Quadrotor Flight Simulator
Architecture/Design Document

Table of Contents

Change History

Version: <1>

Modifier: <Mayank Khichar>

Date: 11/13/2016

Description of Change:

1. Created 2d graphics to show quadrotor on screen.
2. Added functionality to capture keystrokes and accordingly update the quadrotor on screen.
3. Added background music to the simulator to enhance the user experience.

Version: <2>

Modifier: <Vedant Chittlangia>

Date: 11/16/2016

Description of Change:

1. Created class Motor and class Battery
2. Derived class Motor Software from the above mentioned classes.

Version: <3>

Modifier: <Ishana Shekhawat>

Date: 11/22/2016

Description of Change:

3. Derived and coded the quadrotor dynamics as a quadcopter class, with functionality to include external inputs in the form of 4 motor speeds.
4. Added the boost functionality and ode-int. This was done as a separate project, later will be included in the QT based code.

Version: <4>

Modifier: <Mayank Khichar,Ishana Shekhawat>

Date: 11/24/2016

Description of Change:

1. Removed 2d graphics code and instead added QT based classes to generate 3D representation of the quadrotor.
2. Added classes to translate and rotate the coordinate matrix of the quadrotor.
3. Added functionality to move quadrotor in 3D using keystrokes.
4. Included boost code in the QT based code as a separate class

---

Version: <5>

Modifier: <Mayank Khichar, Vedant Chittlangia, Ishana Shekhawat>

Date: 12/04/2016

Description of Change:

5. Modified quadrotor shape (from a simple cuboid to a shape which looks more like a quadrotor).
6. Added motor software class to calculate instantaneous motor rotational speeds (rpm) based on instantaneous keyboard input.
7. Added quadrotor dynamics classes which calculates quadrotor motion parameters based on instantaneous motor speeds.
8. Removed direct control of quadrotor motion on screen though keyboard input and instead used outputs from quadrotor dynamics classes to update the quadrotor position.
9. Included the drag forces in order to stabilize and enhance the visualization
10. Included the file logging functionality.

Introduction

Architecture and Design

The purpose of the software is to let the user test the quadrotor in the virtual environment using the keyboard inputs analogous to a handheld controller. Different parameters for the quadcopter such as the motor rpm, motor-voltage dependence, arm lengths, mass, and the response time can be modified. These parameters are available only to the software developers and can be changed only in the code. For other users, this software doubles up to a quadcopter flight controller game with the fixed dynamics.

This software project closely reflects the use of object oriented programming C++ to solve ordinary differential equations and update the parameters in real time to realise the trajectory of the quadrotor.

This document describes the architecture and design for the Quadrotor Flight Simulator application being developed for control engineers and leisure gaming. It takes the keyboard input to display the flight of a quadrotor.

The purpose of this document is to describe the architecture and design of the Quadrotor Flight Simulator application in a way that addresses the interests and concerns of all major stakeholders. For this application the major stakeholders are:

- Users – they want a software that is not numerical in display and they can have fun with while using it
- Developers – they want an architecture that will minimize complexity and development effort. They want the software to be compatible with readily available open source packages like STL
- Professor/Teaching Assistant - they want a software that is easier to understand and covers all major aspects of object oriented programming in C++. They want an executable code with the instructions to run it on various platforms.
- Maintenance Programmers – they want assurance that the system will be easy to evolve and maintain on into the future.

The architecture and design for a software system is complex and individual stakeholders often have specialized interests. There is no one diagram or model that can easily express a system's architecture and design. For this reason, software architecture and design is often presented in terms of multiple views or perspectives [IEEE Std. 1471]. Here the architecture of the Quadrotor Flight Simulator application is described from 4 different perspectives [1995 Krutchen]:

1. Logical View – major components, their attributes and operations. This view also includes relationships between components and their interactions. When doing OO design, class diagrams and sequence diagrams are often used to express the logical view.
2. Process View – the threads of control and processes used to execute the operations identified in the logical view.
3. Development View – how system modules map to development organization.
4. Use Case View – the use case view is used to both motivate and validate design activity. At the start of design the requirements define the functional objectives for the design. Use cases are also used to validate suggested designs. It should be possible to walk through a use case scenario and follow the interaction between high-level components. The components should have all the necessary behavior to conceptually execute a use case.

Design Goals

There is no absolute measure for distinguishing between good and bad design. The value of a design depends on stakeholder priorities. For example, depending upon the circumstances, an efficient design might be better than a maintainable one, or vice versa. Therefore, before presenting a design it is good practice to state the design priorities. The design that is offered will be judged according to how well it satisfies the stated properties.

The design priorities for the Quadrotor Flight Simulator application are:

- The design should allow individual team member to work independently
- It should be interactive and fun to play or work with
- The design should be such that the team of three graduate students can work on it with equal work distribution among all.
- The design parameters should be modifiable to incorporate range of quadcopter behavior.

