Crazy Game - Developer's Guide

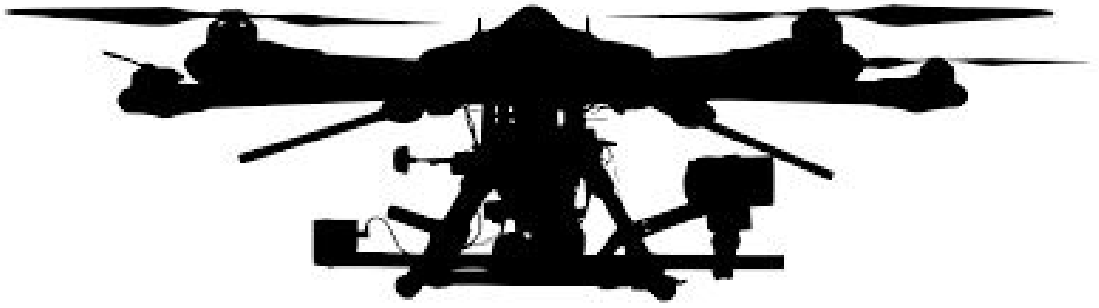


Table of Contents

Introduction

This project was made as a part of the Motion Planning for Robots workshop at Tel-Aviv University, the department of CS, 2018, under the guidance of Prof. Dan Halperin.

In this project, we created a framework for developing games of controlling real world robots on a competition between human and computer players. We also developed two games using this framework, that can be played once the system is up. The system was developed with special emphasis on motion planning algorithms.

We used the Crazyflie drones, Optitrack motion tracking system and various software (further details later in this document).

This guide gives a deeper look into the software we wrote and can be useful for creating your own games and frameworks. The User's Guide provides the links and instructions for the steps needed in order to set up this project and play the games.

Installation and Prerequisites

Hardware

Optitrack

1. Flex 3 capture cameras - There is no standard number of needed cameras for this project to work. The amount depend on the size of the “playing field”. A good “rule of thumb” at least 3 cameras have to be able to see all infrared markers related to a “rigid object” at any given moment.
   We had 6 cameras covering an area of about 1.5X1.5 meters, and up to 0.5 meter in the Z (upward) axis.
2. Infrared markers - Optitrack Cameras are able to identify objects by the reflection from the IR markers connected to the quadcopter.
   For example, 4 markers used for each Crazyflie 2.0 drone, each drone is consider as 1 “rigid object”.
3. Calibration tools - We used CW-500 Calibration Wand Kit for calibrating the cameras and CS-200 Calibration Square to set (X, Y, Z) coordinates directions. Calibration process will be further detailed later in this document.
4. Hardware key - A hardware key dongle is required for using the Motive software.
5. OptiHub 2.0 - All cameras connect to OptiHub device that connects them to the PC.

All the hardware and software mentioned here can be bought together in the following link.

Other required hardware

1. 2 or 4 Crazyflie 2.0 drones - The set of drones we used in this project.
2. Arduino Joystick.
3. WS2811 Controlled Leds.
4. USB - RF antenna for communicating with the Crazyflie drones.

The VM

Bitcraze virtual machine image, a xubuntu-14.04.4 linux VM already containing Ros code and launch files necessary for the system to work. adopted from another project, this Image is unique and has to be download from here.

As mentioned before, the VM acts as a server. The main game (on Windows) communicates with the VM over TCP connection and other code parts in the VM handle the communication with the drones.

Windows

The second software environment is Windows 10, which need to contain the Motive software and other files related to the game. These files can be found in CrazyGame for Windows PC - The host repository (without the motive software).

Software

Motive

Motive is the software used for controlling and processing the information received from the Optitrack cameras set. It is proprietary and requires a license. In Motive, collections of markers, identified by the cameras, are grouped together into trackable ‘rigid bodies’, that represent the location of objects in the real world. In our project, such objects are the Crazyflie Drones and the LEDs. it is very important to follow the next naming convention in order to use our system “as is” without changes.

- For the Drones:
    - Drone number X will be called ‘crazyflieX’ in Motive (as a ‘rigid body’).
    - Its designated MAC address will be ‘E7E7E7E70X’.
    - For example:
        - drone 3 will be called crazyflie3 and have the address E7E7E7E703.
        - drone 8 will be called crazyflie8 and have the address E7E7E7E708.
- For the LEDs:
    - LED number X will be called ‘ledX’ in Motive (as a ‘rigid body’).
    - For example:
        - LED 3 will be called led3.
        - LED 15 will be called led15.

