Quadcopter - Modelling and Design

Group - 19

Bittu Kumar Ray - S20170010027

Rahul Prasad - S20170010118

Shubh Vanvat - S20170010150

Siddhant Jain - S20170010151

Vijendra Kumar Saini - S20170010177

Vineet Sharma - S20170010179

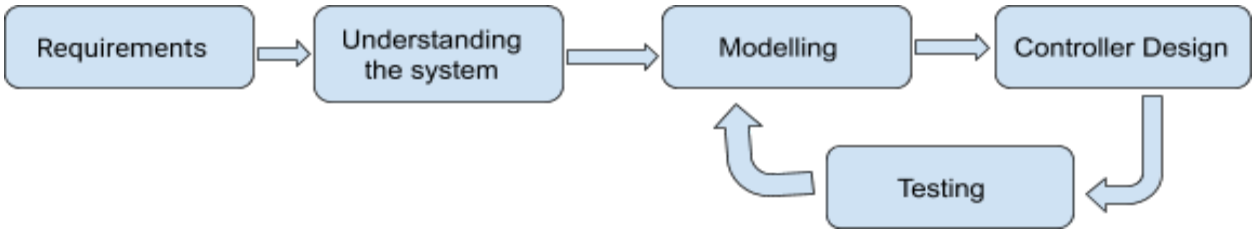
Introduction

A ***quadcopter*****,** also called a quadrotor helicopter or quadrotor, is a multirotor helicopter that is lifted and propelled by four rotors.

Quadcopters are classified as **rotorcraf**t, as opposed to fixed-wing aircraft, because their lift is generated by a set of rotors. Like quadcopter (4wings), hexacopter (6 wings) and so on.

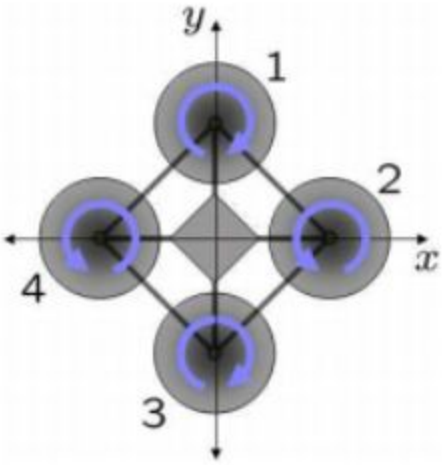
As well as any other machine that uses a flying wing rather than fixed wing to generate lift. Even though these are all rotary wing vehicles, they all have different dynamics, and therefore different control strategies.

Work-flow diagram:-

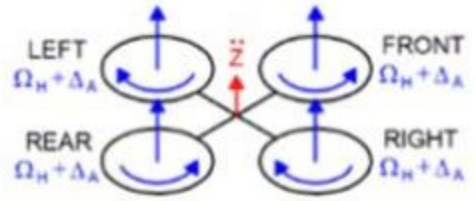


Equation Modelling

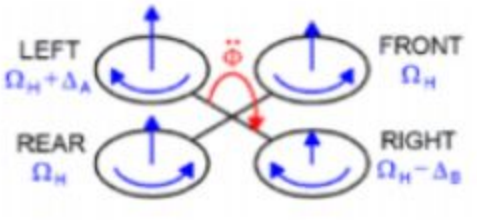
- The altitude and position of the quadrotor can be controlled to desired values by changing the speeds of the motors
- The forces and moments can be performed on the quadrotor are the thrust caused by rotors rotation, the pitching moment and rolling moment caused by the difference of four rotors thrust, the gravity, the gyroscopic effect, and the yawing moment
- Propellers can be divided into two groups:-
    - Front and rear propellers(numbered 2 and 4)
    - Right and left propellers(numbered 1 and 3)



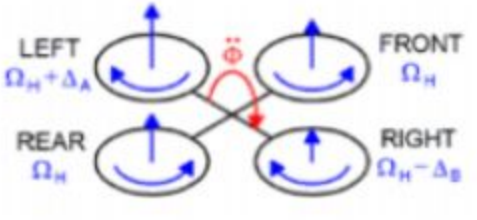
- Motions which any copter can perform include forward and backward movements, lateral movement, vertical motion, roll motion, and pitch and yaw motions.
- The control for six degrees of freedom motions can be implemented by adjusting the rotational speeds of different motors.
- Depending on the speed rotation of each propeller it is possible to identify the four basic movements of the quadrotor :-
    - Thrust



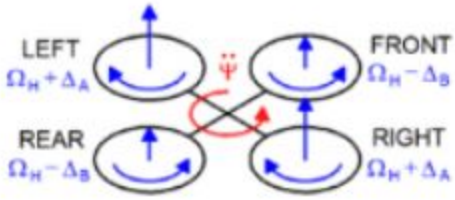
    - Pitch



- Roll



- Yaw



- Euler angles represent a sequence of three elemental rotations, i.e. rotations about the axes of a coordinate system, since any orientation can be achieved by composing three elemental rotations.

- Following rotational matrices describe the orientation of quadcopter:-

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi) & c(\phi) \end{bmatrix}$$

  -

$$\mathbf{R}_y(\theta) = \begin{bmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{bmatrix}$$

$$\mathbf{R}_z(\psi) = \begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  - Therefore, the rotational matrix can be described as:-

$$\mathbf{R}_{zyx}(\phi, \theta, \psi) = \mathbf{R}_z(\psi) \cdot \mathbf{R}_y(\theta) \cdot \mathbf{R}_x(\phi)$$

$$= \begin{bmatrix} c(\theta)c(\psi) & s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi) & c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) \\ c(\theta)s(\psi) & s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{bmatrix}$$

- Let

$$\begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}^T$$

Be the vector representing linear and angular position of the quadrotor in the Earth frame and let

$$\begin{bmatrix} u & v & w & p & q & r \end{bmatrix}^T$$

Be the vector representing linear and angular velocities in the body frame

Then the two reference frame are linked by the equations:-

$$\mathbf{v} = \mathbf{R} \cdot \mathbf{v_B},$$

$$\boldsymbol{\omega} = \mathbf{T} \cdot \boldsymbol{\omega_B},$$

where $\mathbf{v} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T \in \mathbb{R}^3$, $\boldsymbol{\omega} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \in \mathbb{R}^3$, $\mathbf{v_B} = \begin{bmatrix} u & v & w \end{bmatrix}^T \in \mathbb{R}^3$, $\boldsymbol{\omega_B} = \begin{bmatrix} p & q & r \end{bmatrix}^T \in \mathbb{R}^3$, and $\mathbf{T}$ is a matrix for angular transformations [27]

$$\mathbf{T} = \begin{bmatrix} 1 & s(\phi)t(\theta) & c(\phi)t(\theta) \\ 0 & c(\phi) & -s(\phi) \\ 0 & \frac{s(\phi)}{c(\theta)} & \frac{c(\phi)}{c(\theta)} \end{bmatrix}, \tag{2.7}$$