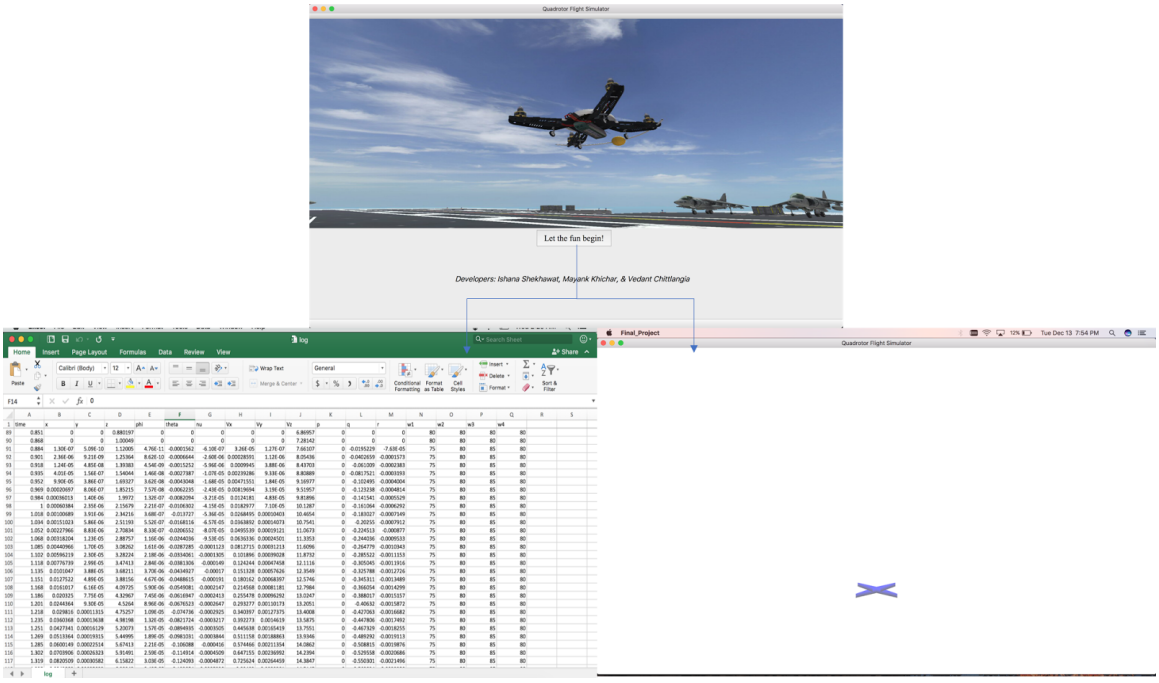
System Behavior

The use case view is used to both drive the design phase and validate the output of the design phase. The architecture description presented here starts with a review of the expected system behavior in order to set the stage for the architecture description that follows. For a more detailed account of software requirements, see the requirements document.

When the software executable is launched the GUI for the simulator appears. This GUI provides developer information. This GUI also works as an entry point for the main quadrotor simulator. To launch the simulator, user needs to press the button *‘Let the fun begin!’*. If the GUI window is closed without pressing the button simulator won’t launch.

GUI also plays audio user instructions for operating the quadrotor; launching the simulator would kill the instruction audio in between.

On clicking the GUI button, GUI gets closed and a *'simulatorWindow'* class object is instantiated. This window captures all the keystrokes and updates the quadrotor position accordingly. Along with this visualization of quadrotor in 3D, the software also stores the quadrotor trajectory (x,y,z) and its other state parameters with time. This stored data can be used for further analysis of the trajector.



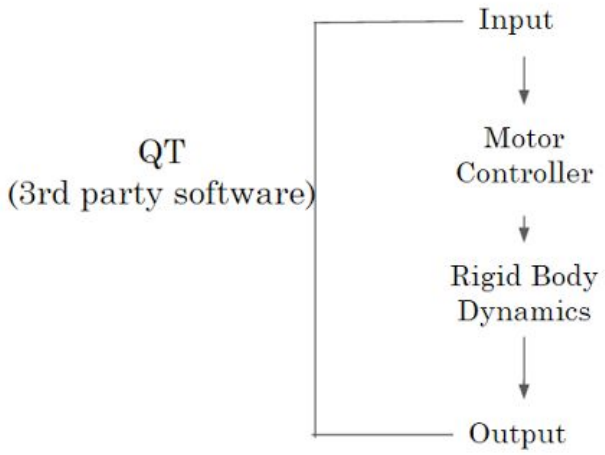
Logical View

The logical view describes the main functional components of the system. This includes modules, the static relationships between modules, and their dynamic patterns of interaction.

In this section the modules of the system are first expressed in terms of high level components (architecture) and progressively refined into more detailed components and eventually classes with specific attributes and operations.

1.1 High-Level Design (Architecture)

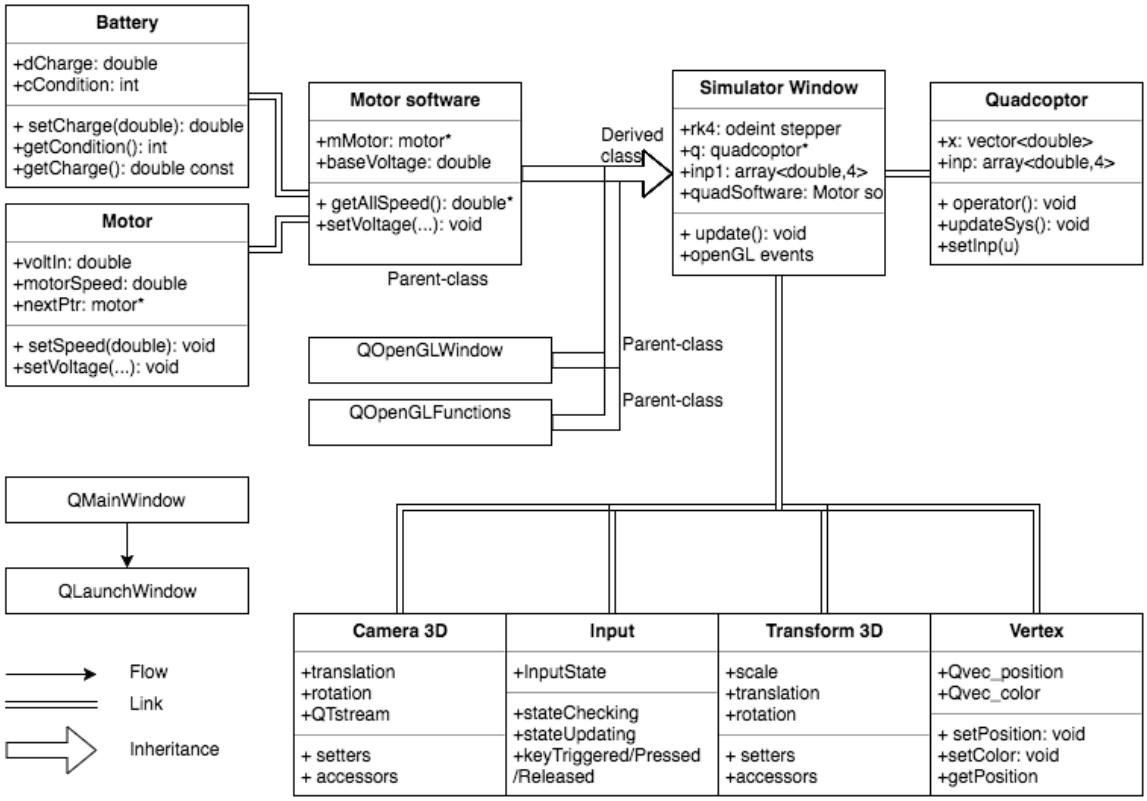
The high-level view or architecture consists of 4 major components:



