this range the selected flight mode will be applied. Values: <ul style="list-style-type: none"> • -1: Unassigned • 0: Manual • 1: Altitude • 2: Position • 3: Mission • 4: Hold • 5: Return • 6: Acro • 7: Offboard • 8: Stabilized • 9: Rattitude • 10: Takeoff • 11: Land • 12: Follow Me 		-1	
COM_FLTMODE4 (INT32)	Fourth flightmode slot (1480-1640) Comment: If the main switch channel is in this range the selected flight mode will be applied. Values: <ul style="list-style-type: none"> • -1: Unassigned • 0: Manual • 1: Altitude • 2: Position • 3: Mission • 4: Hold • 5: Return • 6: Acro • 7: Offboard • 8: Stabilized • 9: Rattitude • 10: Takeoff • 11: Land • 12: Follow Me 		-1	

COM_FLTMODE5 (INT32)	Fifth flightmode slot (1640-1800) Comment: If the main switch channel is in this range the selected flight mode will be applied. Values: <ul style="list-style-type: none"> • -1: Unassigned • 0: Manual • 1: Altitude • 2: Position • 3: Mission • 4: Hold • 5: Return • 6: Acro • 7: Offboard • 8: Stabilized • 9: Rattitude • 10: Takeoff • 11: Land • 12: Follow Me 		-1	
COM_FLTMODE6 (INT32)	Sixth flightmode slot (1800-2000) Comment: If the main switch channel is in this range the selected flight mode will be applied. Values: <ul style="list-style-type: none"> • -1: Unassigned • 0: Manual • 1: Altitude • 2: Position • 3: Mission • 4: Hold • 5: Return • 6: Acro • 7: Offboard • 8: Stabilized • 9: Rattitude • 10: Takeoff • 11: Land • 12: Follow Me 		-1	
COM_HOME_H_T (FLOAT)	Home set horizontal threshold Comment: The home position will be set if the estimated positioning accuracy is below the threshold.	2 > 15 (0.5)	5.0	m

COM_HOME_V_T (FLOAT)	Home set vertical threshold Comment: The home position will be set if the estimated positioning accuracy is below the threshold.	5 > 25 (0.5)	10.0	m
COM_LOW_BAT_ACT (INT32)	Battery failsafe mode Comment: Action the system takes on low battery. Defaults to off Values: <ul style="list-style-type: none"> • 0: Warning • 1: Return to land • 2: Land at current position • 3: Return to land at critically low level, land at current position if reaching dangerously low levels 	(1)	0	
COM_OF_LOSS_T (FLOAT)	Time-out to wait when offboard connection is lost before triggering offboard lost action. See COM_OBL_ACT and COM_OBL_RC_ACT to configure action	0 > 60 (1)	0.0	second
COM_POS_FS_DELAY (INT32)	Loss of position failsafe activation delay Comment: This sets number of seconds that the position checks need to be failed before the failsafe will activate. The default value has been optimised for rotary wing applications. For fixed wing applications, a larger value between 5 and 10 should be used. Reboot required: true		1	sec
COM_POS_FS_GAIN (INT32)	Loss of position probation gain factor Comment: This sets the rate that the loss of position probation time grows when position checks are failing. The default value has been optimised for rotary wing applications. For fixed wing applications a value of 0 should be used. Reboot required: true		10	

COM_POS_FS_PROB (INT32)	<p>Loss of position probation delay at takeoff</p> <p>Comment: The probation delay is the number of seconds that the EKF innovation checks need to pass for the position to be declared good after it has been declared bad. The probation delay will be reset to this parameter value when takeoff is detected. After takeoff, if position checks are passing, the probation delay will reduce by one second for every lapsed second of valid position down to a minimum of 1 second. If position checks are failing, the probation delay will increase by COM_POS_FS_GAIN seconds for every lapsed second up to a maximum of 100 seconds. The default value has been optimised for rotary wing applications. For fixed wing applications, a value of 1 should be used.</p> <p>Reboot required: true</p>		30	sec
COM_RC_ARM_HYST (INT32)	<p>RC input arm/disarm command duration</p> <p>Comment: The default value of 1000 requires the stick to be held in the arm or disarm position for 1 second.</p>	100 > 1500	1000	
COM_RC_IN_MODE (INT32)	<p>RC control input mode</p> <p>Comment: The default value of 0 requires a valid RC transmitter setup. Setting this to 1 allows joystick control and disables RC input handling and the associated checks. A value of 2 will generate RC control data from manual input received via MAVLink instead of directly forwarding the manual input data.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: RC Transmitter • 1: Joystick/No RC Checks • 2: Virtual RC by Joystick 	0 > 2	0	
COM_RC_LOSS_T (FLOAT)	<p>RC loss time threshold</p> <p>Comment: After this amount of seconds without RC connection the rc lost flag is set to true</p>	0 > 35 (0.1)	0.5	s

COM_RC_OVERRIDE (INT32)	Enable RC stick override of auto modes		0	
COM_RC_STICK_OV (FLOAT)	RC stick override threshold Comment: If an RC stick is moved more than by this amount the system will interpret this as override request by the pilot.	5 > 40 (0.05)	12.0	%

Data Link Loss

The module where these parameters are defined is: *modules/navigator*.

Name	Description	Min > Max (Incr.)	Default	Units
NAV_AH_ALT (FLOAT)	Airfield home alt Comment: Altitude of airfield home waypoint	-50 > ? (0.5)	600.0	m
NAV_AH_LAT (INT32)	Airfield home Lat Comment: Latitude of airfield home waypoint	-900000000 > 900000000	-265847810	deg * 1e7
NAV_AH_LON (INT32)	Airfield home Lon Comment: Longitude of airfield home waypoint	-1800000000 > 1800000000	1518423250	deg * 1e7
NAV_DLL_AH_T (FLOAT)	Airfield home wait time Comment: The amount of time in seconds the system should wait at the airfield home waypoint	0.0 > 3600.0 (1)	120.0	s
NAV_DLL_CHSK (INT32)	Skip comms hold wp Comment: If set to 1 the system will skip the comms hold wp on data link loss and will directly fly to airfield home		0	

NAV_DLL_CH_ALT (FLOAT)	Comms hold alt Comment: Altitude of comms hold waypoint	-50 > 30000 (0.5)	600.0	m
NAV_DLL_CH_LAT (INT32)	Comms hold Lat Comment: Latitude of comms hold waypoint	-900000000 > 900000000	-266072120	deg * 1e7
NAV_DLL_CH_LON (INT32)	Comms hold Lon Comment: Longitude of comms hold waypoint	-1800000000 > 1800000000	1518453890	deg * 1e7
NAV_DLL_CH_T (FLOAT)	Comms hold wait time Comment: The amount of time in seconds the system should wait at the comms hold waypoint	0.0 > 3600.0 (1)	120.0	s
NAV_DLL_N (INT32)	Number of allowed Datalink timeouts Comment: After more than this number of data link timeouts the aircraft returns home directly	0 > 1000	2	

EKF2

The module where these parameters are defined is: *modules/ekf2*.

Name	Description	Min > Max (Incr.)	Default	Units
EKF2_ABIAS_INIT (FLOAT)	1-sigma IMU accelerometer switch-on bias Reboot required: true	0.0 > 0.5	0.2	m/s/s
EKF2_ABL_ACCLIM (FLOAT)	Maximum IMU accel magnitude that allows IMU bias learning. If the magnitude of the IMU accelerometer vector exceeds this value, the EKF delta velocity state estimation will be inhibited. This reduces the adverse effect of high manoeuvre accelerations and IMU nonlinearity and scale factor errors on the delta velocity bias estimates	20.0 > 200.0	25.0	m/s/s
EKF2_ABL_GYRLIM (FLOAT)	Maximum IMU gyro angular rate magnitude that allows IMU bias learning. If the magnitude of the IMU angular rate vector exceeds this value, the EKF delta velocity state estimation will be inhibited. This reduces the adverse effect of rapid rotation rates and associated errors on the delta velocity bias estimates	2.0 > 20.0	3.0	rad/s

EKF2_ABL_LIM (FLOAT)	Accelerometer bias learning limit. The ekf delta velocity bias states will be limited to within a range equivalent to +- of this value	0.0 > 0.8	0.4	m/s/s
EKF2_ABL_TAU (FLOAT)	Time constant used by acceleration and angular rate magnitude checks used to inhibit delta velocity bias learning. The vector magnitude of angular rate and acceleration used to check if learning should be inhibited has a peak hold filter applied to it with an exponential decay. This parameter controls the time constant of the decay	0.1 > 1.0	0.5	s
EKF2_ACC_B_NOISE (FLOAT)	Process noise for IMU accelerometer bias prediction	0.0 > 0.01	3.0e-3	m/s**3
EKF2_ACC_NOISE (FLOAT)	Accelerometer noise for covariance prediction	0.01 > 1.0	3.5e-1	m/s/s

EKF2_AID_MASK (INT32)	<p>Integer bitmask controlling data fusion and aiding methods</p> <p>Comment: Set bits in the following positions to enable: 0 : Set to true to use GPS data if available 1 : Set to true to use optical flow data if available 2 : Set to true to inhibit IMU bias estimation 3 : Set to true to enable vision position fusion 4 : Set to true to enable vision yaw fusion 5 : Set to true to enable multi-rotor drag specific force fusion 6 : set to true if the EV observations are in a non NED reference frame and need to be rotated before being used</p> <p>Bitmask:</p> <ul style="list-style-type: none"> • 0: use GPS • 1: use optical flow • 2: inhibit IMU bias estimation • 3: vision position fusion • 4: vision yaw fusion • 5: multi-rotor drag fusion • 6: rotate external vision <p>Reboot required: true</p>	0 > 127	1	
EKF2_ANGERR_INIT (FLOAT)	<p>1-sigma tilt angle uncertainty after gravity vector alignment</p> <p>Reboot required: true</p>	0.0 > 0.5	0.1	rad

EKF2_ARSP_THR (FLOAT)	Airspeed fusion threshold. A value of zero will deactivate airspeed fusion. Any other positive value will determine the minimum airspeed which will still be fused. Set to about 90% of the vehicles stall speed. Both airspeed fusion and sideslip fusion must be active for the EKF to continue navigating after loss of GPS. Use EKF2_FUSE_BETA to activate sideslip fusion	0.0 > ?	0.0	m/s
EKF2_ASPD_MAX (FLOAT)	Upper limit on airspeed along individual axes used to correct baro for position error effects	5.0 > 50.0	20.0	m/s
EKF2_ASP_DELAY (FLOAT)	Airspeed measurement delay relative to IMU measurements Reboot required: true	0 > 300	100	ms
EKF2_AVEL_DELAY (FLOAT)	Auxillary Velocity Estimate (e.g from a landing target) delay relative to IMU measurements Reboot required: true	0 > 300	5	ms
EKF2_BARO_DELAY (FLOAT)	Barometer measurement delay relative to IMU measurements Reboot required: true	0 > 300	0	ms

EKF2_BARO_GATE (FLOAT)	Gate size for barometric and GPS height fusion Comment: Sets the number of standard deviations used by the innovation consistency test.	1.0 > ?	5.0	SD
EKF2_BARO_NOISE (FLOAT)	Measurement noise for barometric altitude	0.01 > 15.0	2.0	m
EKF2_BCOEF_X (FLOAT)	X-axis ballistic coefficient used by the multi-rotor specific drag force model. This should be adjusted to minimise variance of the X-axis drag specific force innovation sequence	1.0 > 100.0	25.0	kg/m**2
EKF2_BCOEF_Y (FLOAT)	Y-axis ballistic coefficient used by the multi-rotor specific drag force model. This should be adjusted to minimise variance of the Y-axis drag specific force innovation sequence	1.0 > 100.0	25.0	kg/m**2
EKF2_BETA_GATE (FLOAT)	Gate size for synthetic sideslip fusion Comment: Sets the number of standard deviations used by the innovation consistency test.	1.0 > ?	5.0	SD
EKF2_BETA_NOISE (FLOAT)	Noise for synthetic sideslip fusion	0.1 > 1.0	0.3	m/s
EKF2_DECL_TYPE (INT32)	Integer bitmask controlling handling of magnetic declination Comment: Set bits in the following positions to enable functions. 0 : Set to true to use the declination from the geo_lookup library when the GPS position becomes available, set to false to always use the EKF2_MAG_DECL value. 1 : Set to true to save the EKF2_MAG_DECL parameter to the value returned by the EKF when the vehicle disarms. 2 : Set to true to always use the declination as an observation when 3-axis magnetometer fusion is being used. Bitmask: <ul style="list-style-type: none"> • 0: use geo_lookup declination • 1: save EKF2_MAG_DECL on disarm • 2: use declination as an observation Reboot required: true	0 > 7	7	

EKF2_DRAG_NOISE (FLOAT)	Specific drag force observation noise variance used by the multi-rotor specific drag force model. Increasing it makes the multi-rotor wind estimates adjust more slowly	0.5 > 10.0	2.5	(m/sec**2)**2
EKF2_EAS_NOISE (FLOAT)	Measurement noise for airspeed fusion	0.5 > 5.0	1.4	m/s
EKF2_EVA_NOISE (FLOAT)	Measurement noise for vision angle observations used when the vision system does not supply error estimates	0.01 > ?	0.05	rad
EKF2_EVP_NOISE (FLOAT)	Measurement noise for vision position observations used when the vision system does not supply error estimates	0.01 > ?	0.05	m
EKF2_EV_DELAY (FLOAT)	Vision Position Estimator delay relative to IMU measurements Reboot required: true	0 > 300	175	ms
EKF2_EV_GATE (FLOAT)	Gate size for vision estimate fusion Comment: Sets the number of standard deviations used by the innovation consistency test.	1.0 > ?	5.0	SD
EKF2_EV_POS_X (FLOAT)	X position of VI sensor focal point in body frame		0.0	m
EKF2_EV_POS_Y (FLOAT)	Y position of VI sensor focal point in body frame		0.0	m
EKF2_EV_POS_Z (FLOAT)	Z position of VI sensor focal point in body frame		0.0	m
EKF2_FUSE_BETA (INT32)	Boolean determining if synthetic sideslip measurements should fused Comment: A value of 1 indicates that fusion is active Both sideslip fusion and airspeed fusion must be active for the EKF to continue navigating after loss of GPS. Use EKF2_ARSP_THR to activate airspeed fusion.		0	
EKF2_GBIAS_INIT (FLOAT)	1-sigma IMU gyro switch-on bias Reboot required: true	0.0 > 0.2	0.1	rad/sec

EKF2_GPS_CHECK (INT32)	Integer bitmask controlling GPS checks Comment: Set bits to 1 to enable checks. Checks enabled by the following bit positions 0 : Minimum required sat count set by EKF2_REQ_NSATS 1 : Minimum required GDoP set by EKF2_REQ_GDOP 2 : Maximum allowed horizontal position error set by EKF2_REQ_EPH 3 : Maximum allowed vertical position error set by EKF2_REQ_EPV 4 : Maximum allowed speed error set by EKF2_REQ_SACC 5 : Maximum allowed horizontal position rate set by EKF2_REQ_HDRIFT. This check can only be used if the vehicle is stationary during alignment. 6 : Maximum allowed vertical position rate set by EKF2_REQ_VDRIFT. This check can only be used if the vehicle is stationary during alignment. 7 : Maximum allowed horizontal speed set by EKF2_REQ_HDRIFT. This check can only be used if the vehicle is stationary during alignment. 8 : Maximum allowed vertical velocity discrepancy set by EKF2_REQ_VDRIFT	0 > 511	21	
----------------------------------	--	---------	----	--

	Bitmask: <ul style="list-style-type: none"> • 0: Min sat count (EKF2_REQ_NSATS) • 1: Min GDoP (EKF2_REQ_GDOP) • 2: Max horizontal position error (EKF2_REQ_EPH) • 3: Max vertical position error (EKF2_REQ_EPV) • 4: Max speed error (EKF2_REQ_SACC) • 5: Max horizontal position rate (EKF2_REQ_HDRIFT) • 6: Max vertical position rate (EKF2_REQ_VDRIFT) • 7: Max horizontal speed (EKF2_REQ_HDRIFT) • 8: Max vertical velocity discrepancy (EKF2_REQ_VDRIFT) 			
EKF2_GPS_DELAY (FLOAT)	GPS measurement delay relative to IMU measurements Reboot required: true	0 > 300	110	ms
EKF2_GPS_POS_X (FLOAT)	X position of GPS antenna in body frame		0.0	m
EKF2_GPS_POS_Y (FLOAT)	Y position of GPS antenna in body frame		0.0	m

EKF2_GPS_POS_Z (FLOAT)	Z position of GPS antenna in body frame		0.0	m
EKF2_GPS_P_GATE (FLOAT)	Gate size for GPS horizontal position fusion Comment: Sets the number of standard deviations used by the innovation consistency test.	1.0 > ?	5.0	SD
EKF2_GPS_P_NOISE (FLOAT)	Measurement noise for gps position	0.01 > 10.0	0.5	m
EKF2_GPS_V_GATE (FLOAT)	Gate size for GPS velocity fusion Comment: Sets the number of standard deviations used by the innovation consistency test.	1.0 > ?	5.0	SD
EKF2_GPS_V_NOISE (FLOAT)	Measurement noise for gps horizontal velocity	0.01 > 5.0	0.5	m/s
EKF2_GYR_B_NOISE (FLOAT)	Process noise for IMU rate gyro bias prediction	0.0 > 0.01	1.0e-3	rad/s**2
EKF2_GYR_NOISE (FLOAT)	Rate gyro noise for covariance prediction	0.0001 > 0.1	1.5e-2	rad/s

EKF2_HDG_GATE (FLOAT)	Gate size for magnetic heading fusion Comment: Sets the number of standard deviations used by the innovation consistency test.	1.0 > ?	2.6	SD
EKF2_HEAD_NOISE (FLOAT)	Measurement noise for magnetic heading fusion	0.01 > 1.0	0.3	rad
EKF2_HGT_MODE (INT32)	Determines the primary source of height data used by the EKF Comment: The range sensor option should only be used when for operation over a flat surface as the local NED origin will move up and down with ground level. Values: <ul style="list-style-type: none"> • 0: Barometric pressure • 1: GPS • 2: Range sensor • 3: Vision Reboot required: true		0	
EKF2_IMU_POS_X (FLOAT)	X position of IMU in body frame		0.0	m
EKF2_IMU_POS_Y (FLOAT)	Y position of IMU in body frame		0.0	m

EKF2_IMU_POS_Z (FLOAT)	Z position of IMU in body frame		0.0	m
EKF2_MAGBIAS_ID (INT32)	ID of Magnetometer the learned bias is for Reboot required: true		0	
EKF2_MAGBIAS_X (FLOAT)	Learned value of magnetometer X axis bias. This is the amount of X-axis magnetometer bias learned by the EKF and saved from the last flight. It must be set to zero if the ground based magnetometer calibration is repeated Reboot required: true	-0.5 > 0.5	0.0	mGauss
EKF2_MAGBIAS_Y (FLOAT)	Learned value of magnetometer Y axis bias. This is the amount of Y-axis magnetometer bias learned by the EKF and saved from the last flight. It must be set to zero if the ground based magnetometer calibration is repeated Reboot required: true	-0.5 > 0.5	0.0	mGauss

EKF2_MAGBIAS_Z (FLOAT)	<p>Learned value of magnetometer Z axis bias. This is the amount of Z-axis magnetometer bias learned by the EKF and saved from the last flight. It must be set to zero if the ground based magnetometer calibration is repeated</p> <p>Reboot required: true</p>	-0.5 > 0.5	0.0	mGauss
EKF2_MAGB_K (FLOAT)	<p>Maximum fraction of learned mag bias saved at each disarm. Smaller values make the saved mag bias learn slower from flight to flight. Larger values make it learn faster. Must be > 0.0 and <= 1.0</p>	0.0 > 1.0	0.2	
EKF2_MAGB_VREF (FLOAT)	<p>State variance assumed for magnetometer bias storage. This is a reference variance used to calculate the fraction of learned magnetometer bias that will be used to update the stored value. Smaller values will make the stored bias data adjust more slowly from flight to flight. Larger values will make it adjust faster</p> <p>Reboot required: true</p>		2.5E-7	mGauss**2

EKF2_MAG_ACCLIM (FLOAT)	Horizontal acceleration threshold used by automatic selection of magnetometer fusion method. This parameter is used when the magnetometer fusion method is set automatically (EKF2_MAG_TYPE = 0). If the filtered horizontal acceleration is greater than this parameter value, then the EKF will use 3-axis magnetomer fusion	0.0 > 5.0	0.5	m/s**2
EKF2_MAG_B_NOISE (FLOAT)	Process noise for body magnetic field prediction	0.0 > 0.1	1.0e-4	Gauss/s
EKF2_MAG_DECL (FLOAT)	Magnetic declination		0	deg
EKF2_MAG_DELAY (FLOAT)	Magnetometer measurement delay relative to IMU measurements Reboot required: true	0 > 300	0	ms
EKF2_MAG_E_NOISE (FLOAT)	Process noise for earth magnetic field prediction	0.0 > 0.1	1.0e-3	Gauss/s
EKF2_MAG_GATE (FLOAT)	Gate size for magnetometer XYZ component fusion Comment: Sets the number of standard deviations used by the innovation consistency test.	1.0 > ?	3.0	SD

EKF2_MAG_NOISE (FLOAT)	Measurement noise for magnetometer 3-axis fusion	0.001 > 1.0	5.0e-2	Gauss
EKF2_MAG_TYPE (INT32)	Type of magnetometer fusion Comment: Integer controlling the type of magnetometer fusion used - magnetic heading or 3-axis magnetometer. If set to automatic: heading fusion on-ground and 3-axis fusion in-flight with fallback to heading fusion if there is insufficient motion to make yaw or mag biases observable. Values: <ul style="list-style-type: none"> • 0: Automatic • 1: Magnetic heading • 2: 3-axis fusion • 3: None Reboot required: true		0	
EKF2_MAG_YAWLIM (FLOAT)	Yaw rate threshold used by automatic selection of magnetometer fusion method. This parameter is used when the magnetometer fusion method is set automatically (EKF2_MAG_TYPE = 0). If the filtered yaw rate is greater than this parameter value, then the EKF will use 3-axis magnetomer fusion	0.0 > 0.5	0.25	rad/s

EKF2_MIN_OBS_DT (INT32)	Minimum time of arrival delta between non-IMU observations before data is downsampled. Baro and Magnetometer data will be averaged before downsampling, other data will be point sampled resulting in loss of information Reboot required: true	10 > 50	20	ms
EKF2_MIN_RNG (FLOAT)	Minimum valid range for the range finder	0.01 > ?	0.1	m
EKF2_NOAID_NOISE (FLOAT)	Measurement noise for non-aiding position hold	0.5 > 50.0	10.0	m
EKF2_OF_DELAY (FLOAT)	Optical flow measurement delay relative to IMU measurements Assumes measurement is timestamped at trailing edge of integration period Reboot required: true	0 > 300	5	ms
EKF2_OF_GATE (FLOAT)	Gate size for optical flow fusion Comment: Sets the number of standard deviations used by the innovation consistency test.	1.0 > ?	3.0	SD

EKF2_OF_N_MAX (FLOAT)	Measurement noise for the optical flow sensor Comment: (when it's reported quality metric is at the minimum set by EKF2_OF_QMIN). The following condition must be met: EKF2_OF_N_MAXN >= EKF2_OF_N_MIN	0.05 > ?	0.5	rad/s
EKF2_OF_N_MIN (FLOAT)	Measurement noise for the optical flow sensor when it's reported quality metric is at the maximum	0.05 > ?	0.15	rad/s
EKF2_OF_POS_X (FLOAT)	X position of optical flow focal point in body frame		0.0	m
EKF2_OF_POS_Y (FLOAT)	Y position of optical flow focal point in body frame		0.0	m
EKF2_OF_POS_Z (FLOAT)	Z position of optical flow focal point in body frame		0.0	m
EKF2_OF_QMIN (INT32)	Optical Flow data will only be used if the sensor reports a quality metric >= EKF2_OF_QMIN	0 > 255	1	

EKF2_OF_RMAX (FLOAT)	Optical Flow data will not fused if the magnitude of the flow rate > EKF2_OF_RMAX. Control loops will be instructed to limit ground speed such that the flow rate produced by movement over ground is less than 50% of EKF2_OF_RMAX	1.0 > ?	2.5	rad/s
EKF2_PCOEF_XN (FLOAT)	Static pressure position error coefficient for the negative X axis. This is the ratio of static pressure error to dynamic pressure generated by a negative wind relative velocity along the X body axis. If the baro height estimate rises during backwards flight, then this will be a negative number	-0.5 > 0.5	0.0	
EKF2_PCOEF_XP (FLOAT)	Static pressure position error coefficient for the positive X axis This is the ratio of static pressure error to dynamic pressure generated by a positive wind relative velocity along the X body axis. If the baro height estimate rises during forward flight, then this will be a negative number	-0.5 > 0.5	0.0	

EKF2_PCOEF_Y (FLOAT)	Pressure position error coefficient for the Y axis. This is the ratio of static pressure error to dynamic pressure generated by a wind relative velocity along the Y body axis. If the baro height estimate rises during sideways flight, then this will be a negative number	-0.5 > 0.5	0.0	
EKF2_PCOEF_Z (FLOAT)	Static pressure position error coefficient for the Z axis. This is the ratio of static pressure error to dynamic pressure generated by a wind relative velocity along the Z body axis	-0.5 > 0.5	0.0	
EKF2_REQ_EPH (FLOAT)	Required EPH to use GPS	2 > 100	5.0	m
EKF2_REQ_EPV (FLOAT)	Required EPV to use GPS	2 > 100	8.0	m
EKF2_REQ_GDOP (FLOAT)	Required GDoP to use GPS	1.5 > 5.0	2.5	
EKF2_REQ_HDRIFT (FLOAT)	Maximum horizontal drift speed to use GPS	0.1 > 1.0	0.3	m/s
EKF2_REQ_NSATS (INT32)	Required satellite count to use GPS	4 > 12	6	
EKF2_REQ_SACC (FLOAT)	Required speed accuracy to use GPS	0.5 > 5.0	1.0	m/s
EKF2_REQ_VDRIFT (FLOAT)	Maximum vertical drift speed to use GPS	0.1 > 1.5	0.5	m/s
EKF2_RNG_AID (INT32)	<p>Range sensor aid</p> <p>Comment: If this parameter is enabled then the estimator will make use of the range finder measurements to estimate it's height even if range sensor is not the primary height source. It will only do so if conditions for range measurement fusion are met.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Range aid disabled • 1: Range aid enabled 		0	

EKF2_RNG_A_HMAX (FLOAT)	Maximum absolute altitude (height above ground level) allowed for range aid mode Comment: If the vehicle absolute altitude exceeds this value then the estimator will not fuse range measurements to estimate it's height. This only applies when range aid mode is activated (EKF2_RNG_AID = enabled).	1.0 > 10.0	5.0	
EKF2_RNG_A_IGATE (FLOAT)	Gate size used for innovation consistency checks for range aid fusion Comment: A lower value means HAGL needs to be more stable in order to use range finder for height estimation in range aid mode	0.1 > 5.0	1.0	SD
EKF2_RNG_A_VMAX (FLOAT)	Maximum horizontal velocity allowed for range aid mode Comment: If the vehicle horizontal speed exceeds this value then the estimator will not fuse range measurements to estimate it's height. This only applies when range aid mode is activated (EKF2_RNG_AID = enabled).	0.1 > 2	1.0	

EKF2_RNG_DELAY (FLOAT)	Range finder measurement delay relative to IMU measurements Reboot required: true	0 > 300	5	ms
EKF2_RNG_GATE (FLOAT)	Gate size for range finder fusion Comment: Sets the number of standard deviations used by the innovation consistency test.	1.0 > ?	5.0	SD
EKF2_RNG_NOISE (FLOAT)	Measurement noise for range finder fusion	0.01 > ?	0.1	m
EKF2_RNG_PITCH (FLOAT)	Range sensor pitch offset	-0.75 > 0.75	0.0	rad
EKF2_RNG_POS_X (FLOAT)	X position of range finder origin in body frame		0.0	m
EKF2_RNG_POS_Y (FLOAT)	Y position of range finder origin in body frame		0.0	m
EKF2_RNG_POS_Z (FLOAT)	Z position of range finder origin in body frame		0.0	m

EKF2_RNG_SFE (FLOAT)	Range finder range dependant noise scaler Comment: Specifies the increase in range finder noise with range.	0.0 > 0.2	0.05	m/m
EKF2_TAS_GATE (FLOAT)	Gate size for TAS fusion Comment: Sets the number of standard deviations used by the innovation consistency test.	1.0 > ?	3.0	SD
EKF2_TAU_POS (FLOAT)	Time constant of the position output prediction and smoothing filter. Controls how tightly the output track the EKF states	0.1 > 1.0	0.25	s
EKF2_TAU_VEL (FLOAT)	Time constant of the velocity output prediction and smoothing filter	? > 1.0	0.25	s
EKF2_TERR_GRAD (FLOAT)	Magnitude of terrain gradient	0.0 > ?	0.5	m/m
EKF2_TERR_NOISE (FLOAT)	Terrain altitude process noise - accounts for instability in vehicle height estimate	0.5 > ?	5.0	m/s
EKF2_WIND_NOISE (FLOAT)	Process noise for wind velocity prediction	0.0 > 1.0	1.0e-1	m/s/s

FW Attitude Control

Name	Description	Min > Max (Incr.)	Default	Units
FW_ACRO_X_MAX (FLOAT)	Acro body x max rate Comment: This is the rate the controller is trying to achieve if the user applies full roll stick input in acro mode. Module: modules/fw_att_control	45 > 720	90	degrees
FW_ACRO_Y_MAX (FLOAT)	Acro body y max rate Comment: This is the body y rate the controller is trying to achieve if the user applies full pitch stick input in acro mode. Module: modules/fw_att_control	45 > 720	90	degrees
FW_ACRO_Z_MAX (FLOAT)	Acro body z max rate Comment: This is the body z rate the controller is trying to achieve if the user applies full yaw stick input in acro mode. Module: modules/fw_att_control	10 > 180	45	degrees

