User Guide for Multicopter Design 0.2

Tao Du
taodu@csail.mit.edu

Adriana Schulz
aschulz@csail.mit.edu

Bo Zhu
boolzhu@csail.mit.edu

MIT Computer Science and Artificial Intelligence Laboratory

Contents

1 Introduction

This project is about developing a design, simulation and optimization tool for casual users to build their own multicopters. Initially it was a research project led by the Computational Fabrication Group at MIT CSAIL in 2015. After we published a paper [1] we got inquiries from many different sources so we decided to release our source code for the benefits of both the research and hobbyist communities.

For those who have read that paper, please note that the code here only covers a subset of our implementation in the paper, as we are still in the middle of cleaning it up. We expect to release all features in the next version (0.3).

1.1 Problem Definition

Suppose you are a drone hobbyist who wants to have more fun in building your own copters. Below is a list of questions you might want to ask yourself before you start:

- What components to use: This includes choosing the right material to build the frame (carbon fiber, wood, plastic, etc), picking the proper motors and propellers, selecting a good battery, and many more.
- How to assemble them: You may want to follow standard designs like a quadcopter, or try some fancy configurations like a tricopter, or even come up with your own, unique configuration, like the bunny-shaped example in our paper.
- How to control its flight: Normally, you can find a mature controller design for classic quadcopters, which is typically a multi-layer PID controller, for example. If you want to have more degrees of freedom in our design, you probably need to develop your own controllers, or at least spend some time modifying and tweaking an existing controller from similar models.
- How to verify the design: Crashing a real drone is a huge pain and loss, both psychologically and financially. This is why people turn to software simulation for help. Ideally, it should verify both the geometry and controller accurately, and it should be embedded in the design loop to allow for interactive changes.

Obviously, this is a very involved process requiring a diverse set of skills, including mechanical engineering, electrical engineering, programming, control theory, and hands-on experience in building stuff. The goal of our software is to expedite and simplify this process by hiding low level technical details so that people can focus more on expressing and exploring the design space. Specifically, we provide:

- A dataset of standard components that users can freely choose from and assemble by defining their own scripts.
- An easy-to-use user interface to explore designs in an interactive way.
- An automatically determined LQR controller that is generalizable for most existing designs we have seen so far.
- A real-time simulation tool seamlessly embedded in the design loop to visualize results instantly.

Please keep in mind that this software is not meant to replace all the work involved in building a customized drone. You are expected to do your own measurement, or at least give a rough estimation of the physical properties of the motors, propellers and batteries. The output of our software is a hardware specification you can follow, but of course you still need to fabricate or buy these components and assemble them manually. Finally, we include a controller in the solution, but uploading it into your own flight controller board is beyond the scope of this software. You are expected to have a working knowledge of the flight controller (both the hardware and software) you plan to use, and be comfortable with modifying its code to include a new controller.

1.2 Inputs and Outputs

The input is an XML file that defines the initial copter design. It provides a straightforward way to format all the essential information needed in the software. We provide a few sample XML files for you to start with, including a standard quadcopter, a V-Tail quadcopter, a Y6 hexacopter and a "pentacopter" with 5 motors. Section 4 provides a detailed explanation on the file format for your reference.

The output includes a hardware fabrication plan and a matrix of control parameters. This plan provides hardware specifications like rotor positions and tube sizes so that by following it you should be able to replicate your virtual design. The control parameters fully determine the LQR controller, and if you are familiar with your flight software it should be fairly straightforward to translate them into the code on board.

1.3 An Example

As a concrete example, here we demonstrate how our software can be used to help design and simulate a V-Tail copter 1. When the window first pop up, you will see a V-Tail model centered on screen, and a few windows floating on both sides. Below we will briefly explain the user interface and show a simple design and simulation loop. A more thorough description can be found in Section 3.

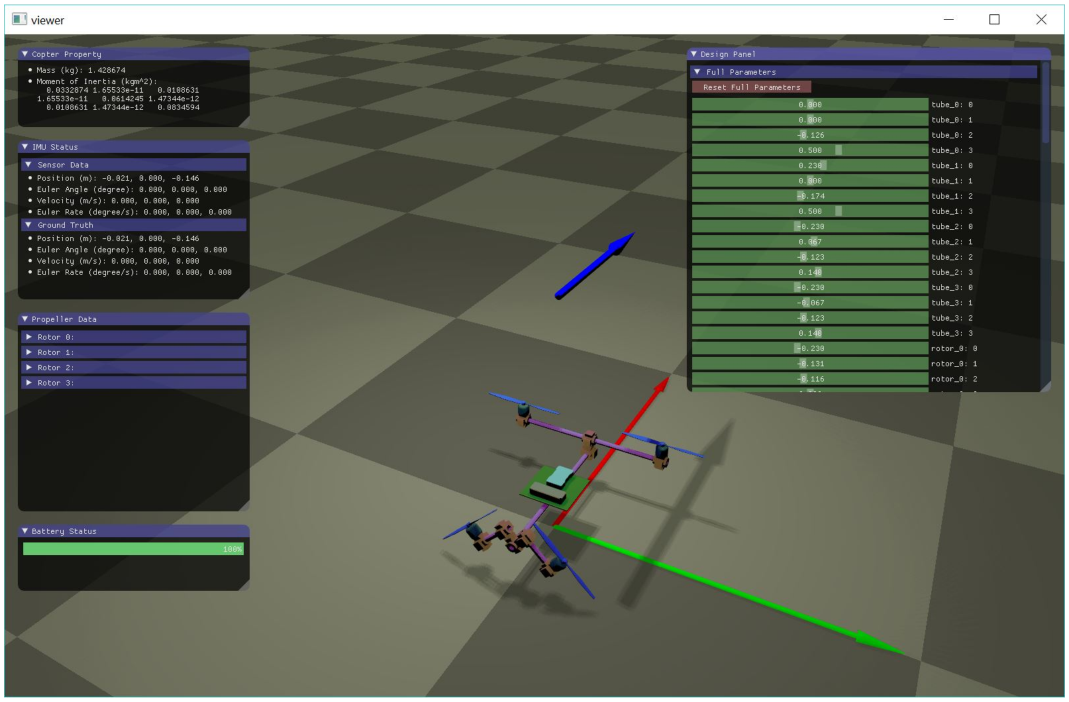
1.3.1 Basic Information

On the top left you will find a window named "Copter Property". This consists of the mass and inertia of the current design, both of which are computed by collecting information from each component in the XML file. Later, when you change your design, this panel will get updated automatically.

1.3.2 Design Panel

On the right there is a long window for you to explore more design possibilities. In our software a copter is parametrized, meaning its shape is fully determined by a list of underlying parameters. These include the length of a tube, 3D position of a connector, height of a plate, and so on. Certain constraints are imposed to ensure the parameters are feasible. For example, a motor should always be attached to a connector instead of freely floating in the air. Please refer to Section 4 for more details about parameters in each component and its constraints.

Full Parameters The first panel named "Full Parameters" lists a union of parameters from each component. When you use a slider, it specifies a desired value of that particular parameter. A linear least square problem is then solved to find parameters that respect all constraints and are closest to the desired value. Once a solution is found, the copter design in the middle gets updated automatically, and usually in real time (Figure 2).



```
▼ Copter Property
• Mass (kg): 1.428674
• Moment of Inertia (kgm^2):
    0.0332874 1.65533e-11   0.0108631
  1.65533e-11   0.0614245 1.47344e-12
    0.0108631 1.47344e-12   0.0834594

▼ IMU Status
  ▼ Sensor Data
• Position (m): -0.021, 0.000, -0.146
• Euler Angle (degree): 0.000, 0.000, 0.000
• Velocity (m/s): 0.000, 0.000, 0.000
• Euler Rate (degree/s): 0.000, 0.000, 0.000
  ▼ Ground Truth
• Position (m): -0.021, 0.000, -0.146
• Euler Angle (degree): 0.000, 0.000, 0.000
• Velocity (m/s): 0.000, 0.000, 0.000
• Euler Rate (degree/s): 0.000, 0.000, 0.000

▼ Propeller Data
  ▶ Rotor 0:
  ▶ Rotor 1:
  ▶ Rotor 2:
  ▶ Rotor 3:

▼ Battery Status
  100%

▼ Design Panel
  ▼ Full Parameters
  Reset Full Parameters
   0.000  tube_0: 0
   0.000  tube_0: 1
  -0.126  tube_0: 2
   0.500  tube_0: 3
   0.230  tube_1: 0
   0.000  tube_1: 1
  -0.174  tube_1: 2
   0.500  tube_1: 3
  -0.230  tube_2: 0
   0.067  tube_2: 1
  -0.123  tube_2: 2
   0.140  tube_2: 3
  -0.230  tube_3: 0
  -0.067  tube_3: 1
  -0.123  tube_3: 2
   0.140  tube_3: 3
  -0.230  rotor_0: 0
  -0.131  rotor_0: 1
  -0.116  rotor_0: 2
```

Figure 1: Loading the given V-Tail design into our software. The left windows show the current status of the copter, and the right panel allows you to freely explore and simulate different designs.

Reduced Parameters A copter design consists of up to hundreds of parameters. However, most of them are over constrained by the relative locations between different components. As a result, we also provide a panel to manipulate the reduced parameters, which is usually less intuitive but has way fewer degrees of freedom. Technically, this is done by replacing the linear equality constraints with vectors in its null space.