System Architecture

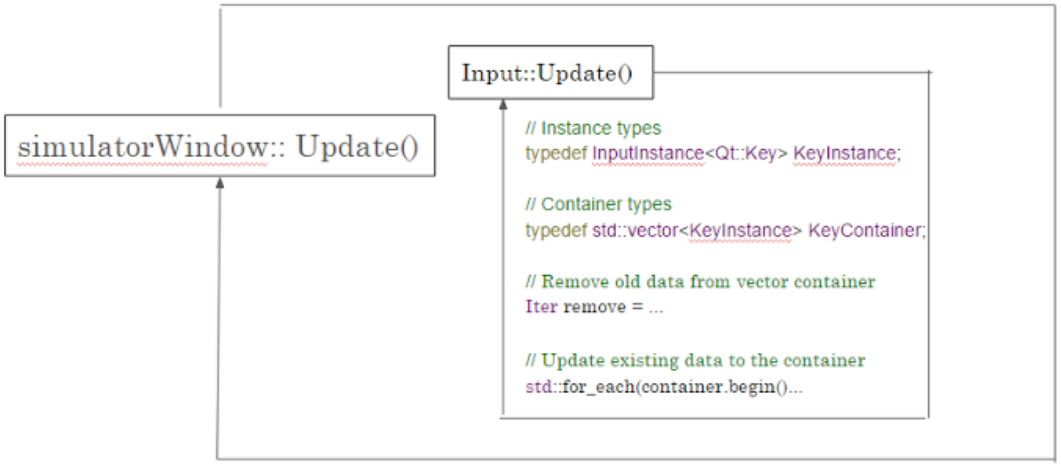
- The **QT** is the 3rd party software used to assemble all the modules
- The **Input** provides the user interface to input the desired movement of the quadcopter using the keyboard
- The **Motor Controller** calculates motor rpm based on the input given by the user
- Given the updated motor rpm, the **Rigid Body Dynamics** will the updated velocity of the quadcopter based on the previous speed and acceleration.
- The **Output** displays all the dynamics in a graphical user interface.

1.2 Mid-Level Design



QT overview

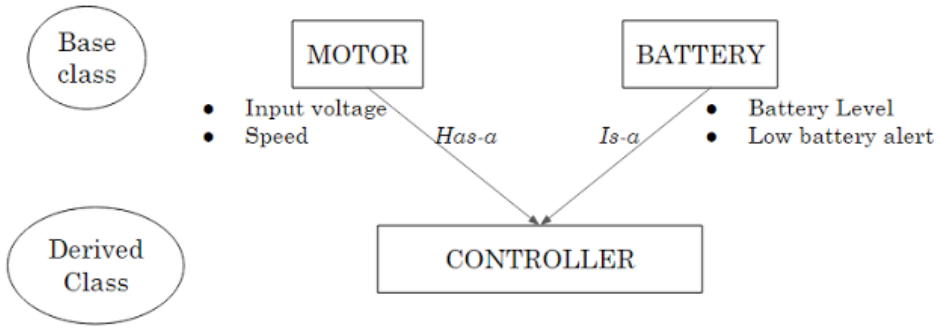
Simulator Window overview (read as Input, Output, GUI)



The aim of this class is to display an object (here quadrotor) at a particular coordinate location. This class takes the keyboard inputs and pass this information to Motor

controller for speed calculations, and then obtain the quadrotor position at next time step from Rotor Dynamics.
The object is defined by its vertices. This coordinate group is converted to final position in GPU coordinates using Vertex Shader program. Fragment shader takes information from vertex shader and provides output to Back buffer. OpenGL Vertex array object is used to combine all vertices and store it as a single object.

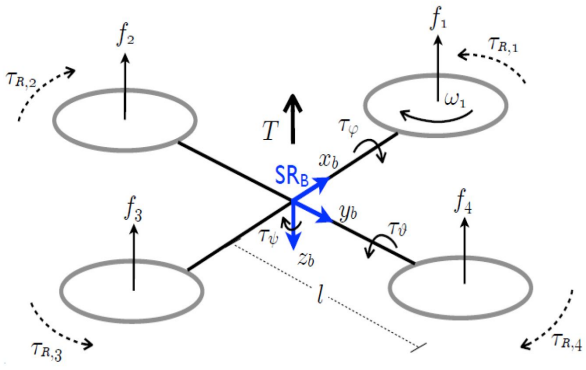
Motor Controller overview (class: Motor software)



Motor Controller is implemented in the class “Motor software”. Motor software is derived from two base classes - Motor and Battery. Battery stores the information of the charge in the battery and can send the low battery signal. There is a single battery for the whole quadrotor so it posses “Is-a” relationship to the Motor software. It is publicly inherited and its functions and members can be accessed by the object of class Motor Software as its own member functions. There are 4 motors for a quadrotor so class Motor possess “has-a” relationship with class Motor Software that is Motor software has objects of class Motor type. Four motors are created dynamically using pointers when the constructor of Motor Software is invoked and joined together in a linked list. Motor Software receives the input signals for movement from the Simulator Window - throttle, front, back, left and right. Throttle sets the base speed of the motors and depending upon other inputs rpm of the four motors are calculated. These rpm can be accessed by Simulator Window through Motor Software object and then can be passed to the Rigid Body Dynamics.

Rotor Dynamics overview (Quadcopter)

The dynamics were derived from using the free body diagram of a quadrotor, where the gyroscopic effects were neglected. The resultant system has 12 states which include the translation position and velocities and the orientations and rate of change of orientations.