Git

Our project is divided into five different Git repositories. Each one can be used by its own, but of course the parts were built to fit together, and therefore some adjustments will be

necessary. In order to install the full framework, follow the steps provided in the installation guide.

Repositories

1. CrazyGame computer host - The host.
2. CrazyGame VM code - Communication with the drones.
3. CrazyGame Arduino code - For joystick and leds.
4. CrazyGame Automatic player - path solver.
5. CrazyGame Website

Dependencies

Different parts of the project are made using different environments and software dependencies. Best practise of using this project is following the installation guide, on a Windows computer strong enough to carry the simultaneous load of the VM and the host softwares, Which means minimum 8 GB of RAM and virtualization capabilities. Alternatively it is possible to use it in other configurations but we don't offer additional support for it.

Installation Guide

Assuming:

- You are using a Windows computer that meets the demands in the dependencies section.
- That the Optitrack + motive system is installed and ready to go.
- That the Crazyflie antenna dongle is connected.

1. Download and install Oracle VirtualBox or alternative VM host software of your choice.
2. Download the Linux xubuntu VM we supply in our git repository. It is based on the original Crazyflie project VM. You can move to the next steps while it is downloading.
3. Install python 3.6 on the windows PC.
4. Install the needed Python dependencies by opening Windows CMD and typing:

   ```
   pip install shapely pyserial munch
   ```
5. Download and Install Git on the Windows PC.
6. Clone the Windows-side repository by opening the now installed Git bash program, and typing in the command:

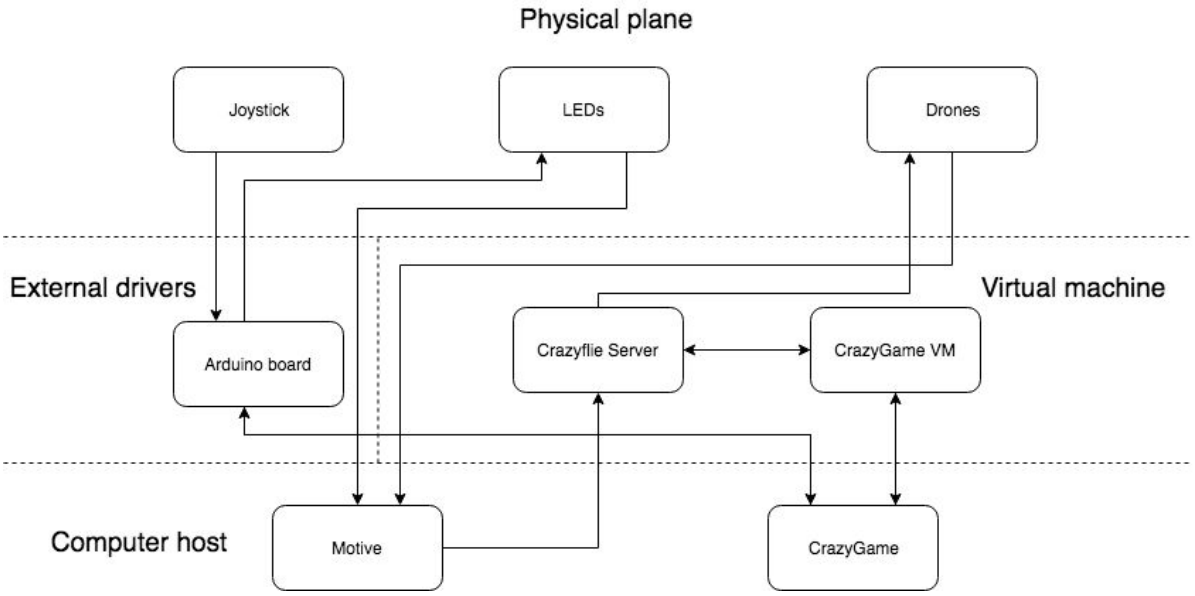
git clone https://github.com/xqgex/CrazyFlie

It is very important to add the main crazyFlie directory to python path.

7. Import the downloaded VM from step 2, in the VM host software from step 1. You can use this guide.
8. Set the following configuration ("How To" guide) for the VM:
    a. At least 4GB of RAM
    b. static IP - 172.16.1.2
    c. Network adapter attached to - "Host-Only adapter" (in the Virtualbox VM settings)
9. Start the VM (root password is "crazyflie").
10. optional: if you wish to control the drone using an Arduino Joystick + LED targets, install the Arduino IDE, and clone/use the code from the - CrazyGame Arduino code - For joystick and led strips repository.