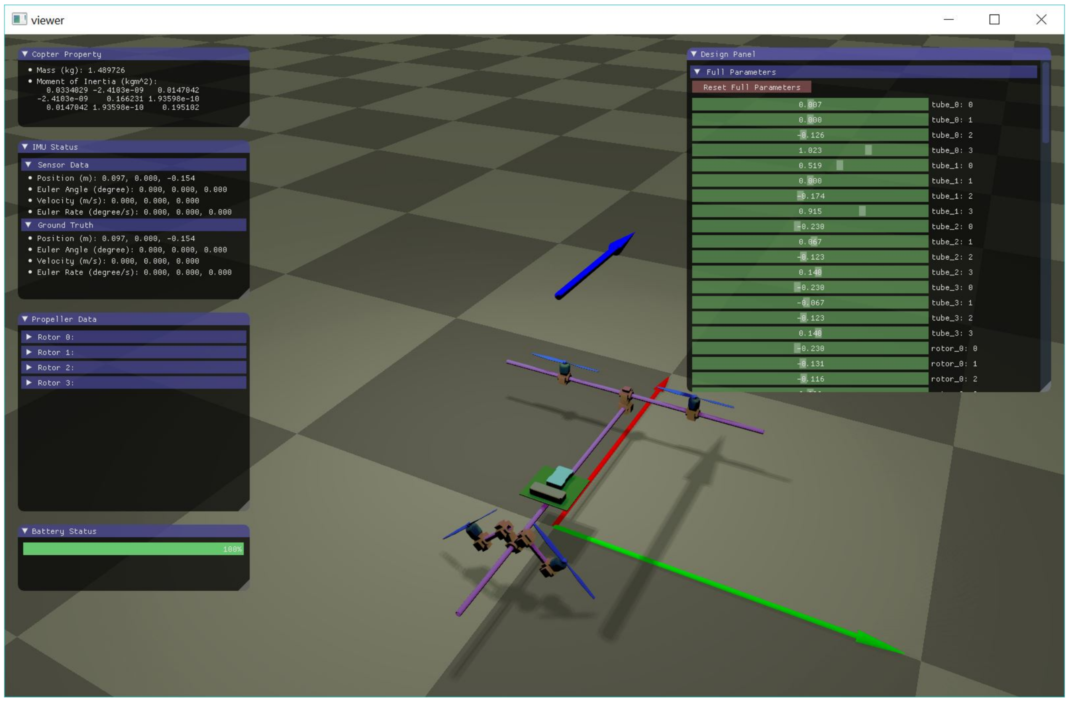
When you slide the reduced parameters, you may notice the background color sometimes changes between green and red. This color indicates the current parameter is feasible (green) or breaks certain constraints (red). If you slide it into red, the new value will be rejected and the current design remains unchanged.

Finally, there is a "Reset" button in both "Full Parameters" and "Reduced Parameters" windows. Simply clicking it will reset the copter to its original design.

1.3.3 Simulation

Once you are satisfied with the current copter, scroll down in the design panel, and you will find a "Simulate" button. Clicking it will trigger the software to compute a proper LQR controller and start simulating a virtual flight (Figure 3). Press this button will invalidate the "Full Parameters" and "Reduced Parameters" windows. The button itself will change to "Reset" so that you can switch back to the design process if more iterations are needed.

During the flight, this copter will try to track the blue arrow as close as possible. Specifically, it interprets the tail as the target position and the arrow as the desired heading. You can use the four



```
• Mass (kg): 1.489726
• Moment of Inertia (kgm^2):
    0.0334029 -2.4103e-09   0.0147042
  -2.4103e-09    0.166231 1.93598e-10
    0.0147042 1.93598e-10    0.195102

Sensor Data
• Position (m): 0.097, 0.000, -0.154
• Euler Angle (degree): 0.000, 0.000, 0.000
• Velocity (m/s): 0.000, 0.000, 0.000
• Euler Rate (degree/s): 0.000, 0.000, 0.000
Ground Truth
• Position (m): 0.097, 0.000, -0.154
• Euler Angle (degree): 0.000, 0.000, 0.000
• Velocity (m/s): 0.000, 0.000, 0.000
• Euler Rate (degree/s): 0.000, 0.000, 0.000
```

```
 0.007    tube_0: 0
 0.000    tube_0: 1
-0.126    tube_0: 2
 1.023    tube_0: 3
 0.519    tube_1: 0
 0.000    tube_1: 1
-0.174    tube_1: 2
 0.915    tube_1: 3
-0.230    tube_2: 0
 0.067    tube_2: 1
-0.123    tube_2: 2
 0.140    tube_2: 3
-0.230    tube_3: 0
-0.067    tube_3: 1
-0.123    tube_3: 2
 0.140    tube_3: 3
-0.230    rotor_0: 0
-0.131    rotor_0: 1
-0.116    rotor_0: 2
```

Figure 2: By adjusting the full parameters, we elongate two tubes and push the front rotors farther from the center.

arrow keys plus ‘a’, ‘d’, ‘w’ and ‘s’ to adjust the blue arrow and guide your copter to a desired position. You can also change the camera view by mouse gestures like scrolling or panning.

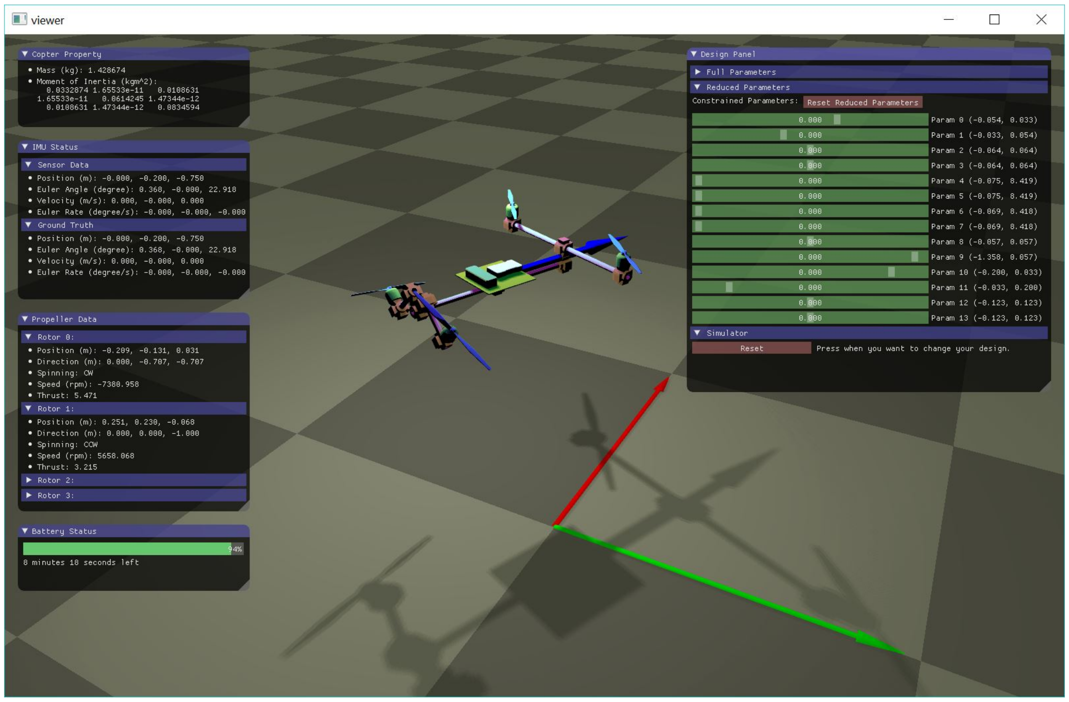
1.3.4 Flight Status

On the middle and lower left we provide you more windows to monitor the current status of your copter during flight. The “IMU Status” window displays the ground truth position, velocity, rotation and angular rates as well as their readings from the sensor. Currently no noise is added to the sensor so it gives perfect data identical to the ground truth.

Below the sensor data is the “Propeller Data” window. Here you can monitor the spinning rate and direction of each propeller. This window also includes basic information like the position and direction of each rotor.

Finally, a “Battery Status” window is on the lower left. This shows an estimation of the battery life by summing up the current flowing to each rotor. The progress bar will turn red when the battery is about to drain.

The remaining sections in this document are organized as follows: Section 2 provides a step-to-step tutorial about downloading, compiling and running our code on different platforms (Windows, Linux, and macOS). Section 3 is a detailed description of our user interface, where you can learn more about the design and simulation features. Finally, if you want to try our own copter in our software, Section 4 gives you a thorough description of the XML file format used to define a copter design. Follow that section to learn how to write your own XML file and load it into our software design and simulation



```
▼ Copter Property
• Mass (kg): 1.428674
• Moment of Inertia (kgm^2):
   0.0332874 1.65533e-11   0.0108631
 1.65533e-11   0.0614245 1.47344e-12
   0.0108631 1.47344e-12   0.0834594

▼ IMU Status
 ▼ Sensor Data
• Position (m): -0.000, -0.200, -0.750
• Euler Angle (degree): 0.368, -0.000, 22.918
• Velocity (m/s): 0.000, -0.000, 0.000
• Euler Rate (degree/s): -0.000, -0.000, -0.000
 ▼ Ground Truth
• Position (m): -0.000, -0.200, -0.750
• Euler Angle (degree): 0.368, -0.000, 22.918
• Velocity (m/s): 0.000, -0.000, 0.000
• Euler Rate (degree/s): -0.000, -0.000, -0.000

▼ Propeller Data
 ▼ Rotor 0:
• Position (m): -0.209, -0.131, 0.031
• Direction (m): 0.000, -0.707, -0.707
• Spinning: CW
• Speed (rpm): -7380.958
• Thrust: 5.471
 ▼ Rotor 1:
• Position (m): 0.251, 0.230, -0.068
• Direction (m): 0.000, 0.000, -1.000
• Spinning: CCW
• Speed (rpm): 5658.068
• Thrust: 3.215
 ▶ Rotor 2:
 ▶ Rotor 3:

▼ Battery Status
94%
8 minutes 18 seconds left

▼ Design Panel
 ▶ Full Parameters
 ▼ Reduced Parameters
Constrained Parameters: Reset Reduced Parameters
0.000   Param 0 (-0.054, 0.033)
0.000   Param 1 (-0.033, 0.054)
0.000   Param 2 (-0.064, 0.064)
0.000   Param 3 (-0.064, 0.064)
0.000   Param 4 (-0.075, 8.419)
0.000   Param 5 (-0.075, 8.419)
0.000   Param 6 (-0.069, 8.418)
0.000   Param 7 (-0.069, 8.418)
0.000   Param 8 (-0.057, 0.057)
0.000   Param 9 (-1.358, 0.057)
0.000   Param 10 (-0.200, 0.033)
0.000   Param 11 (-0.033, 0.200)
0.000   Param 12 (-0.123, 0.123)
0.000   Param 13 (-0.123, 0.123)
 ▼ Simulator
Reset   Press when you want to change your design.
```

Figure 3: The V-Tail quad is executing a virtual flight in simulation. The copter is trying to follow the position and heading specified by the blue arrow.

loop.