The equations for the state derivatives are as follows:

$$\dot{x} = v_x$$

$$\dot{y} = v_y$$

$$\dot{z} = v_z$$

$$\dot{v}_x = F_{A,x} - (\cos(\psi)\sin(\nu)\cos(\varphi) + \sin(\psi)\sin(\varphi))\frac{T}{m}$$

$$\dot{v}_y = F_{A,y} - (\sin(\psi)\sin(\nu)\cos(\varphi) - \sin(\varphi)\cos(\psi))\frac{T}{m}$$

$$\dot{v}_z = F_{A,z} + g - (\cos(\nu)\cos(\varphi))\frac{T}{m}$$

$$\dot{\varphi} = p + \sin(\varphi)\tan(\nu)q + \cos(\varphi)\tan(\nu)r$$

$$\dot{\nu} = \cos(\varphi)q - \sin(\varphi)r$$

$$\dot{\psi} = \sin(\varphi)\sec(\nu)q + \cos(\varphi)\sec(\nu)r$$

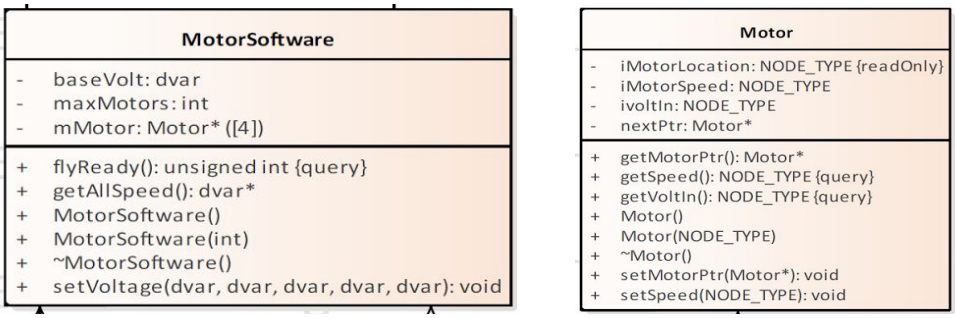
$$\dot{p} = \tau_{A,x} + \frac{I_r}{I_x} q\Omega_r + \frac{I_y - I_z}{I_x} qr + \frac{\tau_\varphi}{I_x}$$

$$\dot{q} = \tau_{A,y} + \frac{I_r}{I_y} q\Omega_r + \frac{I_z - I_x}{I_y} pr + \frac{\tau_\nu}{I_y}$$

$$\dot{r} = \tau_{A,z} + \frac{I_x - I_y}{I_z} pq + \frac{\tau_\psi}{I_z}$$

The differential equations are implemented in the overloaded operator () of the class Quadcoptor. This is accessed by the ode_solver object which was initialized using the boost::numeric library.

1.3 Detailed Class Design

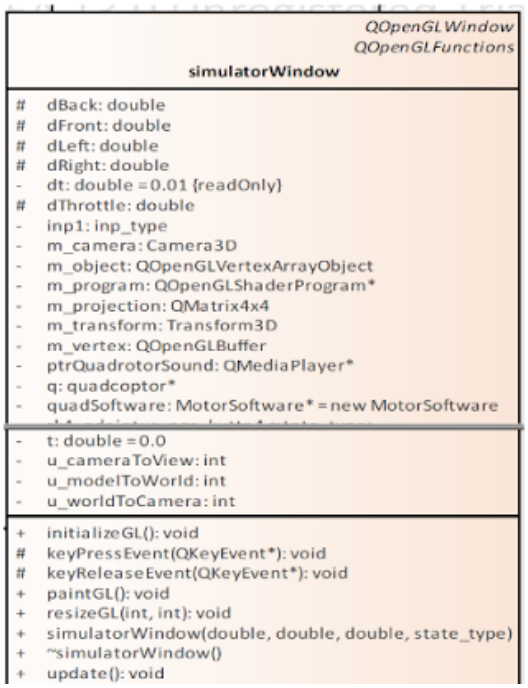


class MotorSoftware

It has 4 pointers of Motor type through which Motor objects are dynamically created and connected in a linked list upon initialisation through the constructor. Function setVoltage() sends the appropriate voltage to the Motor object function setSpeed() to calculate the rpm of the motors. Function getAllSpeed() invokes getSpeed() function to get the updated rpm and return it to the user.

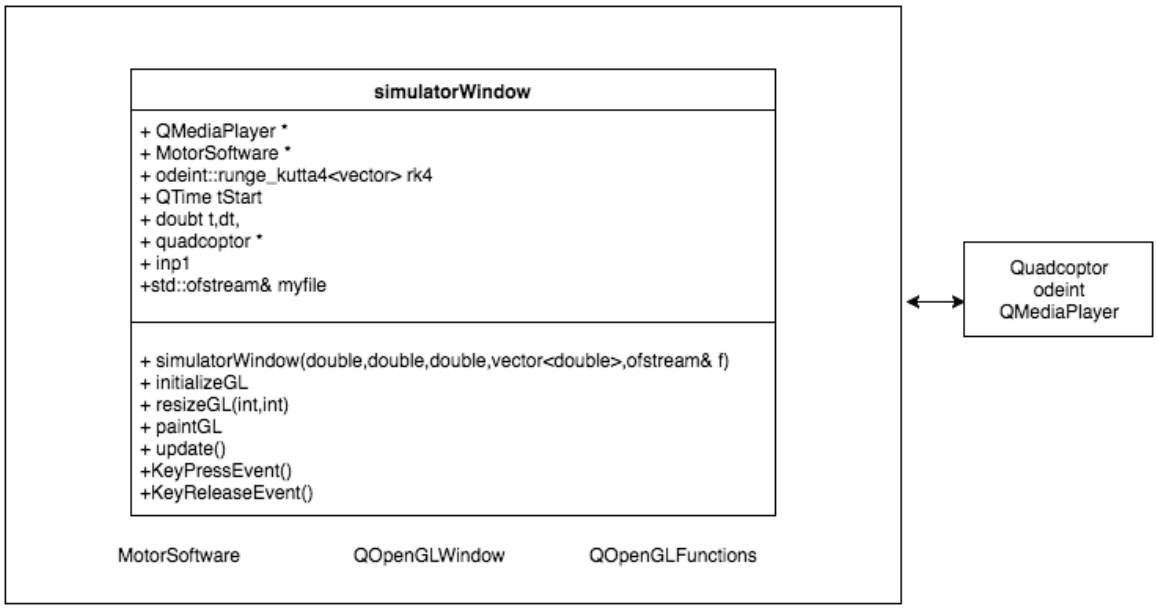
class Motor

This class is created such that it can store the location of next motor type nextPtr pointer. The class can return voltage by calling getVoltIn() and the speed by calling getSpeed(). Speed is updated by the function setSpeed() which takes the voltage as the input and calculates the speed based on the speed manipulator function.