So, the kinematic model of the quadrotor is:-

$$\begin{cases} \dot{x} = w[s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta)] - v[c(\phi)s(\psi) - c(\psi)s(\phi)s(\theta)] + u[c(\psi)c(\theta)] \\ \dot{y} = v[c(\phi)c(\psi) + s(\phi)s(\psi)s(\theta)] - w[c(\psi)s(\phi) - c(\phi)s(\psi)s(\theta)] + u[c(\theta)s(\psi)] \\ \dot{z} = w[c(\phi)c(\theta)] - u[s(\theta)] + v[c(\theta)s(\phi)] \\ \dot{\phi} = p + r[c(\phi)t(\theta)] + q[s(\phi)t(\theta)] \\ \dot{\theta} = q[c(\phi)] - r[s(\phi)] \\ \dot{\psi} = r\frac{c(\phi)}{c(\theta)} + q\frac{s(\phi)}{c(\theta)} \end{cases}$$

Total force acting on quadrotor can be described as:-

$$m(\boldsymbol{\omega_B} \wedge \mathbf{v_B} + \dot{\mathbf{v}}_\mathbf{B}) = \mathbf{f_B},$$

Where **m** is mass and :-

$$\mathbf{f_B} = \begin{bmatrix} f_x & f_y & f_z \end{bmatrix}^T$$

Total torque applied to the quadrotor can be defined by:-

$$\mathbf{I} \cdot \dot{\boldsymbol{\omega}}_\mathbf{B} + \boldsymbol{\omega}_B \wedge (\mathbf{I} \cdot \boldsymbol{\omega}_B) = \mathbf{m_B},$$

Where :-

$$\mathbf{m_B} = \begin{bmatrix} m_x & m_y & m_z \end{bmatrix}^T$$

Is the total torque and **I** is the diagonal inertial matrix:-

$$\mathbf{I} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}$$

So, the dynamic model of the quadrotor in the body frame is :-

$$\begin{cases} f_x = m(\dot{u} + qw - rv) \\ f_y = m(\dot{v} - pw + ru) \\ f_z = m(\dot{w} + pv - qu) \\ m_x = \dot{p}I_x - qrI_y + qrI_z \\ m_y = \dot{q}I_y + prI_x - prI_z \\ m_z = \dot{r}I_z - pqI_x + pqI_y \end{cases}$$

The external forces on quadrotor in the body frame are defined by:-

$$\mathbf{f_B} = mg\mathbf{R}^T \cdot \hat{\mathbf{e}}_\mathbf{z} - f_t\hat{\mathbf{e}}_\mathbf{3} + \mathbf{f_w},$$

Where:-

$\hat{\mathbf{e}}_\mathbf{z}$ is the unit vector in the inertial z-axis

$\hat{\mathbf{e}}_3$ is the unit vector in the body z-axis

$\mathbf{f_w} = \begin{bmatrix} f_{wx} & f_{wy} & f_{wz} \end{bmatrix}^T$ are the forces by wind

The external moments in the body frame are given by:-

$$\mathbf{m_B} = \boldsymbol{\tau_B} - \mathbf{g_a} + \boldsymbol{\tau_w},$$

Where:-

$\mathbf{g_a}$ represents gyroscopic moments caused by the combined rotation of the four rotors and the vehicle body

$\boldsymbol{\tau_B} = \begin{bmatrix} \tau_x & \tau_y & \tau_z \end{bmatrix}^T$ are the control torques generated by differences in the rotor speeds

$\boldsymbol{\tau_w} = \begin{bmatrix} \tau_{wx} & \tau_{wy} & \tau_{wz} \end{bmatrix}^T$ are the torques produced by wind

So, the complete dynamic model of the quadrotor in the body frame is obtained by substituting the force expression:-

$$\begin{cases} -mg[s(\theta)] + f_{wx} = m(\dot{u} + qw - rv) \\ mg[c(\theta)s(\phi)] + f_{wy} = m(\dot{v} - pw + ru) \\ mg[c(\theta)c(\phi)] + f_{wz} - f_t = m(\dot{w} + pv - qu) \\ \tau_x + \tau_{wx} = \dot{p}I_x - qrI_y + qrI_z \\ \tau_y + \tau_{wy} = \dot{q}I_y + prI_x - prI_z \\ \tau_z + \tau_{wz} = \dot{r}I_z - pqI_x + pqI_y \end{cases}$$

Controller Design

Now in order to set up our control problem we need to spend a little time to understand hardware as follows:-

Control Requirements:-

1. **Sensors**
    a. ***Ultrasound*** - It used to measure vertical distances. It sends a high-frequency sound pulse and measures how long it takes that pulse to bounce off the floor and return back to the sensor.
    b. ***Camera*** - It used to take images and use an image processing technique called optical flow to determine how objects are moving between one frame and the next.
    c. ***Pressure*** - As the drone climbs in altitude the air pressure drops slightly and we can use this slight change in pressure to estimate how the altitude of the drone is changing. Like is it going up or down?
    d. ***IMU*** *(Inertial Measurement Unit)* - This is made up of a three axis accelerometer which measures linear acceleration and 3-axis gyroscope which measures angular rate.
    From the IMU and our knowledge of acceleration due to gravity, we can estimate the drone's altitude relative to gravity and how fast it's rotating.