2 Installation

This section provides a step-by-step tutorial about how you can build the software from source code on a Windows, Linux, or macOS machine.

2.1 Prerequisites

Check the list below before you start building from the source code:

- **Update your graphics card driver**: OpenGL 3.3 or later version is required to display the user interface correctly. If your computer is not very old, then you can probably skip this step. Keep in mind that if any weird display issues occur after you compile the code, you might want to try updating your graphics card driver first.
- **Install Git**: Git is a widely used version control system in the open source community. If you want to modify our code then you probably should install Git. If you are an end user, Git is not necessary, but we still recommend it, otherwise you will have to download the source code and its dependencies (for those who know Git, we use submodules in our code base), and manually place them in the right folder. Moreover, Git allows you to easily track the latest patch we release for bug fixes, code improvement, etc.
- **Install CMake**: CMake is a tool for building cross-platform softwares. You specify your building rules in a file typically named as `CMakeLists.txt`, and CMake will generate build files depending on your compiler and platform, which could be a Visual Studio solution file on Windows, or a Makefile on Linux, for example. If you are an end user, don't worry about learning how to write the CMakeLists.txt file, just install CMake and you are good to go.
- **Install a C++ compiler**: There are multiple options here and this is really up to you. For Windows users we recommend the free community version of Microsoft Visual Studio. For Linux users, it is usually gcc/g++. For macOS users we have tested with AppleClang provided in Xcode. Make sure your computer can run a basic "Hello, world!" C++ program before you move on.

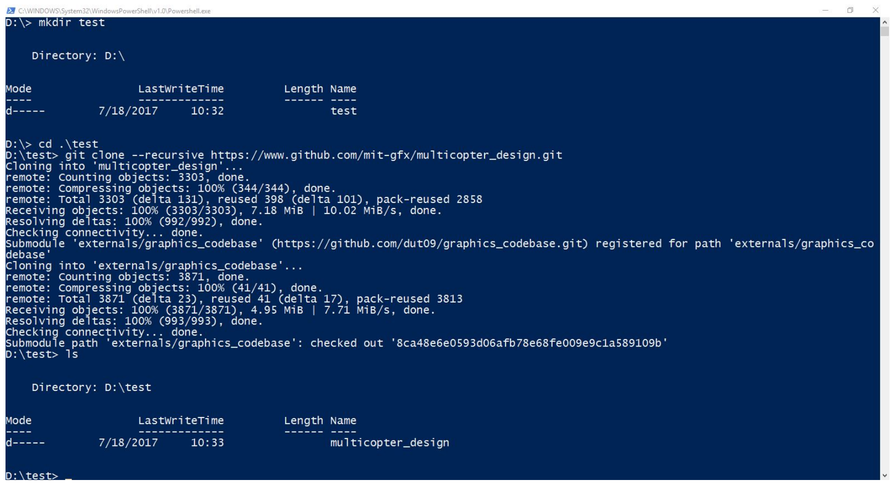
2.2 Build on Windows

This tutorial has been tested on Windows 10 64bit with Microsoft Visual Studio 2015 Community.

2.2.1 Get the Source Code from GitHub

Create an empty folder, open Git shell, navigate to that folder, and use `git clone` to download the source code. Figure 4 uses an empty folder called `test` and clones the project into it:

```
> mkdir test
> cd test
> git clone --recursive https://github.com/mit-gfx/multicopter_design.git
```



```
D:\> mkdir test


    Directory: D:\


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        7/18/2017     10:32                test


D:\> cd .\test
D:\test> git clone --recursive https://www.github.com/mit-gfx/multicopter_design.git
Cloning into 'multicopter_design'...
remote: Counting objects: 3303, done.
remote: Compressing objects: 100% (344/344), done.
remote: Total 3303 (delta 131), reused 398 (delta 101), pack-reused 2858
Receiving objects: 100% (3303/3303), 7.18 MiB | 10.02 MiB/s, done.
Resolving deltas: 100% (992/992), done.
Checking connectivity... done.
Submodule 'externals/graphics_codebase' (https://github.com/dut09/graphics_codebase.git) registered for path 'externals/graphics_co
debase'
Cloning into 'externals/graphics_codebase'...
remote: Counting objects: 3871, done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 3871 (delta 23), reused 41 (delta 17), pack-reused 3813
Receiving objects: 100% (3871/3871), 4.95 MiB | 7.71 MiB/s, done.
Resolving deltas: 100% (993/993), done.
Checking connectivity... done.
Submodule path 'externals/graphics_codebase': checked out '8ca48e6e0593d06afb78e68fe009e9c1a589109b'
D:\test> ls


    Directory: D:\test


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        7/18/2017     10:33                multicopter_design


D:\test> _
```

Figure 4: (Windows) Cloning the project into an empty folder named `test`.

2.2.2 Download Libraries

Our program relies on some 3rd party libraries. Please use the provided PowerShell script to download and configure them:

```
> cd multicopter_design
> ./windows_setup.ps1
```

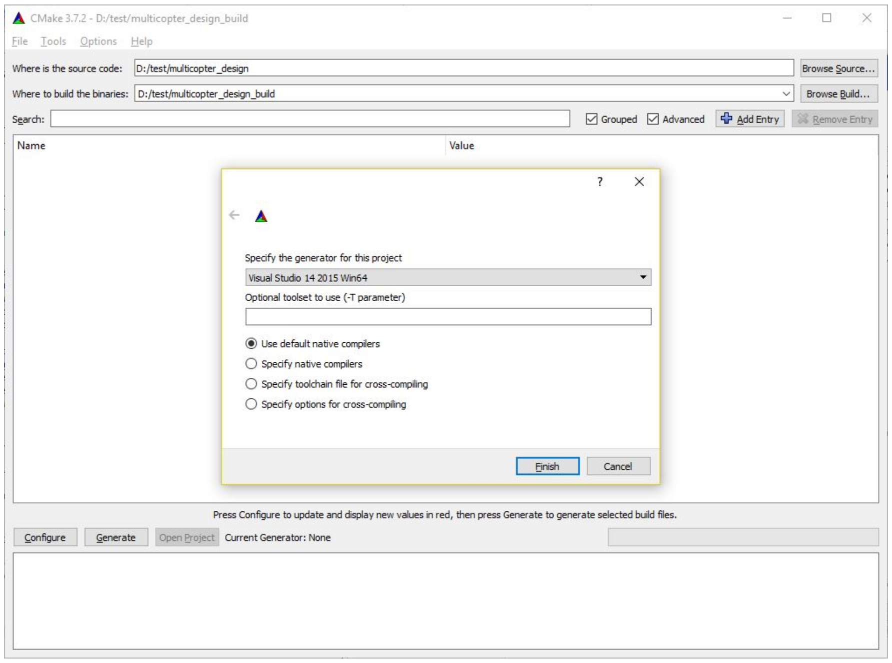
2.2.3 Generate the Build Solution

Open CMake GUI. In "Where is the source code", type the project location you cloned just now, which is `D:/test/multicopter_design` in our case. For "Where to build the binaries", use the location you want to put the generated solution file into. Ideally it should be an empty folder out of the source. In our demo we use an empty folder located at `D:/test/multicopter_design_build`. Click "Configure", CMake will pop up a window and ask you to choose the generator. Here `Visual Studio 14 2015 Win64` is select, and you should use the C++ IDE you have installed on your computer (Figure 5), click "Finish" when you are ready.

During the configuration step, CMake will display a few messages and finally "Configuring done" if no error occurs. Click "Generate" to create the Visual Studio solution file, which can be found in the location you specified in "Where to build the binaries". Click "Open Project" and wait for Visual Studio to open your solution.

2.2.4 Build and Test

After Visual Studio is open and ready, press `F7` or click "Build" on the menu and choose "Build Solution". The default configuration is built in "Debug" mode but you can also switch to "Release" for faster solution. If no error occurs, multiple executable files will be generated in the folder you used for building binaries. Now set `copter_viewer` as the startup project and press `F5`. A window should pop up and you will see a copter with 5 rotors in the middle (Figure 6).


CMake 3.7.2 - D:/test/multicopter_design_build

Where is the source code: D:/test/multicopter_design

Where to build the binaries: D:/test/multicopter_design_build

Search:

Grouped ✓ Advanced ✓

Name | Value

Specify the generator for this project: Visual Studio 14 2015 Win64

Optional toolset to use (-T parameter):

● Use default native compilers

○ Specify native compilers

○ Specify toolchain file for cross-compiling

○ Specify options for cross-compiling

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: None

Figure 5: (Windows) Using CMake to generate the build solution.