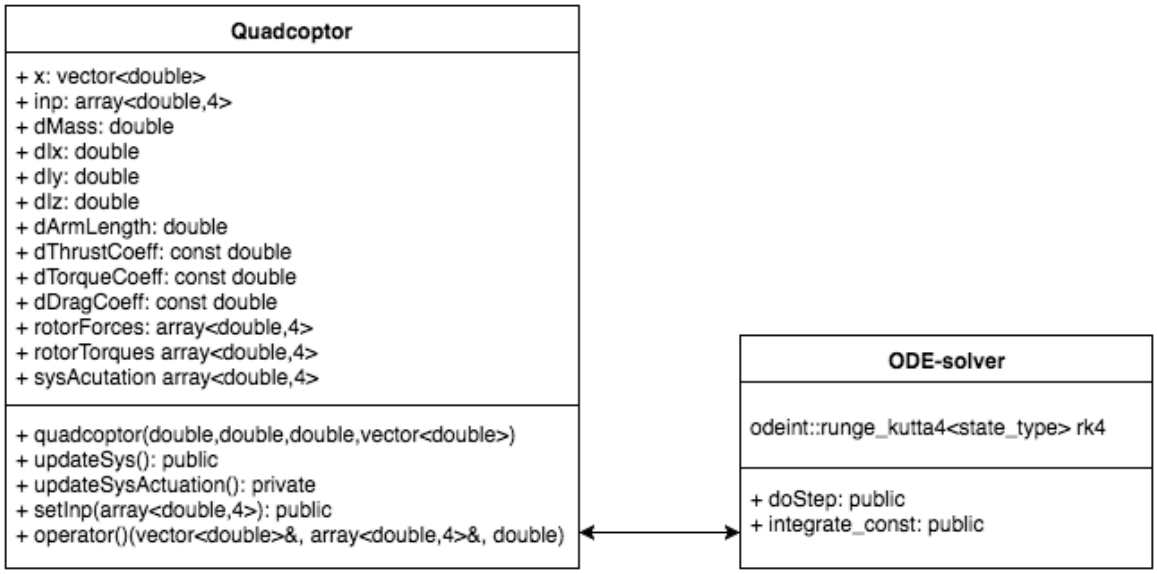
class simulatorWindow

This is the main interface between the other classes, This is where the inputs are taken from the keyboard and given to the MotorSoftware which processes and returns the 4 motor speeds. These motor speeds are further input to the Quadcoptor and the next state is returned. These states are logged in a file over each run of the update function. The class is a sub-class of MotorSoftware, QOpenGLWindow and QOpenGLFunctions, further it extensively interacts with the Quadcoptor, Odeint, and QMediaPlayer classes.



class Quadcoptor

The overloaded operator () calculates the derivatives of the 12 dimensional state vector. The setInp function is used to update the rotor velocities which are obtained from the MotorSoftware class. Further, the updateSys function updates the values of the state and inputs at each time step. The dMass, dIx, dIy, dIz are initialized at the time of construction and are not changed. This class is accessed by an ODE solver object which is used to calculate the state at the next time step, which is called by using integrate_const function.



Process View

"Not Required"

Development View

"Not Required"

Physical View

"Not Required"

Use Case View

This software has been developed in **QT** environment which can be installed on both macOS and Windows. The installation requirements and instructions can be found on the official QT website, *http://doc.qt.io/qt-4.8/installation.html*. Using QT makes it possible to develop a platform independent application, which can be compiled and executed across multiple operating systems with minimal changes.

For the development of this software's code we have used macOS and the QT configuration file (***Final_Project.pro***) is written in a way which is compatible with **macOS**. Most part of the configuration file is also OS independent and contains information about the required header and .cpp files. Only the following four lines are OS dependent and needs to be modified while choosing an operating system other than macOS.

QMAKE_CFLAGS += -std=c++11 -stdlib=libc++ -mmacosx-version-min=10.8
QMAKE_CXXFLAGS += -std=c++11 -stdlib=libc++ -mmacosx-version-min=10.8
LIBS += -L"/usr/local/Cellar/boost/1.62.0/lib"
INCLUDEPATH += "/usr/local/Cellar/boost/1.62.0/include"

These four lines informs QT about the locations of boost libraries. Since the software uses boost class methods we must provide this information otherwise there will be compilation error. Note: Make sure your system has boost already installed.

To change the configuration file for windows operating system please refer *http://stackoverflow.com/questions/12113679/configure-qt-creator-to-use-boost-on-windows*

Instructions to run the executable (project deployment on mac)

1. unzip the folder and run the file *Final_Project.app*
2. (In case there is an error due to security settings) Go to system-preferences > security & privacy > open this time.
3. The project will run.