Software Architecture - Overview

Crazy game contains multiple layers, from the physical plane, through external drivers and a virtual machine, to the host in which the logic is implemented. The project contain both hardware and software parts, an overview software architecture is shown in the following diagram:



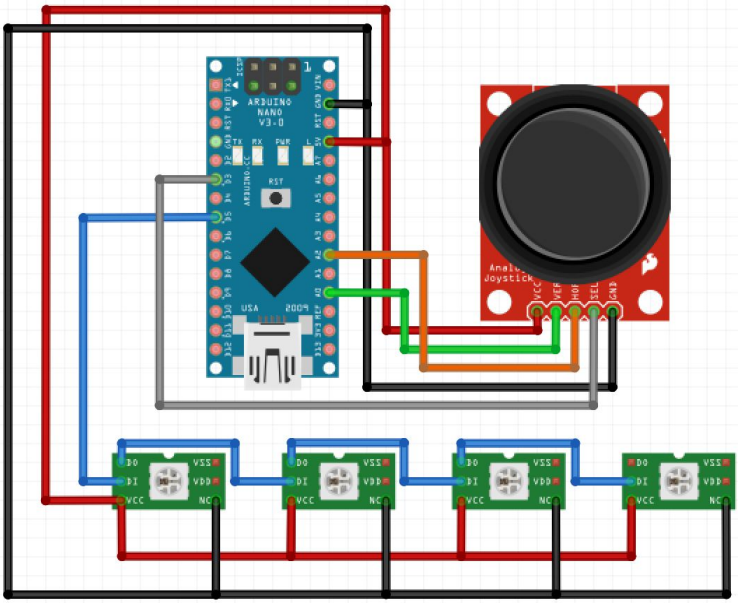
Arduino board

Our system allows the user to play games in the real world while the software is running on a computer. In order to create the full experience of playing a game in the real world, we let the user navigate the drones with a joystick and represent the targets with LEDs.

The controller is an Arduino Nano based on ATMEGA328, 16 MHz. Its code is written in C++. The LEDs are WS2811. We used the well known FastLed library to communicate with them. The Joystick is two potentiometers that connect to two analog input pins(A0 and A2) and a button to pin 3.

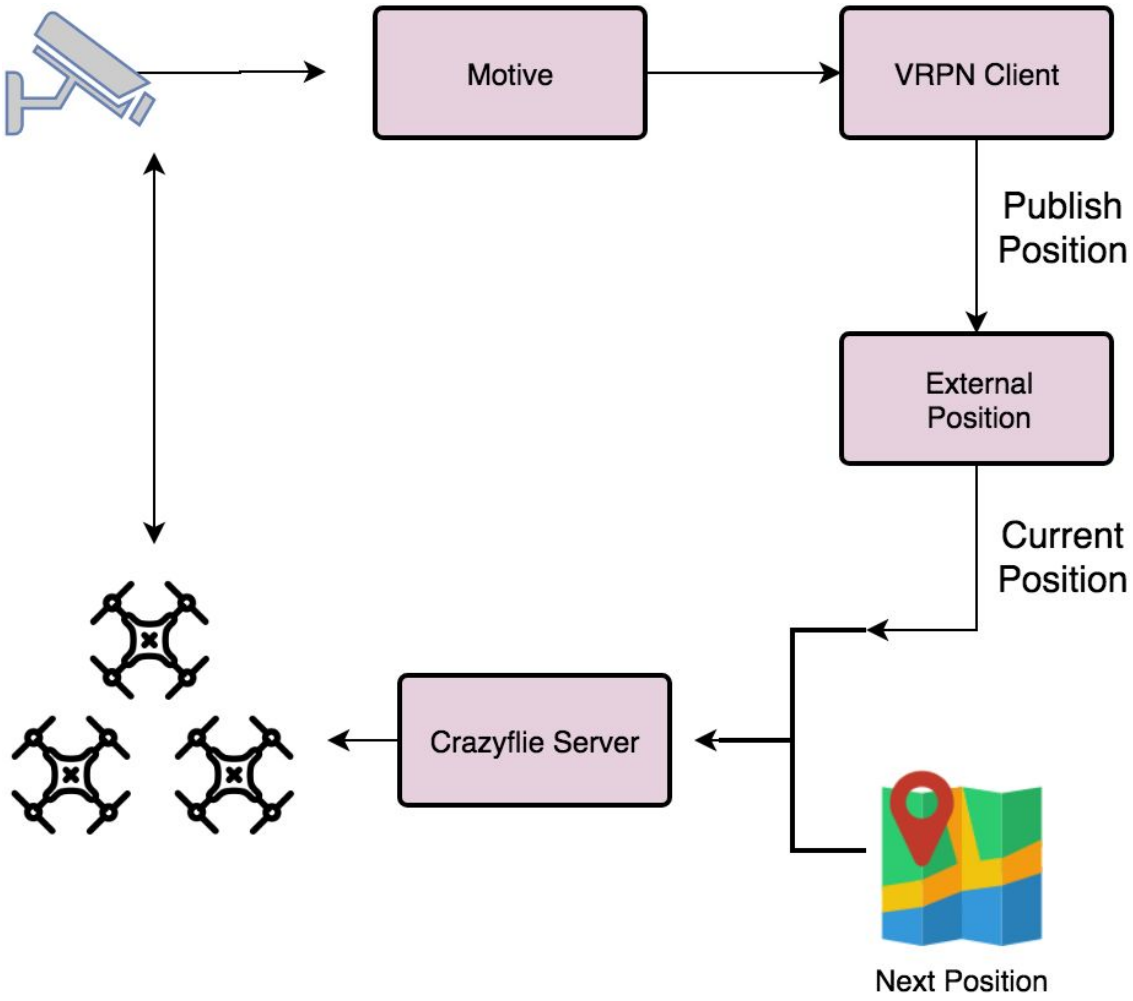
The communication between the computer and the control is via serial port, the baud rate is 9600 bps. It follows a simple protocol we created that documented in the file ArduinoDriver.cpp.