FW_ARSP_MODE (INT32)	Airspeed mode Comment: For small wings or VTOL without airspeed sensor this parameter can be used to enable flying without an airspeed reading Values: <ul style="list-style-type: none"> • 0: Normal (use airspeed if available) • 1: Airspeed disabled Module: modules/fw_att_control		0	
FW_BAT_SCALE_EN (INT32)	Whether to scale throttle by battery power level Comment: This compensates for voltage drop of the battery over time by attempting to normalize performance across the operating range of the battery. The fixed wing should constantly behave as if it was fully charged with reduced max thrust at lower battery percentages. i.e. if cruise speed is at 0.5 throttle at 100% battery, it will still be 0.5 at 60% battery. Module: modules/fw_att_control		0	

FW_FLAPERON_SCL (FLOAT)	Scale factor for flaperons Module: modules/fw_att_control	0.0 > 1.0 (0.01)	0.0	norm
FW_FLAPS_SCL (FLOAT)	Scale factor for flaps Module: modules/fw_att_control	0.0 > 1.0 (0.01)	1.0	norm
FW_MAN_P_MAX (FLOAT)	Max manual pitch Comment: Max pitch for manual control in attitude stabilized mode Module: modules/fw_att_control	0.0 > 90.0 (0.5)	45.0	deg
FW_MAN_P_SC (FLOAT)	Manual pitch scale Comment: Scale factor applied to the desired pitch actuator command in full manual mode. This parameter allows to adjust the throws of the control surfaces. Module: modules/fw_att_control	0.0 > ? (0.01)	1.0	norm
FW_MAN_R_MAX (FLOAT)	Max manual roll Comment: Max roll for manual control in attitude stabilized mode Module: modules/fw_att_control	0.0 > 90.0 (0.5)	45.0	deg
FW_MAN_R_SC (FLOAT)	Manual roll scale Comment: Scale factor applied to the desired roll actuator command in full manual mode. This parameter allows to adjust the throws of the control surfaces. Module: modules/fw_att_control	0.0 > 1.0 (0.01)	1.0	norm
FW_MAN_Y_SC (FLOAT)	Manual yaw scale Comment: Scale factor applied to the desired yaw actuator command in full manual mode. This parameter allows to adjust the throws of the control surfaces. Module: modules/fw_att_control	0.0 > ? (0.01)	1.0	norm
FW_PR_FF (FLOAT)	Pitch rate feed forward Comment: Direct feed forward from rate setpoint to control surface output Module: modules/fw_att_control	0.0 > 10.0 (0.05)	0.5	%/rad/s

FW_PR_I (FLOAT)	<p>Pitch rate integrator gain</p> <p>Comment: This gain defines how much control response will result out of a steady state error. It trims any constant error.</p> <p>Module: modules/fw_att_control</p>	0.005 > 0.5 (0.005)	0.02	%/rad
FW_PR_IMAX (FLOAT)	<p>Pitch rate integrator limit</p> <p>Comment: The portion of the integrator part in the control surface deflection is limited to this value</p> <p>Module: modules/fw_att_control</p>	0.0 > 1.0 (0.05)	0.4	
FW_PR_P (FLOAT)	<p>Pitch rate proportional gain</p> <p>Comment: This defines how much the elevator input will be commanded depending on the current body angular rate error.</p> <p>Module: modules/fw_att_control</p>	0.005 > 1.0 (0.005)	0.08	%/rad/s
FW_PSP_OFF (FLOAT)	<p>Pitch setpoint offset</p> <p>Comment: An airframe specific offset of the pitch setpoint in degrees, the value is added to the pitch setpoint and should correspond to the typical cruise speed of the airframe.</p> <p>Module: modules/fw_att_control</p>	-90.0 > 90.0 (0.5)	0.0	deg
FW_P_RMAX_NEG (FLOAT)	<p>Maximum negative / down pitch rate</p> <p>Comment: This limits the maximum pitch down up angular rate the controller will output (in degrees per second). Setting a value of zero disables the limit.</p> <p>Module: modules/fw_att_control</p>	0.0 > 90.0 (0.5)	60.0	deg/s
FW_P_RMAX_POS (FLOAT)	<p>Maximum positive / up pitch rate</p> <p>Comment: This limits the maximum pitch up angular rate the controller will output (in degrees per second). Setting a value of zero disables the limit.</p> <p>Module: modules/fw_att_control</p>	0.0 > 90.0 (0.5)	60.0	deg/s

FW_P_TC (FLOAT)	Attitude pitch time constant Comment: This defines the latency between a pitch step input and the achieved setpoint (inverse to a P gain). Half a second is a good start value and fits for most average systems. Smaller systems may require smaller values, but as this will wear out servos faster, the value should only be decreased as needed. Module: modules/fw_att_control	0.2 > 1.0 (0.05)	0.4	s
FW_RATT_TH (FLOAT)	Threshold for Rattitude mode Comment: Manual input needed in order to override attitude control rate setpoints and instead pass manual stick inputs as rate setpoints Module: modules/fw_att_control	0.0 > 1.0 (0.01)	0.8	
FW_RLL_TO_YAW_FF (FLOAT)	Roll control to yaw control feedforward gain Comment: This gain can be used to counteract the "adverse yaw" effect for fixed wings. When the plane enters a roll it will tend to yaw the nose out of the turn. This gain enables the use of a yaw actuator (rudder, airbrakes, ...) to counteract this effect. Module: modules/fw_att_control	0.0 > ? (0.01)	0.0	
FW_RR_FF (FLOAT)	Roll rate feed forward Comment: Direct feed forward from rate setpoint to control surface output. Use this to obtain a tighter response of the controller without introducing noise amplification. Module: modules/fw_att_control	0.0 > 10.0 (0.05)	0.5	%/rad/s
FW_RR_I (FLOAT)	Roll rate integrator Gain Comment: This gain defines how much control response will result out of a steady state error. It trims any constant error. Module: modules/fw_att_control	0.005 > 0.2 (0.005)	0.01	%/rad
FW_RR_IMAX (FLOAT)	Roll integrator anti-windup Comment: The portion of the integrator part in the control surface deflection is limited to this value. Module: modules/fw_att_control	0.0 > 1.0 (0.05)	0.2	

FW_RR_P (FLOAT)	Roll rate proportional Gain Comment: This defines how much the aileron input will be commanded depending on the current body angular rate error. Module: modules/fw_att_control	0.005 > 1.0 (0.005)	0.05	%/rad/s
FW_RSP_OFF (FLOAT)	Roll setpoint offset Comment: An airframe specific offset of the roll setpoint in degrees, the value is added to the roll setpoint and should correspond to the typical cruise speed of the airframe. Module: modules/fw_att_control	-90.0 > 90.0 (0.5)	0.0	deg
FW_R_RMAX (FLOAT)	Maximum roll rate Comment: This limits the maximum roll rate the controller will output (in degrees per second). Setting a value of zero disables the limit. Module: modules/fw_att_control	0.0 > 90.0 (0.5)	70.0	deg/s
FW_R_TC (FLOAT)	Attitude Roll Time Constant Comment: This defines the latency between a roll step input and the achieved setpoint (inverse to a P gain). Half a second is a good start value and fits for most average systems. Smaller systems may require smaller values, but as this will wear out servos faster, the value should only be decreased as needed. Module: modules/fw_att_control	0.4 > 1.0 (0.05)	0.4	s
FW_WR_FF (FLOAT)	Wheel steering rate feed forward Comment: Direct feed forward from rate setpoint to control surface output Module: modules/fw_att_control	0.0 > 10.0 (0.05)	0.2	%/rad/s
FW_WR_I (FLOAT)	Wheel steering rate integrator gain Comment: This gain defines how much control response will result out of a steady state error. It trims any constant error. Module: modules/fw_att_control	0.005 > 0.5 (0.005)	0.1	%/rad

FW_WR_IMAX (FLOAT)	Wheel steering rate integrator limit Comment: The portion of the integrator part in the control surface deflection is limited to this value Module: modules/fw_att_control	0.0 > 1.0 (0.05)	1.0	
FW_WR_P (FLOAT)	Wheel steering rate proportional gain Comment: This defines how much the wheel steering input will be commanded depending on the current body angular rate error. Module: modules/fw_att_control	0.005 > 1.0 (0.005)	0.5	%/rad/s
FW_W_EN (INT32)	Enable wheel steering controller Module: modules/fw_att_control		0	
FW_W_RMAX (FLOAT)	Maximum wheel steering rate Comment: This limits the maximum wheel steering rate the controller will output (in degrees per second). Setting a value of zero disables the limit. Module: modules/fw_att_control	0.0 > 90.0 (0.5)	0.0	deg/s

FW_YR_FF (FLOAT)	<p>Yaw rate feed forward</p> <p>Comment: Direct feed forward from rate setpoint to control surface output</p> <p>Module: modules/fw_att_control</p>	0.0 > 10.0 (0.05)	0.3	%/rad/s
FW_YR_I (FLOAT)	<p>Yaw rate integrator gain</p> <p>Comment: This gain defines how much control response will result out of a steady state error. It trims any constant error.</p> <p>Module: modules/fw_att_control</p>	0.0 > 50.0 (0.5)	0.0	%/rad
FW_YR_IMAX (FLOAT)	<p>Yaw rate integrator limit</p> <p>Comment: The portion of the integrator part in the control surface deflection is limited to this value</p> <p>Module: modules/fw_att_control</p>	0.0 > 1.0 (0.05)	0.2	
FW_YR_P (FLOAT)	<p>Yaw rate proportional gain</p> <p>Comment: This defines how much the rudder input will be commanded depending on the current body angular rate error.</p> <p>Module: modules/fw_att_control</p>	0.005 > 1.0 (0.005)	0.05	%/rad/s
FW_Y_RMAX (FLOAT)	<p>Maximum yaw rate</p> <p>Comment: This limits the maximum yaw rate the controller will output (in degrees per second). Setting a value of zero disables the limit.</p> <p>Module: modules/fw_att_control</p>	0.0 > 90.0 (0.5)	0.0	deg/s
GND_WR_TC (FLOAT)	<p>Attitude Wheel Time Constant</p> <p>Comment: This defines the latency between a steering step input and the achieved setpoint (inverse to a P gain). Half a second is a good start value and fits for most average systems. Smaller systems may require smaller values, but as this will wear out servos faster, the value should only be decreased as needed.</p> <p>Module: modules/gnd_att_control</p>	0.4 > 1.0 (0.05)	0.4	s

FW L1 Control

The module where these parameters are defined is: *modules/fw_pos_control_l1*.

Name	Description	Min > Max (Incr.)	Default	Units
FW_CLMBOUT_DIFF (FLOAT)	Climbout Altitude difference Comment: If the altitude error exceeds this parameter, the system will climb out with maximum throttle and minimum airspeed until it is closer than this distance to the desired altitude. Mostly used for takeoff waypoints / modes. Set to 0 to disable climbout mode (not recommended).	0.0 > 150.0 (0.5)	10.0	m
FW_L1_DAMPING (FLOAT)	L1 damping Comment: Damping factor for L1 control.	0.6 > 0.9 (0.05)	0.75	
FW_L1_PERIOD (FLOAT)	L1 period Comment: This is the L1 distance and defines the tracking point ahead of the aircraft its following. A value of 18-25 meters works for most aircraft. Shorten slowly during tuning until response is sharp without oscillation.	12.0 > 50.0 (0.5)	20.0	m

FW_LND_AIRSPD_SC (FLOAT)	Min. airspeed scaling factor for landing Comment: Multiplying this factor with the minimum airspeed of the plane gives the target airspeed the landing approach. FW_AIRSPD_MIN * FW_LND_AIRSPD_SC	1.0 > 1.5 (0.01)	1.3	norm
FW_LND_ANG (FLOAT)	Landing slope angle	1.0 > 15.0 (0.5)	5.0	deg
FW_LND_FLALT (FLOAT)	Landing flare altitude (relative to landing altitude)	0.0 > 25.0 (0.5)	8.0	m
FW_LND_FL_PMAX (FLOAT)	Flare, maximum pitch Comment: Maximum pitch during flare, a positive sign means nose up Applied once FW_LND_TLALT is reached	0 > 45.0 (0.5)	15.0	deg
FW_LND_FL_PMIN (FLOAT)	Flare, minimum pitch Comment: Minimum pitch during flare, a positive sign means nose up Applied once FW_LND_TLALT is reached	0 > 15.0 (0.5)	2.5	deg
FW_LND_HHDIST (FLOAT)	Landing heading hold horizontal distance. Set to 0 to disable heading hold	0 > 30.0 (0.5)	15.0	m

FW_LND_HVIRT (FLOAT)		1.0 > 15.0 (0.5)	10.0	m
FW_LND_TLALT (FLOAT)	Landing throttle limit altitude (relative landing altitude) Comment: Default of -1.0 lets the system default to applying throttle limiting at 2/3 of the flare altitude.	-1.0 > 30.0 (0.5)	-1.0	m
FW_LND_USETER (INT32)	Use terrain estimate during landing		0	
FW_P_LIM_MAX (FLOAT)	Positive pitch limit Comment: The maximum positive pitch the controller will output.	0.0 > 60.0 (0.5)	45.0	deg
FW_P_LIM_MIN (FLOAT)	Negative pitch limit Comment: The minimum negative pitch the controller will output.	-60.0 > 0.0 (0.5)	-45.0	deg
FW_R_LIM (FLOAT)	Controller roll limit Comment: The maximum roll the controller will output.	35.0 > 65.0 (0.5)	50.0	deg

FW_THR_ALT_SCL (FLOAT)	Scale throttle by pressure change Comment: Automatically adjust throttle to account for decreased air density at higher altitudes. Start with a scale factor of 1.0 and adjust for different propulsion systems. When flying without airspeed sensor this will help to keep a constant performance over large altitude ranges. The default value of 0 will disable scaling.	0.0 > 2.0 (0.1)	0.0	
FW_THR_CRUISE (FLOAT)	Cruise throttle Comment: This is the throttle setting required to achieve the desired cruise speed. Most airframes have a value of 0.5-0.7.	0.0 > 1.0 (0.01)	0.6	norm
FW_THR_IDLE (FLOAT)	Idle throttle Comment: This is the minimum throttle while on the ground For aircraft with internal combustion engine this parameter should be set above desired idle rpm.	0.0 > 0.4 (0.01)	0.15	norm
FW_THR_LND_MAX (FLOAT)	Throttle limit value before flare Comment: This throttle value will be set as throttle limit at FW_LND_TLALT, before aircraft will flare.	0.0 > 1.0 (0.01)	1.0	norm

FW_THR_MAX (FLOAT)	Throttle limit max Comment: This is the maximum throttle % that can be used by the controller. For overpowered aircraft, this should be reduced to a value that provides sufficient thrust to climb at the maximum pitch angle PTCH_MAX.	0.0 > 1.0 (0.01)	1.0	norm
FW_THR_MIN (FLOAT)	Throttle limit min Comment: This is the minimum throttle % that can be used by the controller. For electric aircraft this will normally be set to zero, but can be set to a small non-zero value if a folding prop is fitted to prevent the prop from folding and unfolding repeatedly in-flight or to provide some aerodynamic drag from a turning prop to improve the descent rate. For aircraft with internal combustion engine this parameter should be set for desired idle rpm.	0.0 > 1.0 (0.01)	0.0	norm
FW_THR_SLEW_MAX (FLOAT)	Throttle max slew rate Comment: Maximum slew rate for the commanded throttle	0.0 > 1.0	0.0	

FW Launch detection

The module where these parameters are defined is: *modules/fw_pos_control_11/launchdetection*.

Name	Description	Min > Max (Incr.)	Default	Units
LAUN_ALL_ON (INT32)	Launch detection		0	
LAUN_CAT_A (FLOAT)	Catapult accelerometer threshold Comment: LAUN_CAT_A for LAUN_CAT_T serves as threshold to trigger launch detection.	0 > ? (0.5)	30.0	m/s/s
LAUN_CAT_MDEL (FLOAT)	Motor delay Comment: Delay between starting attitude control and powering up the throttle (giving throttle control to the controller) Before this timespan is up the throttle will be set to FW_THR_IDLE, set to 0 to deactivate	0.0 > 10.0 (0.5)	0.0	s
LAUN_CAT_PMAX (FLOAT)	Maximum pitch before the throttle is powered up (during motor delay phase) Comment: This is an extra limit for the maximum pitch which is imposed in the phase before the throttle turns on. This allows to limit the maximum pitch angle during a bungee launch (make the launch less steep).	0.0 > 45.0 (0.5)	30.0	deg
LAUN_CAT_T (FLOAT)	Catapult time threshold Comment: LAUN_CAT_A for LAUN_CAT_T serves as threshold to trigger launch detection.	0.0 > 5.0 (0.05)	0.05	s

FW TECS

Name	Description	Min > Max (Incr.)	Default	Units
FW_AIRSPD_MAX (FLOAT)	<p>Maximum Airspeed</p> <p>Comment: If the airspeed is above this value, the TECS controller will try to decrease airspeed more aggressively.</p> <p>Module: modules/fw_pos_control_l1</p>	0.0 > 40 (0.5)	20.0	m/s
FW_AIRSPD_MIN (FLOAT)	<p>Minimum Airspeed</p> <p>Comment: If the airspeed falls below this value, the TECS controller will try to increase airspeed more aggressively.</p> <p>Module: modules/fw_pos_control_l1</p>	0.0 > 40 (0.5)	10.0	m/s
FW_AIRSPD_TRIM (FLOAT)	<p>Cruise Airspeed</p> <p>Comment: The fixed wing controller tries to fly at this airspeed.</p> <p>Module: modules/fw_pos_control_l1</p>	0.0 > 40 (0.5)	15.0	m/s
FW_T_CLMB_MAX (FLOAT)	<p>Maximum climb rate</p> <p>Comment: This is the best climb rate that the aircraft can achieve with the throttle set to THR_MAX and the airspeed set to the default value. For electric aircraft make sure this number can be achieved towards the end of flight when the battery voltage has reduced. The setting of this parameter can be checked by commanding a positive altitude change of 100m in loiter, RTL or guided mode. If the throttle required to climb is close to THR_MAX and the aircraft is maintaining airspeed, then this parameter is set correctly. If the airspeed starts to reduce, then the parameter is set to high, and if the throttle demand required to climb and maintain speed is noticeably less than FW_THR_MAX, then either FW_T_CLMB_MAX should be increased or FW_THR_MAX reduced.</p> <p>Module: modules/fw_pos_control_l1</p>	1.0 > 15.0 (0.5)	5.0	m/s

FW_T_HGT_OMEGA (FLOAT)	Complementary filter "omega" parameter for height Comment: This is the cross-over frequency (in radians/second) of the complementary filter used to fuse vertical acceleration and barometric height to obtain an estimate of height rate and height. Increasing this frequency weights the solution more towards use of the barometer, whilst reducing it weights the solution more towards use of the accelerometer data. Module: modules/fw_pos_control_l1	1.0 > 10.0 (0.5)	3.0	rad/s
FW_T_HRATE_FF (FLOAT)	Height rate feed forward Module: modules/fw_pos_control_l1	0.0 > 1.0 (0.05)	0.8	
FW_T_HRATE_P (FLOAT)	Height rate proportional factor Module: modules/fw_pos_control_l1	0.0 > 1.0 (0.05)	0.05	
FW_T_INTEG_GAIN (FLOAT)	Integrator gain Comment: This is the integrator gain on the control loop. Increasing this gain increases the speed at which speed and height offsets are trimmed out, but reduces damping and increases overshoot. Module: modules/fw_pos_control_l1	0.0 > 2.0 (0.05)	0.1	
FW_T_PTCH_DAMP (FLOAT)	Pitch damping factor Comment: This is the damping gain for the pitch demand loop. Increase to add damping to correct for oscillations in height. The default value of 0.0 will work well provided the pitch to servo controller has been tuned properly. Module: modules/fw_pos_control_l1	0.0 > 2.0 (0.1)	0.0	

FW_T_RLL2THR (FLOAT)	<p>Roll -> Throttle feedforward</p> <p>Comment: Increasing this gain turn increases the amount of throttle that will be used to compensate for the additional drag created by turning. Ideally this should be set to approximately 10 x the extra sink rate in m/s created by a 45 degree bank turn. Increase this gain if the aircraft initially loses energy in turns and reduce if the aircraft initially gains energy in turns. Efficient high aspect-ratio aircraft (eg powered sailplanes) can use a lower value, whereas inefficient low aspect-ratio models (eg delta wings) can use a higher value.</p> <p>Module: modules/fw_pos_control_l1</p>	0.0 > 20.0 (0.5)	15.0	
FW_T_SINK_MAX (FLOAT)	<p>Maximum descent rate</p> <p>Comment: This sets the maximum descent rate that the controller will use. If this value is too large, the aircraft can over-speed on descent. This should be set to a value that can be achieved without exceeding the lower pitch angle limit and without over-speeding the aircraft.</p> <p>Module: modules/fw_pos_control_l1</p>	2.0 > 15.0 (0.5)	5.0	m/s

FW_T_SINK_MIN (FLOAT)	<p>Minimum descent rate</p> <p>Comment: This is the sink rate of the aircraft with the throttle set to THR_MIN and flown at the same airspeed as used to measure FW_T_CLMB_MAX.</p> <p>Module: modules/fw_pos_control_l1</p>	1.0 > 5.0 (0.5)	2.0	m/s
FW_T_SPDWEIGHT (FLOAT)	<p>Speed <--> Altitude priority</p> <p>Comment: This parameter adjusts the amount of weighting that the pitch control applies to speed vs height errors. Setting it to 0.0 will cause the pitch control to control height and ignore speed errors. This will normally improve height accuracy but give larger airspeed errors. Setting it to 2.0 will cause the pitch control loop to control speed and ignore height errors. This will normally reduce airspeed errors, but give larger height errors. The default value of 1.0 allows the pitch control to simultaneously control height and speed. Note to Glider Pilots - set this parameter to 2.0 (The glider will adjust its pitch angle to maintain airspeed, ignoring changes in height).</p> <p>Module: modules/fw_pos_control_l1</p>	0.0 > 2.0 (1.0)	1.0	

FW_T_SPD_OMEGA (FLOAT)	Complementary filter "omega" parameter for speed Comment: This is the cross-over frequency (in radians/second) of the complementary filter used to fuse longitudinal acceleration and airspeed to obtain an improved airspeed estimate. Increasing this frequency weights the solution more towards use of the airspeed sensor, whilst reducing it weights the solution more towards use of the accelerometer data. Module: modules/fw_pos_control_l1	1.0 > 10.0 (0.5)	2.0	rad/s
FW_T_SRATE_P (FLOAT)	Speed rate P factor Module: modules/fw_pos_control_l1	0.0 > 2.0 (0.01)	0.02	
FW_T_THRO_CONST (FLOAT)	TECS Throttle time constant Comment: This is the time constant of the TECS throttle control algorithm (in seconds). Smaller values make it faster to respond, larger values make it slower to respond. Module: modules/fw_pos_control_l1	1.0 > 10.0 (0.5)	8.0	s

FW_T_THR_DAMP (FLOAT)	Throttle damping factor Comment: This is the damping gain for the throttle demand loop. Increase to add damping to correct for oscillations in speed and height. Module: modules/fw_pos_control_l1	0.0 > 2.0 (0.1)	0.5	
FW_T_TIME_CONST (FLOAT)	TECS time constant Comment: This is the time constant of the TECS control algorithm (in seconds). Smaller values make it faster to respond, larger values make it slower to respond. Module: modules/fw_pos_control_l1	1.0 > 10.0 (0.5)	5.0	s
FW_T_VERT_ACC (FLOAT)	Maximum vertical acceleration Comment: This is the maximum vertical acceleration (in m/s/s) either up or down that the controller will use to correct speed or height errors. The default value of 7 m/s/s (equivalent to +/- 0.7 g) allows for reasonably aggressive pitch changes if required to recover from under-speed conditions. Module: modules/fw_pos_control_l1	1.0 > 10.0 (0.5)	7.0	m/s/s
GND_SPEED_MAX (FLOAT)	Maximum ground speed Module: modules/gnd_pos_control	0.0 > 40 (0.5)	10.0	m/s
GND_SPEED_TRIM (FLOAT)	Trim ground speed Module: modules/gnd_pos_control	0.0 > 40 (0.5)	3.0	m/s

Follow target

The module where these parameters are defined is: *modules/navigator*.

Name	Description	Min > Max (Incr.)	Default	Units
NAV_FT_DST (FLOAT)	Distance to follow target from Comment: The distance in meters to follow the target at	1.0 > ?	8.0	meters
NAV_FT_FS (INT32)	Side to follow target from Comment: The side to follow the target from (front right = 0, behind = 1, front = 2, front left = 3)	0 > 3	1	n/a
NAV_FT_RS (FLOAT)	Dynamic filtering algorithm responsiveness to target movement lower numbers increase the responsiveness to changing long lat but also ignore less noise	0.0 > 1.0	0.5	n/a
NAV_MIN_FT_HT (FLOAT)	Minimum follow target altitude Comment: The minimum height in meters relative to home for following a target	8.0 > ?	8.0	meters

GND Attitude Control

Name	Description	Min > Max (Incr.)	Default	Units
GND_BAT_SCALE_EN (INT32)	<p>Whether to scale throttle by battery power level</p> <p>Comment: This compensates for voltage drop of the battery over time by attempting to normalize performance across the operating range of the battery. The fixed wing should constantly behave as if it was fully charged with reduced max thrust at lower battery percentages. i.e. if cruise speed is at 0.5 throttle at 100% battery, it will still be 0.5 at 60% battery.</p> <p>Module: modules/gnd_att_control</p>		0	
GND_GSPD_SP_TRIM (FLOAT)	<p>Groundspeed speed trim</p> <p>Comment: This allows to scale the turning radius depending on the speed.</p> <p>Module: modules/gnd_att_control</p>	0.0 > ? (0.1)	1.0	norm

GND_MAN_Y_SC (FLOAT)	Manual yaw scale Comment: Scale factor applied to the desired yaw actuator command in full manual mode. This parameter allows to adjust the throws of the control surfaces. Module: modules/gnd_att_control	0.0 > ? (0.01)	1.0	norm
GND_SPEED_D (FLOAT)	Speed proportional gain Comment: This is the derivative gain for the speed closed loop controller Module: modules/gnd_pos_control	0.00 > 50.0 (0.005)	0.0	%m/s
GND_SPEED_I (FLOAT)	Speed Integral gain Comment: This is the integral gain for the speed closed loop controller Module: modules/gnd_pos_control	0.00 > 50.0 (0.005)	0.1	%m/s
GND_SPEED_IMAX (FLOAT)	Speed integral maximum value Comment: This is the maxim value the integral can reach to prevent wind-up. Module: modules/gnd_pos_control	0.005 > 50.0 (0.005)	1.0	%m/s

GND_SPEED_P (FLOAT)	Speed proportional gain Comment: This is the proportional gain for the speed closed loop controller Module: modules/gnd_pos_control	0.005 > 50.0 (0.005)	2.0	%m/s
GND_SPEED_THR_SC (FLOAT)	Speed to throttle scaler Comment: This is a gain to map the speed control output to the throttle linearly. Module: modules/gnd_pos_control	0.005 > 50.0 (0.005)	1.0	%m/s
GND_SP_CTRL_MODE (INT32)	Control mode for speed Comment: This allows the user to choose between closed loop gps speed or open loop cruise throttle speed Values: <ul style="list-style-type: none"> • 0: open loop control • 1: close the loop with gps speed Module: modules/gnd_pos_control	0 > 1	1	
GND_WR_D (FLOAT)	Wheel steering rate integrator gain Module: modules/gnd_att_control	0.00 > 30 (0.005)	0.00	%/rad

GND_WR_FF (FLOAT)	Wheel steering rate feed forward Comment: Direct feed forward from rate setpoint to control surface output Module: modules/gnd_att_control	0.0 > 10.0 (0.05)	0.0	%/rad/s
GND_WR_I (FLOAT)	Wheel steering rate integrator gain Comment: This gain defines how much control response will result out of a steady state error. It trims any constant error. Module: modules/gnd_att_control	0.00 > 0.5 (0.005)	0.00	%/rad
GND_WR_IMAX (FLOAT)	Wheel steering rate integrator limit Comment: The portion of the integrator part in the control surface deflection is limited to this value Module: modules/gnd_att_control	0.0 > 1.0 (0.05)	0.0	
GND_WR_P (FLOAT)	Wheel steering rate proportional gain Comment: This defines how much the wheel steering input will be commanded depending on the current body angular rate error. Module: modules/gnd_att_control	0.005 > 1.0 (0.005)	1.0	%/rad/s
GND_W_RMAX (FLOAT)	Maximum wheel steering rate Comment: This limits the maximum wheel steering rate the controller will output (in degrees per second). Setting a value of zero disables the limit. Module: modules/gnd_att_control	0.0 > 90.0 (0.5)	90.0	deg/s

GND POS Control

The module where these parameters are defined is: *modules/gnd_pos_control*.