Instruction to build and run the software (.pro file)

1. Mac OS X
    a. Install latest Xcode version from mac app store. It's free.
    b. Install boost. A simple way to install boost on mac is via homebrew. If you want to use homebrew make sure you have homebrew already installed on your system before using *brew install boost* command.
    c. Install QT. The installer can be found here, *https://www.qt.io*

d. Once QT is installed properly, launch it. If you don't get any errors, no worries, move to next step. But, if you get the following error, "*Project ERROR: Xcode not setup properly. You may need to confirm the license agreement by running /usr/bin/xcodebuild"*. Note that this has nothing to do with your Xcode. QT is simply not able to interact with Xcode properly. The following two steps does the trick:
   i. Open the file:

   Qt_install_folder/5.7/clang_64/mkspecs/features/mac/default_pre.prf

   ii. Replace

   isEmpty($$list($$system("/usr/bin/xcrun -find xcrun 2>/dev/null")))

   with

   isEmpty($$list($$system("/usr/bin/xcrun -find xcodebuild 2>/dev/null")))

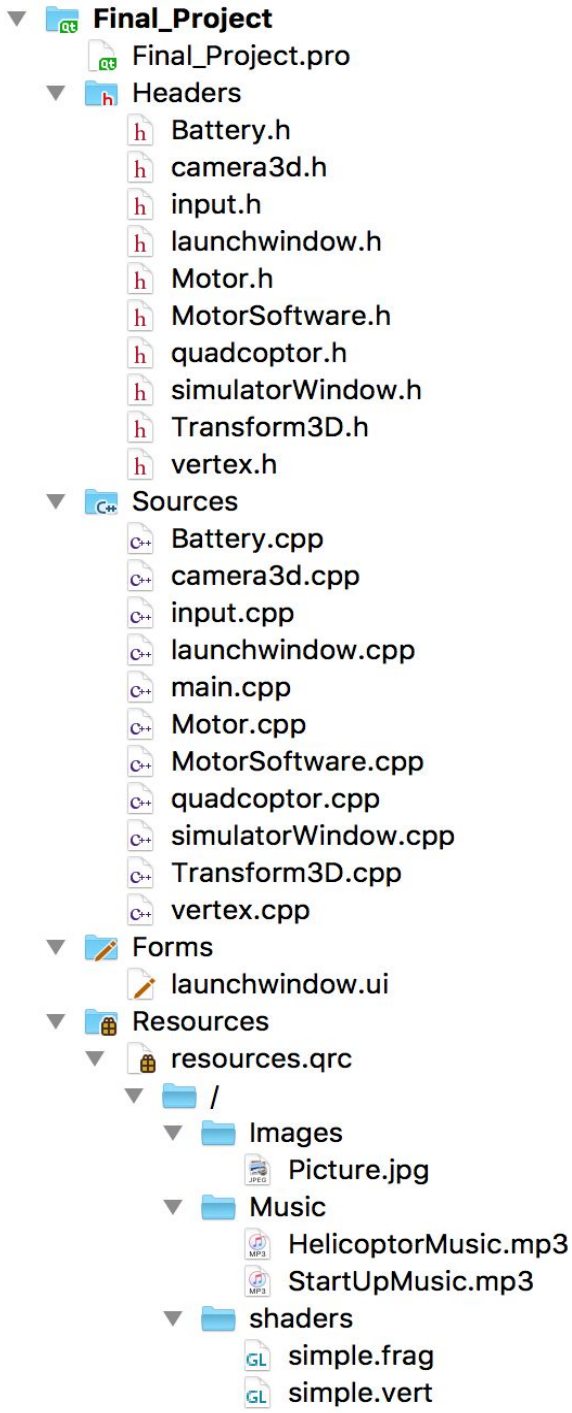
e. Now, you have everything required to build the software. Open the software in QT by double clicking the QT configuration file of the software (***Final_Project.pro***). Next, click on the run button, it will first build and then run the software.

2. Windows OS
   a. Install boost. MinGW can be used to install boost on windows.
   b. Install QT. The installer can be found here, *https://www.qt.io*
   c. Once, both boost and QT are installed properly, click the QT configuration file of the software (***Final_Project.pro***) to open it in QT environment. The last four lines of the configuration file needs to be changed now as explained above in this section. Depending upon how you install boost, the path at which its libraries are stored in the hard disk may vary. Find the path for these libraries and provide it in the configuration file.
   d. Click on the run button, it will first build and then run the software.

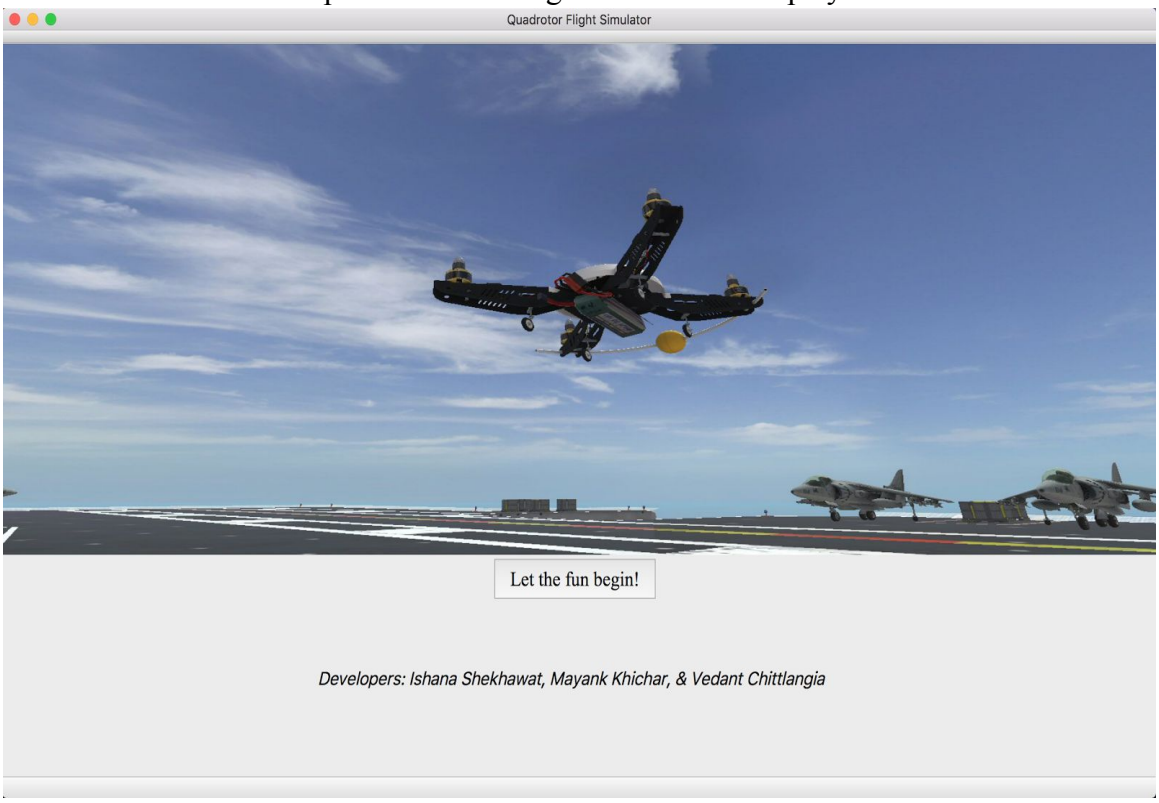
Note: The software uses resource files whose relative path is defined in the code. For this reason the three folders, named *'Images'*, *'Music'*, and *'shaders',* must be placed inside the folder in which all the header files, .cpp files, and the .pro file is placed.