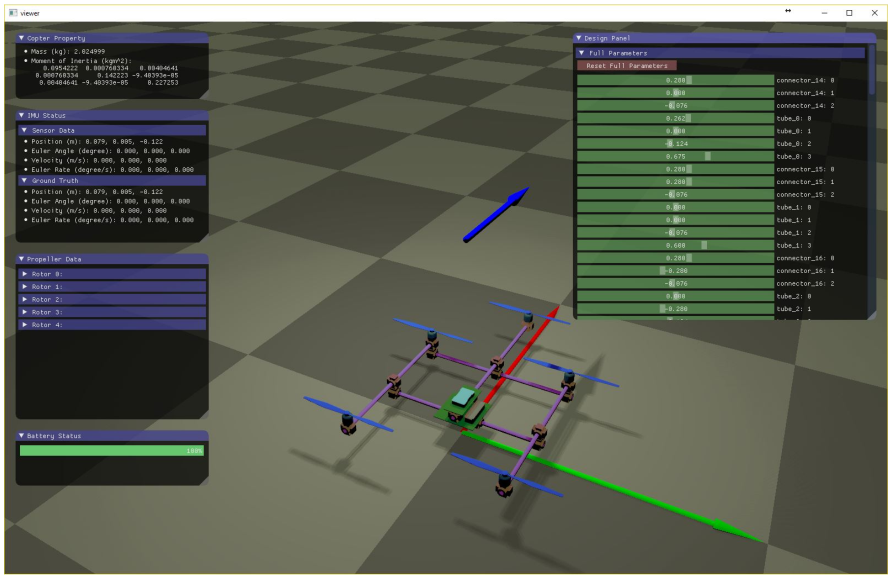

```
▼ Copter Property
 • Mass (kg): 2.024999
 • Moment of Inertia (kgm^2):
      0.0954222  0.000760334   0.00404641
    0.000760334     0.142223 -9.40393e-05
     0.00404641 -9.40393e-05     0.227253

▼ IMU Status
 ▼ Sensor Data
 • Position (m): 0.079, 0.005, -0.122
 • Euler Angle (degree): 0.000, 0.000, 0.000
 • Velocity (m/s): 0.000, 0.000, 0.000
 • Euler Rate (degree/s): 0.000, 0.000, 0.000
 ▼ Ground Truth
 • Position (m): 0.079, 0.005, -0.122
 • Euler Angle (degree): 0.000, 0.000, 0.000
 • Velocity (m/s): 0.000, 0.000, 0.000
 • Euler Rate (degree/s): 0.000, 0.000, 0.000

▼ Propeller Data
 ▶ Rotor 0:
 ▶ Rotor 1:
 ▶ Rotor 2:
 ▶ Rotor 3:
 ▶ Rotor 4:

▼ Battery Status
 100%

▼ Design Panel
 ▼ Full Parameters
 Reset Full Parameters
   0.280  connector_14: 0
   0.000  connector_14: 1
  -0.076  connector_14: 2
   0.262  tube_0: 0
   0.000  tube_0: 1
  -0.124  tube_0: 2
   0.675  tube_0: 3
   0.280  connector_15: 0
   0.280  connector_15: 1
  -0.076  connector_15: 2
   0.000  tube_1: 0
   0.000  tube_1: 1
  -0.076  tube_1: 2
   0.600  tube_1: 3
   0.280  connector_16: 0
  -0.280  connector_16: 1
  -0.076  connector_16: 2
   0.000  tube_2: 0
  -0.280  tube_2: 1
```

Figure 6: (Windows) A screenshot of the user interface after a successful build.

2.3 Build on Linux

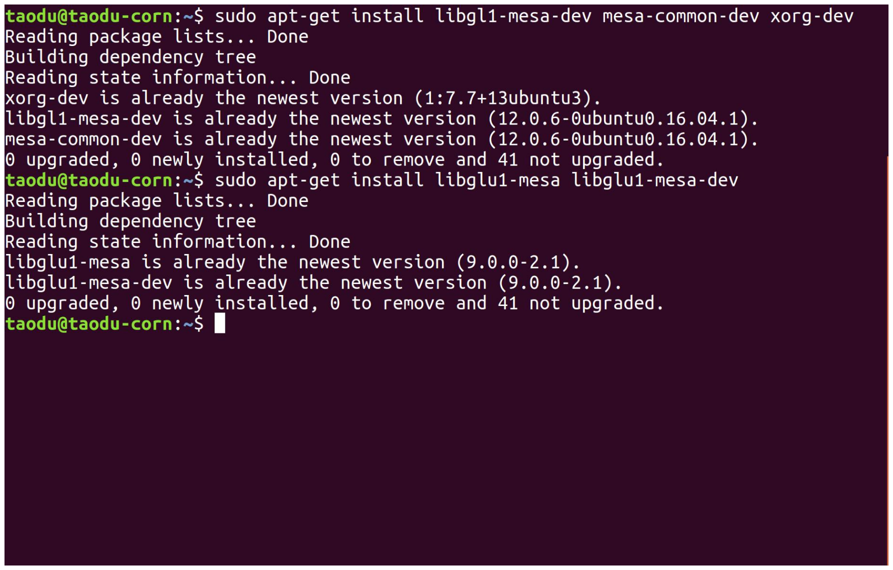
This tutorial has been tested on Ubuntu 16.04 LTS 64bit with gcc/g++ 4.9.4.

2.3.1 Install OpenGL libraries

Before you clone our project, make sure all the following libraries have been installed:

```
> sudo apt-get update
> sudo apt-get upgrade
> sudo apt-get install libgl1-mesa-dev mesa-common-dev xorg-dev
> sudo apt-get install libglu1-mesa libglu1-mesa-dev
```

Figure 7 is provided for your reference if you encounter any errors and are interested in knowing the exact version of these libraries we are using:



```
taodu@taodu-corn:~$ sudo apt-get install libgl1-mesa-dev mesa-common-dev xorg-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
xorg-dev is already the newest version (1:7.7+13ubuntu3).
libgl1-mesa-dev is already the newest version (12.0.6-0ubuntu0.16.04.1).
mesa-common-dev is already the newest version (12.0.6-0ubuntu0.16.04.1).
0 upgraded, 0 newly installed, 0 to remove and 41 not upgraded.
taodu@taodu-corn:~$ sudo apt-get install libglu1-mesa libglu1-mesa-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libglu1-mesa is already the newest version (9.0.0-2.1).
libglu1-mesa-dev is already the newest version (9.0.0-2.1).
0 upgraded, 0 newly installed, 0 to remove and 41 not upgraded.
taodu@taodu-corn:~$
```

Figure 7: (Ubuntu) Installing prerequisites.

2.3.2 Get the Source Code from GitHub

Create an empty folder, then use `git` to clone the project.

```
> mkdir test
> cd test
> git clone --recursive https://github.com/mit-gfx/multicopter_design.git
```

2.3.3 Download Libraries

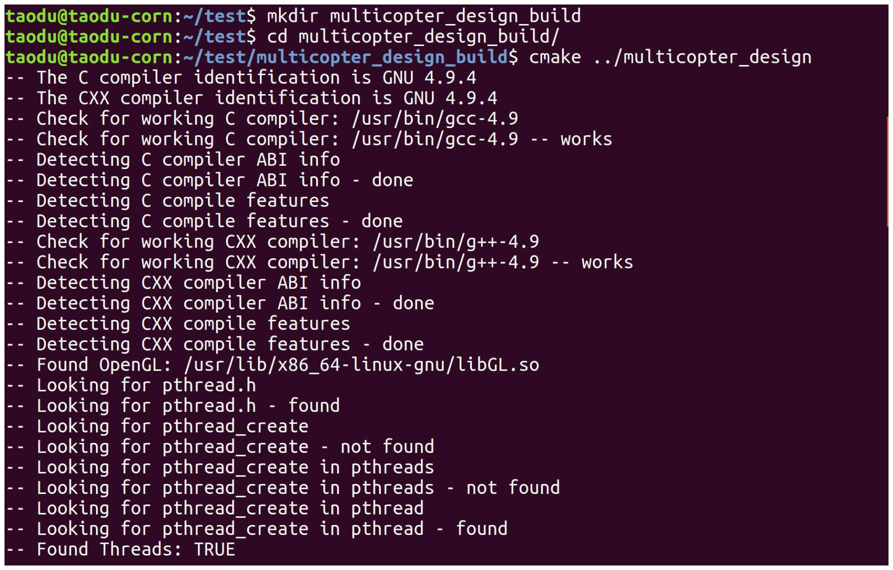
Our program relies on some 3rd party libraries. Please use the provided script to download and configure them:

```
> cd multicopter_design
> ./linux_setup.sh
> cd ../
```

2.3.4 Generate Makefile

Next, you will need CMake to generate a Makefile for compiling the code. As before we recommend an out-of-source build. In Figure 8, we create an empty folder named `test/multicopter_design_build`, and a Makefile is generated inside.

```
> mkdir multicopter_design_build
> cd multicopter_design_build
> cmake ../multicopter_design
```



```
taodu@taodu-corn:~/test$ mkdir multicopter_design_build
taodu@taodu-corn:~/test$ cd multicopter_design_build/
taodu@taodu-corn:~/test/multicopter_design_build$ cmake ../multicopter_design
-- The C compiler identification is GNU 4.9.4
-- The CXX compiler identification is GNU 4.9.4
-- Check for working C compiler: /usr/bin/gcc-4.9
-- Check for working C compiler: /usr/bin/gcc-4.9 -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/g++-4.9
-- Check for working CXX compiler: /usr/bin/g++-4.9 -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenGL: /usr/lib/x86_64-linux-gnu/libGL.so
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Looking for pthread_create
-- Looking for pthread_create - not found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
```

Figure 8: (Ubuntu) Using CMake to generate a Makefile.