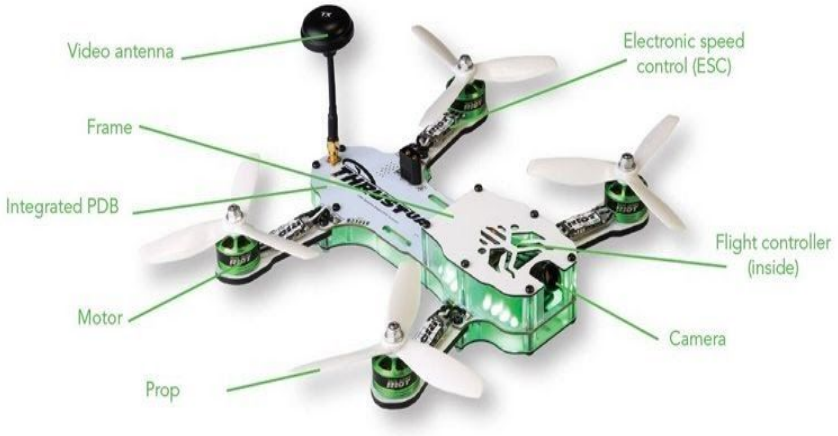
So, We can use Ultrasound and pressure sensors to determine ***altitude*** and the IMU and the Camera to determine ***rotational*** and ***translational*** motion**.**

2. **Actuators**
    a. ***Motors*** - it used to recognize configuration and spin direction of rotors. All these four motors are laid out in an 'X' configuration as opposed to a '+' configuration. The only difference is which motors we send commands to when pitching and rolling the drone. In general the underlying concepts are the same for both.

- Opposing motors spin in the same direction as each other, but opposite direction as the other pair.
- **Thrust, Roll, Yaw and Pitch can be commended independently**. That means we can command one motion without affecting the others.
- In reality complex fluid dynamics around the drone means that all motion is coupled together in some way.
- This is an underactuated system - 4 motors with 6 degree of freedom (DOF)
    - Translational Directions -
        - Up/Down

        - Left/Right
        - Forward/Backward
    - Rotational Directions -
        - Roll
        - Pitch
        - Yaw
- With the configuration of 4 motors we can hover by accelerating each other motor until they each produce a force one fourth that of gravity.
- And as long as we have two counter rotating motors the torque from propellers will balance out and the drone will not spin.

Hardware components -

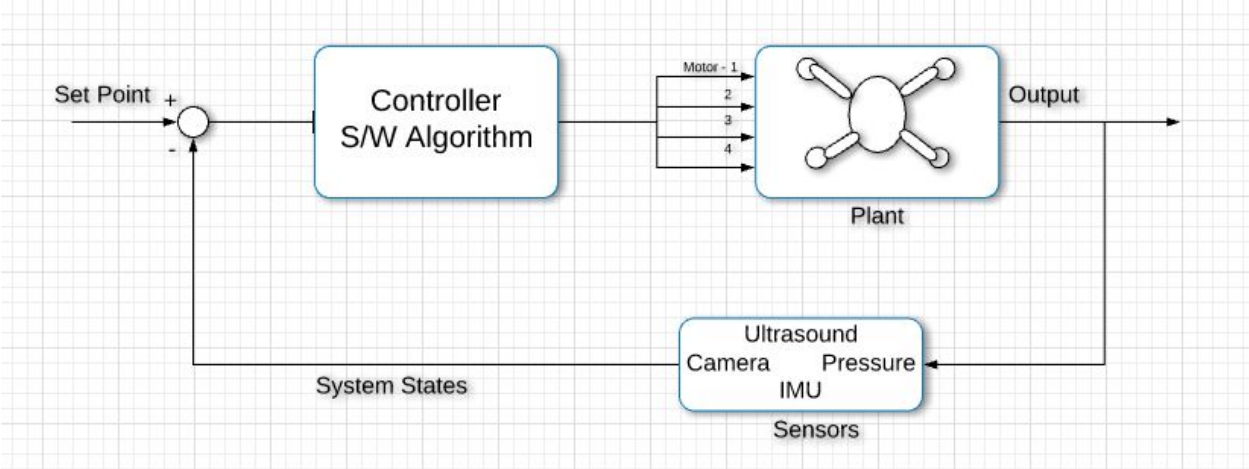


Motor Mixing Algorithm (MMA):-

This algorithm helps us to convert between the intuitive Roll, Pitch, Yaw, and Thrust, and the less intuitive motor and speeds.

$Motor_{frontRight}$	$Thrust_{cmd} + Yaw_{cmd} + Pitch_{cmd} + Roll_{cmd}$
$Motor_{frontLeft}$	$Thrust_{cmd} - Yaw_{cmd} + Pitch_{cmd} - Roll_{cmd}$
$Motor_{backRight}$	$Thrust_{cmd} - Yaw_{cmd} - Pitch_{cmd} + Roll_{cmd}$
$Motor_{backLeft}$	$Thrust_{cmd} + Yaw_{cmd} - Pitch_{cmd} - Roll_{cmd}$

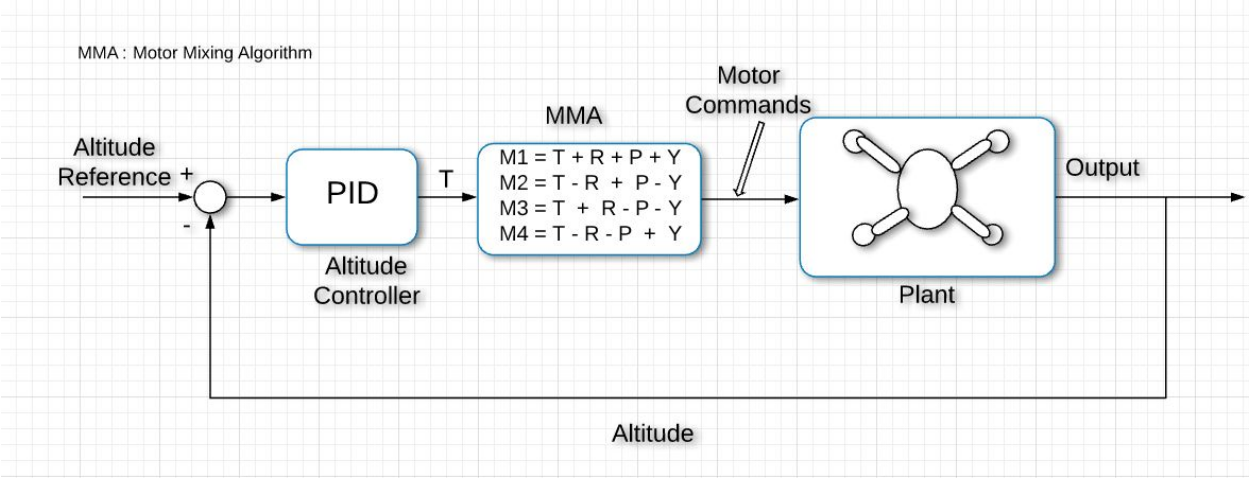
Control Problem:-



- We know that manipulating the 4 motors in specific ways will allow us to control the drone in 3D space.
- And we have a set of sensors that we can use to estimate the state of the system.
- And we have an onboard processor that can run on control logic.