Name	Description	Min > Max (Incr.)	Default	Units
GND_L1_DAMPING (FLOAT)	L1 damping Comment: Damping factor for L1 control.	0.6 > 0.9 (0.05)	0.75	
GND_L1_DIST (FLOAT)	L1 distance Comment: This is the waypoint radius	0.0 > 100.0 (0.1)	5.0	m
GND_L1_PERIOD (FLOAT)	L1 period Comment: This is the L1 distance and defines the tracking point ahead of the rover it's following. Using values around 2-5 for a traxxas stampede. Shorten slowly during tuning until response is sharp without oscillation.	0.0 > 50.0 (0.5)	10.0	m
GND_THR_CRUISE (FLOAT)	Cruise throttle Comment: This is the throttle setting required to achieve the desired cruise speed. 10% is ok for a traxxas stampede vx1 with ESC set to training mode	0.0 > 1.0 (0.01)	0.1	norm
GND_THR_IDLE (FLOAT)	Idle throttle Comment: This is the minimum throttle while on the ground, it should be 0 for a rover	0.0 > 0.4 (0.01)	0.0	norm
GND_THR_MAX (FLOAT)	Throttle limit max Comment: This is the maximum throttle % that can be used by the controller. For a Traxxas stampede vx1 with the ESC set to training, 30 % is enough	0.0 > 1.0 (0.01)	0.3	norm
GND_THR_MIN (FLOAT)	Throttle limit min Comment: This is the minimum throttle % that can be used by the controller. Set to 0 for rover	0.0 > 1.0 (0.01)	0.0	norm

GPS

The module where these parameters are defined is: *drivers/gps*.

Name	Description	Min > Max (Incr.)	Default	Units
GPS_DUMP_COMM (INT32)	Dump GPS communication to a file Comment: If this is set to 1, all GPS communication data will be published via uORB, and written to the log file as <code>gps_dump</code> message. Values: <ul style="list-style-type: none">• 0: Disable• 1: Enable	0 > 1	0	
GPS_UBX_DYNMODEL (INT32)	u-blox GPS dynamic platform model Comment: u-blox receivers support different dynamic platform models to adjust the navigation engine to the expected application environment. Values: <ul style="list-style-type: none">• 2: stationary• 4: automotive• 6: airborne with <1g acceleration<="" j="">• 7: airborne with <2g acceleration<="" j="">• 8: airborne with <4g acceleration<="" j=""> Reboot required: true	0 > 9	7	

GPS Failure Navigation

The module where these parameters are defined is: *modules/navigator*.

Name	Description	Min > Max (Incr.)	Default	Units
NAV_GPSF_LT (FLOAT)	Loiter time Comment: The time in seconds the system should do open loop loiter and wait for GPS recovery before it goes into flight termination. Set to 0 to disable.	0.0 > 3600.0 (1)	0.0	s
NAV_GPSF_P (FLOAT)	Fixed pitch angle Comment: Pitch in degrees during the open loop loiter	-30.0 > 30.0 (0.5)	0.0	deg
NAV_GPSF_R (FLOAT)	Fixed bank angle Comment: Roll in degrees during the loiter	0.0 > 30.0 (0.5)	15.0	deg
NAV_GPSF_TR (FLOAT)	Thrust Comment: Thrust value which is set during the open loop loiter	0.0 > 1.0 (0.05)	0.0	norm

Geofence

The module where these parameters are defined is: modules/navigator.

Name	Description	Min > Max (Incr.)	Default	Units
GF_ACTION (INT32)	<p>Geofence violation action</p> <p>Comment: Note: Setting this value to 4 enables flight termination, which will kill the vehicle on violation of the fence. Due to the inherent danger of this, this function is disabled using a software circuit breaker, which needs to be reset to 0 to really shut down the system.</p> <p>Values:</p> <ul style="list-style-type: none">• 0: None• 1: Warning• 2: Loiter• 3: Return to Land• 4: Flight terminate	0 > 4	1	
GF_ALTMODE (INT32)	<p>Geofence altitude mode</p> <p>Comment: Select which altitude reference should be used 0 = WGS84, 1 = AMSL</p> <p>Values:</p> <ul style="list-style-type: none">• 0: WGS84• 1: AMSL	0 > 1	0	

GF_COUNT (INT32)	Geofence counter limit Comment: Set how many subsequent position measurements outside of the fence are needed before geofence violation is triggered	-1 > 10 (1)	-1	
GF_MAX_HOR_DIST (FLOAT)	Max horizontal distance in meters Comment: Maximum horizontal distance in meters the vehicle can be from home before triggering a geofence action. Disabled if 0.	0 > 10000 (1)	0	m
GF_MAX_VER_DIST (FLOAT)	Max vertical distance in meters Comment: Maximum vertical distance in meters the vehicle can be from home before triggering a geofence action. Disabled if 0.	0 > 10000 (1)	0	m
GF_SOURCE (INT32)	Geofence source Comment: Select which position source should be used. Selecting GPS instead of global position makes sure that there is no dependence on the position estimator 0 = global position, 1 = GPS Values: <ul style="list-style-type: none"> • 0: GPOS • 1: GPS 	0 > 1	0	

Iridium SBD

The module where these parameters are defined is: *drivers/telemetry/iridiumsgbd*.

Name	Description	Min > Max (Incr.)	Default	Units
ISBD_READINT (INT32)	Satellite radio read interval	0 > 300	10	s

Land Detector

The module where these parameters are defined is: *modules/land_detector*.

Name	Description	Min > Max (Incr.)	Default	Units
LNDFW_AIRSPD_MAX (FLOAT)	Airspeed max Comment: Maximum airspeed allowed in the landed state (m/s)	4 > 20	8.00	m/s
LNDFW_VELI_MAX (FLOAT)	Fixedwing max short-term velocity Comment: Maximum velocity integral in flight direction allowed in the landed state (m/s)	2 > 10	8.0	m/s
LNDFW_VEL_XY_MAX (FLOAT)	Fixedwing max horizontal velocity Comment: Maximum horizontal velocity allowed in the landed state (m/s)	0.5 > 10	5.0	m/s
LNDFW_VEL_Z_MAX (FLOAT)	Fixedwing max climb rate Comment: Maximum vertical velocity allowed in the landed state (m/s up and down)	5 > 20	10.0	m/s
LNDMC_ALT_MAX (FLOAT)	Maximum altitude for multicopters Comment: The system will obey this limit as a hard altitude limit. This setting will be consolidated with the GF_MAX_VER_DIST parameter. A negative value indicates no altitude limitation.	-1 > 10000	-1.0	m
LNDMC_FFALL_THR (FLOAT)	Multicopter specific force threshold Comment: Multicopter threshold on the specific force measured by accelerometers in m/s^2 for free-fall detection	0.1 > 10	2.0	m/s^2
LNDMC_FFALL_TTRI (FLOAT)	Multicopter free-fall trigger time Comment: Seconds (decimal) that freefall conditions have to met before triggering a freefall. Minimal value is limited by LAND_DETECTOR_UPDATE_RATE=50Hz in landDetector.h	0.02 > 5	0.3	s
LNDMC_ROT_MAX (FLOAT)	Multicopter max rotation Comment: Maximum allowed angular velocity around each axis allowed in the landed state.		20.0	deg/s

LNDMC_THR_RANGE (FLOAT)	Multicopter sub-hover throttle scaling Comment: The range between throttle_min and throttle_hover is scaled by this parameter to define how close to minimum throttle the current throttle value needs to be in order to get accepted as landed.	0.05 > 0.5	0.1	
LNDMC_XY_VEL_MAX (FLOAT)	Multicopter max horizontal velocity Comment: Maximum horizontal velocity allowed in the landed state (m/s)		1.5	m/s
LNDMC_Z_VEL_MAX (FLOAT)	Multicopter max climb rate Comment: Maximum vertical velocity allowed in the landed state (m/s up and down)		0.50	m/s
LND_FLIGHT_T_HI (INT32)	Total flight time in microseconds Comment: Total flight time of this autopilot. Higher 32 bits of the value. Flight time in microseconds = (LND_FLIGHT_T_HI << 32) LND_FLIGHT_T_LO.	0 > ?	0	
LND_FLIGHT_T_LO (INT32)	Total flight time in microseconds Comment: Total flight time of this autopilot. Lower 32 bits of the value. Flight time in microseconds = (LND_FLIGHT_T_HI << 32) LND_FLIGHT_T_LO.	0 > ?	0	

Landing target Estimator

The module where these parameters are defined is: *modules/landing_target_estimator*.

Name	Description	Min > Max (Incr.)	Default	Units
LTEST_ACC_UNC (FLOAT)	Acceleration uncertainty Comment: Variance of acceleration measurement used for landing target position prediction. Higher values results in tighter following of the measurements and more lenient outlier rejection	0.01 > ?	10.0	(m/s ²) ²
LTEST_MEAS_UNC (FLOAT)	Landing target measurement uncertainty Comment: Variance of the landing target measurement from the driver. Higher values results in less agresive following of the measurement and a smoother output as well as fewer rejected measurements.		0.005	tan(rad) ²

LTEST_MODE (INT32)	Landing target mode Comment: Configure the mode of the landing target. Depending on the mode, the landing target observations are used differently to aid position estimation. Mode Moving: The landing target may be moving around while in the field of view of the vehicle. Landing target measurements are not used to aid positioning. Mode Stationary: The landing target is stationary. Measured velocity w.r.t. the landing target is used to aid velocity estimation. Values: <ul style="list-style-type: none"> • 0: Moving • 1: Stationary 	0 > 1	0	
LTEST_POS_UNC_IN (FLOAT)	Initial landing target position uncertainty Comment: Initial variance of the relative landing target position in x and y direction	0.001 > ?	0.1	m ²
LTEST_SCALE_X (FLOAT)	Scale factor for sensor measurements in sensor x axis Comment: Landing target x measurements are scaled by this factor before being used	0.01 > ?	1.0	
LTEST_SCALE_Y (FLOAT)	Scale factor for sensor measurements in sensor y axis Comment: Landing target y measurements are scaled by this factor before being used	0.01 > ?	1.0	
LTEST_VEL_UNC_IN (FLOAT)	Initial landing target velocity uncertainty Comment: Initial variance of the relative landing target velocity in x and y direction	0.001 > ?	0.1	(m/s) ²

Local Position Estimator

The module where these parameters are defined is: *modules/local_position_estimator*.

Name	Description	Min > Max (Incr.)	Default	Units
LPE_ACC_XY (FLOAT)	Accelerometer xy noise density Comment: Data sheet noise density = 150ug/sqrt(Hz) = 0.0015 m/s ² /sqrt(Hz) Larger than data sheet to account for tilt error.	0.00001 > 2	0.012	m/s ² /sqrt(Hz)
LPE_ACC_Z (FLOAT)	Accelerometer z noise density Comment: Data sheet noise density = 150ug/sqrt(Hz) = 0.0015 m/s ² /sqrt(Hz)	0.00001 > 2	0.02	m/s ² /sqrt(Hz)
LPE_BAR_Z (FLOAT)	Barometric pressure altitude z standard deviation	0.01 > 100	3.0	m
LPE_EPH_MAX (FLOAT)	Max EPH allowed for GPS initialization	1.0 > 5.0	3.0	m
LPE_EPV_MAX (FLOAT)	Max EPV allowed for GPS initialization	1.0 > 5.0	5.0	m
LPE_FAKE_ORIGIN (INT32)	Enable publishing of a fake global position (e.g for AUTO missions using Optical Flow) by initializing the estimator to the LPE_LAT/LON parameters when global information is unavailable	0 > 1	0	
LPE_FGYRO_HP (FLOAT)	Flow gyro high pass filter cut off frequency	0 > 2	0.001	Hz
LPE_FLW_OFF_Z (FLOAT)	Optical flow z offset from center	-1 > 1	0.0	m
LPE_FLW_QMIN (INT32)	Optical flow minimum quality threshold	0 > 255	150	
LPE_FLW_R (FLOAT)	Optical flow rotation (roll/pitch) noise gain	0.1 > 10.0	7.0	m/s / (rad)
LPE_FLW_RR (FLOAT)	Optical flow angular velocity noise gain	0.0 > 10.0	7.0	m/s / (rad/s)
LPE_FLW_SCALE (FLOAT)	Optical flow scale	0.1 > 10.0	1.3	m

LPE_FUSION (INT32)	<p>Integer bitmask controlling data fusion</p> <p>Comment: Set bits in the following positions to enable: 0 : Set to true to fuse GPS data if available, also requires GPS for altitude init 1 : Set to true to fuse optical flow data if available 2 : Set to true to fuse vision position 3 : Set to true to enable landing target 4 : Set to true to fuse land detector 5 : Set to true to publish AGL as local position down component 6 : Set to true to enable flow gyro compensation 7 : Set to true to enable baro fusion default (145 - GPS, baro, land detector)</p> <p>Bitmask:</p> <ul style="list-style-type: none"> • 0: fuse GPS, requires GPS for alt. init • 1: fuse optical flow • 2: fuse vision position • 3: fuse landing target • 4: fuse land detector • 5: pub agl as lpos down • 6: flow gyro compensation • 7: fuse baro 	0 > 255	145	
---------------------------	--	---------	-----	--

LPE_GPS_DELAY (FLOAT)	GPS delay compensaton	0 > 0.4	0.29	sec
LPE_GPS_VXY (FLOAT)	GPS xy velocity standard deviation. EPV used if greater than this value	0.01 > 2	0.25	m/s
LPE_GPS_VZ (FLOAT)	GPS z velocity standard deviation	0.01 > 2	0.25	m/s
LPE_GPS_XY (FLOAT)	Minimum GPS xy standard deviation, uses reported EPH if greater	0.01 > 5	1.0	m
LPE_GPS_Z (FLOAT)	Minimum GPS z standard deviation, uses reported EPV if greater	0.01 > 200	3.0	m
LPE_LAND_VXY (FLOAT)	Land detector xy velocity standard deviation	0.01 > 10.0	0.05	m/s
LPE_LAND_Z (FLOAT)	Land detector z standard deviation	0.001 > 10.0	0.03	m
LPE_LAT (FLOAT)	Local origin latitude for nav w/o GPS	-90 > 90	47.397742	deg

LPE_LDR_OFF_Z (FLOAT)	Lidar z offset from center of vehicle +down	-1 > 1	0.00	m
LPE_LDR_Z (FLOAT)	Lidar z standard deviation	0.01 > 1	0.03	m
LPE_LON (FLOAT)	Local origin longitude for nav w/o GPS	-180 > 180	8.545594	deg
LPE_LT_COV (FLOAT)	Minimum landing target standard covariance, uses reported covariance if greater	0.0 > 10	0.0001	m ²
LPE_PN_B (FLOAT)	Accel bias propagation noise density	0 > 1	1e-3	(m/s ²)/s/sqrt(Hz)
LPE_PN_P (FLOAT)	Position propagation noise density Comment: Increase to trust measurements more. Decrease to trust model more.	0 > 1	0.1	m/s/sqrt(Hz)
LPE_PN_T (FLOAT)	Terrain random walk noise density, hilly/outdoor (0.1), flat/Indoor (0.001)	0 > 1	0.001	(m/s)/(sqrt(hz))
LPE_PN_V (FLOAT)	Velocity propagation noise density Comment: Increase to trust measurements more. Decrease to trust model more.	0 > 1	0.1	(m/s)/s/sqrt(Hz)
LPE_SNR_OFF_Z (FLOAT)	Sonar z offset from center of vehicle +down	-1 > 1	0.00	m
LPE_SNR_Z (FLOAT)	Sonar z standard deviation	0.01 > 1	0.05	m
LPE_T_MAX_GRADE (FLOAT)	Terrain maximum percent grade, hilly/outdoor (100 = 45 deg), flat/Indoor (0 = 0 deg) Used to calculate increased terrain random walk noise due to movement	0 > 100	1.0	%
LPE_VIC_P (FLOAT)	Vicon position standard deviation	0.0001 > 1	0.001	m
LPE_VIS_DELAY (FLOAT)	Vision delay compensaton Comment: Set to zero to enable automatic compensation from measurement timestamps	0 > 0.1	0.1	sec

LPE_VIS_XY (FLOAT)	Vision xy standard deviation	0.01 > 1	0.1	m
LPE_VIS_Z (FLOAT)	Vision z standard deviation	0.01 > 100	0.5	m
LPE_VXY_PUB (FLOAT)	Required velocity xy standard deviation to publish position	0.01 > 1.0	0.3	m/s
LPE_X_LP (FLOAT)	Cut frequency for state publication	5 > 1000	5.0	Hz
LPE_Z_PUB (FLOAT)	Required z standard deviation to publish altitude/ terrain	0.3 > 5.0	1.0	m

MAVLink

The module where these parameters are defined is: *modules/mavlink*.

Name	Description	Min > Max (Incr.)	Default	Units
MAV_BROADCAST (INT32)	Broadcast heartbeats on local network Comment: This allows a ground control station to automatically find the drone on the local network. Values: <ul style="list-style-type: none"> • 0: Never broadcast • 1: Always broadcast 		0	
MAV_COMP_ID (INT32)	MAVLink component ID Reboot required: true	1 > 250	1	
MAV_FWDEXTSP (INT32)	Forward external setpoint messages Comment: If set to 1 incoming external setpoint messages will be directly forwarded to the controllers if in offboard control mode		1	
MAV_PROTO_VER (INT32)	MAVLink protocol version Values: <ul style="list-style-type: none"> • 0: Default to 1, switch to 2 if GCS sends version 2 • 1: Always use version 1 • 2: Always use version 2 		0	
MAV_RADIO_ID (INT32)	MAVLink Radio ID Comment: When non-zero the MAVLink app will attempt to configure the radio to this ID and reset the parameter to 0. If the value is negative it will reset the complete radio config to factory defaults.	-1 > 240	0	
MAV_SYS_ID (INT32)	MAVLink system ID Reboot required: true	1 > 250	1	

MAV_TYPE (INT32)	MAVLink airframe type Values: <ul style="list-style-type: none"> • 0: Generic micro air vehicle • 1: Fixed wing aircraft • 2: Quadrotor • 3: Coaxial helicopter • 4: Normal helicopter with tail rotor • 5: Ground installation • 6: Operator control unit / ground control station • 7: Airship, controlled • 8: Free balloon, uncontrolled • 9: Rocket • 10: Ground rover • 11: Surface vessel, boat, ship • 12: Submarine • 13: Hexarotor • 14: Octorotor • 15: Tricopter • 16: Flapping wing • 17: Kite • 18: Onboard companion controller • 19: Two-rotor VTOL using control surfaces in vertical operation in addition. Tailsitter. • 20: Quad-rotor VTOL using a V-shaped quad config in vertical operation. Tailsitter. • 21: Tiltrotor VTOL • 22: VTOL reserved 2 • 23: VTOL reserved 3 • 24: VTOL reserved 4 • 25: VTOL reserved 5 • 26: Onboard gimbal • 27: Onboard ADSB peripheral 	1 > 27	2	
MAV_USEHILGPS (INT32)	Use/Accept HIL GPS message even if not in HIL mode Comment: If set to 1 incoming HIL GPS messages are parsed.		0	

MKBLCTRL Testmode

The module where these parameters are defined is: *drivers/mkblctrl*.

Name	Description	Min > Max (Incr.)	Default	Units
MKBLCTRL_TEST (INT32)	Test mode (Identify) of MKBLCTRL Driver		0	

MPU9x50 Configuration

The module where these parameters are defined is: *platforms/qurt/fc_addon/mpu_spi*.

Name	Description	Min > Max (Incr.)	Default	Units
MPU_ACC_LPF_ENM (INT32)	Low pass filter frequency for Accelerometer Values: <ul style="list-style-type: none">0: MPU9X50_ACC_LPF_460HZ1: MPU9X50_ACC_LPF_184HZ2: MPU9X50_ACC_LPF_92HZ3: MPU9X50_ACC_LPF_41HZ4: MPU9X50_ACC_LPF_20HZ5: MPU9X50_ACC_LPF_10HZ6: MPU9X50_ACC_LPF_5HZ7: MPU9X50_ACC_LPF_460HZ_NOLPF		4	
MPU_GYRO_LPF_ENM (INT32)	Low pass filter frequency for Gyro Values: <ul style="list-style-type: none">0: MPU9X50_GYRO_LPF_250HZ1: MPU9X50_GYRO_LPF_184HZ2: MPU9X50_GYRO_LPF_92HZ3: MPU9X50_GYRO_LPF_41HZ4: MPU9X50_GYRO_LPF_20HZ5: MPU9X50_GYRO_LPF_10HZ6: MPU9X50_GYRO_LPF_5HZ7: MPU9X50_GYRO_LPF_3600HZ_NOLPF		4	

MPU_SAMPLE_R_ENM (INT32)	Sample rate in Hz Values: <ul style="list-style-type: none"> • 0: MPU9x50_SAMPLE_RATE_100HZ • 1: MPU9x50_SAMPLE_RATE_200HZ • 2: MPU9x50_SAMPLE_RATE_500HZ • 3: MPU9x50_SAMPLE_RATE_1000HZ 		2	
------------------------------------	---	--	---	--

Mission

Name	Description	Min > Max (Incr.)	Default	Units
COM_OBL_ACT (INT32)	Set offboard loss failsafe mode Comment: The offboard loss failsafe will only be entered after a timeout, set by COM_OF_LOSS_T in seconds. Values: <ul style="list-style-type: none"> • 0: Land at current position • 1: Loiter • 2: Return to Land Module: modules/commander		0	
COM_OBL_RC_ACT (INT32)	Set offboard loss failsafe mode when RC is available Comment: The offboard loss failsafe will only be entered after a timeout, set by COM_OF_LOSS_T in seconds. Values: <ul style="list-style-type: none"> • 0: Position control • 1: Altitude control • 2: Manual • 3: Return to Land • 4: Land at current position • 5: Loiter Module: modules/commander		0	

COM_POSCTL_NAVL (INT32)	<p>Position control navigation loss response</p> <p>Comment: This sets the flight mode that will be used if navigation accuracy is no longer adequate for position control. Navigation accuracy checks can be disabled using the CBRK_VELPOSERR parameter, but doing so will remove protection for all flight modes.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Assume use of remote control after fallback. Switch to ALTCTL if a height estimate is available, else switch to MANUAL. • 1: Assume no use of remote control after fallback. Switch to DESCEND if a height estimate is available, else switch to TERMINATION. <p>Module: modules/commander</p>		0	
COM_TAKEOFF_ACT (INT32)	<p>Action after TAKEOFF has been accepted</p> <p>Comment: The mode transition after TAKEOFF has completed successfully.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Hold • 1: Mission (if valid) <p>Module: modules/commander</p>		0	

MIS_ALTMODE (INT32)	Altitude setpoint mode Comment: 0: the system will follow a zero order hold altitude setpoint 1: the system will follow a first order hold altitude setpoint values follow the definition in enum mission_altitude_mode Values: <ul style="list-style-type: none"> • 0: Zero Order Hold • 1: First Order Hold Module: modules/navigator	0 > 1	1	
MIS_DIST_1WP (FLOAT)	Maximal horizontal distance from home to first waypoint Comment: Failsafe check to prevent running mission stored from previous flight at a new takeoff location. Set a value of zero or less to disable. The mission will not be started if the current waypoint is more distant than MIS_DIST_1WP from the home position. Module: modules/navigator	0 > 10000 (100)	900	m
MIS_DIST_WPS (FLOAT)	Maximal horizontal distance between waypoint Comment: Failsafe check to prevent running missions which are way too big. Set a value of zero or less to disable. The mission will not be started if any distance between two subsequent waypoints is greater than MIS_DIST_WPS. Module: modules/navigator	0 > 10000 (100)	900	m
MIS_LTRMIN_ALT (FLOAT)	Minimum Loiter altitude Comment: This is the minimum altitude the system will always obey. The intent is to stay out of ground effect. set to -1, if there shouldn't be a minimum loiter altitude Module: modules/navigator	-1 > 80 (0.5)	-1.0	m

MIS_MNT_YAW_CTL (INT32)	<p>Enable yaw control of the mount. (Only affects multicopters and ROI mission items)</p> <p>Comment: If enabled, yaw commands will be sent to the mount and the vehicle will follow its heading mode as specified by MIS_YAWMODE. If disabled, the vehicle will yaw towards the ROI.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Disable • 1: Enable <p>Module: modules/navigator</p>	0 > 1	0	
MIS_TAKEOFF_ALT (FLOAT)	<p>Take-off altitude</p> <p>Comment: This is the minimum altitude the system will take off to.</p> <p>Module: modules/navigator</p>	0 > 80 (0.5)	2.5	m
MIS_YAWMODE (INT32)	<p>Multicopter only. Yaw setpoint mode</p> <p>Comment: The values are defined in the enum mission_altitude_mode</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Heading as set by waypoint • 1: Heading towards waypoint • 2: Heading towards home • 3: Heading away from home <p>Module: modules/navigator</p>	0 > 3	1	
MIS_YAW_ERR (FLOAT)	<p>Max yaw error in degrees needed for waypoint heading acceptance</p> <p>Module: modules/navigator</p>	0 > 90 (1)	12.0	deg
MIS_YAW_TMT (FLOAT)	<p>Time in seconds we wait on reaching target heading at a waypoint if it is forced</p> <p>Comment: If set > 0 it will ignore the target heading for normal waypoint acceptance. If the waypoint forces the heading the timeout will matter. For example on VTOL forwards transition. Mainly useful for VTOLs that have less yaw authority and might not reach target yaw in wind. Disabled by default.</p> <p>Module: modules/navigator</p>	-1 > 20 (1)	-1.0	s

NAV_ACC_RAD (FLOAT)	Acceptance Radius Comment: Default acceptance radius, overridden by acceptance radius of waypoint if set. For fixed wing the L1 turning distance is used for horizontal acceptance. Module: modules/navigator	0.05 > 200.0 (0.5)	10.0	m
NAV_DLL_ACT (INT32)	Set data link loss failsafe mode Comment: The data link loss failsafe will only be entered after a timeout, set by COM_DL_LOSS_T in seconds. Once the timeout occurs the selected action will be executed. Setting this parameter to 4 will enable CASA Outback Challenge rules, which are only recommended to participants of that competition. Values: <ul style="list-style-type: none"> • 0: Disabled • 1: Loiter • 2: Return to Land • 3: Land at current position • 4: Data Link Auto Recovery (CASA Outback Challenge rules) • 5: Terminate • 6: Lockdown Module: modules/navigator		0	

NAV_FORCE_VT (INT32)	Force VTOL mode takeoff and land Module: modules/navigator		1	
NAV_FW_ALT_RAD (FLOAT)	FW Altitude Acceptance Radius Comment: Acceptance radius for fixedwing altitude. Module: modules/navigator	0.05 > 200.0 (0.5)	10.0	m
NAV_LOITER_RAD (FLOAT)	Loiter radius (FW only) Comment: Default value of loiter radius for missions, loiter, RTL, etc. (fixedwing only). Module: modules/navigator	25 > 1000 (0.5)	50.0	m
NAV_MC_ALT_RAD (FLOAT)	MC Altitude Acceptance Radius Comment: Acceptance radius for multicopter altitude. Module: modules/navigator	0.05 > 200.0 (0.5)	0.8	m
NAV_RCL_ACT (INT32)	Set RC loss failsafe mode Comment: The RC loss failsafe will only be entered after a timeout, set by COM_RC_LOSS_T in seconds. If RC input checks have been disabled by setting the COM_RC_IN_MODE param it will not be triggered. Setting this parameter to 4 will enable CASA Outback Challenge rules, which are only recommended to participants of that competition. Values: <ul style="list-style-type: none"> • 0: Disabled • 1: Loiter • 2: Return to Land • 3: Land at current position • 4: RC Auto Recovery (CASA Outback Challenge rules) • 5: Terminate • 6: Lockdown Module: modules/navigator		2	

NAV_RCL_LT (FLOAT)	RC Loss Loiter Time (CASA Outback Challenge rules) Comment: The amount of time in seconds the system should loiter at current position before termination. Only applies if NAV_RCL_ACT is set to 2 (CASA Outback Challenge rules). Set to -1 to make the system skip loitering. Module: modules/navigator	-1.0 > ? (0.1)	120.0	s
NAV_TRAFF_AVOID (INT32)	Set traffic avoidance mode Comment: Enabling this will allow the system to respond to transponder data from e.g. ADSB transponders Values: <ul style="list-style-type: none"> • 0: Disabled • 1: Warn only • 2: Return to Land • 3: Land immediately Module: modules/navigator		1	
VT_WV_LND_EN (INT32)	Weather-vane mode landings for missions Module: modules/vtol_att_control		0	
VT_WV_LTR_EN (INT32)	Weather-vane mode for loiter Module: modules/vtol_att_control		0	
VT_WV_TKO_EN (INT32)	Enable weather-vane mode takeoff for missions Module: modules/vtol_att_control		0	

Mount

The module where these parameters are defined is: drivers/vmount.