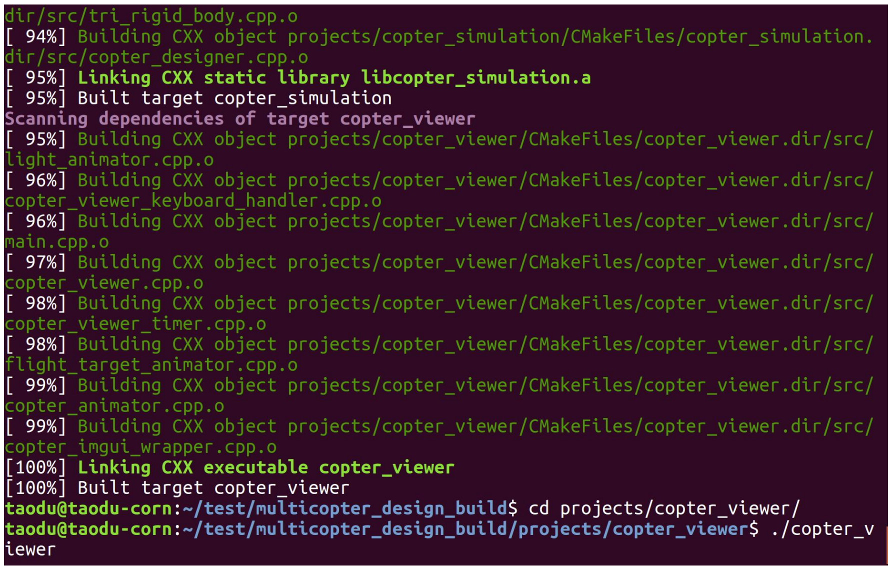
2.3.5 Build and Test

Once a Makefile is generated, you can call `make` to build it. If no errors occur, an executable file `copter_viewer` will be generated. To test if the build is successful, try running `copter_viewer` without any arguments (Figure 9), and a window like Figure 10 should pop up.

```
> make
> cd projects/copter_viewer/
> ./copter_viewer
```

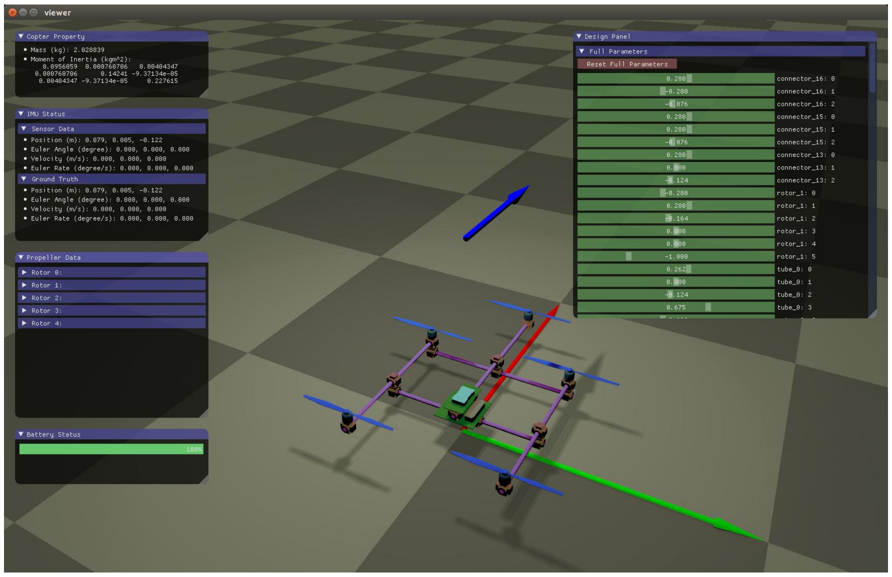
2.4 Build on macOS

The following tutorial has been tested on macOS Sierra (10.12.3) with AppleClang 8.1.0 provided in Xcode. The build steps are almost identical to those on Linux, except that no manual OpenGL installation is needed.



```
dir/src/tri_rigid_body.cpp.o
[ 94%] Building CXX object projects/copter_simulation/CMakeFiles/copter_simulation.
dir/src/copter_designer.cpp.o
[ 95%] Linking CXX static library libcopter_simulation.a
[ 95%] Built target copter_simulation
Scanning dependencies of target copter_viewer
[ 95%] Building CXX object projects/copter_viewer/CMakeFiles/copter_viewer.dir/src/
light_animator.cpp.o
[ 96%] Building CXX object projects/copter_viewer/CMakeFiles/copter_viewer.dir/src/
copter_viewer_keyboard_handler.cpp.o
[ 96%] Building CXX object projects/copter_viewer/CMakeFiles/copter_viewer.dir/src/
main.cpp.o
[ 97%] Building CXX object projects/copter_viewer/CMakeFiles/copter_viewer.dir/src/
copter_viewer.cpp.o
[ 98%] Building CXX object projects/copter_viewer/CMakeFiles/copter_viewer.dir/src/
copter_viewer_timer.cpp.o
[ 98%] Building CXX object projects/copter_viewer/CMakeFiles/copter_viewer.dir/src/
flight_target_animator.cpp.o
[ 99%] Building CXX object projects/copter_viewer/CMakeFiles/copter_viewer.dir/src/
copter_animator.cpp.o
[ 99%] Building CXX object projects/copter_viewer/CMakeFiles/copter_viewer.dir/src/
copter_imgui_wrapper.cpp.o
[100%] Linking CXX executable copter_viewer
[100%] Built target copter_viewer
taodu@taodu-corn:~/test/multicopter_design_build$ cd projects/copter_viewer/
taodu@taodu-corn:~/test/multicopter_design_build/projects/copter_viewer$ ./copter_v
iewer
```

Figure 9: (Ubuntu) Running `copter_viewer` without any arguments.



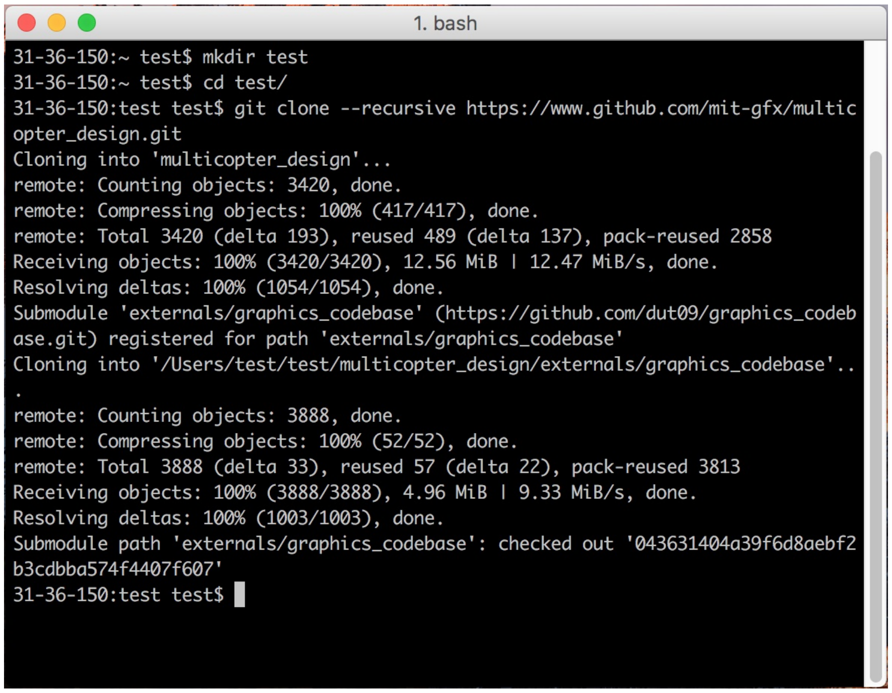
```
• Mass (kg): 2.028839
• Moment of Inertia (kgm^2):
     0.0956059  0.000760706    0.00404347
   0.000760706      0.14241 -9.37134e-05
    0.00404347 -9.37134e-05      0.227615

• Position (m): 0.079, 0.005, -0.122
• Euler Angle (degree): 0.000, 0.000, 0.000
• Velocity (m/s): 0.000, 0.000, 0.000
• Euler Rate (degree/s): 0.000, 0.000, 0.000

• Position (m): 0.079, 0.005, -0.122
• Euler Angle (degree): 0.000, 0.000, 0.000
• Velocity (m/s): 0.000, 0.000, 0.000
• Euler Rate (degree/s): 0.000, 0.000, 0.000

 0.280   connector_16: 0
-0.280   connector_16: 1
-0.076   connector_16: 2
 0.280   connector_15: 0
 0.280   connector_15: 1
-0.076   connector_15: 2
 0.280   connector_13: 0
 0.000   connector_13: 1
-0.124   connector_13: 2
-0.280   rotor_1: 0
 0.280   rotor_1: 1
-0.164   rotor_1: 2
 0.000   rotor_1: 3
 0.000   rotor_1: 4
-1.000   rotor_1: 5
 0.262   tube_0: 0
 0.000   tube_0: 1
-0.124   tube_0: 2
 0.675   tube_0: 3
```

Figure 10: (Ubuntu) A screenshot of the user interface after we build from the source code.

2.4.1 Get the Source Code from GitHub

Create an empty folder, then use `git` to clone the project. In Figure 11 we cloned it into a newly created empty folder `test/`:

```
> mkdir test
```



```
31-36-150:~ test$ mkdir test
31-36-150:~ test$ cd test/
31-36-150:test test$ git clone --recursive https://www.github.com/mit-gfx/multic
opter_design.git
Cloning into 'multicopter_design'...
remote: Counting objects: 3420, done.
remote: Compressing objects: 100% (417/417), done.
remote: Total 3420 (delta 193), reused 489 (delta 137), pack-reused 2858
Receiving objects: 100% (3420/3420), 12.56 MiB | 12.47 MiB/s, done.
Resolving deltas: 100% (1054/1054), done.
Submodule 'externals/graphics_codebase' (https://github.com/dut09/graphics_codeb
ase.git) registered for path 'externals/graphics_codebase'
Cloning into '/Users/test/test/multicopter_design/externals/graphics_codebase'..
.
remote: Counting objects: 3888, done.
remote: Compressing objects: 100% (52/52), done.
remote: Total 3888 (delta 33), reused 57 (delta 22), pack-reused 3813
Receiving objects: 100% (3888/3888), 4.96 MiB | 9.33 MiB/s, done.
Resolving deltas: 100% (1003/1003), done.
Submodule path 'externals/graphics_codebase': checked out '043631404a39f6d8aebf2
b3cdbba574f4407f607'
31-36-150:test test$
```

Figure 11: (macOS) Cloning the project into an empty `test/` folder.

```
> cd test
> git clone --recursive https://github.com/mit-gfx/multicopter_design.git
```