Step by step explanation of an example use case with screenshots

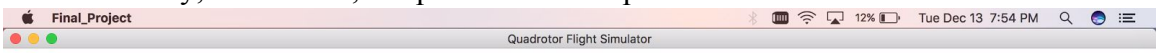
1. Open the software in QT by double clicking the ***Final_Project.pro*** file. Make sure you have all the header and .cpp files along with *'resources.qrc'*, *'launchwindow.ui'*, and three folders, named *'Images'*, *'Music'*, and *'shaders'*. It should look like this



2. Build and execute the code by clicking on the *'run'* button on the QT screen. A successful build opens the following GUI. It will also play audio instructions.



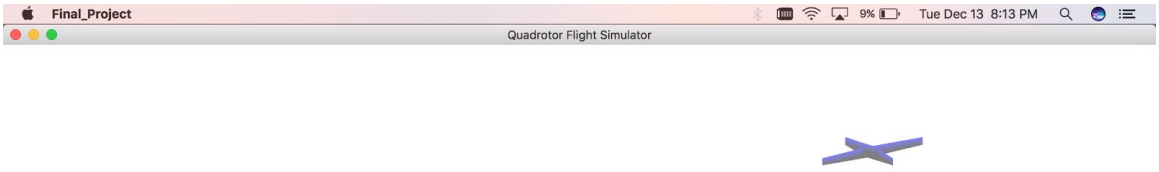
3. Click on *'Let the fun begin!'* button. It will close the GUI and open the quadrotor simulator. The simulator screen appears with quadrotor at rest (motors turned off). Initially, we assume, it's placed on some platform before motors are turned on.



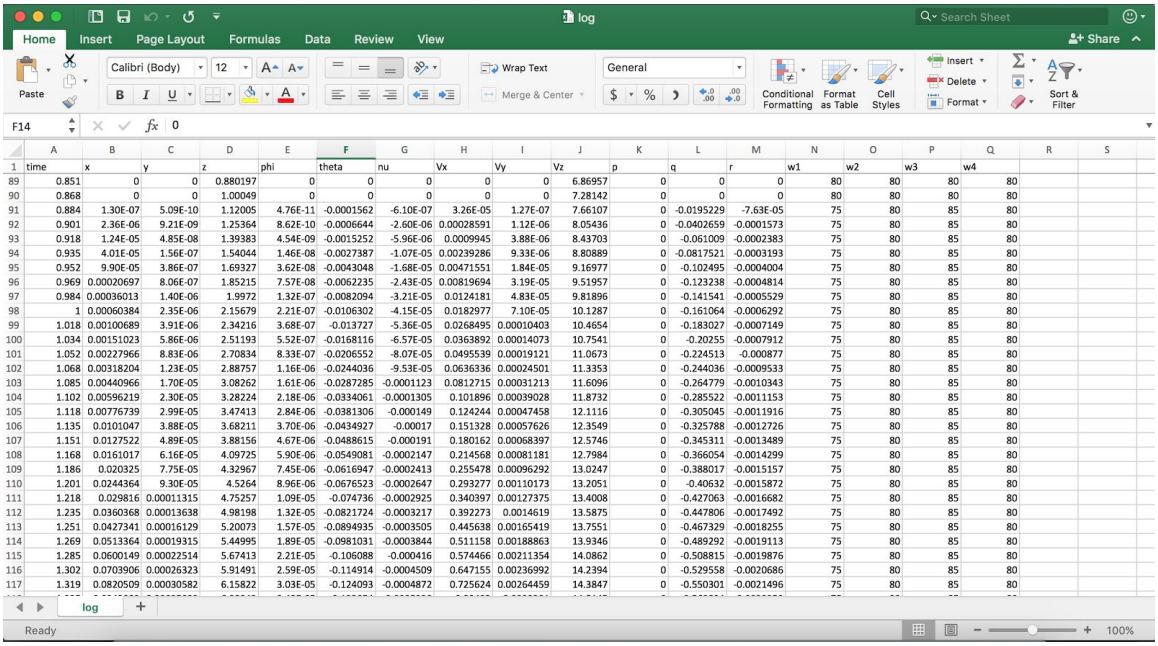
4. To start the simulator, press a key between 1-5 on keyboard. It will turn on the motors and from this moment the quadrotor dynamics module will decide its motion in 3 dimensional space. To give this quadrotor a direction to move press arrows keys. Pressing the arrow keys have the following effect:
    a. Press left key: quadrotor goes left on the screen
    b. Press right key: quadrotor goes right on the screen
    c. Press up key: quadrotor seems to go inside the screen
    d. Press down key: quadrotor seems to come out of the screen

Note: Since laptop/PC screen is two dimensional. Sometimes it is difficult to visualize the in and out motion of the quadrotor on the screen.

The following screenshot shows the virtual quadrotor which is following the trajectory that an actual quadrotor would have followed given the same inputs are provided to both of them.