The control system is done in Simulink, where we have built and simulated the quadcopter model, tuned the controller and tested in a closed loop simulation, and then finally automatically generated flight code that we loaded into the onboard microcontroller on the parrot mini-drone.

Control System Architecture



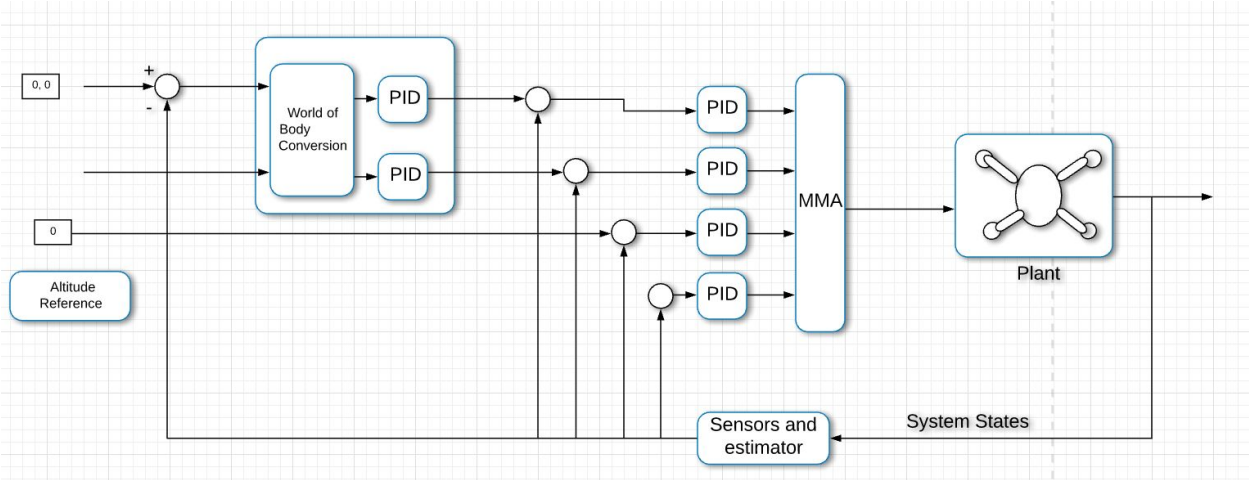
So, basically this is a simple **altitude controller** (feedback) that will hover our mini drone.
But, that's not the case because there are disturbances like wind gusts that will induce a little roll or pitch into the system and when that happens the thrust will not only adjust altitude, but also create some horizontal motion and the drone will start to move away from us.

- We clearly need a better controller architecture and we should start by trying to maintain level flight by controlling roll and pitch angles to zero degrees.
- If we can keep the mini drone level then thrust once again will only impact altitude and the drone won't wander away.
- As we know that we can command thrust, roll, pitch, and yaw independently. And knowing this we can create three more feedback controllers. One for roll, pitch, and yaw.
- So, We can add three more (PID) feedback controllers from sensors to MMA.It is a better hover control control system, but still it is not perfect.

- In this case roll and pitch controller angle may need to be non-zero in order to hover. To do so, We can feedback the mini-drones measuring XY position and compare it to the reference to get the position error.

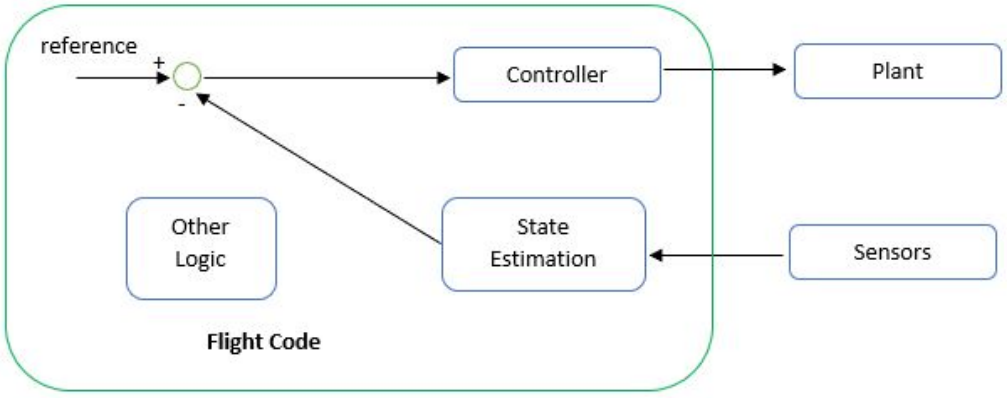
Initially we can assume the reference position is zero-zero. This way our controller will cause the drone to hover right above the take-off point.

Position controller takes position error as inputs and outputs roll and pitch angles. And these are the reference angles that roll and pitch angles are trying to follow. So, instead of us as designer picking roll and pitch angles we are letting the position controller create them for us.
So, following is our final and complete control system architecture in which drone can hover:-
All 6 PIDs (2 [roll, pitch] for inner loop controller/position controller + 4 [roll, pitch, yaw, thrust] for outer loop controller)



Here, **MMA** is a Motor Mixing Algorithm.

Designing the control logic (Flight code)



Other logic:

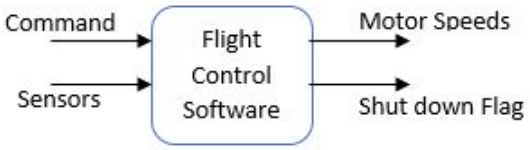
- Sensor interface
- Battery Management
- Bluetooth Interface
- LEDs
- Motor Interface
- Memory management, etc

Higher level workflow
This Flight code will be created in Simulink using GUI components → Build it to generate C code → Compile to get the Binary → put this to the drone

Using the Simulink Support package for Parrot Minidrone and we are keeping other logic components intact and customizing the Flight Control System(FCS).