Name	Description	Min > Max (Incr.)	Default	Units
MNT_DO_STAB (INT32)	Stabilize the mount (set to true for servo gimbal, false for passthrough). Does not affect MAVLINK_ROI input		0	
MNT_MAN_PITCH (INT32)	Auxiliary channel to control pitch (in AUX input or manual mode) Values: <ul style="list-style-type: none">• 0: Disable• 1: AUX1• 2: AUX2• 3: AUX3• 4: AUX4• 5: AUX5	0 > 5	0	
MNT_MAN_ROLL (INT32)	Auxiliary channel to control roll (in AUX input or manual mode) Values: <ul style="list-style-type: none">• 0: Disable• 1: AUX1• 2: AUX2• 3: AUX3• 4: AUX4• 5: AUX5	0 > 5	0	

MNT_MAN_YAW (INT32)	<p>Auxiliary channel to control yaw (in AUX input or manual mode)</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Disable • 1: AUX1 • 2: AUX2 • 3: AUX3 • 4: AUX4 • 5: AUX5 	0 > 5	0	
MNT_MAV_COMPID (INT32)	<p>Mavlink Component ID of the mount</p> <p>Comment: If MNT_MODE_OUT is MAVLINK, mount configure/control commands will be sent with this component ID.</p>		154	
MNT_MAV_SYSID (INT32)	<p>Mavlink System ID of the mount</p> <p>Comment: If MNT_MODE_OUT is MAVLINK, mount configure/control commands will be sent with this target ID.</p>		1	
MNT_MODE_IN (INT32)	<p>Mount input mode</p> <p>Comment: RC uses the AUX input channels (see MNT_MAN_* parameters), MAVLINK_ROI uses the MAV_CMD_DO_SET_ROI Mavlink message, and MAVLINK_DO_MOUNT the MAV_CMD_DO_MOUNT_CONFIGURE and MAV_CMD_DO_MOUNT_CONTROL messages to control a mount.</p> <p>Values:</p> <ul style="list-style-type: none"> • -1: DISABLED • 0: AUTO • 1: RC • 2: MAVLINK_ROI • 3: MAVLINK_DO_MOUNT <p>Reboot required: true</p>	-1 > 3	-1	

MNT_MODE_OUT (INT32)	Mount output mode Comment: AUX uses the mixer output Control Group #2. MAVLINK uses the MAV_CMD_DO_MOUNT_CONFIGURE and MAV_CMD_DO_MOUNT_CONTROL MavLink messages to control a mount (set MNT_MAV_SYSID & MNT_MAV_COMPID) Values: <ul style="list-style-type: none"> • 0: AUX • 1: MAVLINK 	0 > 1	0	
MNT_OB_LOCK_MODE (FLOAT)	Mixer value for selecting a locking mode if required for the gimbal (only in AUX output mode)	-1.0 > 1.0	0.0	
MNT_OB_NORM_MODE (FLOAT)	Mixer value for selecting normal mode if required by the gimbal (only in AUX output mode)	-1.0 > 1.0	-1.0	
MNT_OFF_PITCH (FLOAT)	Offset for pitch channel output in degrees	-360.0 > 360.0	0.0	
MNT_OFF_ROLL (FLOAT)	Offset for roll channel output in degrees	-360.0 > 360.0	0.0	
MNT_OFF_YAW (FLOAT)	Offset for yaw channel output in degrees	-360.0 > 360.0	0.0	

MNT_RANGE_PITCH (FLOAT)	Range of pitch channel output in degrees (only in AUX output mode)	1.0 > 720.0	360.0	
MNT_RANGE_ROLL (FLOAT)	Range of roll channel output in degrees (only in AUX output mode)	1.0 > 720.0	360.0	
MNT_RANGE_YAW (FLOAT)	Range of yaw channel output in degrees (only in AUX output mode)	1.0 > 720.0	360.0	

Multicopter Attitude Control

The module where these parameters are defined is: *modules/mc_att_control*.

Name	Description	Min > Max (Incr.)	Default	Units
MC_ACRO_EXPO (FLOAT)	Acro Expo factor applied to input of all axis: roll, pitch, yaw Comment: 0 Purely linear input curve 1 Purely cubic input curve	0 > 1	0.69	
MC_ACRO_P_MAX (FLOAT)	Max acro pitch rate default: 2 turns per second	0.0 > 1000.0 (5)	720.0	deg/s
MC_ACRO_R_MAX (FLOAT)	Max acro roll rate default: 2 turns per second	0.0 > 1000.0 (5)	720.0	deg/s
MC_ACRO_SUPEXPO (FLOAT)	Acro SuperExpo factor applied to input of all axis: roll, pitch, yaw Comment: 0 Pure Expo function 0.7 resonable shape enhancement for intuitive stick feel 0.95 very strong bent input curve only near maxima have effect	0 > 0.95	0.7	
MC_ACRO_Y_MAX (FLOAT)	Max acro yaw rate default 1.5 turns per second	0.0 > 1000.0 (5)	540.0	deg/s

MC_BAT_SCALE_EN (INT32)	Battery power level scaler Comment: This compensates for voltage drop of the battery over time by attempting to normalize performance across the operating range of the battery. The copter should constantly behave as if it was fully charged with reduced max acceleration at lower battery percentages. i.e. if hover is at 0.5 throttle at 100% battery, it will still be 0.5 at 60% battery.		0	
MC_DTERM_CUTOFF (FLOAT)	Cutoff frequency for the low pass filter on the D-term in the rate controller Comment: The D-term uses the derivative of the rate and thus is the most susceptible to noise. Therefore, using a D-term filter allows to decrease the driver-level filtering, which leads to reduced control latency and permits to increase the P gains. A value of 0 disables the filter.	0 > 1000 (10)	0.	Hz
MC_PITCHRATE_D (FLOAT)	Pitch rate D gain Comment: Pitch rate differential gain. Small values help reduce fast oscillations. If value is too big oscillations will appear again.	0.0 > ? (0.0005)	0.003	

MC_PITCHRATE_FF (FLOAT)	Pitch rate feedforward Comment: Improves tracking performance.	0.0 > ?	0.0	
MC_PITCHRATE_I (FLOAT)	Pitch rate I gain Comment: Pitch rate integral gain. Can be set to compensate static thrust difference or gravity center offset.	0.0 > ? (0.01)	0.05	
MC_PITCHRATE_MAX (FLOAT)	Max pitch rate Comment: Limit for pitch rate in manual and auto modes (except acro). Has effect for large rotations in autonomous mode, to avoid large control output and mixer saturation. This is not only limited by the vehicle's properties, but also by the maximum measurement rate of the gyro.	0.0 > 1800.0 (5)	220.0	deg/s
MC_PITCHRATE_P (FLOAT)	Pitch rate P gain Comment: Pitch rate proportional gain, i.e. control output for angular speed error 1 rad/s.	0.0 > 0.6 (0.01)	0.15	
MC_PITCH_P (FLOAT)	Pitch P gain Comment: Pitch proportional gain, i.e. desired angular speed in rad/s for error 1 rad.	0.0 > 12 (0.1)	6.5	1/s

MC_PITCH_TC (FLOAT)	Pitch time constant Comment: Reduce if the system is too twitchy, increase if the response is too slow and sluggish.	0.15 > 0.25 (0.01)	0.2	s
MC_PR_INT_LIM (FLOAT)	Pitch rate integrator limit Comment: Pitch rate integrator limit. Can be set to increase the amount of integrator available to counteract disturbances or reduced to improve settling time after large pitch moment trim changes.	0.0 > ? (0.01)	0.30	
MC_RATT_TH (FLOAT)	Threshold for Rattitude mode Comment: Manual input needed in order to override attitude control rate setpoints and instead pass manual stick inputs as rate setpoints	0.0 > 1.0 (0.01)	0.8	
MC_ROLLRATE_D (FLOAT)	Roll rate D gain Comment: Roll rate differential gain. Small values help reduce fast oscillations. If value is too big oscillations will appear again.	0.0 > 0.01 (0.0005)	0.003	
MC_ROLLRATE_FF (FLOAT)	Roll rate feedforward Comment: Improves tracking performance.	0.0 > ?	0.0	

MC_ROLLRATE_I (FLOAT)	Roll rate I gain Comment: Roll rate integral gain. Can be set to compensate static thrust difference or gravity center offset.	0.0 > ? (0.01)	0.05	
MC_ROLLRATE_MAX (FLOAT)	Max roll rate Comment: Limit for roll rate in manual and auto modes (except acro). Has effect for large rotations in autonomous mode, to avoid large control output and mixer saturation. This is not only limited by the vehicle's properties, but also by the maximum measurement rate of the gyro.	0.0 > 1800.0 (5)	220.0	deg/s
MC_ROLLRATE_P (FLOAT)	Roll rate P gain Comment: Roll rate proportional gain, i.e. control output for angular speed error 1 rad/s.	0.0 > 0.5 (0.01)	0.15	
MC_ROLL_P (FLOAT)	Roll P gain Comment: Roll proportional gain, i.e. desired angular speed in rad/s for error 1 rad.	0.0 > 12 (0.1)	6.5	1/s
MC_ROLL_TC (FLOAT)	Roll time constant Comment: Reduce if the system is too twitchy, increase if the response is too slow and sluggish.	0.15 > 0.25 (0.01)	0.2	s
MC_RR_INT_LIM (FLOAT)	Roll rate integrator limit Comment: Roll rate integrator limit. Can be set to increase the amount of integrator available to counteract disturbances or reduced to improve settling time after large roll moment trim changes.	0.0 > ? (0.01)	0.30	
MC_TPA_BREAK_D (FLOAT)	TPA D Breakpoint Comment: Throttle PID Attenuation (TPA) Magnitude of throttle setpoint at which to begin attenuating roll/pitch D gain	0.0 > 1.0 (0.1)	1.0	
MC_TPA_BREAK_I (FLOAT)	TPA I Breakpoint Comment: Throttle PID Attenuation (TPA) Magnitude of throttle setpoint at which to begin attenuating roll/pitch I gain	0.0 > 1.0 (0.1)	1.0	

MC_TPA_BREAK_P (FLOAT)	TPA P Breakpoint Comment: Throttle PID Attenuation (TPA) Magnitude of throttle setpoint at which to begin attenuating roll/pitch P gain	0.0 > 1.0 (0.1)	1.0	
MC_TPA_RATE_D (FLOAT)	TPA Rate D Comment: Throttle PID Attenuation (TPA) Rate at which to attenuate roll/pitch D gain Attenuation factor is 1.0 when throttle magnitude is below the setpoint Above the setpoint, the attenuation factor is $(1 - \text{rate} * (\text{throttle} - \text{breakpoint}) / (1.0 - \text{breakpoint}))$	0.0 > 1.0 (0.05)	0.0	
MC_TPA_RATE_I (FLOAT)	TPA Rate I Comment: Throttle PID Attenuation (TPA) Rate at which to attenuate roll/pitch I gain Attenuation factor is 1.0 when throttle magnitude is below the setpoint Above the setpoint, the attenuation factor is $(1 - \text{rate} * (\text{throttle} - \text{breakpoint}) / (1.0 - \text{breakpoint}))$	0.0 > 1.0 (0.05)	0.0	
MC_TPA_RATE_P (FLOAT)	TPA Rate P Comment: Throttle PID Attenuation (TPA) Rate at which to attenuate roll/pitch P gain Attenuation factor is 1.0 when throttle magnitude is below the setpoint Above the setpoint, the attenuation factor is $(1 - \text{rate} * (\text{throttle} - \text{breakpoint}) / (1.0 - \text{breakpoint}))$	0.0 > 1.0 (0.05)	0.0	
MC_YAWRATE_D (FLOAT)	Yaw rate D gain Comment: Yaw rate differential gain. Small values help reduce fast oscillations. If value is too big oscillations will appear again.	0.0 > ? (0.01)	0.0	
MC_YAWRATE_FF (FLOAT)	Yaw rate feedforward Comment: Improves tracking performance.	0.0 > ? (0.01)	0.0	
MC_YAWRATE_I (FLOAT)	Yaw rate I gain Comment: Yaw rate integral gain. Can be set to compensate static thrust difference or gravity center offset.	0.0 > ? (0.01)	0.1	
MC_YAWRATE_MAX (FLOAT)	Max yaw rate	0.0 > 1800.0 (5)	200.0	deg/s

MC_YAWRATE_P (FLOAT)	Yaw rate P gain Comment: Yaw rate proportional gain, i.e. control output for angular speed error 1 rad/s.	0.0 > 0.6 (0.01)	0.2	
MC_YAWRAUTO_MAX (FLOAT)	Max yaw rate in auto mode Comment: Limit for yaw rate, has effect for large rotations in autonomous mode, to avoid large control output and mixer saturation.	0.0 > 360.0 (5)	45.0	deg/s
MC_YAW_FF (FLOAT)	Yaw feed forward Comment: Feed forward weight for manual yaw control. 0 will give slow response and no overshoot, 1 - fast response and big overshoot.	0.0 > 1.0 (0.01)	0.5	
MC_YAW_P (FLOAT)	Yaw P gain Comment: Yaw proportional gain, i.e. desired angular speed in rad/s for error 1 rad.	0.0 > 5 (0.1)	2.8	1/s
MC_YR_INT_LIM (FLOAT)	Yaw rate integrator limit Comment: Yaw rate integrator limit. Can be set to increase the amount of integrator available to counteract disturbances or reduced to improve settling time after large yaw moment trim changes.	0.0 > ? (0.01)	0.30	

Multicopter Position Control

The module where these parameters are defined is: *modules/mc_pos_control*.

Name	Description	Min > Max (Incr.)	Default	Units
MPC_ACC_DOWN_MAX (FLOAT)	Maximum vertical acceleration in velocity controlled modes down	2.0 > 15.0 (1)	10.0	m/s/s
MPC_ACC_HOR (FLOAT)	Acceleration for auto and for manual	2.0 > 15.0 (1)	5.0	m/s/s
MPC_ACC_HOR_FLOW (FLOAT)	Horizontal acceleration in manual modes when optical flow ground speed limit is removed. If full stick is being applied and the EKF starts using GPS whilst using optical flow, the vehicle will accelerate at this rate until the normal position control speed is achieved	0.2 > 2.0 (0.1)	0.5	m/s/s
MPC_ACC_HOR_MAX (FLOAT)	Maximum horizontal acceleration for auto mode and maximum deceleration for manual mode	2.0 > 15.0 (1)	10.0	m/s/s
MPC_ACC_UP_MAX (FLOAT)	Maximum vertical acceleration in velocity controlled modes upward	2.0 > 15.0 (1)	10.0	m/s/s

MPC_ALT_MODE (INT32)	Altitude control mode, note mode 1 only tested with LPE Values: <ul style="list-style-type: none"> • 0: Altitude following • 1: Terrain following 	0 > 1	0	
MPC_CRUISE_90 (FLOAT)	Cruise speed when angle prev-current/current-next setpoint is 90 degrees. It should be lower than MPC_XY_CRUISE Comment: Applies only in AUTO modes (includes also RTL / hold / etc.)	1.0 > 20.0 (1)	3.0	m/s
MPC_DEC_HOR_SLOW (FLOAT)	Slow horizontal manual deceleration for manual mode	0.5 > 10.0 (1)	5.0	m/s/s
MPC_HOLD_DZ (FLOAT)	Deadzone of sticks where position hold is enabled	0.0 > 1.0	0.1	
MPC_HOLD_MAX_XY (FLOAT)	Maximum horizontal velocity for which position hold is enabled (use 0 to disable check)	0.0 > 3.0	0.8	m/s
MPC_HOLD_MAX_Z (FLOAT)	Maximum vertical velocity for which position hold is enabled (use 0 to disable check)	0.0 > 3.0	0.6	m/s

MPC_JERK_MAX (FLOAT)	Maximum jerk in manual controlled mode for BRAKING to zero. If this value is below MPC_JERK_MIN, the acceleration limit in xy and z is MPC_ACC_HOR_MAX and MPC_ACC_UP_MAX respectively instantaneously when the user demands brake (=zero stick input). Otherwise the acceleration limit increases from current acceleration limit towards MPC_ACC_HOR_MAX/MPC_ACC_UP_MAX with jerk limit	0.0 > 15.0 (1)	0.0	m/s/s/s
MPC_JERK_MIN (FLOAT)	Minimum jerk in manual controlled mode for BRAKING to zero	0.5 > 10.0 (1)	1.0	m/s/s/s
MPC_LAND_ALT1 (FLOAT)	Altitude for 1. step of slow landing (descend) Comment: Below this altitude descending velocity gets limited to a value between "MPC_Z_VEL_MAX" and "MPC_LAND_SPEED" to enable a smooth descent experience Value needs to be higher than "MPC_LAND_ALT2"	0 > 122	10.0	m

MPC_LAND_ALT2 (FLOAT)	Altitude for 2. step of slow landing (landing) Comment: Below this altitude descending velocity gets limited to "MPC_LAND_SPEED" Value needs to be lower than "MPC_LAND_ALT1"	0 > 122	5.0	m
MPC_LAND_SPEED (FLOAT)	Landing descend rate	0.6 > ?	0.7	m/s
MPC_MANTHR_MAX (FLOAT)	Maximum manual thrust Comment: Limit max allowed thrust. Setting a value of one can put the system into actuator saturation as no spread between the motors is possible any more. A value of 0.8 - 0.9 is recommended.	0.0 > 1.0 (0.01)	0.9	norm
MPC_MANTHR_MIN (FLOAT)	Minimum manual thrust Comment: Minimum vertical thrust. It's recommended to set it > 0 to avoid free fall with zero thrust.	0.0 > 1.0 (0.01)	0.08	norm
MPC_MAN_TILT_MAX (FLOAT)	Maximal tilt angle in manual or altitude mode	0.0 > 90.0	35.0	deg
MPC_MAN_Y_MAX (FLOAT)	Max manual yaw rate	0.0 > 400	200.0	deg/s

MPC_THR_HOVER (FLOAT)	<p>Hover thrust</p> <p>Comment: Vertical thrust required to hover. This value is mapped to center stick for manual throttle control. With this value set to the thrust required to hover, transition from manual to ALTCTL mode while hovering will occur with the throttle stick near center, which is then interpreted as (near) zero demand for vertical speed.</p>	0.2 > 0.8 (0.01)	0.5	norm
MPC_THR_MAX (FLOAT)	<p>Maximum thrust in auto thrust control</p> <p>Comment: Limit max allowed thrust. Setting a value of one can put the system into actuator saturation as no spread between the motors is possible any more. A value of 0.8 - 0.9 is recommended.</p>	0.0 > 0.95 (0.01)	0.9	norm
MPC_THR_MIN (FLOAT)	<p>Minimum thrust in auto thrust control</p> <p>Comment: It's recommended to set it > 0 to avoid free fall with zero thrust.</p>	0.05 > 1.0 (0.01)	0.12	norm
MPC_TILTMAX_AIR (FLOAT)	<p>Maximum tilt angle in air</p> <p>Comment: Limits maximum tilt in AUTO and POSCTRL modes during flight.</p>	0.0 > 90.0	45.0	deg

MPC_TILTMAX_LND (FLOAT)	Maximum tilt during landing Comment: Limits maximum tilt angle on landing.	0.0 > 90.0	12.0	deg
MPC_TKO_RAMP_T (FLOAT)	Position control smooth takeoff ramp time constant Comment: Increasing this value will make automatic and manual takeoff slower. If it's too slow the drone might scratch the ground and tip over.	0.1 > 1	0.4	
MPC_TKO_SPEED (FLOAT)	Takeoff climb rate	1 > 5	1.5	m/s
MPC_VELD_LP (FLOAT)	Low pass filter cut freq. for numerical velocity derivative	0.0 > 10	5.0	Hz
MPC_VEL_MANUAL (FLOAT)	Maximum horizontal velocity setpoint for manual controlled mode If velocity setpoint larger than MPC_XY_VEL_MAX is set, then the setpoint will be capped to MPC_XY_VEL_MAX	3.0 > 20.0 (1)	10.0	m/s

MPC_XY_CRUISE (FLOAT)	Maximum horizontal velocity in mission Comment: Normal horizontal velocity in AUTO modes (includes also RTL / hold / etc.) and endpoint for position stabilized mode (POSCTRL).	3.0 > 20.0 (1)	5.0	m/s
MPC_XY_MAN_EXPO (FLOAT)	Manual control stick exponential curve sensitivity attenuation with small velocity setpoints Comment: The higher the value the less sensitivity the stick has around zero while still reaching the maximum value with full stick deflection. 0 Purely linear input curve (default) 1 Purely cubic input curve	0 > 1	0.0	
MPC_XY_P (FLOAT)	Proportional gain for horizontal position error	0.0 > 2.0	0.95	
MPC_XY_VEL_D (FLOAT)	Differential gain for horizontal velocity error. Small values help reduce fast oscillations. If value is too big oscillations will appear again	0.005 > 0.1	0.01	
MPC_XY_VEL_I (FLOAT)	Integral gain for horizontal velocity error Comment: Non-zero value allows to resist wind.	0.0 > 0.1	0.02	

MPC_XY_VEL_MAX (FLOAT)	Maximum horizontal velocity Comment: Maximum horizontal velocity in AUTO mode. If higher speeds are commanded in a mission they will be capped to this velocity.	0.0 > 20.0 (1)	12.0	m/s
MPC_XY_VEL_P (FLOAT)	Proportional gain for horizontal velocity error	0.06 > 0.15	0.09	
MPC_Z_MAN_EXPO (FLOAT)	Manual control stick vertical exponential curve Comment: The higher the value the less sensitivity the stick has around zero while still reaching the maximum value with full stick deflection. 0 Purely linear input curve (default) 1 Purely cubic input curve	0 > 1	0.0	
MPC_Z_P (FLOAT)	Proportional gain for vertical position error	0.0 > 1.5	1.0	
MPC_Z_VEL_D (FLOAT)	Differential gain for vertical velocity error	0.0 > 0.1	0.0	
MPC_Z_VEL_I (FLOAT)	Integral gain for vertical velocity error Comment: Non zero value allows hovering thrust estimation on stabilized or autonomous takeoff.	0.01 > 0.1	0.02	
MPC_Z_VEL_MAX_DN (FLOAT)	Maximum vertical descent velocity Comment: Maximum vertical velocity in AUTO mode and endpoint for stabilized modes (ALTCTRL, POSCTRL).	0.5 > 4.0	1.0	m/s
MPC_Z_VEL_MAX_UP (FLOAT)	Maximum vertical ascent velocity Comment: Maximum vertical velocity in AUTO mode and endpoint for stabilized modes (ALTCTRL, POSCTRL).	0.5 > 8.0	3.0	m/s
MPC_Z_VEL_P (FLOAT)	Proportional gain for vertical velocity error	0.1 > 0.4	0.2	

PWM Outputs

Name	Description	Min > Max (Incr.)	Default	Units
MOT_SLEW_MAX (FLOAT)	<p>Minimum motor rise time (slew rate limit)</p> <p>Comment: Minimum time allowed for the motor input signal to pass through a range of 1000 PWM units. A value x means that the motor signal can only go from 1000 to 2000 PWM in maximum x seconds. Zero means that slew rate limiting is disabled.</p> <p>Module: drivers/px4fmu</p>	0.0 > ?	0.0	s/(1000*PWM)
PWM_AUX_DIS1 (INT32)	<p>Set the disarmed PWM for the AUX 1 output</p> <p>Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_AUX_DISARMED will be used</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	-1 > 2200	-1	us
PWM_AUX_DIS2 (INT32)	<p>Set the disarmed PWM for the AUX 2 output</p> <p>Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_AUX_DISARMED will be used</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	-1 > 2200	-1	us
PWM_AUX_DIS3 (INT32)	<p>Set the disarmed PWM for the AUX 3 output</p> <p>Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_AUX_DISARMED will be used</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	-1 > 2200	-1	us

PWM_AUX_DIS4 (INT32)	<p>Set the disarmed PWM for the AUX 4 output</p> <p>Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_AUX_DISARMED will be used</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	-1 > 2200	-1	us
PWM_AUX_DIS5 (INT32)	<p>Set the disarmed PWM for the AUX 5 output</p> <p>Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_AUX_DISARMED will be used</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	-1 > 2200	-1	us
PWM_AUX_DIS6 (INT32)	<p>Set the disarmed PWM for the AUX 6 output</p> <p>Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_AUX_DISARMED will be used</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	-1 > 2200	-1	us
PWM_AUX_DISARMED (INT32)	<p>Set the disarmed PWM for auxiliary outputs</p> <p>Comment: This is the PWM pulse the autopilot is outputting if not armed. The main use of this parameter is to silence ESCs when they are disarmed.</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	0 > 2200	1500	us

PWM_AUX_MAX (INT32)	Set the maximum PWM for the auxiliary outputs Comment: Set to 2000 for default or 2100 to increase servo travel Reboot required: true Module: modules/sensors	1600 > 2200	2000	us
PWM_AUX_MIN (INT32)	Set the minimum PWM for the auxiliary outputs Comment: Set to 1000 for default or 900 to increase servo travel Reboot required: true Module: modules/sensors	800 > 1400	1000	us
PWM_AUX_REV1 (INT32)	Invert direction of aux output channel 1 Comment: Enable to invert the channel. Module: drivers/px4fmu		0	
PWM_AUX_REV2 (INT32)	Invert direction of aux output channel 2 Comment: Enable to invert the channel. Module: drivers/px4fmu		0	

PWM_AUX_REV3 (INT32)	Invert direction of aux output channel 3 Comment: Enable to invert the channel. Module: drivers/px4fmu		0	
PWM_AUX_REV4 (INT32)	Invert direction of aux output channel 4 Comment: Enable to invert the channel. Module: drivers/px4fmu		0	
PWM_AUX_REV5 (INT32)	Invert direction of aux output channel 5 Comment: Enable to invert the channel. Module: drivers/px4fmu		0	
PWM_AUX_REV6 (INT32)	Invert direction of aux output channel 6 Comment: Enable to invert the channel. Module: drivers/px4fmu		0	

PWM_AUX_TRIM1 (FLOAT)	Trim value for FMU PWM output channel 1 Comment: Set to normalized offset Module: drivers/px4fmu	-0.2 > 0.2	0	
PWM_AUX_TRIM2 (FLOAT)	Trim value for FMU PWM output channel 2 Comment: Set to normalized offset Module: drivers/px4fmu	-0.2 > 0.2	0	
PWM_AUX_TRIM3 (FLOAT)	Trim value for FMU PWM output channel 3 Comment: Set to normalized offset Module: drivers/px4fmu	-0.2 > 0.2	0	
PWM_AUX_TRIM4 (FLOAT)	Trim value for FMU PWM output channel 4 Comment: Set to normalized offset Module: drivers/px4fmu	-0.2 > 0.2	0	

PWM_AUX_TRIM5 (FLOAT)	Trim value for FMU PWM output channel 5 Comment: Set to normalized offset Module: drivers/px4fmu	-0.2 > 0.2	0	
PWM_AUX_TRIM6 (FLOAT)	Trim value for FMU PWM output channel 6 Comment: Set to normalized offset Module: drivers/px4fmu	-0.2 > 0.2	0	
PWM_DISARMED (INT32)	Set the disarmed PWM for the main outputs Comment: This is the PWM pulse the autopilot is outputting if not armed. The main use of this parameter is to silence ESCs when they are disarmed. Reboot required: true Module: modules/sensors	0 > 2200	900	us
PWM_MAIN_DIS1 (INT32)	Set the disarmed PWM for the main 1 output Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_DISARMED will be used Reboot required: true Module: modules/sensors	-1 > 2200	-1	us
PWM_MAIN_DIS2 (INT32)	Set the disarmed PWM for the main 2 output Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_DISARMED will be used Reboot required: true Module: modules/sensors	-1 > 2200	-1	us

PWM_MAIN_DIS3 (INT32)	<p>Set the disarmed PWM for the main 3 output</p> <p>Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_DISARMED will be used</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	-1 > 2200	-1	us
PWM_MAIN_DIS4 (INT32)	<p>Set the disarmed PWM for the main 4 output</p> <p>Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_DISARMED will be used</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	-1 > 2200	-1	us
PWM_MAIN_DIS5 (INT32)	<p>Set the disarmed PWM for the main 5 output</p> <p>Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_DISARMED will be used</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	-1 > 2200	-1	us
PWM_MAIN_DIS6 (INT32)	<p>Set the disarmed PWM for the main 6 output</p> <p>Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_DISARMED will be used</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	-1 > 2200	-1	us

PWM_MAIN_DIS7 (INT32)	Set the disarmed PWM for the main 7 output Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_DISARMED will be used Reboot required: true Module: modules/sensors	-1 > 2200	-1	us
PWM_MAIN_DIS8 (INT32)	Set the disarmed PWM for the main 8 output Comment: This is the PWM pulse the autopilot is outputting if not armed. When set to -1 the value for PWM_DISARMED will be used Reboot required: true Module: modules/sensors	-1 > 2200	-1	us
PWM_MAIN_REV1 (INT32)	Invert direction of main output channel 1 Comment: Enable to invert the channel. Module: drivers/px4io		0	

PWM_MAIN_REV2 (INT32)	Invert direction of main output channel 2 Comment: Enable to invert the channel. Module: drivers/px4io		0	
PWM_MAIN_REV3 (INT32)	Invert direction of main output channel 3 Comment: Enable to invert the channel. Module: drivers/px4io		0	
PWM_MAIN_REV4 (INT32)	Invert direction of main output channel 4 Comment: Enable to invert the channel. Module: drivers/px4io		0	
PWM_MAIN_REV5 (INT32)	Invert direction of main output channel 5 Comment: Enable to invert the channel. Module: drivers/px4io		0	

PWM_MAIN_REV6 (INT32)	Invert direction of main output channel 6 Comment: Enable to invert the channel. Module: drivers/px4io		0	
PWM_MAIN_REV7 (INT32)	Invert direction of main output channel 7 Comment: Enable to invert the channel. Module: drivers/px4io		0	
PWM_MAIN_REV8 (INT32)	Invert direction of main output channel 8 Comment: Enable to invert the channel. Module: drivers/px4io		0	
PWM_MAIN_TRIM1 (FLOAT)	Trim value for main output channel 1 Comment: Set to normalized offset Module: drivers/px4io	-0.2 > 0.2	0	

PWM_MAIN_TRIM2 (FLOAT)	Trim value for main output channel 2 Comment: Set to normalized offset Module: drivers/px4io	-0.2 > 0.2	0	
PWM_MAIN_TRIM3 (FLOAT)	Trim value for main output channel 3 Comment: Set to normalized offset Module: drivers/px4io	-0.2 > 0.2	0	
PWM_MAIN_TRIM4 (FLOAT)	Trim value for main output channel 4 Comment: Set to normalized offset Module: drivers/px4io	-0.2 > 0.2	0	
PWM_MAIN_TRIM5 (FLOAT)	Trim value for main output channel 5 Comment: Set to normalized offset Module: drivers/px4io	-0.2 > 0.2	0	
PWM_MAIN_TRIM6 (FLOAT)	Trim value for main output channel 6 Comment: Set to normalized offset Module: drivers/px4io	-0.2 > 0.2	0	

PWM_MAIN_TRIM7 (FLOAT)	Trim value for main output channel 7 Comment: Set to normalized offset Module: drivers/px4io	-0.2 > 0.2	0	
PWM_MAIN_TRIM8 (FLOAT)	Trim value for main output channel 8 Comment: Set to normalized offset Module: drivers/px4io	-0.2 > 0.2	0	
PWM_MAX (INT32)	Set the maximum PWM for the main outputs Comment: Set to 2000 for industry default or 2100 to increase servo travel. Reboot required: true Module: modules/sensors	1600 > 2200	2000	us
PWM_MIN (INT32)	Set the minimum PWM for the main outputs Comment: Set to 1000 for industry default or 900 to increase servo travel. Reboot required: true Module: modules/sensors	800 > 1400	1000	us

PWM_RATE (INT32)	<p>Set the PWM output frequency for the main outputs</p> <p>Comment: Set to 400 for industry default or 1000 for high frequency ESCs. Set to 0 for Oneshot125.</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	-1 > 2000	400	Hz
PWM_SBUS_MODE (INT32)	<p>S.BUS out</p> <p>Comment: Set to 1 to enable S.BUS version 1 output instead of RSSI.</p> <p>Module: drivers/px4io</p>		0	
THR_MDL_FAC (FLOAT)	<p>Thrust to PWM model parameter</p> <p>Comment: Parameter used to model the relationship between static thrust and motor input PWM. Model is: thrust = (1-factor)*PWM + factor * PWM^2</p> <p>Module: drivers/px4fmu</p>	0.0 > 1.0	0.0	

Payload drop

The module where these parameters are defined is: *examples/bottle_drop*.