5. The output from quadrotor dynamics module is stored in a .csv file for further analysis of the trajectory. The following screenshot depicts the data stored for a sample run.



log

Search Sheet

Home Insert Page Layout Formulas Data Review View Share

Paste Calibri (Body) 12 Wrap Text General Conditional Formatting Format as Table Cell Styles Insert Delete Format Sort & Filter

F14 0

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	time	x	y	z	phi	theta	nu	Vx	Vy	Vz	p	q	r	w1	w2	w3	w4		
89	0.851	0	0	0.880197	0	0	0	0	0	6.86957	0	0	0	80	80	80	80		
90	0.868	0	0	1.00049	0	0	0	0	0	7.28142	0	0	0	80	80	80	80		
91	0.884	1.30E-07	5.09E-10	1.12005	4.76E-11	-0.0001562	-6.10E-07	3.26E-05	1.27E-07	7.66107	0	-0.0195229	-7.63E-05	75	80	85	80		
92	0.901	2.36E-06	9.21E-09	1.25364	8.62E-10	-0.0006644	-2.60E-06	0.00028591	1.12E-06	8.05436	0	-0.0402659	-0.0001573	75	80	85	80		
93	0.918	1.24E-05	4.85E-08	1.39383	4.54E-09	-0.0015252	-5.96E-06	0.0009945	3.88E-06	8.43703	0	-0.061009	-0.0002383	75	80	85	80		
94	0.935	4.01E-05	1.56E-07	1.54044	1.46E-08	-0.0027387	-1.07E-05	0.00239286	9.33E-06	8.80889	0	-0.0817521	-0.0003193	75	80	85	80		
95	0.952	9.90E-05	3.86E-07	1.69327	3.62E-08	-0.0043048	-1.68E-05	0.00471551	1.84E-05	9.16977	0	-0.102495	-0.0004004	75	80	85	80		
96	0.969	0.00020697	8.06E-07	1.85215	7.57E-08	-0.0062235	-2.43E-05	0.00819694	3.19E-05	9.51957	0	-0.123238	-0.0004814	75	80	85	80		
97	0.984	0.00036013	1.40E-06	1.9972	1.32E-07	-0.0082094	-3.21E-05	0.0124181	4.83E-05	9.81896	0	-0.141541	-0.0005529	75	80	85	80		
98	1	0.00060384	2.35E-06	2.15679	2.21E-07	-0.0106302	-4.15E-05	0.0182977	7.10E-05	10.1287	0	-0.161064	-0.0006292	75	80	85	80		
99	1.018	0.00100689	3.91E-06	2.34216	3.68E-07	-0.013727	-5.36E-05	0.0268495	0.00010403	10.4654	0	-0.183027	-0.0007149	75	80	85	80		
100	1.034	0.00151023	5.86E-06	2.51193	5.52E-07	-0.0168116	-6.57E-05	0.0363892	0.00014073	10.7541	0	-0.20255	-0.0007912	75	80	85	80		
101	1.052	0.00227966	8.83E-06	2.70834	8.33E-07	-0.0206552	-8.07E-05	0.0495539	0.00019121	11.0673	0	-0.224513	-0.000877	75	80	85	80		
102	1.068	0.00318204	1.23E-05	2.88757	1.16E-06	-0.0244036	-9.53E-05	0.0636336	0.00024501	11.3353	0	-0.244036	-0.0009533	75	80	85	80		
103	1.085	0.00440966	1.70E-05	3.08262	1.61E-06	-0.0287285	-0.0001123	0.0812715	0.00031213	11.6096	0	-0.264779	-0.0010343	75	80	85	80		
104	1.102	0.00596219	2.30E-05	3.28224	2.18E-06	-0.0334061	-0.0001305	0.101896	0.00039028	11.8732	0	-0.285522	-0.0011153	75	80	85	80		
105	1.118	0.00776739	2.99E-05	3.47413	2.84E-06	-0.0381306	-0.000149	0.124244	0.00047458	12.1116	0	-0.305045	-0.0011916	75	80	85	80		
106	1.135	0.0101047	3.88E-05	3.68211	3.70E-06	-0.0434927	-0.00017	0.151328	0.00057626	12.3549	0	-0.325788	-0.0012726	75	80	85	80		
107	1.151	0.0127522	4.89E-05	3.88156	4.67E-06	-0.0488615	-0.000191	0.180162	0.00068397	12.5746	0	-0.345311	-0.0013489	75	80	85	80		
108	1.168	0.0161017	6.16E-05	4.09725	5.90E-06	-0.0549081	-0.0002147	0.214568	0.00081181	12.7984	0	-0.366054	-0.0014299	75	80	85	80		
109	1.186	0.020325	7.75E-05	4.32967	7.45E-06	-0.0616947	-0.0002413	0.255478	0.00096292	13.0247	0	-0.388017	-0.0015157	75	80	85	80		
110	1.201	0.0244364	9.30E-05	4.5264	8.96E-06	-0.0676523	-0.0002647	0.293277	0.00110173	13.2051	0	-0.40632	-0.0015872	75	80	85	80		
111	1.218	0.029816	0.00011315	4.75257	1.09E-05	-0.074736	-0.0002925	0.340397	0.00127375	13.4008	0	-0.427063	-0.0016682	75	80	85	80		
112	1.235	0.0360368	0.00013638	4.98198	1.32E-05	-0.0821724	-0.0003217	0.392273	0.0014619	13.5875	0	-0.447806	-0.0017492	75	80	85	80		
113	1.251	0.0427341	0.00016129	5.20073	1.57E-05	-0.0894935	-0.0003505	0.445638	0.00165419	13.7551	0	-0.467329	-0.0018255	75	80	85	80		
114	1.269	0.0513364	0.00019315	5.44995	1.89E-05	-0.0981031	-0.0003844	0.511158	0.00188863	13.9346	0	-0.489292	-0.0019113	75	80	85	80		
115	1.285	0.0600149	0.00022514	5.67413	2.21E-05	-0.106088	-0.000416	0.574466	0.00211354	14.0862	0	-0.508815	-0.0019876	75	80	85	80		
116	1.302	0.0703906	0.00026323	5.91491	2.59E-05	-0.114914	-0.0004509	0.647155	0.00236992	14.2394	0	-0.529558	-0.0020686	75	80	85	80		
117	1.319	0.0820509	0.00030582	6.15822	3.03E-05	-0.124093	-0.0004872	0.725624	0.00264459	14.3847	0	-0.550301	-0.0021496	75	80	85	80		

log

Ready 100%