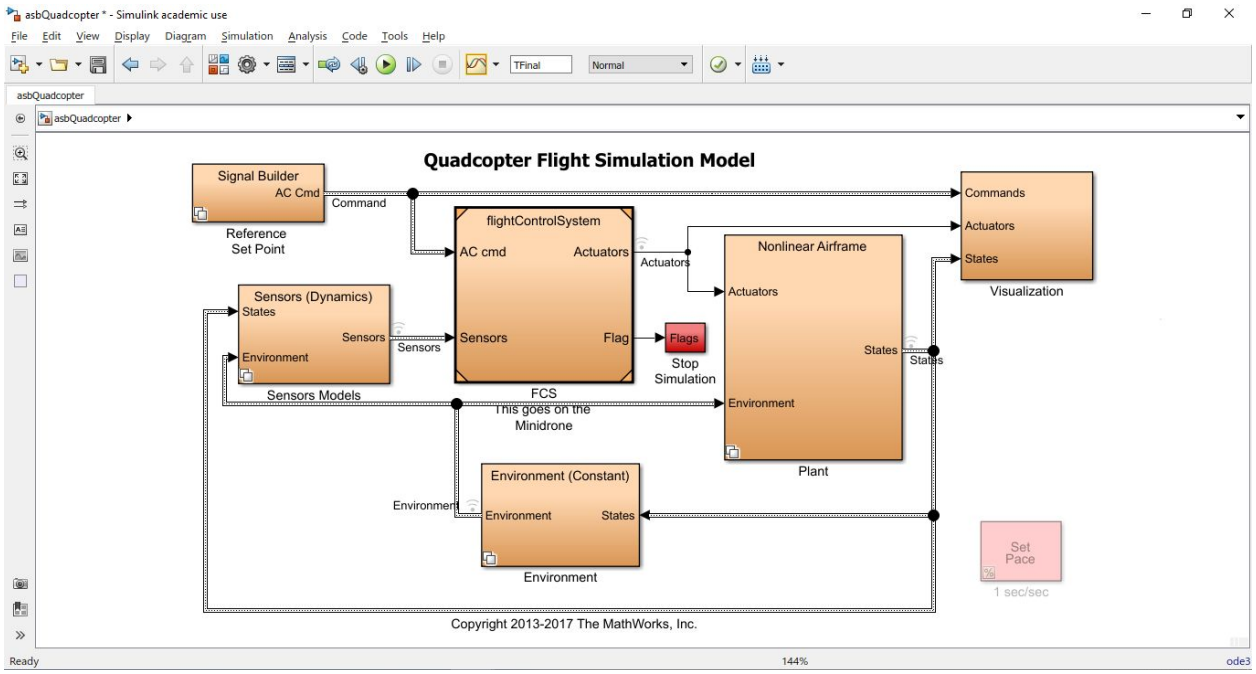
FCS includes:

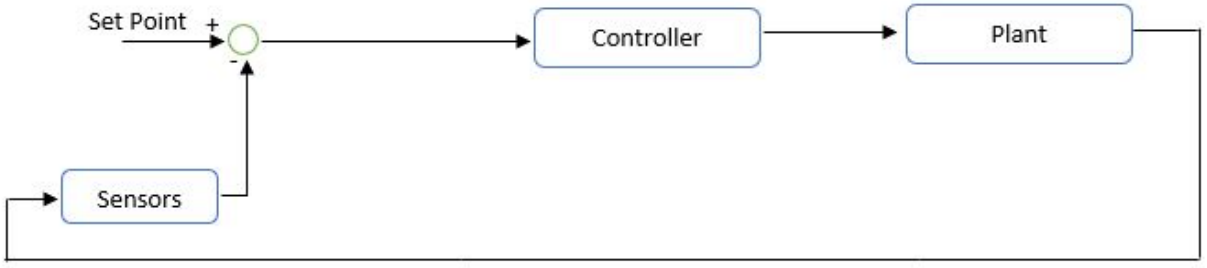
- Controller
- State Estimator
- Data Logging - Store data as .mat file locally on drone
- Fault Protection - Setting constraints on position and altitude