2.4.2 Download Libraries

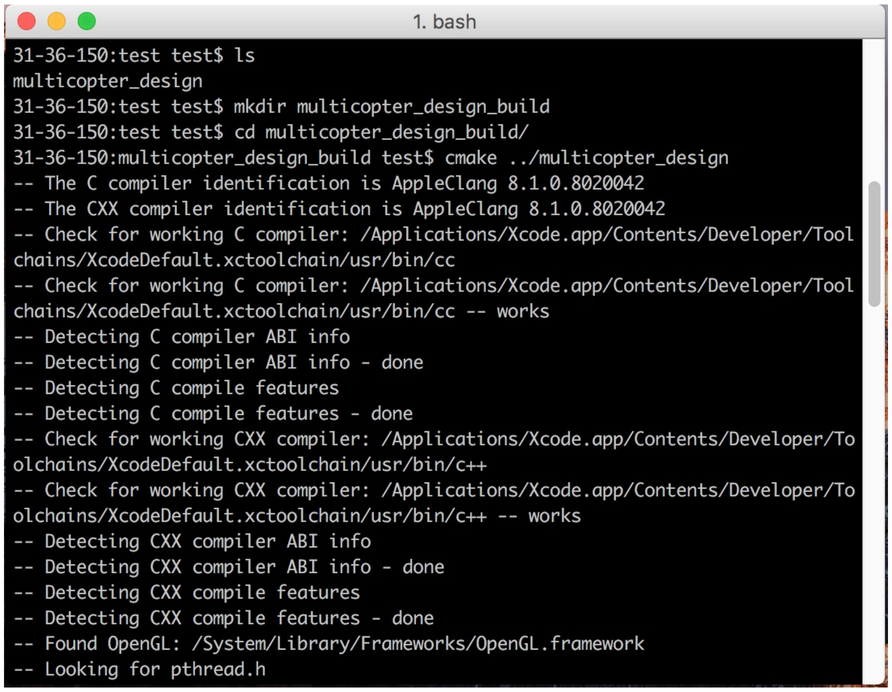
Our program relies on some 3rd party libraries. Please use the provided script to download and configure them:

```
> cd multicopter_design
> ./macos_setup.sh
> cd ../
```

2.4.3 Generate Makefile

Next, you will need CMake to generate a Makefile for compiling the code. We recommend an out-of-source build. In Figure 12, we create an empty folder named `test/multicopter_design_build`, and a Makefile is generated inside.

```
> mkdir multicopter_design_build
> cd multicopter_design_build
> cmake ../multicopter_design
```



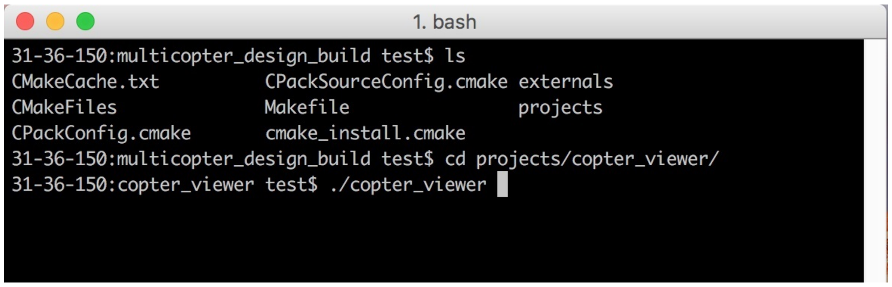
```
31-36-150:test test$ ls
multicopter_design
31-36-150:test test$ mkdir multicopter_design_build
31-36-150:test test$ cd multicopter_design_build/
31-36-150:multicopter_design_build test$ cmake ../multicopter_design
-- The C compiler identification is AppleClang 8.1.0.8020042
-- The CXX compiler identification is AppleClang 8.1.0.8020042
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Tool
chains/XcodeDefault.xctoolchain/usr/bin/cc
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Tool
chains/XcodeDefault.xctoolchain/usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/To
olchains/XcodeDefault.xctoolchain/usr/bin/c++
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/To
olchains/XcodeDefault.xctoolchain/usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenGL: /System/Library/Frameworks/OpenGL.framework
-- Looking for pthread.h
```

Figure 12: (macOS) Using CMake to generate a Makefile.

2.4.4 Build and Test

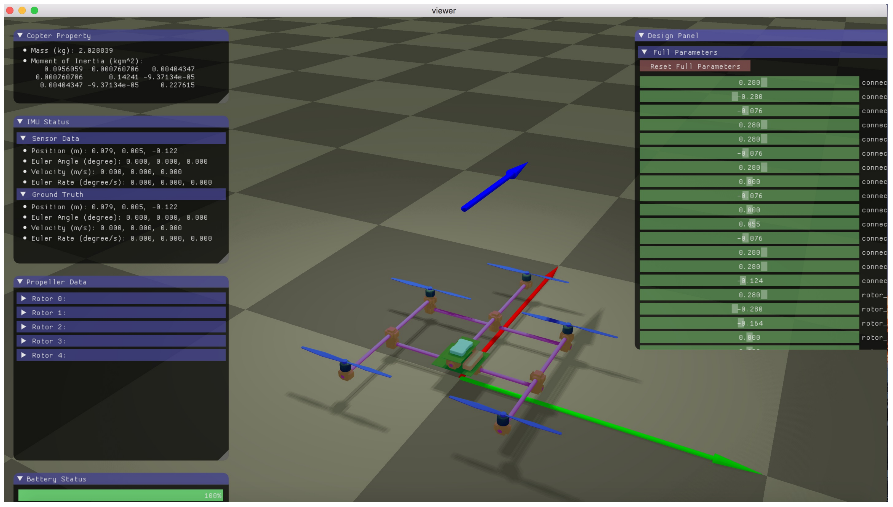
Once a Makefile is generated, you can call `make` to build it. If no errors occur, an executable file `copter_viewer` will be generated. To test if the build is successful, try running `copter_viewer` without any arguments (Figure 13), and a window like Figure 14 should pop up.

```
> make
> cd projects/copter_viewer/
> ./copter_viewer
```



```
31-36-150:multicopter_design_build test$ ls
CMakeCache.txt          CPackSourceConfig.cmake externals
CMakeFiles              Makefile                projects
CPackConfig.cmake       cmake_install.cmake
31-36-150:multicopter_design_build test$ cd projects/copter_viewer/
31-36-150:copter_viewer test$ ./copter_viewer
```

Figure 13: (macOS) Running `copter_viewer` without any arguments.



```
• Mass (kg): 2.028839
• Moment of Inertia (kgm^2):
    0.0956059  0.000760706   0.00404347
  0.000760706      0.14241 -9.37134e-05
   0.00404347 -9.37134e-05     0.227615
```

```
Sensor Data
• Position (m): 0.079, 0.005, -0.122
• Euler Angle (degree): 0.000, 0.000, 0.000
• Velocity (m/s): 0.000, 0.000, 0.000
• Euler Rate (degree/s): 0.000, 0.000, 0.000
Ground Truth
• Position (m): 0.079, 0.005, -0.122
• Euler Angle (degree): 0.000, 0.000, 0.000
• Velocity (m/s): 0.000, 0.000, 0.000
• Euler Rate (degree/s): 0.000, 0.000, 0.000
```

```
 0.280  connec
-0.280  connec
-0.076  connec
 0.280  connec
 0.280  connec
-0.076  connec
 0.280  connec
 0.000  connec
-0.076  connec
 0.000  connec
 0.055  connec
-0.076  connec
 0.280  connec
 0.280  connec
-0.124  connec
 0.280  rotor_
-0.280  rotor_
-0.164  rotor_
 0.000  rotor_
```

Figure 14: (macOS) A screenshot of the user interface after we build from the source code.

3 User Interface

We have briefly covered the user interface in Section 1. Here we provide more instructions on interacting with each window. We will also explain keyboard shortcuts and mouse operations in this section.

3.1 Copter Property Window

This window displays mass and moment of inertia information:

- **Mass** Computing mass is straightforward as it only requires summing up the mass of each part. Technically, each component is defined by a closed surface mesh and a density value. We will assume this mesh is solid and use the divergence theorem to analytically compute its mass.
- **Moment of Inertia** The moment of inertia is a 3×3 symmetric positive semidefinite matrix, defined on a local frame $\mathcal{F}$ rigidly attached to the copter. $\mathcal{F}$ is computed and fixed once an initial design is loaded. The origin of $\mathcal{F}$ is the center of mass and its three axes are parallel to the world frame axes. Recall that the world frame we use is north-east-down, where x, y and z point to the north (red), east (green) and down (not shown in the window as it is below the ground) direction.

The moment of inertia determines the torque need to rotate the copter. It is generally recommended that you place your initial design in a way that its heading is along the x axis with zero roll angle. In this way the diagonal elements in the matrix reveal the resistance of the copter to rolling, pitching, and yawing respectively.

3.2 IMU Status Window

This window shows the reading from a virtual sensor attached to the copter, and the ground truth status. Both consist of the following information:

- **Position** These three numbers show where the center of mass is located in the world frame. As z axis points down to the ground, positive altitude corresponds to a negative z value.
- **Euler Angle** These are the roll, pitch and yaw angles. Initially the body axes are parallel to world axes. We first rotate it around z by yaw, then rotate along the new y axis of the body frame by pitch, then finally rotate around the new x axis of the body frame by roll.
- **Velocity** The time derivative of position.
- **Euler Rate** The time derivative of Euler angles.

3.3 Propeller Data Window

This window contains information about rotors, with one sub-window for each. The rotors are ordered by their occurrence in the file. Each sub-window contains:

- **Position** This shows where the motor is located in the **body** frame. Their values get updated during the design process and fixed in simulation.
- **Direction** This unit vector shows where this motor points to in the **body** frame. Typically it is close to $(0, 0, -1)$.

- **Spinning** A string shown here indicates whether the propeller spins clockwisely (CW) or counter-clockwisely (CCW), defined in a view where the motor is pointing towards you.
- **Speed** This shows the rotation speed of the propeller, in RPM (round per minute).
- **Thrust** This is the magnitude of the thrust generated by the propeller.

Both speed and thrust are calculated from fitting measurement data provided in the file. Thrust is also clamped between 0 and the measured maximum.

3.4 Battery Status Window

This progress bar works as an indicator of the battery life. During a flight we collect the thrust of each rotor, and based on the measurement data provided in the file, we estimate the current sent to each rotor. Dividing the remaining capacity by the current gives a rough estimation of time left before the battery becomes completely drained. The progress bar will turn yellow when it is 60%, and become red when below 20%.

3.5 Design Panel Window

This is the main window people will be working on when designing the copter. It consists of a sub-window that allows you to directly change full parameters, another to manipulate equivalent reduced parameters, and a simulation panel.