Schematics Arduino board design:



Virtual Machine

The virtual machine (xubuntu 14.04 distribution) is downloaded from the Bitcraze project, the developer and manufacturer of the Crazyflie quadcopter. It contains prerequisites needed for the Crazyflie project. For more information on the original VM see the link.
The project runs on top of ROS (Robot Operating System), which coordinates the flow of the data in a close loop between the Motive and the drones, as described in the next diagram:



As can be seen from the diagram, the motive publishes the locations of the drones it received from the OptiTrack cameras, and after a short process of the current position and the desired position, the Crazyflie server sends a move command to the drones using RF.

roslaunch

roslaunch is a tool for easily launching multiple ROS nodes. We created a custom script (generateLaunch.py) that create a .launch file at "CrazyFlie_ros/crazyflie_demo/launch/crazy_game.launch" based on input of:

- Motive ip and port
- The list of Crazyflie's
- The list of led's

The main.py generates a new .launch file using generateLaunch.py (based on hard-coded values) and executes it.

VM Server

The virtual machine server is called crazyGameVM.py, and interfaces between the main game, runs over the host on Windows, and the drones, coordinated by the ROS.
Amongst the rest, it receives move commands from the game player. There are two types of movement commands, a relative command that receives a unit vector pointing to the desired direction, and an absolute command that receives X and Y coordinates.
The relative movement is calculated out of two parameters received from the host at the beginning of the communication - move speed and step size, whereas the absolute one uses only the move speed for the calculation.
Moreover, because the real world coordinate system not necessary start from (0, 0), In order to abstract this from the higher level code, The server make a conversion between the two coordinate systems.

Testers

The tester.py file is a single file used to execute any local test related to the Crazyflies. It provides the basic modules for controlling the Crazyflie, needed by the tests.
Each test is a python file located in the "./tests/" directory containing a function called "run" that received a variable contains a list of all the drones for the test.
The tests are executed with the following call:

- Python tester.py <test name> <names of drones>...

Where "test name" stands for the name of the test python file, excluding the .py suffix (for instance, hover.py will be run as "hover").

CrazyGame

The entire logic of the project is implemented as a python project on the host computer, while the CrazyGame VM is a pipe between the host and the CrazyGame server that controls the drones. We address this part of our program as CrazyGame. It's implemented in python 3 and developed in the agile method.

Drivers

Due to its tasks, CrazyGame has full access to all the components of the project. Those drivers are implemented under "phripherals" directory.

Arduino Controller

This script tries to establish a serial communication with the Arduino Nano board. It scans all the available serial ports and tries to connect to the last one. If it succeeds it starts a new thread that polls the joystick parameters every 20ms and then find the initial position of the potentiometers.

This driver allows reading the last valid joystick position and setting the LED's color. In order to ensure thread safety, a mutex manages the access to the serial port object.

Drones Controller

This is a stateless API to communicate with the CrazyGame VM. This API allows us to send commands to the drones and fetch information regarding the world and variables.

The main functions are takeoff, landing, move drone, go to, battery status.