Simulink

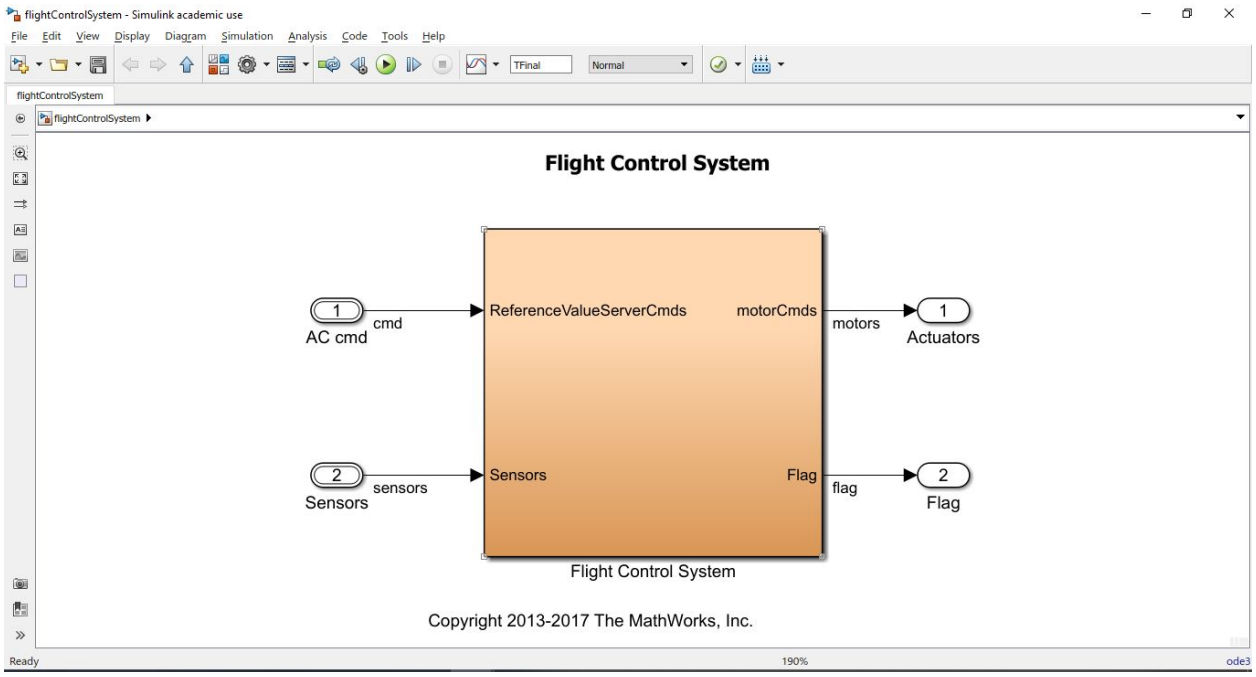
The Overall Model



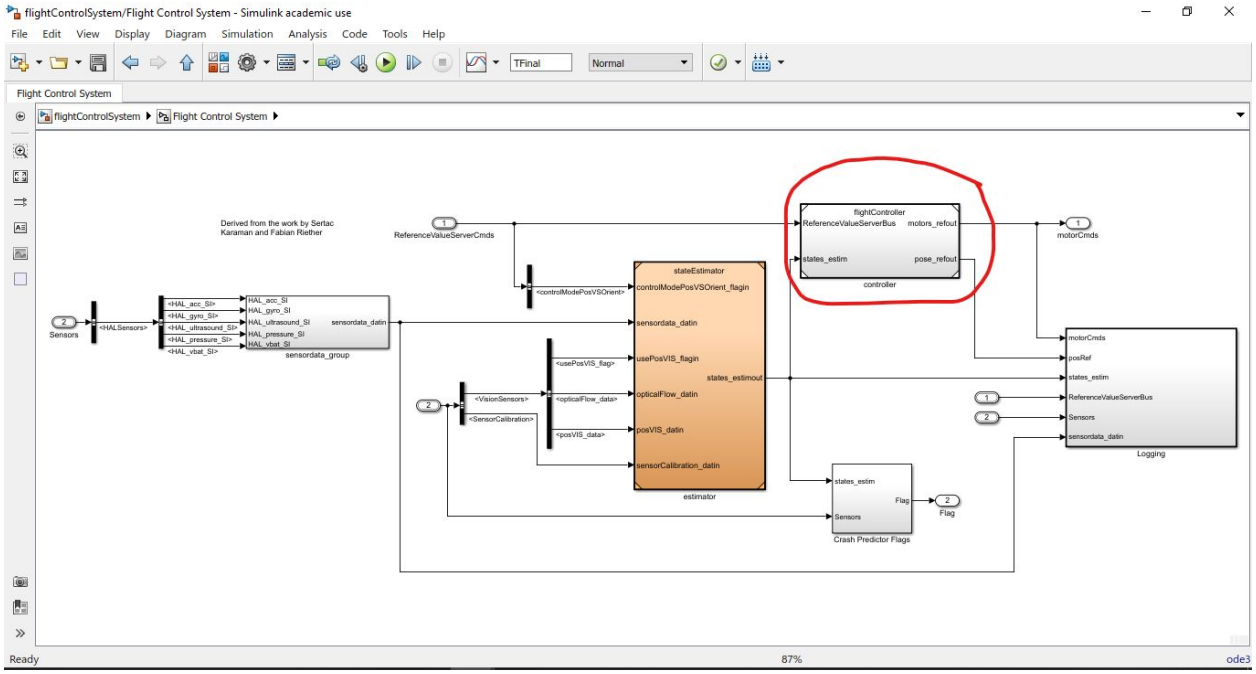


The feedback loop in this

In FCS

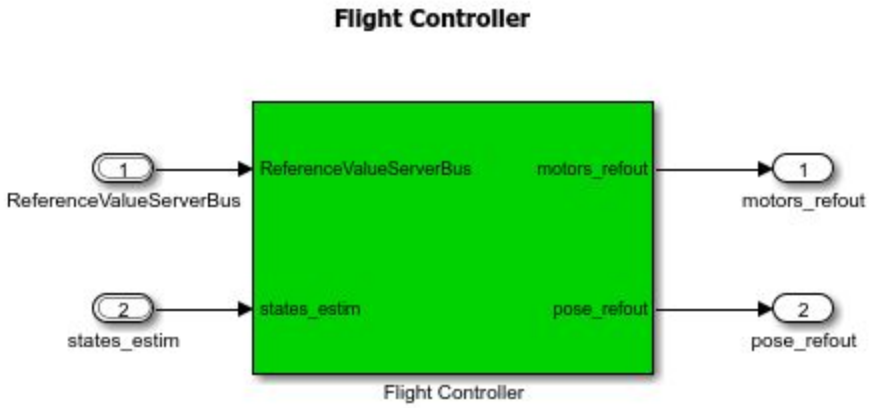


1. Controller

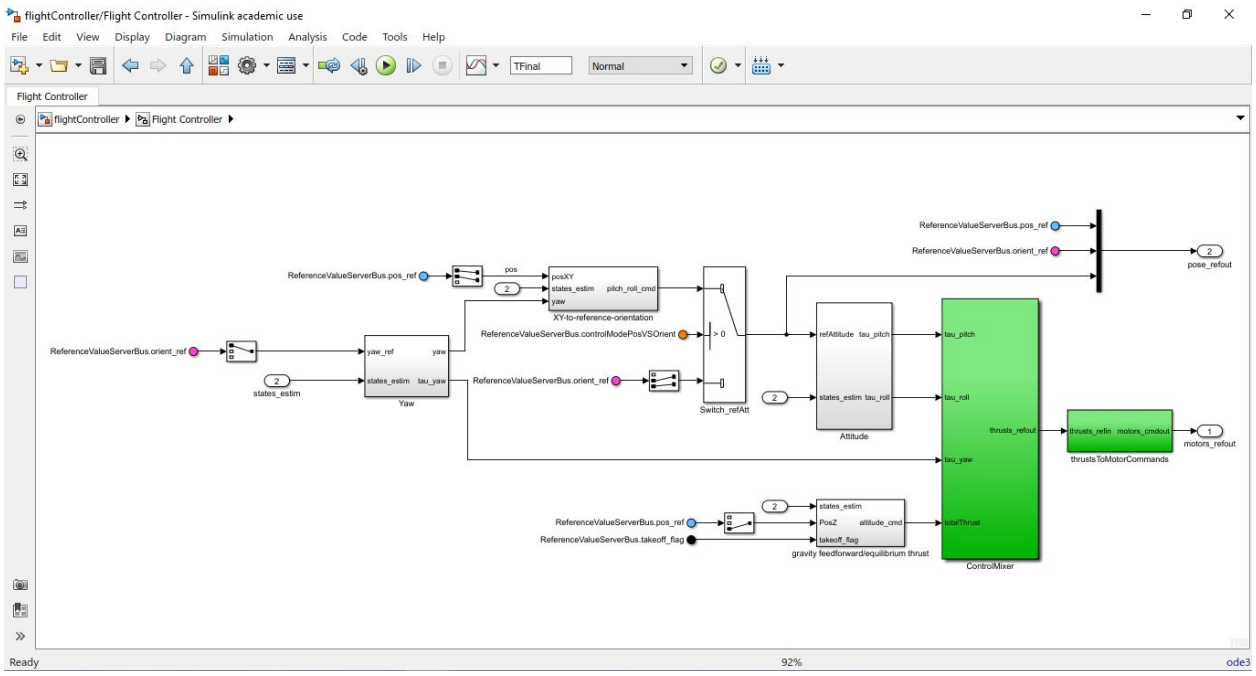


This control subsystem takes the reference signal and compares the estimated states to get the error signals. The errors are then fed into several PID controllers to generate the required Force and Torque commands.