3.5.1 Full Parameters

As we will see in Section 4, parts like tubes, connectors and motors are parametrized by their geometric properties. This window displays all parameters labeled by their names, with an extension of indices. Although you are allowed to slide each single value, note that these parameters are constrained by the way you build the copter, so changing one parameter may affect the others as well.

Technically, let $\mathbf{x}$ be a vector of all parameters listed in the "Full Parameter" window. We build all constraints as linear functions of $\mathbf{x}$, which are compactly represented by $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{Cx} \leq \mathbf{d}$. When you slide a single parameter in the window, we store all parameter values currently shown in the window into a vector $\mathbf{x}_0$, and the new $\mathbf{x}$ is determined by solving the optimization problem below:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \|\mathbf{x} - \mathbf{x}_0\| \\ s.t. \quad & \mathbf{Ax} = \mathbf{b} \\ & \mathbf{Cx} \leq \mathbf{d} \end{aligned} \tag{1}$$

In other words, we find in the feasible set the closest solution to the parameter values indicated by users from the window.

3.5.2 Reduced Parameters

It is common that tens or hundreds of parameters are introduced in a design while most of them are limited by equality constraints. In this case using reduced parameters can significantly reduce the design space. Mathematically, the solution of $\mathbf{Ax} = \mathbf{b}$ can be fully characterized by vectors in its null space. Let $\mathbf{Ax}_0 = \mathbf{b}$, and write $\mathbf{x} = \mathbf{x}_0 + \mathbf{Ey}$ where $\mathbf{E}$ spans the null space of $\mathbf{A}$. Now the equality constraints are removed and we can focus on a much lower dimensional vector $\mathbf{y}$.

The reduced parametric representation brings its own drawbacks though. Perhaps the biggest issue here is that $\mathbf{y}$ does not have a straightforward physical meaning any more. Each element does not correspond to a single component, but might affect the global design in an unintuitive way.

Unlike the full parameters, it is easier to update the reduced parameters when they get updated in the window. Let $\mathbf{Fy} \leq g$ be the only inequality constraints on $\mathbf{y}$. When one y_i is updated, we fix other y_is and calculate the feasible range of y_i and reflect it in the window. If some y_i freezes on the boundary of $\mathbf{Fy} \leq \mathbf{g}$, its background then becomes red to imply this y_i is not changeable.

3.5.3 Simulation

This window contains a button to trigger the simulation process. Your design is then set to track the position and heading of the blue arrow in the scene, which can be changed by keyboard shortcuts discussed below. All design parameters are frozen during flight unless you press the button again and switch back to the design process.

Behind this simulation button we execute a 30Hz loop to control the flight. Within each iteration we do the following things in order:

- Collect the sensor data and target information.
- Execute the control policy based on them.
- Send control signals to each rotor.
- Use thrust and torque from each rotor to update the motion.

Note that depending on your graphics card the actual FPS (frame per second) may be above or below 30. So the time flow you feel in the window is not exactly the actual motion you expect to see in your real copter.

3.6 Keyboard Shortcuts

The following keys are used as shortcuts. Most are activated only in simulation.

- **Left/Right/Up/Down Arrow** This four keys control the horizontal position of the target (blue arrow in the window). The default controller will try to drag the copter to track the blue arrow as close as possible.
- **'w' and 's'** These two keys adjust the vertical location of the blue arrow, which in turn controls the altitude of the copter.
- **'a' and 'd'** These keys rotate the blue arrow on the horizontal plane, which changes the desired heading of the copter in simulation.
- **'p'** When pressing 'p' you can pause/resume the simulation.

3.7 Mouse Operations

Mouse gestures are used to control the camera view. Specifically, the following operations are supported:

- **Zooming in/out** By scrolling the mouse wheel you can zoom the camera to get closer/farther to the scene.

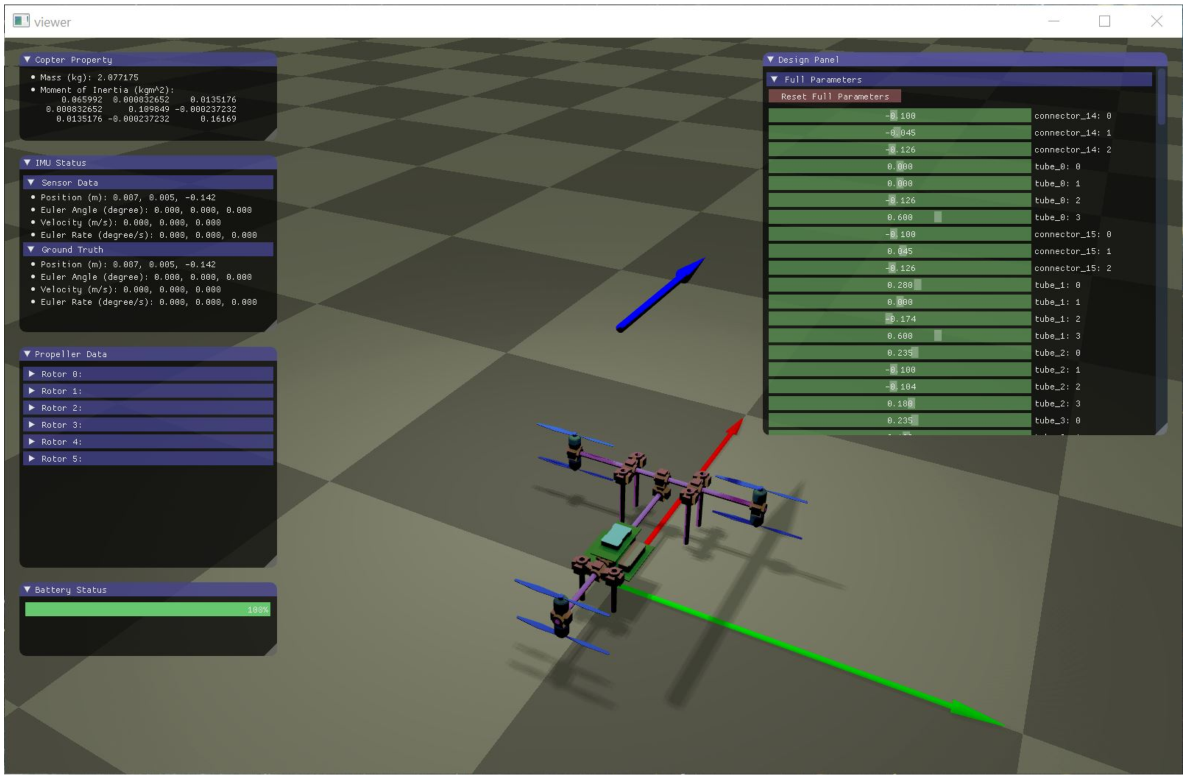
- **Rotation** You can press the mouse wheel and drag to rotate the camera.
- **Panning** Press shift and the mouse wheel and drag your mouse at the same time to change the camera location.

4 File Format

We use XML files to define the copter used in our design and simulation tool. A list of sample files are provided in the folder `resources/copter/` for your reference. To load an XML file, simply use its name as the first argument when you run `copter_viewer`. For example:

```
> ./copter_viewer y6copter.xml
```

will load a Y6 hexacopter defined in `y6copter.xml` (Figure 15). If no XML file is given, then `pentacopter.xml` is used by default.



```
• Mass (kg): 2.077175
• Moment of Inertia (kgm^2):
     0.065992  0.000832652    0.0135176
  0.000832652     0.109849 -0.000237232
    0.0135176 -0.000237232      0.16169

• Position (m): 0.087, 0.005, -0.142
• Euler Angle (degree): 0.000, 0.000, 0.000
• Velocity (m/s): 0.000, 0.000, 0.000
• Euler Rate (degree/s): 0.000, 0.000, 0.000

• Position (m): 0.087, 0.005, -0.142
• Euler Angle (degree): 0.000, 0.000, 0.000
• Velocity (m/s): 0.000, 0.000, 0.000
• Euler Rate (degree/s): 0.000, 0.000, 0.000
```

```
-0.100  connector_14: 0
-0.045  connector_14: 1
-0.126  connector_14: 2
 0.000  tube_0: 0
 0.000  tube_0: 1
-0.126  tube_0: 2
 0.600  tube_0: 3
-0.100  connector_15: 0
 0.045  connector_15: 1
-0.126  connector_15: 2
 0.280  tube_1: 0
 0.000  tube_1: 1
-0.174  tube_1: 2
 0.600  tube_1: 3
 0.235  tube_2: 0
-0.100  tube_2: 1
-0.104  tube_2: 2
 0.180  tube_2: 3
 0.235  tube_3: 0
```

Figure 15: Loading a Y6 hexacopter from the given XML file `y6copter.xml`.

The XML file itself its pretty straightforward: once you learn the rules, you are free to create your own XML files and use the software to design and simulate them. For simplicity, below we will walk you through `quadcopter.xml` so that you can get an idea of how to write your own copter file.

4.1 Basic Structure

The XML file starts with a line defining the version:

```
<?xml version="1.0" ?>
```

The whole copter is then wrapped by a pair of tags `<copter>` and `</copter>`. You should put all your components inside these opening and closing tags. To define a complete copter, you will have to provide definitions for the following components: plates, a battery, an electronic device, tubes, connectors, propellers, and motors. All meshes used here must be watertight, otherwise we won't be able to estimate their moment of inertia.

4.2 3D Transforms

Many of the components are defined in their own frames in the mesh file. To place them correctly you need to apply a series of transforms. In the XML file you can specify it by using `translate` and `rotate` tags. Scaling is in general not supported except for plates and tubes.

To apply a translation, use the following tag:

```
<translate xyz="0 1 2"/>
```

Applying it will move the mesh by $(0, 1, 2)$.

Rotation is defined by Euler angles (ϕ, θ, ψ), which are also known as roll, pitch and yaw. We first rotate the mesh along its own z axis by ψ, then around its own (new) y axis by θ, finally along its (new) x axis by ϕ. The angles are in degrees and its sign is defined by the right-hand rule.

```
<rotate xyz="0 0 90"/>
```

In this example, we rotate the mesh by 90 degrees around its own z axis.