Name	Description	Min > Max (Incr.)	Default	Units
BD_GPROPERTIES (FLOAT)	Ground drag property Comment: This parameter encodes the ground drag coefficient and the corresponding decrease in wind speed from the plane altitude to ground altitude.	0.001 > 0.1	0.03	
BD_OBJ_CD (FLOAT)	Payload drag coefficient of the dropped object Comment: The drag coefficient (cd) is the typical drag constant for air. It is in general object specific, but the closest primitive shape to the actual object should give good results: http://en.wikipedia.org/wiki/Drag_coefficient	0.08 > 1.5	0.1	
BD_OBJ_MASS (FLOAT)	Payload mass Comment: A typical small toy ball: 0.025 kg OBC water bottle: 0.6 kg	0.001 > 5.0	0.6	kg
BD_OBJ_SURFACE (FLOAT)	Payload front surface area Comment: A typical small toy ball: $(0.045 * 0.045) / 4.0 * \pi = 0.001590 \text{ m}^2$ OBC water bottle: $(0.063 * 0.063) / 4.0 * \pi = 0.003117 \text{ m}^2$	0.001 > 0.5	0.00311724531	m ²
BD_PRECISION (FLOAT)	Drop precision Comment: If the system is closer than this distance on passing over the drop position, it will release the payload. This is a safeguard to prevent a drop out of the required accuracy.	1.0 > 80.0	30.0	m
BD_TURNRADIUS (FLOAT)	Plane turn radius Comment: The planes known minimal turn radius - use a higher value to make the plane maneuver more distant from the actual drop position. This is to ensure the wings are level during the drop.	30.0 > 500.0	120.0	m

Position Estimator INAV

The module where these parameters are defined is: *modules/position_estimator_inav*.

Name	Description	Min > Max (Incr.)	Default	Units
CBRK_NO_VISION (INT32)	Disable vision input Comment: Set to the appropriate key (328754) to disable vision input. Reboot required: true	0 > 328754	0	
INAV_DELAY_GPS (FLOAT)	GPS delay Comment: GPS delay compensation	0.0 > 1.0	0.2	s
INAV_DISAB_MOCAP (FLOAT)	Mo-cap Comment: Set to 0 if using fake GPS Values: <ul style="list-style-type: none">• 0: Mo-cap enabled• 1: Mo-cap disabled		0	
INAV_FLOW_DIST_X (FLOAT)	Flow module offset (center of rotation) in X direction Comment: Yaw X flow compensation	-1.0 > 1.0	0.0	m

INAV_FLOW_DIST_Y (FLOAT)	Flow module offset (center of rotation) in Y direction Comment: Yaw Y flow compensation	-1.0 > 1.0	0.0	m
INAV_FLOW_K (FLOAT)	Optical flow scale factor Comment: Factor to scale optical flow	0.0 > 10.0	1.35	
INAV_FLOW_Q_MIN (FLOAT)	Minimal acceptable optical flow quality Comment: 0 - lowest quality, 1 - best quality.	0.0 > 1.0	0.3	
INAV_LAND_DISP (FLOAT)	Land detector altitude dispersion threshold Comment: Dispersion threshold for triggering land detector.	0.0 > 10.0	0.7	m
INAV_LAND_T (FLOAT)	Land detector time Comment: Vehicle assumed landed if no altitude changes happened during this time on low throttle.	0.0 > 10.0	3.0	s
INAV_LAND_THR (FLOAT)	Land detector throttle threshold Comment: Value should be lower than minimal hovering thrust. Half of it is good choice.	0.0 > 1.0	0.2	

INAV_LIDAR_ERR (FLOAT)	Sonar maximal error for new surface Comment: If sonar measurement error is larger than this value it skipped (spike) or accepted as new surface level (if offset is stable).	0.0 > 1.0	0.2	m
INAV_LIDAR_EST (FLOAT)	LIDAR for altitude estimation		0	
INAV_LIDAR_OFF (FLOAT)	LIDAR calibration offset Comment: LIDAR calibration offset. Value will be added to the measured distance	-20 > 20	0.0	m
INAV_W_ACC_BIAS (FLOAT)	Accelerometer bias estimation weight Comment: Weight (cutoff frequency) for accelerometer bias estimation. 0 to disable.	0.0 > 0.1	0.05	
INAV_W_GPS_FLOW (FLOAT)	XY axis weight factor for GPS when optical flow available Comment: When optical flow data available, multiply GPS weights (for position and velocity) by this factor.	0.0 > 1.0	0.1	
INAV_W_MOCAP_P (FLOAT)	Weight for mocap system Comment: Weight (cutoff frequency) for mocap position measurements.	0.0 > 10.0	10.0	

INAV_W_XY_FLOW (FLOAT)	XY axis weight for optical flow Comment: Weight (cutoff frequency) for optical flow (velocity) measurements.	0.0 > 10.0	0.8	
INAV_W_XY_GPS_P (FLOAT)	XY axis weight for GPS position Comment: Weight (cutoff frequency) for GPS position measurements.	0.0 > 10.0	1.0	
INAV_W_XY_GPS_V (FLOAT)	XY axis weight for GPS velocity Comment: Weight (cutoff frequency) for GPS velocity measurements.	0.0 > 10.0	2.0	
INAV_W_XY_RES_V (FLOAT)	XY axis weight for resetting velocity Comment: When velocity sources lost slowly decrease estimated horizontal velocity with this weight.	0.0 > 10.0	0.5	
INAV_W_XY_VIS_P (FLOAT)	XY axis weight for vision position Comment: Weight (cutoff frequency) for vision position measurements.	0.0 > 10.0	7.0	
INAV_W_XY_VIS_V (FLOAT)	XY axis weight for vision velocity Comment: Weight (cutoff frequency) for vision velocity measurements.	0.0 > 10.0	0.0	

INAV_W_Z_BARO (FLOAT)	Z axis weight for barometer Comment: Weight (cutoff frequency) for barometer altitude measurements.	0.0 > 10.0	0.5	
INAV_W_Z_GPS_P (FLOAT)	Z axis weight for GPS Comment: Weight (cutoff frequency) for GPS altitude measurements. GPS altitude data is very noisy and should be used only as slow correction for baro offset.	0.0 > 10.0	0.005	
INAV_W_Z_GPS_V (FLOAT)	Z velocity weight for GPS Comment: Weight (cutoff frequency) for GPS altitude velocity measurements.	0.0 > 10.0	0.0	
INAV_W_Z_LIDAR (FLOAT)	Z axis weight for lidar Comment: Weight (cutoff frequency) for lidar measurements.	0.0 > 10.0	3.0	
INAV_W_Z_VIS_P (FLOAT)	Z axis weight for vision Comment: Weight (cutoff frequency) for vision altitude measurements. vision altitude data is very noisy and should be used only as slow correction for baro offset.	0.0 > 10.0	5.0	

Precision Land

The module where these parameters are defined is: *modules/navigator*.

Name	Description	Min > Max (Incr.)	Default	Units
PLD_BTOUT (FLOAT)	Landing Target Timeout Comment: Time after which the landing target is considered lost without any new measurements.	0.0 > 50 (0.5)	5.0	s
PLD_FAPPR_ALT (FLOAT)	Final approach altitude Comment: Allow final approach (without horizontal positioning) if losing landing target closer than this to the ground.	0.0 > 10 (0.1)	0.1	m
PLD_HACC_RAD (FLOAT)	Horizontal acceptance radius Comment: Start descending if closer above landing target than this.	0.0 > 10 (0.1)	0.2	m
PLD_MAX_SRCH (INT32)	Maximum number of search attempts Comment: Maximum number of times to search for the landing target if it is lost during the precision landing.	0 > 100	3	
PLD_SRCH_ALT (FLOAT)	Search altitude Comment: Altitude above home to which to climb when searching for the landing target.	0.0 > 100 (0.1)	10.0	m
PLD_SRCH_TOUT (FLOAT)	Search timeout Comment: Time allowed to search for the landing target before falling back to normal landing.	0.0 > 100 (0.1)	10.0	s

RC Receiver Configuration

The module where these parameters are defined is: *platforms/qurt/fc_addon/rc_receiver*.

Name	Description	Min > Max (Incr.)	Default	Units
RC_RECEIVER_TYPE (INT32)	RC receiver type Comment: Acceptable values: - RC_RECEIVER_SPEKTRUM = 1, - RC_RECEIVER_LEMONRX = 2,		1	

Radio Calibration

Name	Description	Min > Max (Incr.)	Default	Units
RC10_DZ (FLOAT)	RC channel 10 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	0.0	
RC10_MAX (FLOAT)	RC channel 10 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us
RC10_MIN (FLOAT)	RC channel 10 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC10_REV (FLOAT)	RC channel 10 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	

RC10_TRIM (FLOAT)	RC channel 10 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us
RC11_DZ (FLOAT)	RC channel 11 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	0.0	
RC11_MAX (FLOAT)	RC channel 11 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us
RC11_MIN (FLOAT)	RC channel 11 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC11_REV (FLOAT)	RC channel 11 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC11_TRIM (FLOAT)	RC channel 11 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us
RC12_DZ (FLOAT)	RC channel 12 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	0.0	
RC12_MAX (FLOAT)	RC channel 12 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us

RC12_MIN (FLOAT)	RC channel 12 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC12_REV (FLOAT)	RC channel 12 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC12_TRIM (FLOAT)	RC channel 12 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us
RC13_DZ (FLOAT)	RC channel 13 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	0.0	

RC13_MAX (FLOAT)	RC channel 13 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us
RC13_MIN (FLOAT)	RC channel 13 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC13_REV (FLOAT)	RC channel 13 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC13_TRIM (FLOAT)	RC channel 13 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us
RC14_DZ (FLOAT)	RC channel 14 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	0.0	
RC14_MAX (FLOAT)	RC channel 14 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us
RC14_MIN (FLOAT)	RC channel 14 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC14_REV (FLOAT)	RC channel 14 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	

RC14_TRIM (FLOAT)	RC channel 14 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us
RC15_DZ (FLOAT)	RC channel 15 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	0.0	
RC15_MAX (FLOAT)	RC channel 15 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us
RC15_MIN (FLOAT)	RC channel 15 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us

RC15_REV (FLOAT)	RC channel 15 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC15_TRIM (FLOAT)	RC channel 15 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us
RC16_DZ (FLOAT)	RC channel 16 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	0.0	
RC16_MAX (FLOAT)	RC channel 16 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us

RC16_MIN (FLOAT)	RC channel 16 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC16_REV (FLOAT)	RC channel 16 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC16_TRIM (FLOAT)	RC channel 16 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us
RC17_DZ (FLOAT)	RC channel 17 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	0.0	

RC17_MAX (FLOAT)	RC channel 17 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us
RC17_MIN (FLOAT)	RC channel 17 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC17_REV (FLOAT)	RC channel 17 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC17_TRIM (FLOAT)	RC channel 17 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us

RC18_MAX (FLOAT)	RC channel 18 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us
RC18_MIN (FLOAT)	RC channel 18 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC18_REV (FLOAT)	RC channel 18 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC18_TRIM (FLOAT)	RC channel 18 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us

RC1_DZ (FLOAT)	RC channel 1 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	10.0	us
RC1_MAX (FLOAT)	RC channel 1 maximum Comment: Maximum value for RC channel 1 Module: modules/sensors	1500.0 > 2200.0	2000.0	us
RC1_MIN (FLOAT)	RC channel 1 minimum Comment: Minimum value for RC channel 1 Module: modules/sensors	800.0 > 1500.0	1000.0	us
RC1_REV (FLOAT)	RC channel 1 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC1_TRIM (FLOAT)	RC channel 1 trim Comment: Mid point value (same as min for throttle) Module: modules/sensors	800.0 > 2200.0	1500.0	us
RC2_DZ (FLOAT)	RC channel 2 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	10.0	us
RC2_MAX (FLOAT)	RC channel 2 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000.0	us
RC2_MIN (FLOAT)	RC channel 2 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000.0	us

RC2_REV (FLOAT)	<p>RC channel 2 reverse</p> <p>Comment: Set to -1 to reverse channel.</p> <p>Values:</p> <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal <p>Module: modules/sensors</p>	-1.0 > 1.0	1.0	
RC2_TRIM (FLOAT)	<p>RC channel 2 trim</p> <p>Comment: Mid point value (has to be set to the same as min for throttle channel).</p> <p>Module: modules/sensors</p>	800.0 > 2200.0	1500.0	us
RC3_DZ (FLOAT)	<p>RC channel 3 dead zone</p> <p>Comment: The +- range of this value around the trim value will be considered as zero.</p> <p>Module: modules/sensors</p>	0.0 > 100.0	10.0	us
RC3_MAX (FLOAT)	<p>RC channel 3 maximum</p> <p>Comment: Maximum value for this channel.</p> <p>Module: modules/sensors</p>	1500.0 > 2200.0	2000	us

RC3_MIN (FLOAT)	RC channel 3 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC3_REV (FLOAT)	RC channel 3 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC3_TRIM (FLOAT)	RC channel 3 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us
RC4_DZ (FLOAT)	RC channel 4 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	10.0	us
RC4_MAX (FLOAT)	RC channel 4 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us
RC4_MIN (FLOAT)	RC channel 4 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC4_REV (FLOAT)	RC channel 4 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC4_TRIM (FLOAT)	RC channel 4 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us

RC5_DZ (FLOAT)	RC channel 5 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	10.0	
RC5_MAX (FLOAT)	RC channel 5 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us
RC5_MIN (FLOAT)	RC channel 5 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC5_REV (FLOAT)	RC channel 5 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC5_TRIM (FLOAT)	RC channel 5 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us
RC6_DZ (FLOAT)	RC channel 6 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	10.0	
RC6_MAX (FLOAT)	RC channel 6 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us
RC6_MIN (FLOAT)	RC channel 6 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us

RC6_REV (FLOAT)	<p>RC channel 6 reverse</p> <p>Comment: Set to -1 to reverse channel.</p> <p>Values:</p> <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal <p>Module: modules/sensors</p>	-1.0 > 1.0	1.0	
RC6_TRIM (FLOAT)	<p>RC channel 6 trim</p> <p>Comment: Mid point value (has to be set to the same as min for throttle channel).</p> <p>Module: modules/sensors</p>	800.0 > 2200.0	1500	us
RC7_DZ (FLOAT)	<p>RC channel 7 dead zone</p> <p>Comment: The +- range of this value around the trim value will be considered as zero.</p> <p>Module: modules/sensors</p>	0.0 > 100.0	10.0	
RC7_MAX (FLOAT)	<p>RC channel 7 maximum</p> <p>Comment: Maximum value for this channel.</p> <p>Module: modules/sensors</p>	1500.0 > 2200.0	2000	us

RC7_MIN (FLOAT)	RC channel 7 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC7_REV (FLOAT)	RC channel 7 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC7_TRIM (FLOAT)	RC channel 7 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us
RC8_DZ (FLOAT)	RC channel 8 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	10.0	
RC8_MAX (FLOAT)	RC channel 8 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us
RC8_MIN (FLOAT)	RC channel 8 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC8_REV (FLOAT)	RC channel 8 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC8_TRIM (FLOAT)	RC channel 8 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us

RC9_DZ (FLOAT)	RC channel 9 dead zone Comment: The +- range of this value around the trim value will be considered as zero. Module: modules/sensors	0.0 > 100.0	0.0	
RC9_MAX (FLOAT)	RC channel 9 maximum Comment: Maximum value for this channel. Module: modules/sensors	1500.0 > 2200.0	2000	us
RC9_MIN (FLOAT)	RC channel 9 minimum Comment: Minimum value for this channel. Module: modules/sensors	800.0 > 1500.0	1000	us
RC9_REV (FLOAT)	RC channel 9 reverse Comment: Set to -1 to reverse channel. Values: <ul style="list-style-type: none"> • -1.0: Reverse • 1.0: Normal Module: modules/sensors	-1.0 > 1.0	1.0	
RC9_TRIM (FLOAT)	RC channel 9 trim Comment: Mid point value (has to be set to the same as min for throttle channel). Module: modules/sensors	800.0 > 2200.0	1500	us
RC_CHAN_CNT (INT32)	RC channel count Comment: This parameter is used by Ground Station software to save the number of channels which were used during RC calibration. It is only meant for ground station use. Module: modules/sensors	0 > 18	0	
RC_FAILS_THR (INT32)	Failsafe channel PWM threshold Comment: Set to a value slightly above the PWM value assumed by throttle in a failsafe event, but ensure it is below the PWM value assumed by throttle during normal operation. Module: modules/sensors	0 > 2200	0	us

RC_FLT_CUTOFF (FLOAT)	Cutoff frequency for the low pass filter on roll, pitch, yaw and throttle Comment: Does not get set unless below RC_FLT_SMP_RATE/2 because of filter instability characteristics. Set to 0 to disable the filter. Module: modules/sensors	0 > ?	10.0	Hz
RC_FLT_SMP_RATE (FLOAT)	Sample rate of the remote control values for the low pass filter on roll, pitch, yaw and throttle Comment: Has an influence on the cutoff frequency precision. Module: modules/sensors	1.0 > ?	50.0	Hz
RC_MAP_AUX1 (INT32)	AUX1 Passthrough RC channel Comment: Default function: Camera pitch Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 Module: modules/sensors	0 > 18	0	

RC_MAP_AUX2 (INT32)	AUX2 Passthrough RC channel Comment: Default function: Camera roll Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 Module: modules/sensors	0 > 18	0	
-------------------------------	--	--------	---	--

RC_MAP_AUX3 (INT32)	AUX3 Passthrough RC channel Comment: Default function: Camera azimuth / yaw Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 Module: modules/sensors	0 > 18	0	
RC_MAP_AUX4 (INT32)	AUX4 Passthrough RC channel Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 Module: modules/sensors	0 > 18	0	

RC_MAP_AUX5 (INT32)	AUX5 Passthrough RC channel Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 Module: modules/sensors	0 > 18	0	
-------------------------------	---	--------	---	--

RC_MAP_FAILSAFE (INT32)	Failsafe channel mapping Comment: The RC mapping index indicates which channel is used for failsafe. If 0, whichever channel is mapped to throttle is used; otherwise, the value indicates the specific RC channel to use. Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 Module: modules/sensors	0 > 18	0	
-----------------------------------	---	--------	---	--

RC_MAP_PARAM1 (INT32)	PARAM1 tuning channel Comment: Can be used for parameter tuning with the RC. This one is further referenced as the 1st parameter channel. Set to 0 to deactivate * Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 Module: modules/sensors	0 > 18	0	
---------------------------------	---	--------	---	--

RC_MAP_PARAM2 (INT32)	PARAM2 tuning channel Comment: Can be used for parameter tuning with the RC. This one is further referenced as the 2nd parameter channel. Set to 0 to deactivate * Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 Module: modules/sensors	0 > 18	0	
---------------------------------	--	--------	---	--

RC_MAP_PARAM3 (INT32)	PARAM3 tuning channel Comment: Can be used for parameter tuning with the RC. This one is further referenced as the 3th parameter channel. Set to 0 to deactivate * Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 Module: modules/sensors	0 > 18	0	
---------------------------------	--	--------	---	--

RC_MAP_PITCH (INT32)	Pitch control channel mapping Comment: The channel index (starting from 1 for channel 1) indicates which channel should be used for reading pitch inputs from. A value of zero indicates the switch is not assigned. Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 Module: modules/sensors	0 > 18	0	
--------------------------------	--	--------	---	--

RC_MAP_ROLL (INT32)	Roll control channel mapping Comment: The channel index (starting from 1 for channel 1) indicates which channel should be used for reading roll inputs from. A value of zero indicates the switch is not assigned. Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 Module: modules/sensors	0 > 18	0	
-------------------------------	--	--------	---	--

RC_MAP_THROTTLE (INT32)	Throttle control channel mapping Comment: The channel index (starting from 1 for channel 1) indicates which channel should be used for reading throttle inputs from. A value of zero indicates the switch is not assigned. Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 Module: modules/sensors	0 > 18	0	
-----------------------------------	--	--------	---	--

RC_MAP_YAW (INT32)	<p>Yaw control channel mapping</p> <p>Comment: The channel index (starting from 1 for channel 1) indicates which channel should be used for reading yaw inputs from. A value of zero indicates the switch is not assigned.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 <p>Module: modules/sensors</p>	0 > 18	0	
---------------------------	--	--------	---	--

RC_RSSI_PWM_CHAN (INT32)	<p>PWM input channel that provides RSSI</p> <p>Comment: 0: do not read RSSI from input channel 1-18: read RSSI from specified input channel Specify the range for RSSI input with RC_RSSI_PWM_MIN and RC_RSSI_PWM_MAX parameters.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 <p>Module: drivers/px4io</p>	0 > 18	0	
------------------------------------	--	--------	---	--

RC_RSSI_PWM_MAX (INT32)	Max input value for RSSI reading Comment: Only used if RC_RSSI_PWM_CHAN > 0 Module: drivers/px4io	0 > 2000	1000	
RC_RSSI_PWM_MIN (INT32)	Min input value for RSSI reading Comment: Only used if RC_RSSI_PWM_CHAN > 0 Module: drivers/px4io	0 > 2000	2000	
TRIM_PITCH (FLOAT)	Pitch trim Comment: The trim value is the actuator control value the system needs for straight and level flight. It can be calibrated by flying manually straight and level using the RC trims and copying them using the GCS. Module: modules/commander	-0.25 > 0.25 (0.01)	0.0	
TRIM_ROLL (FLOAT)	Roll trim Comment: The trim value is the actuator control value the system needs for straight and level flight. It can be calibrated by flying manually straight and level using the RC trims and copying them using the GCS. Module: modules/commander	-0.25 > 0.25 (0.01)	0.0	
TRIM_YAW (FLOAT)	Yaw trim Comment: The trim value is the actuator control value the system needs for straight and level flight. It can be calibrated by flying manually straight and level using the RC trims and copying them using the GCS. Module: modules/commander	-0.25 > 0.25 (0.01)	0.0	

Radio Switches

The module where these parameters are defined is: *modules/sensors*.

Name	Description	Min > Max (Incr.)	Default	Units
RC_ACRO_TH (FLOAT)	Threshold for selecting acro mode Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel	-1 > 1	0.5	
RC_ARMSWITCH_TH (FLOAT)	Threshold for the arm switch Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel	-1 > 1	0.25	
RC_ASSIST_TH (FLOAT)	Threshold for selecting assist mode Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel	-1 > 1	0.25	

RC_AUTO_TH (FLOAT)	<p>Threshold for selecting auto mode</p> <p>Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel</p>	-1 > 1	0.75	
RC_GEAR_TH (FLOAT)	<p>Threshold for the landing gear switch</p> <p>Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel</p>	-1 > 1	0.25	
RC_KILLSWITCH_TH (FLOAT)	<p>Threshold for the kill switch</p> <p>Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel</p>	-1 > 1	0.25	
RC_LOITER_TH (FLOAT)	<p>Threshold for selecting loiter mode</p> <p>Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel</p>	-1 > 1	0.5	

RC_MAN_TH (FLOAT)	Threshold for the manual switch Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel	-1 > 1	0.5	
RC_MAP_ACRO_SW (INT32)	Acro switch channel Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	

RC_MAP_ARM_SW (INT32)	Arm switch channel Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	
RC_MAP_FLAPS (INT32)	Flaps channel Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	

RC_MAP_FLTMODE (INT32)	Single channel flight mode selection Comment: If this parameter is non-zero, flight modes are only selected by this channel and are assigned to six slots. Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	
RC_MAP_GEAR_SW (INT32)	Landing gear switch channel Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	

RC_MAP_KILL_SW (INT32)	Kill switch channel Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	
RC_MAP_LOITER_SW (INT32)	Loiter switch channel Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	

RC_MAP_MAN_SW (INT32)	Manual switch channel mapping Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	
---------------------------------	--	--------	---	--

RC_MAP_MODE_SW (INT32)	<p>Mode switch channel mapping</p> <p>Comment: This is the main flight mode selector. The channel index (starting from 1 for channel 1) indicates which channel should be used for deciding about the main mode. A value of zero indicates the switch is not assigned.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	
----------------------------------	--	--------	---	--

RC_MAP_OFFB_SW (INT32)	Offboard switch channel Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	
RC_MAP_POSCTL_SW (INT32)	Position Control switch channel Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	

RC_MAP_RATT_SW (INT32)	Rattitude switch channel Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	
RC_MAP_RETURN_SW (INT32)	Return switch channel Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	

RC_MAP_STAB_SW (INT32)	Stabilize switch channel mapping Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	
RC_MAP_TRANS_SW (INT32)	VTOL transition switch channel mapping Values: <ul style="list-style-type: none"> • 0: Unassigned • 1: Channel 1 • 2: Channel 2 • 3: Channel 3 • 4: Channel 4 • 5: Channel 5 • 6: Channel 6 • 7: Channel 7 • 8: Channel 8 • 9: Channel 9 • 10: Channel 10 • 11: Channel 11 • 12: Channel 12 • 13: Channel 13 • 14: Channel 14 • 15: Channel 15 • 16: Channel 16 • 17: Channel 17 • 18: Channel 18 	0 > 18	0	

RC_OFFB_TH (FLOAT)	Threshold for selecting offboard mode Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel	-1 > 1	0.5	
RC_POSCTL_TH (FLOAT)	Threshold for selecting posctl mode Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel	-1 > 1	0.5	
RC_RATT_TH (FLOAT)	Threshold for selecting rattitude mode Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel	-1 > 1	0.5	
RC_RETURN_TH (FLOAT)	Threshold for selecting return to launch mode Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel	-1 > 1	0.5	

RC_STAB_TH (FLOAT)	Threshold for the stabilize switch Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel	-1 > 1	0.5	
RC_TRANS_TH (FLOAT)	Threshold for the VTOL transition switch Comment: 0-1 indicate where in the full channel range the threshold sits 0 : min 1 : max sign indicates polarity of comparison positive : true when channel>th negative : true when channel	-1 > 1	0.25	

Return To Land

The module where these parameters are defined is: *modules/navigator*.

Name	Description	Min > Max (Incr.)	Default	Units
RTL_DESCEND_ALT (FLOAT)	RTL loiter altitude Comment: Stay at this altitude above home position after RTL descending. Land (i.e. slowly descend) from this altitude if autoland is allowed.	2 > 100 (0.5)	30	m
RTL_LAND_DELAY (FLOAT)	RTL delay Comment: Delay after descend before landing in RTL mode. If set to -1 the system will not land but loiter at RTL_DESCEND_ALT.	-1 > 300 (0.5)	-1.0	s
RTL_LAND_TYPE (INT32)	RTL land location Comment: Land at the home location or planned mission landing Values: <ul style="list-style-type: none"> • 0: Home Position • 1: Planned Landing (Mission) 		0	
RTL_MIN_DIST (FLOAT)	Minimum distance to trigger rising to a safe altitude Comment: If the system is horizontally closer than this distance to home it will land straight on home instead of raising to the return altitude first.	0.5 > 20 (0.5)	5.0	m
RTL_RETURN_ALT (FLOAT)	RTL altitude Comment: Altitude to fly back in RTL in meters	0 > 150 (0.5)	60	m

Runway Takeoff

The module where these parameters are defined is: *modules/fw_pos_control_l1/runway_takeoff*.