Drones Controller Simulator

In order to develop without being dependant on setting up the entire system in the physical layer each time, and in order to avoid bottlenecks at the system development process, Drone Controller Simulator emulates the physical layer. It contains and updates virtual drone's positions and allows querying the current positions.

Considering that the physical layer contains inconsistencies in the form of noise, the Drone Controller Simulator emulates the inconsistencies by adding Gaussian noise to the positions and destinations of the virtual drones.

Managers

Drones orchestrator

In order to ensure the avoidance of drones colliding or moving out of bounds, the Drones Orchestrator manages the movements of the drones. It forbids movements which would bring the system to unwanted situations. Drones commands are sent through the Drones Orchestrator that assures the drones don’t collide nor go out of bounds, hence protecting them even if the pilot gives a dangerous command.

The orchestrator doesn't prevent collisions when multiple drones move simultaneous. Hence extra care should be applied when running more than one drone.

Landmark Manager

The Landmark Manager manages LEDs and obstacles in the system. It parses the objects, stores the positions and color and allows changing the LED’s color.

Joystick

The Joystick allows the user to navigate the drones by supplying a movement direction . When the Arduino board is connected the direction is received from it, otherwise it receives the direction from the keyboard.

Display

The GUI is implemented with the well known pygame library. This library allows drawing shapes and pictures in a GUI window. In addition, it manages events like mouse clicking and moving, keys pressing and more.

The render function draws all the components that are currently visible. Due to real-time constraints, it should be as fast as possible.

The board is a display of the system situation. So The width/height ratio of the board is according to the size of the playing area. It discretizes the playing area in order to display it, but doesn’t change the system behavior that is continuous.

Initialization process

When the program starts it tries to connect the peripherals hardware. The initialization process includes:

- Create a display manager
- Try to connect the arduino board

- Create a joystick
- Wait to the user to decide which kind of drone controller to connect to
    - Virtual machine
    - Simulator
- Create drones orchestrator
- Create landmark manager
- Set display board and batteries display

Games

Real time issues

Python doesn’t adequate for real-time software and the project requires firm real-time control. The one thread approach(excluding the Arduino polling thread) demands using non-blocking and fast functions that are called few times every second. The non-block approach allows creating responsive GUI while tracking the drones condition.

The rendering frame rate is around 20 Hz, while the drone location frame rate is around 100 Hz. It sufficient in spite of python limits.

Sand Box

Use this part for free control of the drones and LEDs. Note that if you are moving more than one drone, the drones orchestrator won't prevent you from creating a collision between the drones.

Capture the flag

Two drones, each one with a target, are competing to reaching their target first. Each player can be autonomous or controlled by the user. For the autonomous player, the route is calculated with the module PathFinder that runs Dijkstra's algorithm for the shortest path.

Capture them all

One drone that should pass through a few targets, represented by LEDs in the fastest time. The autonomous player might use AlgoLink [work in progress] for optimal path or pass them by order of them in the program in case the AlgoLink is not available.

Tests

Tests are an essential part of program development, especially on complex and multi-modules projects. The tests directory contains twelve tests. Part of them are unit tests for a single module and other are system tests that run the whole system.

We highly recommend to run the tests to ensure the system runs smoothly.

Summery

CrazyGame is a platform for integrating motion planning algorithms and robotics games into the real world. Most of the development time was spent to create and stabilize this platform. The challenges were mostly from the integration of pure software concepts into the inconsistencies in real-world applications. We will be glad to help others to continue the work we started in order to extend the capabilities of this system.

Future work

Autonomous flight

In order to extend the capabilities of path planning, we created a CGAL based service. CGAL is an open source c++ library for computational geometry algorithms. The Crazyflie Autoplayer is a server which supplies solution for motion planning problems. AlgoLink is the python module which communicates with this server and allows the games to send queries for motion planning problems.

The server supports a naive algorithm for traveling salesman in order to find the shortest path between a few targets. The working space may contain obstacles in addition to the drones should be avoided.

In addition, the server includes an algorithm to find the shortest path to pass via few gates. The working space may include other drones and obstacles.

The program is working on Linux machines, but due to the incompatibility of c++, CGAL and other dependent libraries in cross-platform programming we encountered some problems with running it on the windows machine which runs the entire project. So we leave it as a future work to integrate between the programs.

Games

The system that is described in this paper is a platform for creating games and other applications with drones in the physical layer. We implemented three basic games that introduce the capability of this system but much more can be achieved by future developing of applications on top of this platform. We'll be glad to help others to continue working with this system.