You can apply a series of translation/rotation to a single mesh. For example, if you want to translate it first by $(0, 1, 2)$, then rotate it by $(45, 0, 90)$, then translate it by $(3, 4, 5)$:

```
<translate xyz="0 1 2"/>
<rotate rpy="45 0 90"/>
<translate xyz="3 4 5"/>
```

We will see many 3D transform examples shortly.

Finally, the world frame in which we define the copter is NED (north east down), which means **negative** z is the **up** direction. To avoid penetrating the ground, we usually lift our copter a little in the definition.

4.3 Add Plates

To define a plate, you will need to first create the tag `<plate>` and `</plate>` with two attributes: `file` specifies the mesh file, and `density` defines its density:

```
<plate file="plate.obj" density="1180.0">
  ...
</plate>
```

In our code, we look for a file named `plate.obj` in the folder `resources/mesh`, which we have provided along with the source code. This mesh is a $1\text{m}\times1\text{m}\times0.0024\text{m}$ axis-aligned cuboid centered at the origin, and the unit of the density value is kg/m^3.

Between `<plate>` and `</plate>` you can create multiple plates by using `<instance>` tag, one for each plate. An `<instance>` tag needs you to specify the size of your plate along x and y direction first, then specify the location and orientation of the plate by applying translation or rotation.

```
<instance size="0.15 0.15">
  <translate xyz="0 0 -0.1"/>
</instance>
```

Here we define our first plate: it is a 0.15m×0.15m×0.0024m cuboid, and it is lifted by 0.1 meters. The second plate has a smaller size (0.15m×0.07m), and it is lifted by 0.148 meters, leaving a $0.048 - 0.0024 = 0.0456$ meters gap between them. This is exactly the size of one connector, which we will cover shortly:

```
<instance size="0.15 0.07">
  <translate xyz="0 0 -0.1"/>
  <!-- 0.0024m is the thickness of the plate. -->
  <translate xyz="0 0 -0.0024"/>
  <!-- 0.0456m is the size of the connector. -->
  <translate xyz="0 0 -0.0456"/>
  <!-- Instead of copying and pasting the three translations here, we will
       use -0.148 = -0.1 - 0.0024 - 0.0456 below. -->
</instance>
```

Plates do not have any free parameters in the design. Once they are defined in the XML, they cannot be changed any more.

4.4 Add Batteries

Batteries are defined by `<battery>`, which requires a file name, a density value (in kg/m^3) and its capacity (in mAh). The density is estimated by using the mass and size from the manufacturer. Currently only one battery is allowed:

```
<battery file="battery.obj" density="2857.1" capacity="2200">
  <rotate rpy="0 0 90"/>
  <translate xyz="0 0 -0.1"/>
</battery>
```

As before, you can find `battery.obj` in `resources/mesh`. In that file, we intentionally set the mesh in a position such that it is attached to `plate.obj` seamlessly. So by lifting the battery to 0.1 meters above the ground, it is now attached to the lower plate we defined earlier.

Like plates, batteries are not parametrized in the design.

4.5 Add Electronic Devices

An electronic device is defined by `<electronics>` and `</electronics>`. It is only used for visualization. We also assume there is only one device needed in a copter:

```
<electronics file="electronics.obj" density="601.6">
  <!-- The electronic device is placed on top of the upper plate. -->
  <translate xyz="0 0 -0.148"/>
  <!-- Lift by 0.01 more because the mesh itself has thickness. -->
  <translate xyz="0 0 -0.01"/>
</electronics>
```

`electronics.obj` defines a mesh bounded by a 0.0816m×0.05m×0.0154m box and centered at the origin.

Like plates and batteries, the location and size of an electronic device are fixed once it is defined in the XML file.

4.6 Add Tubes

Tubes are defined between `<tube>` and `</tube>`. Each tube needs you to specify the length and location:

```
<tube file="tube.obj" density="1638.9">
  <instance length="0.5">
    <translate xyz="0 0 -0.1"/>
    <!-- 0.0012 is half of the thickness of a plate. -->
    <translate xyz="0 0 0.0012"/>
    <!-- 0.0228 is half of the size of a connector. -->
    <translate xyz="0 0 0.0228"/>
  </instance>
  <instance length="0.5">
    <rotate rpy="0 0 90"/>
    <translate xyz="0 0 -0.1"/>
    <!-- 0.0012 is half of the thickness of a plate. -->
    <translate xyz="0 0 -0.0012"/>
    <!-- 0.0228 is half of the size of a connector. -->
    <translate xyz="0 0 -0.0228"/>
  </instance>
  <!-- For simplicity we will use -0.076 and -0.124 below. -->
</tube>
```

The default `tube.obj` is a 1-meter-long tube centered at the origin, and pointing along the x axis. Here two 0.5-meter-long tubes are defined, one below the lower plate ($z = -0.1$) by 0.24m and the other rotated by 90 degrees and then between two plates ($z = -0.1$ and $z = -0.148$).

A tube is parametrized by its center of mass (3 degrees of freedom) and length (1 degree of freedom), but its direction is fixed in the design.

4.7 Add Connectors

Connectors are defined between `<round_connector` and `</round_connector>`:

```
<round_connector file="connector.obj" density="1170.0">
  <instance>
    ...
  </instance>
  ...
</round_connector>
```

The default mesh file `connector.obj` defines a mesh that is centered at the origin, with a bounding box of size 0.04m×0.0456m×0.0456m. Without any transform, a connector will be placed at the center of `tube.obj` and allow the default tube to pass through. It is crucial to place them in the right position, otherwise our algorithm won't be able to parse their connection and constraints correctly.

As an example, consider the first connector we defined after `<round_connector>`:

```
<instance>
  <rotate rpy="0 0 90"/>
  <translate xyz="0 0 -0.124"/>
</instance>
```

So we first rotate the connector along z axis by 90 degrees, then place it at $(0, 0, -0.124)$. Recall that our second tube is placed at $(0, 0, -0.124)$ and points to the y direction. In this way, the tube passes through the connector precisely. The value -0.124 comes from the half size of the connector (0.0228), the half of the thickness of the plate (0.0012), and the offset of the lower plate (-0.1), which sums up to -0.124.

As another example, consider the third connector in `quadcopter.xml`:

```
<instance>
  <translate xyz="0.23 0 -0.076"/>
</instance>
```

Recall that the first tube is centered at $(0, 0, -0.076)$ and has a length of 0.5 meters. So placing this connector at $(0.23, 0, -0.076)$ means it is at the end of the tube (a 0.02 gap is left here because the connector itself spans 0.04 meters along x axis).

Each connector has 4 faces that can be connected to plates, motors, or other connectors. A connector is parametrized by its center of mass (three degrees of freedom). If it is attached to a plate then the connector becomes fixed (zero degrees of freedom). If two connectors, or a connector and a motor are attached, then their relative position is fixed. Finally, each connector must have a parent tube, which further limits the possible position of the connector: it can only slide along the tube bounded by its two endpoints.

You do not have to explicitly specify which connectors are connected, or which is the parent tube. Our program will parse your XML file and as long as you place them in the world frame in a correct way, the algorithm will identify the connection automatically.

4.8 Add Propellers

The propeller definition in XML file is for visualization only, and it requires a file name to load the corresponding mesh:

```
<propeller file="propeller_10inch.obj"/>
```

We provide two sample propellers in our resource folder: `propeller_10.inch.obj` and `propeller_14inch.obj`, and you are free to use either of them here, or devise your own propeller.

4.9 Add Motors

Finally, motors need to be specified by using `<motor>` and `</motor>`:

```
<motor file="motor.obj" density="4833.3"
  measurement="motor_10inch_prop.txt" propeller_height="0.019">
  ...
</motor>
```

As before, `motor.obj` defines the default motor mesh which spins around the z axis. Its top and bottom surfaces are at $z = \mp 0.018$ meters. This offset will be used later to place motors on top of its supporting connector.

The `measurement` attribute requires you to provide a file that contains data samples of duty cycle, torque, thrust, motor speed and current. Relatively accurate torque and thrust samples are needed to construct a plausible dynamic model, and current samples are used to estimate the battery life. Motor speed (in round-per-minute, or RPM) is mostly for visualization but a good measurement can help you do a sanity check on thrust samples. We have provided our measurement in `resources/measurement` for you to start using the software, but eventually you should provide your own measurement for your model.

The last attribute, `propeller_height`, defines the relative height between the center of motor and its propeller. Since the top surface of the motor is 0.18m tall, we set this value to be 0.19, leaving a 0.01m gap between to avoid any collision in visualization.

Each motor is then defined by using the `instance` tag. A spinning direction, either clockwise or counter-clockwise, needs to be specified in the `spin_dir` attribute. The actual location is then defined by the similar way as above:

```
<instance spin_dir="cw">
  <!-- Location of its parent connector. -->
  <translate xyz="0.23 0 -0.076"/>
  <!-- Half size of its parent connector. -->
  <translate xyz="0 0 -0.0228"/>
  <!-- 0.018 is the distance between the bottom surface and the center of
       the motor. -->
  <translate xyz="0 0 -0.018"/>
</instance>
```

Each motor, parametrized by its center of mass (three degrees of freedom), must be attached to a parent connector, and the relative position between the motor and its connector is fixed. Note that you can rotate the motor along with its parent in the XML file. One example of this can be found in `resources/copter/vtailcopter.xml`.

Finally, each motor has an optional attribute named `flip_prop`, which by default is false. This attribute is used for the case when you want to flip your propeller to provide a downward thrust (relative to the motor) instead of the usual upward thrust, or if you want to install your motor upside down. You can refer to `resources/copter/y6copter.xml`, in which three lower motors are installed upside down.

5 Acknowledgment

We would like to thank Jie Xu for implementing a secondary user interface and Budmonde Duinkhar for debugging an initial version of the controller. We also thank Ryan Gulland for his comments on the draft of this user guide.

References

[1] Tao Du, Adriana Schulz, Bo Zhu, Bernd Bickel, and Wojciech Matusik. Computational multicopter design. *ACM Transaction on Graphics*, 2016.