Name	Description	Min > Max (Incr.)	Default	Units
RWTO_AIRSPD_SCL (FLOAT)	Min. airspeed scaling factor for takeoff. Pitch up will be commanded when the following airspeed is reached: FW_AIRSPD_MIN * RWTO_AIRSPD_SCL	0.0 > 2.0 (0.01)	1.3	norm
RWTO_HDG (INT32)	Specifies which heading should be held during runway takeoff Comment: 0: airframe heading, 1: heading towards takeoff waypoint Values: <ul style="list-style-type: none"> • 0: Airframe • 1: Waypoint 	0 > 1	0	
RWTO_MAX_PITCH (FLOAT)	Max pitch during takeoff. Fixed-wing settings are used if set to 0. Note that there is also a minimum pitch of 10 degrees during takeoff, so this must be larger if set	0.0 > 60.0 (0.5)	20.0	deg
RWTO_MAX_ROLL (FLOAT)	Max roll during climbout. Roll is limited during climbout to ensure enough lift and prevents aggressive navigation before we're on a safe height	0.0 > 60.0 (0.5)	25.0	deg

RWTO_MAX_THR (FLOAT)	Max throttle during runway takeoff. (Can be used to test taxi on runway)	0.0 > 1.0 (0.01)	1.0	norm
RWTO_NAV_ALT (FLOAT)	Altitude AGL at which we have enough ground clearance to allow some roll. Until RWTO_NAV_ALT is reached the plane is held level and only rudder is used to keep the heading (see RWTO_HDG). This should be below FW_CLMBOUT_DIFF if FW_CLMBOUT_DIFF > 0	0.0 > 100.0 (1)	5.0	m
RWTO_PSP (FLOAT)	Pitch setpoint during taxi / before takeoff airspeed is reached. A taildragger with steerable wheel might need to pitch up a little to keep it's wheel on the ground before airspeed to takeoff is reached	0.0 > 20.0 (0.5)	0.0	deg
RWTO_TKOFF (INT32)	Runway takeoff with landing gear		0	

SD Logging

Name	Description	Min > Max (Incr.)	Default	Unit
SDLOG_DIRS_MAX (INT32)	<p>Maximum number of log directories to keep</p> <p>Comment: If there are more log directories than this value, the system will delete the oldest directories during startup. In addition, the system will delete old logs if there is not enough free space left. The minimum amount is 300 MB. If this is set to 0, old directories will only be removed if the free space falls below the minimum.</p> <p>Reboot required: true</p> <p>Module: modules/logger</p>	0 > 1000	0	
SDLOG_EXT (INT32)	<p>Extended logging mode</p> <p>Comment: A value of -1 indicates the command line argument should be obeyed. A value of 0 disables extended logging mode, a value of 1 enables it. This parameter is only read out before logging starts (which commonly is before arming).</p> <p>Values:</p> <ul style="list-style-type: none"> • -1: Command Line • 0: Disable • 1: Enable <p>Module: modules/sdlog2</p>	-1 > 1	-1	

SDLOG_GPSTIME (INT32)	<p>Use timestamps only if GPS 3D fix is available</p> <p>Comment: Constrain the log folder creation to only use the time stamp if a 3D GPS lock is present.</p> <p>Module: modules/sdlog2</p>		1	
SDLOG_MODE (INT32)	<p>Logging Mode</p> <p>Comment: Determines when to start and stop logging. By default, logging is started when arming the system, and stopped when disarming. This parameter is only for the new logger (SYS_LOGGER=1).</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: when armed until disarm (default) • 1: from boot until disarm • 2: from boot until shutdown <p>Reboot required: true</p> <p>Module: modules/logger</p>	0 > 2	0	
SDLOG_PRIO_BOOST (INT32)	<p>Give logging app higher thread priority to avoid data loss. This is used for gathering replay logs for the ekf2 module</p> <p>Comment: A value of 0 indicates that the default priority is used. Increasing the parameter in steps of one increases the priority.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Low priority • 1: Default priority • 2: Medium priority • 3: Max priority <p>Module: modules/sdlog2</p>	0 > 3	2	

SDLOG_PROFILE (INT32)	<p>Logging Topic Profile</p> <p>Comment: This is an integer bitmask controlling the set and rates of logged topics. The default allows for general log analysis and estimator replay, while keeping the log file size reasonably small. Enabling multiple sets leads to higher bandwidth requirements and larger log files. Set bits in the following positions to enable: 0 : Set to true to use the default set (used for general log analysis) 1 : Set to true to enable full rate estimator (EKF2) replay topics 2 : Set to true to enable topics for thermal calibration (high rate raw IMU and Baro sensor data) 3 : Set to true to enable topics for system identification (high rate actuator control and IMU data) 4 : Set to true to enable full rates for analysis of fast maneuvers (RC, attitude, rates and actuators) 5 : Set to true to enable debugging topics (debug_*.msg topics, for custom code) 6 : Set to true to enable topics for sensor comparison (low rate raw IMU, Baro and Magnetomer data)</p> <p>Bitmask:</p> <ul style="list-style-type: none"> • 0: default set (log analysis) • 1: estimator replay (EKF2) • 2: thermal calibration • 3: system identification • 4: high rate • 5: debug • 6: sensor comparison 	0 > 127	3	
---------------------------------	--	------------	---	--

	Reboot required: true Module: modules/logger			
SDLOG_RATE (INT32)	Logging rate Comment: A value of -1 indicates the commandline argument should be obeyed. A value of 0 sets the minimum rate, any other value is interpreted as rate in Hertz. This parameter is only read out before logging starts (which commonly is before arming). Module: modules/sdlog2	-1 > 250	-1	Hz
SDLOG_UTC_OFFSET (INT32)	UTC offset (unit: min) Comment: the difference in hours and minutes from Coordinated Universal Time (UTC) for a your place and date. for example, In case of South Korea(UTC+09:00), UTC offset is 540 min (9*60) refer to https://en.wikipedia.org/wiki/List_of_UTC_time_offsets Module: modules/logger	-1000 > 1000	0	min
SDLOG_UUID (INT32)	Log UUID Comment: If set to 1, add an ID to the log, which uniquely identifies the vehicle Module: modules/logger		1	

SITL

The module where these parameters are defined is: *modules/simulator*.

Name	Description	Min > Max (Incr.)	Default	Units
SIM_BAT_DRAIN (FLOAT)	Simulator Battery drain interval	1 > 86400 (1)	60	s
SITL_UDP_PRT (INT32)	Simulator UDP port		14560	

Sensor Calibration

The module where these parameters are defined is: *modules/sensors*.

Name	Description	Min > Max (Incr.)	Default	Units
CAL_ACC0_EN (INT32)	Accelerometer 0 enabled		1	
CAL_ACC0_ID (INT32)	ID of the Accelerometer that the calibration is for		0	
CAL_ACC0_XOFF (FLOAT)	Accelerometer X-axis offset		0.0	
CAL_ACC0_XSCALE (FLOAT)	Accelerometer X-axis scaling factor		1.0	
CAL_ACC0_YOFF (FLOAT)	Accelerometer Y-axis offset		0.0	
CAL_ACC0_YSCALE (FLOAT)	Accelerometer Y-axis scaling factor		1.0	
CAL_ACC0_ZOFF (FLOAT)	Accelerometer Z-axis offset		0.0	
CAL_ACC0_ZSCALE (FLOAT)	Accelerometer Z-axis scaling factor		1.0	
CAL_ACC1_EN (INT32)	Accelerometer 1 enabled		1	
CAL_ACC1_ID (INT32)	ID of the Accelerometer that the calibration is for		0	

CAL_ACC1_XOFF (FLOAT)	Accelerometer X-axis offset		0.0	
CAL_ACC1_XSCALE (FLOAT)	Accelerometer X-axis scaling factor		1.0	
CAL_ACC1_YOFF (FLOAT)	Accelerometer Y-axis offset		0.0	
CAL_ACC1_YSCALE (FLOAT)	Accelerometer Y-axis scaling factor		1.0	
CAL_ACC1_ZOFF (FLOAT)	Accelerometer Z-axis offset		0.0	
CAL_ACC1_ZSCALE (FLOAT)	Accelerometer Z-axis scaling factor		1.0	
CAL_ACC2_EN (INT32)	Accelerometer 2 enabled		1	
CAL_ACC2_ID (INT32)	ID of the Accelerometer that the calibration is for		0	
CAL_ACC2_XOFF (FLOAT)	Accelerometer X-axis offset		0.0	
CAL_ACC2_XSCALE (FLOAT)	Accelerometer X-axis scaling factor		1.0	
CAL_ACC2_YOFF (FLOAT)	Accelerometer Y-axis offset		0.0	
CAL_ACC2_YSCALE (FLOAT)	Accelerometer Y-axis scaling factor		1.0	
CAL_ACC2_ZOFF (FLOAT)	Accelerometer Z-axis offset		0.0	
CAL_ACC2_ZSCALE (FLOAT)	Accelerometer Z-axis scaling factor		1.0	
CAL_ACC_PRIME (INT32)	Primary accel ID		0	

CAL_AIR_CMODEL (INT32)	<p>Airspeed sensor compensation model for the SDP3x</p> <p>Comment: Model with Pitot CAL_AIR_TUBED_MM: Not used, 1.5 mm tubes assumed. CAL_AIR_TUBELEN: Length of the tubes connecting the pitot to the sensor. Model without Pitot (1.5 mm tubes) CAL_AIR_TUBED_MM: Not used, 1.5 mm tubes assumed. CAL_AIR_TUBELEN: Length of the tubes connecting the pitot to the sensor. Tube Pressure Drop CAL_AIR_TUBED_MM: Diameter in mm of the pitot and tubes, must have the same diameter. CAL_AIR_TUBELEN: Length of the tubes connecting the pitot to the sensor and the static + dynamic port length of the pitot.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Model with Pitot • 1: Model without Pitot (1.5 mm tubes) • 2: Tube Pressure Drop 		0	
----------------------------------	--	--	---	--

CAL_AIR_TUBED_MM (FLOAT)	Airspeed sensor tube diameter. Only used for the Tube Pressure Drop Compensation	0.1 > 100	1.5	millimeter
CAL_AIR_TUBELEN (FLOAT)	Airspeed sensor tube length Comment: See the CAL_AIR_CMODEL explanation on how this parameter should be set.	0.01 > 2.00	0.2	meter
CAL_BARO_PRIME (INT32)	Primary baro ID		0	
CAL_GYRO0_EN (INT32)	Gyro 0 enabled		1	
CAL_GYRO0_ID (INT32)	ID of the Gyro that the calibration is for		0	
CAL_GYRO0_XOFF (FLOAT)	Gyro X-axis offset		0.0	
CAL_GYRO0_XSCALE (FLOAT)	Gyro X-axis scaling factor		1.0	
CAL_GYRO0_YOFF (FLOAT)	Gyro Y-axis offset		0.0	
CAL_GYRO0_YSCALE (FLOAT)	Gyro Y-axis scaling factor		1.0	
CAL_GYRO0_ZOFF (FLOAT)	Gyro Z-axis offset		0.0	

CAL_GYRO0_ZSCALE (FLOAT)	Gyro Z-axis scaling factor		1.0	
CAL_GYRO1_EN (INT32)	Gyro 1 enabled		1	
CAL_GYRO1_ID (INT32)	ID of the Gyro that the calibration is for		0	
CAL_GYRO1_XOFF (FLOAT)	Gyro X-axis offset		0.0	
CAL_GYRO1_XSCALE (FLOAT)	Gyro X-axis scaling factor		1.0	
CAL_GYRO1_YOFF (FLOAT)	Gyro Y-axis offset		0.0	
CAL_GYRO1_YSCALE (FLOAT)	Gyro Y-axis scaling factor		1.0	
CAL_GYRO1_ZOFF (FLOAT)	Gyro Z-axis offset		0.0	
CAL_GYRO1_ZSCALE (FLOAT)	Gyro Z-axis scaling factor		1.0	
CAL_GYRO2_EN (INT32)	Gyro 2 enabled		1	
CAL_GYRO2_ID (INT32)	ID of the Gyro that the calibration is for		0	
CAL_GYRO2_XOFF (FLOAT)	Gyro X-axis offset		0.0	

CAL_GYRO2_XSCALE (FLOAT)	Gyro X-axis scaling factor		1.0	
CAL_GYRO2_YOFF (FLOAT)	Gyro Y-axis offset		0.0	
CAL_GYRO2_YSCALE (FLOAT)	Gyro Y-axis scaling factor		1.0	
CAL_GYRO2_ZOFF (FLOAT)	Gyro Z-axis offset		0.0	
CAL_GYRO2_ZSCALE (FLOAT)	Gyro Z-axis scaling factor		1.0	
CAL_GYRO_PRIME (INT32)	Primary gyro ID		0	
CAL_MAG0_EN (INT32)	Mag 0 enabled		1	
CAL_MAG0_ID (INT32)	ID of Magnetometer the calibration is for		0	

<p>CAL_MAG0_ROT (INT32)</p>	<p>Rotation of magnetometer 0 relative to airframe</p> <p>Comment: An internal magnetometer will force a value of -1, so a GCS should only attempt to configure the rotation if the value is greater than or equal to zero.</p> <p>Values:</p> <ul style="list-style-type: none"> • -1: Internal mag • 0: No rotation • 1: Yaw 45° • 2: Yaw 90° • 3: Yaw 135° • 4: Yaw 180° • 5: Yaw 225° • 6: Yaw 270° • 7: Yaw 315° • 8: Roll 180° • 9: Roll 180°, Yaw 45° • 10: Roll 180°, Yaw 90° • 11: Roll 180°, Yaw 135° • 12: Pitch 180° • 13: Roll 180°, Yaw 225° • 14: Roll 180°, Yaw 270° • 15: Roll 180°, Yaw 315° • 16: Roll 90° • 17: Roll 90°, Yaw 45° • 18: Roll 90°, Yaw 90° • 19: Roll 90°, Yaw 135° 	<p>-1 > 30</p>	<p>-1</p>	
	<ul style="list-style-type: none"> • 20: Roll 270° • 21: Roll 270°, Yaw 45° • 22: Roll 270°, Yaw 90° • 23: Roll 270°, Yaw 135° • 24: Pitch 90° • 25: Pitch 270° <p>Reboot required: true</p>			

CAL_MAG0_XOFF (FLOAT)	Magnetometer X-axis offset		0.0	
CAL_MAG0_XSCALE (FLOAT)	Magnetometer X-axis scaling factor		1.0	
CAL_MAG0_YOFF (FLOAT)	Magnetometer Y-axis offset		0.0	
CAL_MAG0_YSCALE (FLOAT)	Magnetometer Y-axis scaling factor		1.0	
CAL_MAG0_ZOFF (FLOAT)	Magnetometer Z-axis offset		0.0	
CAL_MAG0_ZSCALE (FLOAT)	Magnetometer Z-axis scaling factor		1.0	
CAL_MAG1_EN (INT32)	Mag 1 enabled		1	
CAL_MAG1_ID (INT32)	ID of Magnetometer the calibration is for		0	

CAL_MAG1_ROT (INT32)	<p>Rotation of magnetometer 1 relative to airframe</p> <p>Comment: An internal magnetometer will force a value of -1, so a GCS should only attempt to configure the rotation if the value is greater than or equal to zero.</p> <p>Values:</p> <ul style="list-style-type: none"> • -1: Internal mag • 0: No rotation • 1: Yaw 45° • 2: Yaw 90° • 3: Yaw 135° • 4: Yaw 180° • 5: Yaw 225° • 6: Yaw 270° • 7: Yaw 315° • 8: Roll 180° • 9: Roll 180°, Yaw 45° • 10: Roll 180°, Yaw 90° • 11: Roll 180°, Yaw 135° • 12: Pitch 180° • 13: Roll 180°, Yaw 225° • 14: Roll 180°, Yaw 270° • 15: Roll 180°, Yaw 315° • 16: Roll 90° • 17: Roll 90°, Yaw 45° • 18: Roll 90°, Yaw 90° • 19: Roll 90°, Yaw 135° • 20: Roll 270° 	-1 > 30	-1	
--------------------------------	--	---------	----	--

	<ul style="list-style-type: none"> • 21: Roll 270°, Yaw 45° • 22: Roll 270°, Yaw 90° • 23: Roll 270°, Yaw 135° • 24: Pitch 90° • 25: Pitch 270° <p>Reboot required: true</p>			
--	---	--	--	--

CAL_MAG1_XOFF (FLOAT)	Magnetometer X-axis offset		0.0	
CAL_MAG1_XSCALE (FLOAT)	Magnetometer X-axis scaling factor		1.0	
CAL_MAG1_YOFF (FLOAT)	Magnetometer Y-axis offset		0.0	
CAL_MAG1_YSCALE (FLOAT)	Magnetometer Y-axis scaling factor		1.0	
CAL_MAG1_ZOFF (FLOAT)	Magnetometer Z-axis offset		0.0	
CAL_MAG1_ZSCALE (FLOAT)	Magnetometer Z-axis scaling factor		1.0	
CAL_MAG2_EN (INT32)	Mag 2 enabled		1	
CAL_MAG2_ID (INT32)	ID of Magnetometer the calibration is for		0	

<p>CAL_MAG2_ROT (INT32)</p>	<p>Rotation of magnetometer 2 relative to airframe</p> <p>Comment: An internal magnetometer will force a value of -1, so a GCS should only attempt to configure the rotation if the value is greater than or equal to zero.</p> <p>Values:</p> <ul style="list-style-type: none"> • -1: Internal mag • 0: No rotation • 1: Yaw 45° • 2: Yaw 90° • 3: Yaw 135° • 4: Yaw 180° • 5: Yaw 225° • 6: Yaw 270° • 7: Yaw 315° • 8: Roll 180° • 9: Roll 180°, Yaw 45° • 10: Roll 180°, Yaw 90° • 11: Roll 180°, Yaw 135° • 12: Pitch 180° • 13: Roll 180°, Yaw 225° • 14: Roll 180°, Yaw 270° • 15: Roll 180°, Yaw 315° • 16: Roll 90° • 17: Roll 90°, Yaw 45° • 18: Roll 90°, Yaw 90° • 19: Roll 90°, Yaw 135° 	<p>-1 > 30</p>	<p>-1</p>	
	<ul style="list-style-type: none"> • 20: Roll 270° • 21: Roll 270°, Yaw 45° • 22: Roll 270°, Yaw 90° • 23: Roll 270°, Yaw 135° • 24: Pitch 90° • 25: Pitch 270° <p>Reboot required: true</p>			

CAL_MAG2_XOFF (FLOAT)	Magnetometer X-axis offset		0.0	
CAL_MAG2_XSCALE (FLOAT)	Magnetometer X-axis scaling factor		1.0	
CAL_MAG2_YOFF (FLOAT)	Magnetometer Y-axis offset		0.0	
CAL_MAG2_YSCALE (FLOAT)	Magnetometer Y-axis scaling factor		1.0	
CAL_MAG2_ZOFF (FLOAT)	Magnetometer Z-axis offset		0.0	
CAL_MAG2_ZSCALE (FLOAT)	Magnetometer Z-axis scaling factor		1.0	
CAL_MAG3_EN (INT32)	Mag 3 enabled		1	
CAL_MAG3_ID (INT32)	ID of Magnetometer the calibration is for		0	

CAL_MAG3_ROT (INT32)	<p>Rotation of magnetometer 2 relative to airframe</p> <p>Comment: An internal magnetometer will force a value of -1, so a GCS should only attempt to configure the rotation if the value is greater than or equal to zero.</p> <p>Values:</p> <ul style="list-style-type: none"> • -1: Internal mag • 0: No rotation • 1: Yaw 45° • 2: Yaw 90° • 3: Yaw 135° • 4: Yaw 180° • 5: Yaw 225° • 6: Yaw 270° • 7: Yaw 315° • 8: Roll 180° • 9: Roll 180°, Yaw 45° • 10: Roll 180°, Yaw 90° • 11: Roll 180°, Yaw 135° • 12: Pitch 180° • 13: Roll 180°, Yaw 225° • 14: Roll 180°, Yaw 270° • 15: Roll 180°, Yaw 315° • 16: Roll 90° • 17: Roll 90°, Yaw 45° • 18: Roll 90°, Yaw 90° • 19: Roll 90°, Yaw 135° • 20: Roll 270° 	-1 > 30	-1	
--------------------------------	--	---------	----	--

	<ul style="list-style-type: none"> • 21: Roll 270°, Yaw 45° • 22: Roll 270°, Yaw 90° • 23: Roll 270°, Yaw 135° • 24: Pitch 90° • 25: Pitch 270° <p>Reboot required: true</p>			
CAL_MAG3_XOFF (FLOAT)	Magnetometer X-axis offset		0.0	
CAL_MAG3_XSCALE (FLOAT)	Magnetometer X-axis scaling factor		1.0	
CAL_MAG3_YOFF (FLOAT)	Magnetometer Y-axis offset		0.0	
CAL_MAG3_YSCALE (FLOAT)	Magnetometer Y-axis scaling factor		1.0	
CAL_MAG3_ZOFF (FLOAT)	Magnetometer Z-axis offset		0.0	
CAL_MAG3_ZSCALE (FLOAT)	Magnetometer Z-axis scaling factor		1.0	
CAL_MAG_PRIME (INT32)	Primary mag ID		0	
SENS_DPRES_ANSC (FLOAT)	Differential pressure sensor analog scaling Comment: Pick the appropriate scaling from the datasheet. this number defines the (linear) conversion from voltage to Pascal (pa). For the MPXV7002DP this is 1000. NOTE: If the sensor always registers zero, try switching the static and dynamic tubes.		0	
SENS_DPRES_OFF (FLOAT)	Differential pressure sensor offset Comment: The offset (zero-reading) in Pascal		0.0	
SENS_FLOW_MINRNG (FLOAT)	Optical Flow minimum focus distance Comment: This parameter defines the minimum distance from ground required for the optical flow sensor to operate reliably. The sensor may be usable below this height, but accuracy will progressively reduce to loss of focus. *		0.7	

Sensors

Name	Description	Min > Max (Incr.)	Default	Units
CAL_MAG_SIDES (INT32)	<p>Bitfield selecting mag sides for calibration</p> <p>Comment: DETECT_ORIENTATION_TAIL_DOWN = 1 DETECT_ORIENTATION_NOSE_DOWN = 2 DETECT_ORIENTATION_LEFT = 4 DETECT_ORIENTATION_RIGHT = 8 DETECT_ORIENTATION_UPSIDE_DOWN = 16 DETECT_ORIENTATION_RIGHTSIDE_UP = 32</p> <p>Values:</p> <ul style="list-style-type: none"> • 34: Two side calibration • 38: Three side calibration • 63: Six side calibration <p>Module: modules/sensors</p>	34 > 63	63	
IMU_ACCEL_CUTOFF (FLOAT)	<p>Driver level cutoff frequency for accel</p> <p>Comment: The cutoff frequency for the 2nd order butterworth filter on the accel driver. This features is currently supported by the mpu6000 and mpu9250. This only affects the signal sent to the controllers, not the estimators. 0 disables the filter.</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	0 > 1000	30.0	Hz
IMU_GYRO_CUTOFF (FLOAT)	<p>Driver level cutoff frequency for gyro</p> <p>Comment: The cutoff frequency for the 2nd order butterworth filter on the gyro driver. This features is currently supported by the mpu6000 and mpu9250. This only affects the signal sent to the controllers, not the estimators. 0 disables the filter.</p> <p>Reboot required: true</p> <p>Module: modules/sensors</p>	0 > 1000	30.0	Hz
SENS_BARO_QNH (FLOAT)	<p>QNH for barometer</p> <p>Module: modules/sensors</p>	500 > 1500	1013.25	hPa

<p>SENS_BOARD_ROT (INT32)</p>	<p>Board rotation</p> <p>Comment: This parameter defines the rotation of the FMU board relative to the platform.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: No rotation • 1: Yaw 45° • 2: Yaw 90° • 3: Yaw 135° • 4: Yaw 180° • 5: Yaw 225° • 6: Yaw 270° • 7: Yaw 315° • 8: Roll 180° • 9: Roll 180°, Yaw 45° • 10: Roll 180°, Yaw 90° • 11: Roll 180°, Yaw 135° • 12: Pitch 180° • 13: Roll 180°, Yaw 225° • 14: Roll 180°, Yaw 270° • 15: Roll 180°, Yaw 315° • 16: Roll 90° • 17: Roll 90°, Yaw 45° • 18: Roll 90°, Yaw 90° • 19: Roll 90°, Yaw 135° • 20: Roll 270° • 21: Roll 270°, Yaw 45° • 22: Roll 270°, Yaw 90° • 23: Roll 270°, Yaw 135° • 24: Pitch 90° 		0	
	<ul style="list-style-type: none"> • 25: Pitch 270° • 26: Roll 270°, Yaw 270° • 27: Roll 180°, Pitch 270° • 28: Pitch 90°, Yaw 180 • 29: Pitch 90°, Roll 90° • 30: Yaw 293°, Pitch 68°, Roll 90° (Solo) • 31: Pitch 90°, Roll 270° • 32: Pitch 9°, Yaw 180° • 33: Pitch 45° • 34: Pitch 315° <p>Reboot required: true</p> <p>Module: modules/sensors</p>			

SENS_BOARD_X_OFF (FLOAT)	Board rotation X (Roll) offset Comment: This parameter defines a rotational offset in degrees around the X (Roll) axis. It allows the user to fine tune the board offset in the event of misalignment. Module: modules/sensors		0.0	deg
SENS_BOARD_Y_OFF (FLOAT)	Board rotation Y (Pitch) offset Comment: This parameter defines a rotational offset in degrees around the Y (Pitch) axis. It allows the user to fine tune the board offset in the event of misalignment. Module: modules/sensors		0.0	deg
SENS_BOARD_Z_OFF (FLOAT)	Board rotation Z (YAW) offset Comment: This parameter defines a rotational offset in degrees around the Z (Yaw) axis. It allows the user to fine tune the board offset in the event of misalignment. Module: modules/sensors		0.0	deg
SENS_EN_LEDDAR1 (INT32)	LeddarOne rangefinder Reboot required: true Module: drivers/distance_sensor/leddar_one		0	

SENS_EN_LL40LS (INT32)	Lidar-Lite (LL40LS) Values: <ul style="list-style-type: none"> • 0: Disabled • 1: PWM • 2: I2C Reboot required: true Module: drivers/distance_sensor/ll40ls	0 > 2	0	
SENS_EN_MB12XX (INT32)	Maxbotix Soanr (mb12xx) Reboot required: true Module: drivers/distance_sensor/mb12xx		0	
SENS_EN_SF0X (INT32)	Lightware laser rangefinder (serial) Values: <ul style="list-style-type: none"> • 0: Disabled • 1: SF02 • 2: SF10/a • 3: SF10/b • 4: SF10/c • 5: SF11/c Reboot required: true Module: drivers/distance_sensor/sf0x	0 > 4	0	

SENS_EN_SF1XX (INT32)	Lightware SF1xx/SF20/LW20 laser rangefinder (i2c) Values: <ul style="list-style-type: none"> • 0: Disabled • 1: SF10/a • 2: SF10/b • 3: SF10/c • 4: SF11/c • 5: SF/LW20 Reboot required: true Module: drivers/distance_sensor/sf1xx	0 > 5	0	
SENS_EN_TFMINI (INT32)	Benewake TFmini laser rangefinder Reboot required: true Module: drivers/distance_sensor/tfmini		0	
SENS_EN_THERMAL (INT32)	Thermal control of sensor temperature Values: <ul style="list-style-type: none"> • -1: Thermal control unavailable • 0: Thermal control off Module: modules/sensors		-1	
SENS_EN_TRANGER (INT32)	TeraRanger Rangefinder (i2c) Values: <ul style="list-style-type: none"> • 0: Disabled • 1: Autodetect • 2: TROne • 3: TREvo Reboot required: true Module: drivers/distance_sensor/teraranger	0 > 3	0	

<p>SENS_FLOW_ROT (INT32)</p>	<p>PX4Flow board rotation</p> <p>Comment: This parameter defines the yaw rotation of the PX4FLOW board relative to the vehicle body frame. Zero rotation is defined as X on flow board pointing towards front of vehicle. The recommended installation default for the PX4FLOW board is with the Y axis forward (270 deg yaw).</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: No rotation • 1: Yaw 45° • 2: Yaw 90° • 3: Yaw 135° • 4: Yaw 180° • 5: Yaw 225° • 6: Yaw 270° • 7: Yaw 315° <p>Reboot required: true</p> <p>Module: modules/sensors</p>		6	
---	---	--	---	--

Snapdragon UART ESC

The module where these parameters are defined is: *platforms/qurt/fc_addon/uart_esc*.

Name	Description	Min > Max (Incr.)	Default	Units
UART_ESC_BAUD (INT32)	ESC UART baud rate Comment: Default rate is 250Kbps, which is used in off-the-shelf QRP ESC products.		250000	
UART_ESC_MODEL (INT32)	ESC model Comment: See <code>esc_model_t</code> enum definition in <code>uart_esc_dev.h</code> for all supported ESC model enum values. Values: <ul style="list-style-type: none">• 0: ESC_200QX• 1: ESC_350QX• 2: ESC_210QC		2	
UART_ESC_MOTOR1 (INT32)	Motor 1 Mapping		4	
UART_ESC_MOTOR2 (INT32)	Motor 2 Mapping		2	
UART_ESC_MOTOR3 (INT32)	Motor 3 Mapping		1	
UART_ESC_MOTOR4 (INT32)	Motor 4 Mapping		3	

Subscriber Example

The module where these parameters are defined is: *examples/subscriber*.

Name	Description	Min > Max (Incr.)	Default	Units
SUB_INTERV (INT32)	Interval of one subscriber in the example in ms		100	ms
SUB_TESTF (FLOAT)	Float Demonstration Parameter in the Example		3.14	

Syslink

The module where these parameters are defined is: *modules/syslink*.