In flight controller subsystem block

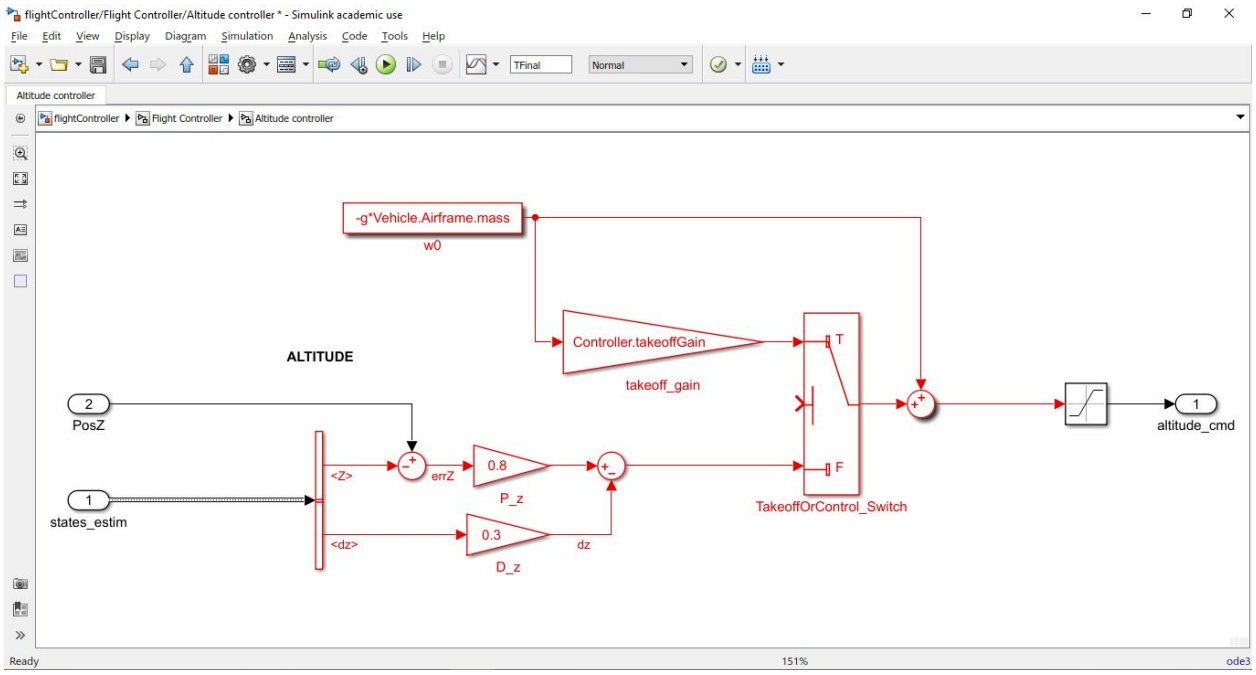


Expanded form



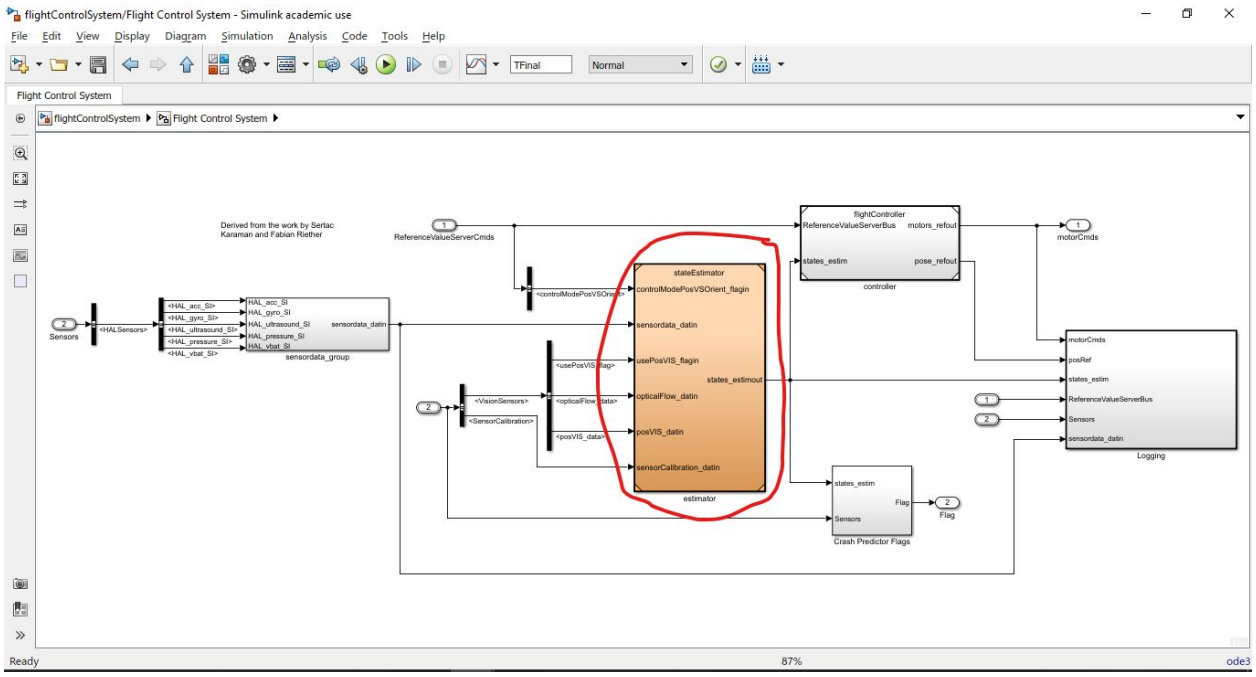
The outputs of the PID controllers are Force and Torque commands.
These commands are fed into the *Motor Mixing algorithm* in the ControlMixer block**.**
The *Motor Mixing algorithm* produces the required thrusts per motor and the net thrust command is converted to motor speed command in **thrustsToMotorCommands** block.