Name	Description	Min > Max (Incr.)	Default	Units
SLNK_RADIO_ADDR1 (INT32)	Operating address of the NRF51 (most significant byte)		231	
SLNK_RADIO_ADDR2 (INT32)	Operating address of the NRF51 (least significant 4 bytes)		3890735079	
SLNK_RADIO_CHAN (INT32)	Operating channel of the NRF51	0 > 125	80	
SLNK_RADIO_RATE (INT32)	Operating datarate of the NRF51	0 > 2	2	

System

Name	Description	Min > Max (Incr.)	Default	Units
LED_RGB_MAXBRT (INT32)	RGB Led brightness limit Comment: Set to 0 to disable, 1 for minimum brightness up to 15 (max) Module: drivers/rgbled	0 > 15	15	
SYS_AUTOCONFIG (INT32)	Automatically configure default values Comment: Set to 1 to reset parameters on next system startup (setting defaults). Platform-specific values are used if available. RC* parameters are preserved. Values: <ul style="list-style-type: none"> • 0: Keep parameters • 1: Reset parameters Module: modules/systemlib	0 > 1	0	
SYS_AUTOSTART (INT32)	Auto-start script index Comment: CHANGING THIS VALUE REQUIRES A RESTART. Defines the auto-start script used to bootstrap the system. Reboot required: true Module: modules/systemlib	0 > 99999	0	

SYS_CAL_ACCEL (INT32)	<p>Enable auto start of accelerometer thermal calibration at the next power up</p> <p>Comment: 0 : Set to 0 to do nothing 1 : Set to 1 to start a calibration at next boot This parameter is reset to zero when the the temperature calibration starts. default (0, no calibration)</p> <p>Module: modules/systemlib</p>	0 > 1	0	
SYS_CAL_BARO (INT32)	<p>Enable auto start of barometer thermal calibration at the next power up</p> <p>Comment: 0 : Set to 0 to do nothing 1 : Set to 1 to start a calibration at next boot This parameter is reset to zero when the the temperature calibration starts. default (0, no calibration)</p> <p>Module: modules/systemlib</p>	0 > 1	0	
SYS_CAL_GYRO (INT32)	<p>Enable auto start of rate gyro thermal calibration at the next power up</p> <p>Comment: 0 : Set to 0 to do nothing 1 : Set to 1 to start a calibration at next boot This parameter is reset to zero when the the temperature calibration starts. default (0, no calibration)</p> <p>Module: modules/systemlib</p>	0 > 1	0	

SYS_CAL_TDEL (INT32)	<p>Required temperature rise during thermal calibration</p> <p>Comment: A temperature increase greater than this value is required during calibration. Calibration will complete for each sensor when the temperature increase above the starting temperature exceeds the value set by SYS_CAL_TDEL. If the temperature rise is insufficient, the calibration will continue indefinitely and the board will need to be repowered to exit.</p> <p>Module: modules/systemlib</p>	10 > ?	24	deg C
SYS_CAL_TMAX (INT32)	<p>Maximum starting temperature for thermal calibration</p> <p>Comment: Temperature calibration will not start if the temperature of any sensor is higher than the value set by SYS_CAL_TMAX.</p> <p>Module: modules/systemlib</p>		10	deg C
SYS_CAL_TMIN (INT32)	<p>Minimum starting temperature for thermal calibration</p> <p>Comment: Temperature calibration for each sensor will ignore data if the temperature is lower than the value set by SYS_CAL_TMIN.</p> <p>Module: modules/systemlib</p>		5	deg C

SYS_COMPANION (INT32)	<p>TELEM2 as companion computer link</p> <p>Comment: CHANGING THIS VALUE REQUIRES A RESTART. Configures the baud rate of the TELEM2 connector as companion computer interface.</p> <p>Values:</p> <ul style="list-style-type: none"> • 0: Disabled • 10: FrSky Telemetry • 20: Crazyflie (Syslink) • 57600: Companion Link (57600 baud, 8N1) • 157600: OSD (57600 baud, 8N1) • 257600: Command Receiver (57600 baud, 8N1) • 319200: Normal Telemetry (19200 baud, 8N1) • 338400: Normal Telemetry (38400 baud, 8N1) • 357600: Normal Telemetry (57600 baud, 8N1) • 419200: Iridium Telemetry (19200 baud, 8N1) • 921600: Companion Link (921600 baud, 8N1) • 1921600: ESP8266 (921600 baud, 8N1) • 3115200: Normal Telemetry (115200 baud, 8N1) 	0 > 1921600	157600	
	<p>Reboot required: true</p> <p>Module: modules/systemlib</p>			

<p>SYS_FMU_TASK (INT32)</p>	<p>Run the FMU as a task to reduce latency</p> <p>Comment: If true, the FMU will run in a separate task instead of on the work queue. Set this if low latency is required, for example for racing. This is a trade-off between RAM usage and latency: running as a task, it requires a separate stack and directly polls on the control topics, whereas running on the work queue, it runs at a fixed update rate.</p> <p>Reboot required: true</p> <p>Module: drivers/px4fmu</p>		0	
<p>SYS_HITL (INT32)</p>	<p>Enable HITL mode on next boot</p> <p>Comment: While enabled the system will boot in HITL mode and not enable all sensors and checks. When disabled the same vehicle can be normally flown outdoors.</p> <p>Reboot required: true</p> <p>Module: modules/systemlib</p>		0	

SYS_LOGGER (INT32)	SD logger Values: <ul style="list-style-type: none"> • 0: sdlog2 (legacy) • 1: logger (default) Reboot required: true Module: modules/systemlib	0 > 1	1	
SYS_MC_EST_GROUP (INT32)	Set multicopter estimator group Comment: Set the group of estimators used for multicopters and VTOLs Values: <ul style="list-style-type: none"> • 1: local_position_estimator, attitude_estimator_q • 2: ekf2 Reboot required: true Module: modules/systemlib	1 > 2	2	
SYS_PARAM_VER (INT32)	Parameter version Comment: This monotonically increasing number encodes the parameter compatibility set. whenever it increases parameters might not be backwards compatible and ground control stations should suggest a fresh configuration. Module: modules/systemlib	0 > ?	1	

SYS_RESTART_TYPE (INT32)	Set restart type Comment: Set by px4io to indicate type of restart Values: <ul style="list-style-type: none"> • 0: Data survives resets • 1: Data survives in-flight resets only • 2: Data does not survive reset Module: modules/systemlib	0 > 2	2	
SYS_STCK_EN (INT32)	Enable stack checking Module: modules/systemlib		1	
SYS_USE_IO (INT32)	Set usage of IO board Comment: Can be used to use a standard startup script but with a FMU only set-up. Set to 0 to force the FMU only set-up. Reboot required: true Module: drivers/px4io	0 > 1	1	

Testing

Name	Description	Min > Max (Incr.)	Default	Units
TEST_1 (INT32)	Module: systemcmds/tests		2	
TEST_2 (INT32)	Module: systemcmds/tests		4	
TEST_3 (FLOAT)	Module: systemcmds/tests		5.0	
TEST_D (FLOAT)	Module: lib/controllib/controllib_test		0.01	
TEST_DEV (FLOAT)	Module: lib/controllib/controllib_test		2.0	
TEST_D_LP (FLOAT)	Module: lib/controllib/controllib_test		10.0	
TEST_HP (FLOAT)	Module: lib/controllib/controllib_test		10.0	
TEST_I (FLOAT)	Module: lib/controllib/controllib_test		0.1	
TEST_I_MAX (FLOAT)	Module: lib/controllib/controllib_test		1.0	
TEST_LP (FLOAT)	Module: lib/controllib/controllib_test		10.0	
TEST_MAX (FLOAT)	Module: lib/controllib/controllib_test		1.0	
TEST_MEAN (FLOAT)	Module: lib/controllib/controllib_test		1.0	
TEST_MIN (FLOAT)	Module: lib/controllib/controllib_test		-1.0	
TEST_P (FLOAT)	Module: lib/controllib/controllib_test		0.2	
TEST_PARAMS (INT32)	Module: systemcmds/tests		12345678	
TEST_RC2_X (INT32)	Module: systemcmds/tests		16	
TEST_RC_X (INT32)	Module: systemcmds/tests		8	
TEST_TRIM (FLOAT)	Module: lib/controllib/controllib_test		0.5	

Thermal Compensation

The module where these parameters are defined is: *modules/sensors*.

Name	Description	Min > Max (Incr.)	Default	Units
TC_A0_ID (INT32)	ID of Accelerometer that the calibration is for		0	
TC_A0_SCL_0 (FLOAT)	Accelerometer scale factor - X axis		1.0	
TC_A0_SCL_1 (FLOAT)	Accelerometer scale factor - Y axis		1.0	
TC_A0_SCL_2 (FLOAT)	Accelerometer scale factor - Z axis		1.0	
TC_A0_TMAX (FLOAT)	Accelerometer calibration maximum temperature		100.0	
TC_A0_TMIN (FLOAT)	Accelerometer calibration minimum temperature		0.0	
TC_A0_TREF (FLOAT)	Accelerometer calibration reference temperature		25.0	
TC_A0_X0_0 (FLOAT)	Accelerometer offset temperature ^0 polynomial coefficient - X axis		0.0	
TC_A0_X0_1 (FLOAT)	Accelerometer offset temperature ^0 polynomial coefficient - Y axis		0.0	
TC_A0_X0_2 (FLOAT)	Accelerometer offset temperature ^0 polynomial coefficient - Z axis		0.0	

TC_A0_X1_0 (FLOAT)	Accelerometer offset temperature ^1 polynomial coefficient - X axis		0.0	
TC_A0_X1_1 (FLOAT)	Accelerometer offset temperature ^1 polynomial coefficient - Y axis		0.0	
TC_A0_X1_2 (FLOAT)	Accelerometer offset temperature ^1 polynomial coefficient - Z axis		0.0	
TC_A0_X2_0 (FLOAT)	Accelerometer offset temperature ^2 polynomial coefficient - X axis		0.0	
TC_A0_X2_1 (FLOAT)	Accelerometer offset temperature ^2 polynomial coefficient - Y axis		0.0	
TC_A0_X2_2 (FLOAT)	Accelerometer offset temperature ^2 polynomial coefficient - Z axis		0.0	
TC_A0_X3_0 (FLOAT)	Accelerometer offset temperature ^3 polynomial coefficient - X axis		0.0	
TC_A0_X3_1 (FLOAT)	Accelerometer offset temperature ^3 polynomial coefficient - Y axis		0.0	
TC_A0_X3_2 (FLOAT)	Accelerometer offset temperature ^3 polynomial coefficient - Z axis		0.0	
TC_A1_ID (INT32)	ID of Accelerometer that the calibration is for		0	

TC_A1_SCL_0 (FLOAT)	Accelerometer scale factor - X axis		1.0	
TC_A1_SCL_1 (FLOAT)	Accelerometer scale factor - Y axis		1.0	
TC_A1_SCL_2 (FLOAT)	Accelerometer scale factor - Z axis		1.0	
TC_A1_TMAX (FLOAT)	Accelerometer calibration maximum temperature		100.0	
TC_A1_TMIN (FLOAT)	Accelerometer calibration minimum temperature		0.0	
TC_A1_TREF (FLOAT)	Accelerometer calibration reference temperature		25.0	
TC_A1_X0_0 (FLOAT)	Accelerometer offset temperature ^{^0} polynomial coefficient - X axis		0.0	
TC_A1_X0_1 (FLOAT)	Accelerometer offset temperature ^{^0} polynomial coefficient - Y axis		0.0	
TC_A1_X0_2 (FLOAT)	Accelerometer offset temperature ^{^0} polynomial coefficient - Z axis		0.0	
TC_A1_X1_0 (FLOAT)	Accelerometer offset temperature ^{^1} polynomial coefficient - X axis		0.0	
TC_A1_X1_1 (FLOAT)	Accelerometer offset temperature ^{^1} polynomial coefficient - Y axis		0.0	

TC_A1_X1_2 (FLOAT)	Accelerometer offset temperature ^1 polynomial coefficient - Z axis		0.0	
TC_A1_X2_0 (FLOAT)	Accelerometer offset temperature ^2 polynomial coefficient - X axis		0.0	
TC_A1_X2_1 (FLOAT)	Accelerometer offset temperature ^2 polynomial coefficient - Y axis		0.0	
TC_A1_X2_2 (FLOAT)	Accelerometer offset temperature ^2 polynomial coefficient - Z axis		0.0	
TC_A1_X3_0 (FLOAT)	Accelerometer offset temperature ^3 polynomial coefficient - X axis		0.0	
TC_A1_X3_1 (FLOAT)	Accelerometer offset temperature ^3 polynomial coefficient - Y axis		0.0	
TC_A1_X3_2 (FLOAT)	Accelerometer offset temperature ^3 polynomial coefficient - Z axis		0.0	
TC_A2_ID (INT32)	ID of Accelerometer that the calibration is for		0	
TC_A2_SCL_0 (FLOAT)	Accelerometer scale factor - X axis		1.0	
TC_A2_SCL_1 (FLOAT)	Accelerometer scale factor - Y axis		1.0	

TC_A2_SCL_2 (FLOAT)	Accelerometer scale factor - Z axis		1.0	
TC_A2_TMAX (FLOAT)	Accelerometer calibration maximum temperature		100.0	
TC_A2_TMIN (FLOAT)	Accelerometer calibration minimum temperature		0.0	
TC_A2_TREF (FLOAT)	Accelerometer calibration reference temperature		25.0	
TC_A2_X0_0 (FLOAT)	Accelerometer offset temperature ^{^0} polynomial coefficient - X axis		0.0	
TC_A2_X0_1 (FLOAT)	Accelerometer offset temperature ^{^0} polynomial coefficient - Y axis		0.0	
TC_A2_X0_2 (FLOAT)	Accelerometer offset temperature ^{^0} polynomial coefficient - Z axis		0.0	
TC_A2_X1_0 (FLOAT)	Accelerometer offset temperature ^{^1} polynomial coefficient - X axis		0.0	
TC_A2_X1_1 (FLOAT)	Accelerometer offset temperature ^{^1} polynomial coefficient - Y axis		0.0	
TC_A2_X1_2 (FLOAT)	Accelerometer offset temperature ^{^1} polynomial coefficient - Z axis		0.0	

TC_A2_X2_0 (FLOAT)	Accelerometer offset temperature ^2 polynomial coefficient - X axis		0.0	
TC_A2_X2_1 (FLOAT)	Accelerometer offset temperature ^2 polynomial coefficient - Y axis		0.0	
TC_A2_X2_2 (FLOAT)	Accelerometer offset temperature ^2 polynomial coefficient - Z axis		0.0	
TC_A2_X3_0 (FLOAT)	Accelerometer offset temperature ^3 polynomial coefficient - X axis		0.0	
TC_A2_X3_1 (FLOAT)	Accelerometer offset temperature ^3 polynomial coefficient - Y axis		0.0	
TC_A2_X3_2 (FLOAT)	Accelerometer offset temperature ^3 polynomial coefficient - Z axis		0.0	
TC_A_ENABLE (INT32)	Thermal compensation for accelerometer sensors	0 > 1	0	
TC_B0_ID (INT32)	ID of Barometer that the calibration is for		0	
TC_B0_SCL (FLOAT)	Barometer scale factor - X axis		1.0	
TC_B0_TMAX (FLOAT)	Barometer calibration maximum temperature		75.0	
TC_B0_TMIN (FLOAT)	Barometer calibration minimum temperature		5.0	

TC_B0_TREF (FLOAT)	Barometer calibration reference temperature		40.0	
TC_B0_X0 (FLOAT)	Barometer offset temperature ^0 polynomial coefficient		0.0	
TC_B0_X1 (FLOAT)	Barometer offset temperature ^1 polynomial coefficients		0.0	
TC_B0_X2 (FLOAT)	Barometer offset temperature ^2 polynomial coefficient		0.0	
TC_B0_X3 (FLOAT)	Barometer offset temperature ^3 polynomial coefficient		0.0	
TC_B0_X4 (FLOAT)	Barometer offset temperature ^4 polynomial coefficient		0.0	
TC_B0_X5 (FLOAT)	Barometer offset temperature ^5 polynomial coefficient		0.0	
TC_B1_ID (INT32)	ID of Barometer that the calibration is for		0	
TC_B1_SCL (FLOAT)	Barometer scale factor - X axis		1.0	
TC_B1_TMAX (FLOAT)	Barometer calibration maximum temperature		75.0	

TC_B1_TMIN (FLOAT)	Barometer calibration minimum temperature		5.0	
TC_B1_TREF (FLOAT)	Barometer calibration reference temperature		40.0	
TC_B1_X0 (FLOAT)	Barometer offset temperature ^0 polynomial coefficient		0.0	
TC_B1_X1 (FLOAT)	Barometer offset temperature ^1 polynomial coefficients		0.0	
TC_B1_X2 (FLOAT)	Barometer offset temperature ^2 polynomial coefficient		0.0	
TC_B1_X3 (FLOAT)	Barometer offset temperature ^3 polynomial coefficient		0.0	
TC_B1_X4 (FLOAT)	Barometer offset temperature ^4 polynomial coefficient		0.0	
TC_B1_X5 (FLOAT)	Barometer offset temperature ^5 polynomial coefficient		0.0	
TC_B2_ID (INT32)	ID of Barometer that the calibration is for		0	
TC_B2_SCL (FLOAT)	Barometer scale factor - X axis		1.0	

TC_B2_TMAX (FLOAT)	Barometer calibration maximum temperature		75.0	
TC_B2_TMIN (FLOAT)	Barometer calibration minimum temperature		5.0	
TC_B2_TREF (FLOAT)	Barometer calibration reference temperature		40.0	
TC_B2_X0 (FLOAT)	Barometer offset temperature ^{^0} polynomial coefficient		0.0	
TC_B2_X1 (FLOAT)	Barometer offset temperature ^{^1} polynomial coefficients		0.0	
TC_B2_X2 (FLOAT)	Barometer offset temperature ^{^2} polynomial coefficient		0.0	
TC_B2_X3 (FLOAT)	Barometer offset temperature ^{^3} polynomial coefficient		0.0	
TC_B2_X4 (FLOAT)	Barometer offset temperature ^{^4} polynomial coefficient		0.0	
TC_B2_X5 (FLOAT)	Barometer offset temperature ^{^5} polynomial coefficient		0.0	
TC_B_ENABLE (INT32)	Thermal compensation for barometric pressure sensors	0 > 1	0	

TC_G0_ID (INT32)	ID of Gyro that the calibration is for		0	
TC_G0_SCL_0 (FLOAT)	Gyro scale factor - X axis		1.0	
TC_G0_SCL_1 (FLOAT)	Gyro scale factor - Y axis		1.0	
TC_G0_SCL_2 (FLOAT)	Gyro scale factor - Z axis		1.0	
TC_G0_TMAX (FLOAT)	Gyro calibration maximum temperature		100.0	
TC_G0_TMIN (FLOAT)	Gyro calibration minimum temperature		0.0	
TC_G0_TREF (FLOAT)	Gyro calibration reference temperature		25.0	
TC_G0_X0_0 (FLOAT)	Gyro rate offset temperature ^{^0} polynomial coefficient - X axis		0.0	
TC_G0_X0_1 (FLOAT)	Gyro rate offset temperature ^{^0} polynomial coefficient - Y axis		0.0	
TC_G0_X0_2 (FLOAT)	Gyro rate offset temperature ^{^0} polynomial coefficient - Z axis		0.0	
TC_G0_X1_0 (FLOAT)	Gyro rate offset temperature ^{^1} polynomial coefficient - X axis		0.0	

TC_G0_X1_1 (FLOAT)	Gyro rate offset temperature ^1 polynomial coefficient - Y axis		0.0	
TC_G0_X1_2 (FLOAT)	Gyro rate offset temperature ^1 polynomial coefficient - Z axis		0.0	
TC_G0_X2_0 (FLOAT)	Gyro rate offset temperature ^2 polynomial coefficient - X axis		0.0	
TC_G0_X2_1 (FLOAT)	Gyro rate offset temperature ^2 polynomial coefficient - Y axis		0.0	
TC_G0_X2_2 (FLOAT)	Gyro rate offset temperature ^2 polynomial coefficient - Z axis		0.0	
TC_G0_X3_0 (FLOAT)	Gyro rate offset temperature ^3 polynomial coefficient - X axis		0.0	
TC_G0_X3_1 (FLOAT)	Gyro rate offset temperature ^3 polynomial coefficient - Y axis		0.0	
TC_G0_X3_2 (FLOAT)	Gyro rate offset temperature ^3 polynomial coefficient - Z axis		0.0	
TC_G1_ID (INT32)	ID of Gyro that the calibration is for		0	
TC_G1_SCL_0 (FLOAT)	Gyro scale factor - X axis		1.0	

TC_G1_SCL_1 (FLOAT)	Gyro scale factor - Y axis		1.0	
TC_G1_SCL_2 (FLOAT)	Gyro scale factor - Z axis		1.0	
TC_G1_TMAX (FLOAT)	Gyro calibration maximum temperature		100.0	
TC_G1_TMIN (FLOAT)	Gyro calibration minimum temperature		0.0	
TC_G1_TREF (FLOAT)	Gyro calibration reference temperature		25.0	
TC_G1_X0_0 (FLOAT)	Gyro rate offset temperature ^{^0} polynomial coefficient - X axis		0.0	
TC_G1_X0_1 (FLOAT)	Gyro rate offset temperature ^{^0} polynomial coefficient - Y axis		0.0	
TC_G1_X0_2 (FLOAT)	Gyro rate offset temperature ^{^0} polynomial coefficient - Z axis		0.0	
TC_G1_X1_0 (FLOAT)	Gyro rate offset temperature ^{^1} polynomial coefficient - X axis		0.0	
TC_G1_X1_1 (FLOAT)	Gyro rate offset temperature ^{^1} polynomial coefficient - Y axis		0.0	

TC_G1_X1_2 (FLOAT)	Gyro rate offset temperature ^1 polynomial coefficient - Z axis		0.0	
TC_G1_X2_0 (FLOAT)	Gyro rate offset temperature ^2 polynomial coefficient - X axis		0.0	
TC_G1_X2_1 (FLOAT)	Gyro rate offset temperature ^2 polynomial coefficient - Y axis		0.0	
TC_G1_X2_2 (FLOAT)	Gyro rate offset temperature ^2 polynomial coefficient - Z axis		0.0	
TC_G1_X3_0 (FLOAT)	Gyro rate offset temperature ^3 polynomial coefficient - X axis		0.0	
TC_G1_X3_1 (FLOAT)	Gyro rate offset temperature ^3 polynomial coefficient - Y axis		0.0	
TC_G1_X3_2 (FLOAT)	Gyro rate offset temperature ^3 polynomial coefficient - Z axis		0.0	
TC_G2_ID (INT32)	ID of Gyro that the calibration is for		0	
TC_G2_SCL_0 (FLOAT)	Gyro scale factor - X axis		1.0	
TC_G2_SCL_1 (FLOAT)	Gyro scale factor - Y axis		1.0	

TC_G2_SCL_2 (FLOAT)	Gyro scale factor - Z axis		1.0	
TC_G2_TMAX (FLOAT)	Gyro calibration maximum temperature		100.0	
TC_G2_TMIN (FLOAT)	Gyro calibration minimum temperature		0.0	
TC_G2_TREF (FLOAT)	Gyro calibration reference temperature		25.0	
TC_G2_X0_0 (FLOAT)	Gyro rate offset temperature ^0 polynomial coefficient - X axis		0.0	
TC_G2_X0_1 (FLOAT)	Gyro rate offset temperature ^0 polynomial coefficient - Y axis		0.0	
TC_G2_X0_2 (FLOAT)	Gyro rate offset temperature ^0 polynomial coefficient - Z axis		0.0	
TC_G2_X1_0 (FLOAT)	Gyro rate offset temperature ^1 polynomial coefficient - X axis		0.0	
TC_G2_X1_1 (FLOAT)	Gyro rate offset temperature ^1 polynomial coefficient - Y axis		0.0	
TC_G2_X1_2 (FLOAT)	Gyro rate offset temperature ^1 polynomial coefficient - Z axis		0.0	

TC_G2_X2_0 (FLOAT)	Gyro rate offset temperature ^2 polynomial coefficient - X axis		0.0	
TC_G2_X2_1 (FLOAT)	Gyro rate offset temperature ^2 polynomial coefficient - Y axis		0.0	
TC_G2_X2_2 (FLOAT)	Gyro rate offset temperature ^2 polynomial coefficient - Z axis		0.0	
TC_G2_X3_0 (FLOAT)	Gyro rate offset temperature ^3 polynomial coefficient - X axis		0.0	
TC_G2_X3_1 (FLOAT)	Gyro rate offset temperature ^3 polynomial coefficient - Y axis		0.0	
TC_G2_X3_2 (FLOAT)	Gyro rate offset temperature ^3 polynomial coefficient - Z axis		0.0	
TC_G_ENABLE (INT32)	Thermal compensation for rate gyro sensors	0 > 1	0	

UAVCAN

Name	Description	Min > Max (Incr.)	Default	Units
CANNODE_BITRATE (INT32)	UAVCAN CAN bus bitrate Module: modules/uavcannode	20000 > 1000000	1000000	
CANNODE_NODE_ID (INT32)	UAVCAN Node ID Comment: Read the specs at http://uavcan.org to learn more about Node ID. Module: modules/uavcannode	1 > 125	120	
ESC_BITRATE (INT32)	UAVCAN CAN bus bitrate Module: modules/uavcanesc	20000 > 1000000	1000000	
ESC_NODE_ID (INT32)	UAVCAN Node ID Comment: Read the specs at http://uavcan.org to learn more about Node ID. Module: modules/uavcanesc	1 > 125	120	
UAVCAN_BITRATE (INT32)	UAVCAN CAN bus bitrate Reboot required: true Module: modules/uavcan	20000 > 1000000	1000000	bit/s

UAVCAN_ENABLE (INT32)	UAVCAN mode Comment: 0 - UAVCAN disabled. 1 - Basic support for UAVCAN actuators and sensors. 2 - Full support for dynamic node ID allocation and firmware update. 3 - Sets the motor control outputs to UAVCAN and enables support for dynamic node ID allocation and firmware update. Values: <ul style="list-style-type: none"> • 0: Disabled • 2: Only Sensors • 3: Sensors and Motors Reboot required: true Module: modules/uavcan	0 > 3	0	
UAVCAN_ESC_IDLT (INT32)	UAVCAN ESC will spin at idle throttle when armed, even if the mixer outputs zero setpoints Reboot required: true Module: modules/uavcan		1	
UAVCAN_NODE_ID (INT32)	UAVCAN Node ID Comment: Read the specs at http://uavcan.org to learn more about Node ID. Reboot required: true Module: modules/uavcan	1 > 125	1	

VTOL Attitude Control

The module where these parameters are defined is: *modules/vtol_att_control*.

Name	Description	Min > Max (Incr.)	Default	Units
VT_ARSP_BLEND (FLOAT)	Transition blending airspeed Comment: Airspeed at which we can start blending both fw and mc controls. Set to 0 to disable.	0.00 > 30.00 (1)	8.0	m/s
VT_ARSP_TRANS (FLOAT)	Transition airspeed Comment: Airspeed at which we can switch to fw mode	0.00 > 30.00 (1)	10.0	m/s
VT_B_DEC_MSS (FLOAT)	Approximate deceleration during back transition Comment: The approximate deceleration during a back transition in m/s/s Used to calculate back transition distance in mission mode. A lower value will make the VTOL transition further from the destination waypoint.	0.00 > 20.00 (1)	2.0	m/s/s
VT_B_REV_DEL (FLOAT)	Delay in seconds before applying back transition throttle Set this to a value greater than 0 to give the motor time to spin down Comment: unit s	0 > 10 (1)	0.0	

VT_B_REV_OUT (FLOAT)	Output on airbrakes channel during back transition Used for airbrakes or with ESCs that have reverse thrust enabled on a separate channel Airbrakes need to be enables for your selected model/mixer	0 > 1 (0.01)	0.0	
VT_B_TRANS_DUR (FLOAT)	Duration of a back transition Comment: Time in seconds used for a back transition	0.00 > 20.00 (1)	4.0	s
VT_B_TRANS_RAMP (FLOAT)	Back transition MC motor ramp up time Comment: This sets the duration during wich the MC motors ramp up to the commanded thrust during the back transition stage.	0.0 > 20.0	3.0	s
VT_B_TRANS_THR (FLOAT)	Thottle output during back transition For ESCs and mixers that support reverse thrust on low PWM values set this to a negative value to apply active breaking For ESCs that support thrust reversal with a control channel please set VT_B_REV_OUT and set this to a positive value to apply active breaking	-1 > 1 (0.01)	0.0	

VT_DWN_PITCH_MAX (FLOAT)	Maximum allowed down-pitch the controller is able to demand. This prevents large, negative lift values being created when facing strong winds. The vehicle will use the pusher motor to accelerate forward if necessary	0.0 > 45.0	5.0	
VT_ELEV_MC_LOCK (INT32)	Lock elevons in multicopter mode Comment: If set to 1 the elevons are locked in multicopter mode		1	
VT_FWD_THRUST_SC (FLOAT)	Fixed wing thrust scale for hover forward flight Comment: Scale applied to fixed wing thrust being used as source for forward acceleration in multicopter mode. This technique can be used to avoid the plane having to pitch down a lot in order to move forward. Setting this value to 0 (default) will disable this strategy.	0.0 > 2.0	0.0	
VT_FW_ALT_ERR (FLOAT)	Adaptive QuadChute Comment: Maximum negative altitude error, when in fixed wing the altitude drops below this compared to the altitude setpoint the vehicle will transition back to MC mode and enter failsafe RTL	0.0 > 200.0	0.0	

VT_FW_DIFTHR_EN (INT32)	Differential thrust in forwards flight Comment: Set to 1 to enable differential thrust in fixed-wing flight.	0 > 1	0	
VT_FW_DIFTHR_SC (FLOAT)	Differential thrust scaling factor Comment: This factor specifies how the yaw input gets mapped to differential thrust in forwards flight.	0.0 > 1.0 (0.1)	0.1	
VT_FW_MIN_ALT (FLOAT)	QuadChute Altitude Comment: Minimum altitude for fixed wing flight, when in fixed wing the altitude drops below this altitude the vehicle will transition back to MC mode and enter failsafe RTL	0.0 > 200.0	0.0	
VT_FW_MOT_OFFID (INT32)	The channel number of motors that must be turned off in fixed wing mode	0 > 12345678 (1)	0	
VT_FW_PERM_STAB (INT32)	Permanent stabilization in fw mode Comment: If set to one this parameter will cause permanent attitude stabilization in fw mode. This parameter has been introduced for pure convenience sake.		0	

VT_FW_PITCH_TRIM (FLOAT)	Fixed wing pitch trim Comment: This parameter allows to adjust the neutral elevon position in fixed wing mode.	-1.0 > 1.0 (0.01)	0.0	
VT_FW_QC_P (INT32)	QuadChute Max Pitch Comment: Maximum pitch angle before QuadChute engages Above this the vehicle will transition back to MC mode and enter failsafe RTL	0 > 180	0	
VT_FW_QC_R (INT32)	QuadChute Max Roll Comment: Maximum roll angle before QuadChute engages Above this the vehicle will transition back to MC mode and enter failsafe RTL	0 > 180	0	
VT_F_TRANS_DUR (FLOAT)	Duration of a front transition Comment: Time in seconds used for a transition	0.00 > 20.00 (1)	5.0	s
VT_F_TR_OL_TM (FLOAT)	Airspeed less front transition time (open loop) Comment: The duration of the front transition when there is no airspeed feedback available.	1.0 > 30.0	6.0	seconds

VT_IDLE_PWM_MC (INT32)	Idle speed of VTOL when in multicopter mode	900 > 2000 (1)	900	us
VT_MOT_COUNT (INT32)	VTOL number of engines	0 > 8 (1)	0	
VT_OPT_RECOV_EN (INT32)	Optimal recovery strategy for pitch-weak tailsitters		0	
VT_TILT_FW (FLOAT)	Position of tilt servo in fw mode	0.0 > 1.0 (0.01)	1.0	
VT_TILT_MC (FLOAT)	Position of tilt servo in mc mode	0.0 > 1.0 (0.01)	0.0	
VT_TILT_TRANS (FLOAT)	Position of tilt servo in transition mode	0.0 > 1.0 (0.01)	0.3	
VT_TRANS_MIN_TM (FLOAT)	Front transition minimum time Comment: Minimum time in seconds for front transition.	0.0 > 20.0	2.0	s
VT_TRANS_P2_DUR (FLOAT)	Duration of front transition phase 2 Comment: Time in seconds it should take for the rotors to rotate forward completely from the point when the plane has picked up enough airspeed and is ready to go into fixed wind mode.	0.1 > 5.0 (0.01)	0.5	s
VT_TRANS_THR (FLOAT)	Target throttle value for pusher/puller motor during the transition to fw mode	0.0 > 1.0 (0.01)	0.6	
VT_TRANS_TIMEOUT (FLOAT)	Front transition timeout Comment: Time in seconds after which transition will be cancelled. Disabled if set to 0.	0.00 > 30.00 (1)	15.0	s
VT_TYPE (INT32)	VTOL Type (Tailsitter=0, Tiltrotor=1, Standard=2) Values: <ul style="list-style-type: none"> • 0: Tailsitter • 1: Tiltrotor • 2: Standard 	0 > 2	0	
VT_WV_YAWR_SCL (FLOAT)	Weather-vane yaw rate scale Comment: The desired yawrate from the controller will be scaled in order to avoid yaw fighting against the wind.	0.0 > 1.0 (0.01)	0.15	

Miscellaneous

Name	Description	Min > Max (Incr.)	Default	Units
EXFW_HDNG_P (FLOAT)	Module: examples/fixedwing_control		0.1	
EXFW_PITCH_P (FLOAT)	Module: examples/fixedwing_control		0.2	
EXFW_ROLL_P (FLOAT)	Module: examples/fixedwing_control		0.2	
RV_YAW_P (FLOAT)	Module: examples/rover_steering_control		0.1	
SEG_Q2V (FLOAT)	Module: examples/segway		1.0	
SEG_TH2V_I (FLOAT)	Module: examples/segway		0.0	
SEG_TH2V_I_MAX (FLOAT)	Module: examples/segway		0.0	
SEG_TH2V_P (FLOAT)	Module: examples/segway		10.0	

11.4 Installing driver on Ubuntu for Intel RealSense R200

This tutorial aims to give instructions on how to install the camera driver of the Intel RealSense R200 camera head in Linux environment such that the gathered images can be accessed via the Robot Operation System (ROS). The RealSense R200 camera head is depicted below:



The installation of the driver package is executed on a Ubuntu operation system (OS) that runs as a guest OS in a Virtual Box. The specifications of the host computer where the Virtual Box is running, the Virtual Box and the guest system are given below:

- Host Operation System: Windows 8
- Processor: Intel(R) Core(TM) i7-4702MQ CPU @ 2.20GHz
- Virtual Box: Oracle VM. Version 5.0.14 r105127
- Extensions: Extension package for Virtual Box installed (Needed for USB3 support)
- Guest Operation System: Linux - Ubuntu 14.04.3 LTS

The tutorial is ordered in the following way: In a first part it is shown how to install Ubuntu 14.04 as a guest OS in the Virtual Box. In a second part is shown how to install ROS Indigo and the camera driver. The ensuing frequently used expressions have the following meaning:

- Virtual Box (VB): Program that runs different Virtual Machines. In this case the Oracle VM.
- Virtual Machine (VM): The operation system that runs in the Virtual Box as a guest system. In this case Ubuntu.

Installing Ubuntu 14.04.3 LTS in Virtual Box

- Create a new Virtual Machine (VM): Linux 64-Bit.
- Download the iso file of Ubuntu 14.04.3 LTS: ([ubuntu-14.04.3-desktop-amd64.iso](#)).
- Installation of Ubuntu:
 - During the installation procedure leave the following two options unchecked:
 - Download updates while installing
 - Install this third party software
 - After the installation you might need to enable the Virtual Box to display Ubuntu on the whole desktop:
 - Start VM Ubuntu and login, Click on **Devices->Insert Guest Additions CD image** in the menu bar of the Virtual Box.
 - Click on **Run** and enter password on the windows that pop up in Ubuntu.
 - Wait until the installation is completed and then restart. Now, it should be possible to display the VM on the whole desktop.
 - If a window pops up in Ubuntu that asks whether to update, reject to update at this point.
 - Enable USB 3 Controller in Virtual Box:
 - Shut down Virtual Machine.
 - Go to the settings of the Virtual Machine to the menu selection USB and choose: "USB 3.0(xHCI)". This is only possible if you have installed the extension package for the Virtual Box.
 - Start the Virtual Machine again.

Installing ROS Indigo

- Follow instructions given at [ROS indigo installation guide](#):
 - Install Desktop-Full version.
 - Execute steps described in the sections "Initialize rosdep" and "Environment setup".

Installing camera driver

- Install git:

```
sudo apt-get install git
```

- Download and install the driver
 - Clone [RealSense_ROS repository](#):

```
git clone https://github.com/bestmodule/RealSense_ROS.git
```

- Follow instructions given in [here](#).
 - Press the enter button when the questions whether to install the following installation packages show up:

```
Intel Low Power Subsystem support in ACPI mode (MFD_INTEL_LPSS_ACPI) [N/m/y/?] (NEW)
Intel Low Power Subsystem support in PCI mode (MFD_INTEL_LPSS_PCI) [N/m/y/?] (NEW)
Dell Airplane Mode Switch driver (DELL_RBTN) [N/m/y/?] (NEW)
```

- The following error message that can appear at the end of the installation process should not lead to a malfunction of the driver:

```
rmmod: ERROR: Module uvcvideo is not currently loaded
```

- After the installation has completed, reboot the Virtual Machine.
- Test camera driver:
 - Connect the Intel RealSense camera head with the computer with a USB3 cable that is plugged into a USB3 receptacle on the computer.
 - Click on Devices->USB-> Intel Corp Intel RealSense 3D Camera R200 in the menu bar of the Virtual Box, in order to forward the camera USB connection to the Virtual Machine.
 - Execute the file [unpacked folder]/Bin/DSReadCameraInfo:
 - If the following error message appears, unplug the camera (physically unplug USB cable from the computer). Plug it in again + Click on Devices->USB-> Intel Corp Intel RealSense 3D Camera R200 in the menu bar of the Virtual Box again and execute again the file [unpacked folder]/Bin/DSReadCameraInfo.

```
DSAPI call failed at ReadCameraInfo.cpp:134!
```

- If the camera driver works and recognises the Intel RealSense R200, you should see specific information about the Intel RealSense R200 camera head.
- Installation and testing of the ROS nodlet:
 - Follow the installation instructions in the "Installation" section given [here](#), to install the ROS nodlet.
 - Follow the instructions in the "Running the R200 nodelet" section given [here](#), to test the ROS nodlet together with the Intel RealSense R200 camera head.
- If everything works, the different data streams from the Intel RealSense R200 camera are published as ROS topics.

11.5 Switching State Estimators

This page shows you which state estimators are available and how you can switch between them.

Available estimators

1. Q attitude estimator

The attitude Q estimator is a very simple, quaternion based complementary filter for attitude.

2. INAV position estimator

The INAV position estimator is a complementary filter for 3D position and velocity states.

3. LPE position estimator

The LPE position estimator is an extended kalman filter for 3D position and velocity states.

4. EKF2 attitude, position and wind states estimator

EKF2 is an extended kalman filter estimating attitude, 3D position / velocity and wind states.

How to enable different estimators

For multirotors and VTOL use the parameter **SYS_MC_EST_GROUP** to chose between the following configurations.

SYS_MC_EST_GROUP	Q Estimator	INAV	LPE	EKF2
0	enabled	enabled		
1	enabled		enabled	
2				enabled

Out-of-tree Modules

This describes the possibility to add an external module to the PX4 build.

External modules can use the same includes as internal modules and can interact with internal modules via uORB.

Usage

- `EXTERNAL_MODULES_LOCATION` needs to point to a directory with the same structure as Firmware (and thus contains a directory called `src`).
- There are two options: copy an existing module (eg. `examples/px4_simple_app`) to the external directory, or directly create a new module.
- Rename the module (including `MODULE` in `CMakeLists.txt`) or remove it from the existing Firmware cmake build config. This is to avoid conflicts with internal modules.
- Add a file `$EXTERNAL_MODULES_LOCATION/CMakeLists.txt` with content:

```
set(config_module_list_external
  modules/<new_module>
  PARENT_SCOPE
)
```

- add a line `EXTERNAL` to the `modules/<new_module>/CMakeLists.txt` within `px4_add_module`, for example like this:

```
px4_add_module(
  MODULE modules__test_app
  MAIN test_app
  STACK_MAIN 2000
  SRCS
    px4_simple_app.c
  DEPENDS
    platforms__common
  EXTERNAL
)
```

- Execute `make posix EXTERNAL_MODULES_LOCATION=<path>`. Any other build target can be used, but the build directory must not yet exist. If it already exists, you can also just set the cmake variable in the build folder. For the following incremental builds `EXTERNAL_MODULES_LOCATION` does not need to be specified anymore.

11.6 STM32 Bootloader

The code for the PX4 bootloader is available from the Github [Bootloader](#) repository.

Supported Boards

- FMUv2 (Pixhawk 1, STM32F4)
- FMUv3 (Pixhawk 2, STM32F4)
- FMUv4 (Pixracer 3 and Pixhawk 3 Pro, STM32F4)
- FMUv5 (Pixhawk 4, STM32F7)
- TAPv1 (TBA, STM32F4)
- ASCv1 (TBA, STM32F4)

Building the Bootloader

```
git clone https://github.com/PX4/Bootloader.git
cd Bootloader
make
```

After this step a range of elf files for all supported boards are present in the Bootloader directory.

Flashing the Bootloader

IMPORTANT: The right power sequence is critical for some boards to allow JTAG / SWD access. Follow these steps exactly as described. The instructions below are valid for a Blackmagic / Dronecode probe. Other JTAG probes will need different but similar steps. Developers attempting to flash the bootloader should have the required knowledge. If you do not know how to do this you probably should reconsider if you really need to change anything about the bootloader.

- Disconnect the JTAG cable
- Connect the USB power cable
- Connect the JTAG cable

Black Magic / Dronecode Probe

Using the right serial port

- On LINUX: `/dev/serial/by-id/usb-Black_Sphere_XXX-if00`

-
- On MAC OS: Make sure to use the cu.xxx port, not the tty.xxx port: `tar ext /dev/tty.usbmodemDDEasdf`

```
arm-none-eabi-gdb
(gdb) tar ext /dev/serial/by-id/usb-Black_Sphere_XXX-if00
(gdb) mon swdp_scan
(gdb) attach 1
(gdb) mon option erase
(gdb) mon erase_mass
(gdb) load tapv1_b1.elf
...
Transfer rate: 17 KB/sec, 828 bytes/write.
(gdb) kill
```

J-Link

These instructions are for the [J-Link GDB server](#).

Prerequisites

[Download the J-Link software](#) from the Segger website and install it according to their instructions.

Run the JLink GDB server

The command below is used to run the server for flight controllers that use the STM32F427VI SoC:

```
JLinkGDBServer -select USB=0 -device STM32F427VI -if SWD-DP -speed 20000
```

The `--device/SoC` for common targets is:

- **FMUv2, FMUv3, FMUv4, aerofc-v1, mindpx-v2:** STM32F427VI
- **px4fmu-v4pro:** STM32F469II
- **px4fmu-v5:** STM32F765II
- **crazyflie:** STM32F405RG

Connect GDB

```
arm-none-eabi-gdb
(gdb) tar ext :2331
(gdb) load aerofcv1_b1.elf
```

Troubleshooting

If any of the commands above are not found, you are either not using a Blackmagic probe or its software is outdated. Upgrade the on-probe software first.

If this error message occurs: `Error erasing flash with vFlashErase packet`
Disconnect the target (while leaving JTAG connected) and run

```
mon tpwr disable  
swdp_scan  
attach 1  
load tapv1_b1.elf
```

This will disable target powering and attempt another flash cycle.

12 Platform Testing and Continuous Integration

PX4 offers extensive unit testing and continuous integration facilities. This page provides an overview.

- [Unit Tests](#)
- [Continuous Integration \(CI\)](#)
- [Jenkins CI](#)
- [Integration Testing](#)
- [Docker](#)
- [Maintenance](#)

12.1 Unit Tests

PX4 provides a simple base [Unittest-class](#). Each developer is encouraged to write unit tests in the process of adding a new feature to the PX4 framework.

Writing a Test

1. Create a new `.cpp` file within `tests` with name `test_[description].cpp`.
2. Within `test_[description].cpp` include the base `unittest-class` `<unit_test.h>` and all files required to write a test for the new feature.
3. Within `test_[description].cpp` create a class `[Description]Test` that inherits from `UnitTest`.
4. Within `[Description]Test` class declare the public method `virtual bool run_tests()`.
5. Within `[Description]Test` class declare all private methods required to test the feature in question (`test1()`, `test2()`,...).
6. Within `test_[description].cpp` implement the `run_tests()` method where each `test[1,2,...]` will be run.
7. Within `test_[description].cpp`, implement the various tests.
8. At the bottom within `test_[description].cpp` declare the test.

```
ut_declare_test_c(test_[description], [Description]Test)
```

Here is a template:

```
#include <unit_test.h>
#include "[new feature].h"
...

class [Description]Test : public UnitTest
```

```

{
public:
    virtual bool run_tests();

private:
    bool test1();
    bool test2();
    ...
};

bool [Description]Test::run_tests()
{
    ut_run_test(test1)
    ut_run_test(test2)
    ...

    return (_tests_failed == 0);
}

bool [Description]Test::test1()
{
    ut_[name of one of the unit test functions](...
    ut_[name of one of the unit test functions](...
    ...

    return true;
}

bool [Description]Test::test2()
{
    ut_[name of one of the unit test functions](...
    ut_[name of one of the unit test functions](...
    ...

    return true;
}
...

```

```
ut_declare_test_c(test_[description], [Description]Test)
```

Note that `ut_[name of one of the unit test functions]` corresponds to one of the `unittest` functions defined within `unit_test.h`.

9. Within `tests_main.h` define the new test:

```
extern int test_[description](int argc, char *argv[]);
```

10. Within `tests_main.c` add description name, test function and option:

```
...
} tests[] = {
    {...
    {"[description]", test_[description], OPTION},
    ...
}
```

OPTION can be `OPT_NOALLTEST`, `OPT_NOJIGTEST` or `0` and is considered if within px4 shell one of the two commands are called:

```
pxh> tests all
```

or

```
pxh> tests jig
```

If a test has option `OPT_NOALLTEST`, then that test will be excluded when calling `tests all`. The same is true for `OPT_NOJITEST` when command `test jig` is called. Option `0` means that the test is never excluded, which is what most developer want to use.

11. Add the test `test_[description].cpp` to the [CMakeLists.txt](#).

Testing on the local machine

The following command is sufficient to start a minimal new shell with the PX4 posix port running.

```
make posix_sitl_shell none
```

The shell can then be used to e.g. execute unit tests:

```
pxh> tests [description]
```

Alternatively it is also possible to run the complete unit-tests right from bash:

```
make tests
```

To see a full list of available tests write within px4 shell:

```
pxh> tests help
```

12.2 PX4 Continuous Integration

PX4 builds and testing are spread out over multiple continuous integration services.

Travis-ci

Travis-ci is responsible for the official stable/beta/development binaries that are flashable through [QGroundControl](#). It currently uses GCC 4.9.3 included in the docker image [px4io/px4-dev-base](#) and compiles `px4fmu-{v2, v4}`, `mindpx-v2`, `tap-v1` with makefile target `qgc_firmware`.

Travis-ci also has a macOS posix sitl build which includes testing.

Semaphore

Semaphore is primarily used to compile changes for the Qualcomm Snapdragon platform, but also serves as a backup to Travis-ci using the the same [px4io/px4-dev-base](#) docker image. In addition to the set of firmware compiled by Travis-ci, Semaphore also builds for the stm32discovery, crazyflie, runs unit testing, and verifies code style.

CircleCI

CircleCI tests the proposed next version of GCC to be used for stable firmware releases using the docker image [px4io/px4-dev-nuttx-gcc_next](#). It uses the makefile target `quick_check` which compiles `px4fmu-v4_default`, `posix_sitl_default`, runs testing, and verifies code style.

12.3 Jenkins CI

Jenkins continuous integration server on [ci.px4.io](#) is used to automatically run integration tests against PX4 SITL.

Overview

- Involved components: Jenkins, Docker, PX4 POSIX SITL
- Tests run inside [Docker Containers](#)
- Jenkins executes 2 jobs: one to check each PR against master, and the other to check every push on master
-

Test Execution

Jenkins uses [run_container.bash](#) to start the container which in turn executes [run_tests.bash](#) to compile and run the tests.

If Docker is installed the same method can be used locally:

```
cd <directory_where_firmware_is_cloned>
sudo WORKSPACE=$(pwd) ./Firmware/integrationtests/run_container.bash
```

Server Setup

Installation

See setup [script/log](#) for details on how Jenkins got installed and maintained.

Configuration

- Jenkins security enabled
- Installed plugins
 - github
 - github pull request builder
 - embeddable build status plugin
 - s3 plugin
 - notification plugin
 - collapsing console sections
 - postbuildscript

Integration Testing

This is about end to end integration testing. Tests are executed automatically ([Jenkins CI](#))

ROS / MAVROS Tests

Prerequisites:

- [jMAVSim Simulator](#)
- [Gazebo Simulator](#)
- [ROS and MAVROS](#)

Execute Tests

To run the complete MAVROS test suite:

```
cd <Firmware_clone>
source integrationtests/setup_gazebo_ros.bash $(pwd)
rostopic echo /mavros/iris/altitude
```

Or with GUI to see what's happening:

```
rostopic echo /mavros/iris/altitude gui:=true headless:=false
```

Write a new MAVROS test (Python)

Currently in early stages, more streamlined support for testing (helper classes/methods etc.) to come.

1.) Create a new test script

Test scripts are located in `integrationtests/python_src/px4_it/mavros/`. See other existing scripts for examples. Also please consult the official ROS documentation on how to use [unittest](#).

Empty test skeleton:

```
#!/usr/bin/env python
# [... LICENSE ...]

#
# @author Example Author <author@example.com>
#
PKG = 'px4'

import unittest
import rospy
import rosbag

from sensor_msgs.msg import NavSatFix

class MavrosNewTest(unittest.TestCase):
    """
    Test description
    """

    def setUp(self):
        rospy.init_node('test_node', anonymous=True)
        rospy.wait_for_service('mavros/cmd/arming', 30)

        rospy.Subscriber("mavros/global_position/global", NavSatFix,
self.global_position_callback)
        self.rate = rospy.Rate(10) # 10hz
        self.has_global_pos = False

    def tearDown(self):
        pass
```

```

#
# General callback functions used in tests
#
def global_position_callback(self, data):
    self.has_global_pos = True

def test_method(self):
    """Test method description"""

    # FIXME: hack to wait for simulation to be ready
    while not self.has_global_pos:
        self.rate.sleep()

    # TODO: execute test

if __name__ == '__main__':
    import rostest
    rostest.rosrun(PKG, 'mavros_new_test', MavrosNewTest)

```

2.) Run the new test only

```

# Start simulation
cd <Firmware_clone>
source integrationtests/setup_gazebo_ros.bash $(pwd)
roslaunch px4 mavros_posix_sitl.launch

# Run test (in a new shell):
cd <Firmware_clone>
source integrationtests/setup_gazebo_ros.bash $(pwd)
roslaunch px4 mavros_new_test.py

```

3.) Add new test node to launch file

In `launch/mavros_posix_tests_iris1.launch` add new entry in test group:

```

<group ns="$(arg ns)">
  [...]
  <test test-name="mavros_new_test" pkg="px4" type="mavros_new_test.py" />
</group>

```

Run the complete test suite as described above.

12.4 PX4 Docker Containers

Docker containers are provided for the complete [PX4 development toolchain](#) including NuttX and Linux based hardware, [Gazebo Simulation](#) and [ROS](#).

This topic shows how to use the [available docker containers](#) to access the build environment in a local Linux computer.

Dockerfiles and README can be found on [Github here](#). They are built automatically on [Docker Hub](#).

Prerequisites

PX4 containers are currently only supported on Linux (if you don't have Linux you can run the container [inside a virtual machine](#)). Do not use `boot2docker` with the default Linux image because it contains no X-Server.

[Install Docker](#) for your Linux computer, preferably using one of the Docker-maintained package repositories to get the latest stable version. You can use either the *Enterprise Edition* or (free) *Community Edition*.

For local installation of non-production setups on *Ubuntu*, the quickest and easiest way to install Docker is to use the [convenience script](#) as shown below (alternative installation methods are found on the same page):

```
curl -fsSL get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

The default installation requires that you invoke *Docker* as the root user (i.e. using `sudo`). If you would like to [use Docker as a non-root user](#), you can optionally add the user to the "docker" group and then log out/in:

```
# Create docker group (may not be required)
sudo groupadd docker
# Add your user to the docker group.
sudo usermod -aG docker $USER
# Log in/out again before using docker!
```

Container Hierarchy

The available containers are listed below (from [Github](#)):

Container	Description
px4-dev-base	Base setup common to all containers
px4-dev-nuttx	NuttX toolchain
px4-dev-simulation	NuttX toolchain + simulation (jMAVSim, Gazebo)
px4-dev-ros	NuttX toolchain, simulation + ROS (incl. MAVROS)
px4-dev-raspi	Raspberry Pi toolchain
px4-dev-snapdragon	Qualcomm Snapdragon Flight toolchain
px4-dev-clang	Clang tools
px4-dev-nuttx-clang	Clang and NuttX tools

The most recent version can be accessed using the `latest` tag: `px4io/px4-dev-ros:latest` (available tags are listed for each container on hub.docker.com. For example, the `px4-dev-ros` tags can be found [here](#)).

Typically you should use a recent container, but not necessarily the latest (as this changes too often).

Use the Docker Container

The following instructions show how to build PX4 source code on the host computer using a toolchain running in a docker container. The information assumes that you have already downloaded the PX4 source code to **src/Firmware**, as shown:

```
mkdir src
cd src
git clone https://github.com/PX4/Firmware.git
cd Firmware
```

Helper Script (`docker_run.sh`)

The easiest way to use the containers is via the `docker_run.sh` helper script. This script takes a PX4 build command as an argument (e.g. `make tests`). It starts up docker with a recent version (hard coded) of the appropriate container and sensible environment settings.

For example, to build SITL you would call (from within the **/Firmware** directory):

```
sudo ./Tools/docker_run.sh 'make posix_sitl_default'
```

Or to start a bash session using the NuttX toolchain:

```
sudo ./Tools/docker_run.sh 'bash'
```

The script is easy because you don't need to know anything much about *Docker* or think about what container to use. However it is not particularly robust! The manual approach discussed in the [section below](#) is more flexible and should be used if you have any problems with the script.

Calling Docker Manually

The syntax of a typical command is shown below. This runs a Docker container that has support for X forwarding (makes the simulation GUI available from inside the container). It maps the directory `<host_src>` from your computer to `<container_src>` inside the container and forwards the UDP port needed to connect *QGroundControl*. With the `--privileged` option it will automatically have access to the devices on your host (e.g. a joystick and GPU). If you connect/disconnect a device you have to restart the container.

```
# enable access to xhost from the container
xhost +

# Run docker
docker run -it --privileged \
  --env=LOCAL_USER_ID="$(id -u)" \
  -v <host_src>:<container_src>:rw \
  -v /tmp/.X11-unix:/tmp/.X11-unix:ro \
  -e DISPLAY=:0 \
  -p 14556:14556/udp \
  --name=<local_container_name> <container>:<tag> <build_command>
```

Where,

- `<host_src>`: The host computer directory to be mapped to `<container_src>` in the container. This should normally be the **Firmware** directory.
- `<container_src>`: The location of the shared (source) directory when inside the container.
- `<local_container_name>`: A name for the docker container being created. This can later be used if we need to reference the container again.
- `<container>:<tag>`: The container with version tag to start - e.g.: `px4io/px4-dev-ros:2017-10-23`.
- `<build_command>`: The command to invoke on the new container. E.g. `bash` is used to open a bash shell in the container.

The concrete example below shows how to open a bash shell and share the directory `~/src/Firmware` on the host computer.

```
# enable access to xhost from the container
xhost +

# Run docker and open bash shell
sudo docker run -it --privileged \
  --env=LOCAL_USER_ID="$(id -u)" \
```

```
-v ~/src/Firmware:/src/firmware/:rw \  
-v /tmp/.X11-unix:/tmp/.X11-unix:ro \  
-e DISPLAY=:0 \  
-p 14556:14556/udp \  
--name=mycontainer px4io/px4-dev-ros:2017-10-23 bash
```

If everything went well you should be in a new bash shell now. Verify if everything works by running, for example, SITL:

```
cd src/firmware #This is <container_src>  
make posix_sitl_default gazebo
```

Re-enter the Container

The `docker run` command can only be used to create a new container. To get back into this container (which will retain your changes) simply do:

```
# start the container  
sudo docker start container_name  
# open a new bash shell in this container  
sudo docker exec -it container_name bash
```

If you need multiple shells connected to the container, just open a new shell and execute that last command again.

Clearing the Container

Sometimes you may need to clear a container altogether. You can do so using its name:

```
$ sudo docker rm mycontainer
```

If you can't remember the name, then you can list inactive container ids and then delete them, as shown below:

```
$ sudo docker ps -a -q  
45eeb98f1dd9  
$ sudo docker rm 45eeb98f1dd9
```

QGroundControl

When running a simulation instance e.g. SITL inside the docker container and controlling it via *QGroundControl* from the host, the communication link has to be set up manually. The autoconnect feature of *QGroundControl* does not work here.

In *QGroundControl*, navigate to [Settings](#) and select Comm Links. Create a new link that uses the UDP protocol. The port depends on the used [configuration](#) e.g. port 14557 for the SITL iris config. The IP address is the one of your docker container, usually 172.17.0.1/16 when using the default network.

Troubleshooting

Permission Errors

The container creates files as needed with a default user - typically "root". This can lead to permission errors where the user on the host computer is not able to access files created by the container.

The example above uses the line `--env=LOCAL_USER_ID="$(id -u)"` to create a user in the container with the same UID as the user on the host. This ensures that all files created within the container will be accessible on the host.

Graphics Driver Issues

It's possible that running Gazebo will result in a similar error message like the following:

```
libGL error: failed to load driver: swrast
```

In that case the native graphics driver for your host system must be installed. Download the right driver and install it inside the container. For Nvidia drivers the following command should be used (otherwise the installer will see the loaded modules from the host and refuse to proceed):

```
./NVIDIA-DRIVER.run -a -N --ui=none --no-kernel-module
```

More information on this can be found [here](#).

Virtual Machine Support

Any recent Linux distribution should work.

The following configuration is tested:

- OS X with VMWare Fusion and Ubuntu 14.04 (Docker container with GUI support on Parallels make the X-Server crash).

Memory

Use at least 4GB memory for the virtual machine.

Compilation problems

If compilation fails with errors like this:

```
The bug is not reproducible, so it is likely a hardware or OS problem.  
c++: internal compiler error: Killed (program cc1plus)
```

Try disabling parallel builds.

Allow Docker Control from the VM Host

Edit `/etc/defaults/docker` and add this line:

```
DOCKER_OPTS="${DOCKER_OPTS} -H unix:///var/run/docker.sock -H 0.0.0.0:2375"
```

You can then control docker from your host OS:

```
export DOCKER_HOST=tcp://<ip of your VM>:2375
# run some docker command to see if it works, e.g. ps
docker ps
```

Legacy

The ROS multiplatform containers are not maintained anymore: <https://github.com/PX4/containers/tree/master/docker/ros-indigo>

12.5 Maintenance notes

This picks and describes some tools to help analyze the state of the codebase and support its maintenance.

Analyze churn

The amount of churn, so the number of changes done to a file can be an indicator which files/parts might need refactoring.

To find churn metrics a tool such as [Churn](#) can be used:

```
gem install churn
An example output as of v1.6.0-rc2 would be:
cd src/Firmware
churn --start_date "6 months ago"
*****
* Revision Changes
*****
Files
+-----+
| file           |
+-----+
| src/modules/navigator/mission.cpp |
| src/modules/navigator/navigator_main.cpp |
| src/modules/navigator/rtl.cpp |
+-----+
```

* Project Churn

Files

```
+-----+
-----+
| file_path |
times_changed |
+-----+
-----+
| src/modules/mc_pos_control/mc_pos_control_main.cpp | 107
|
| src/modules/commander/commander.cpp | 67
|
| ROMFS/px4fmu_common/init.d/rcS | 52
|
| Makefile | 49
|
| src/drivers/px4fmu/fmu.cpp | 47
|
| ROMFS/px4fmu_common/init.d/rc.sensors | 40
|
| src/drivers/boards/aerofc-v1/board_config.h | 31
|
| src/modules/logger/logger.cpp | 29
|
| src/modules/navigator/navigator_main.cpp | 28
|
```