In Altitude controller

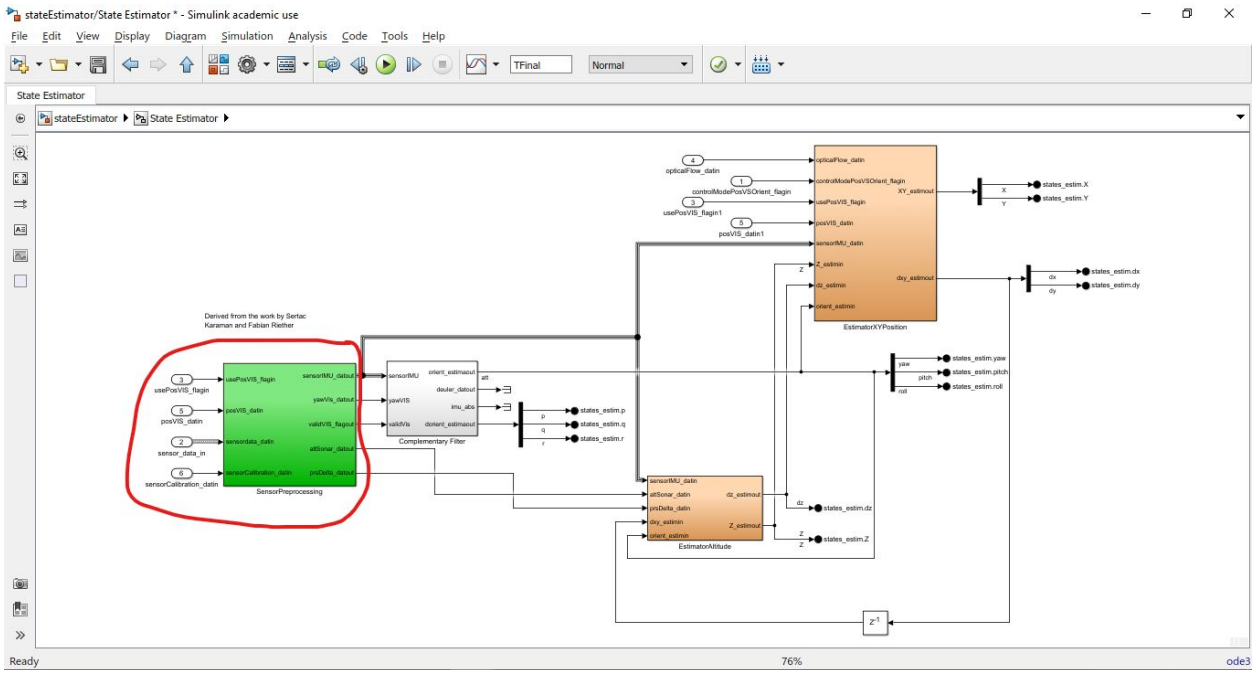


This is a PID controller.
The P_z gain is applied to altitude error derived from the ultrasound whereas D_z gain is applied to the vertical rate estimate that comes from the common filter in the state estimation block.

2. State Estimator



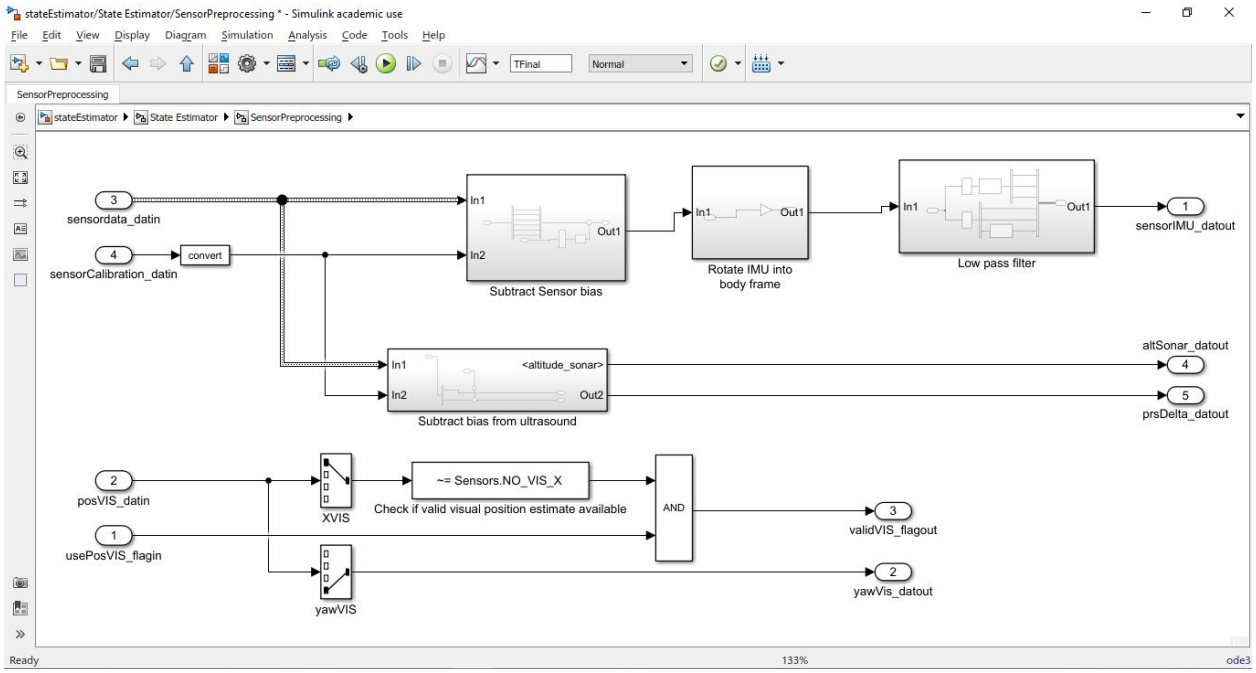
In estimator block



There are 2 steps involved in taking raw sensor measurements and generating estimated states

1. Process the measurements in SensorProcessing
2. Blend them with filters to estimate the control states

In SensorProcessing



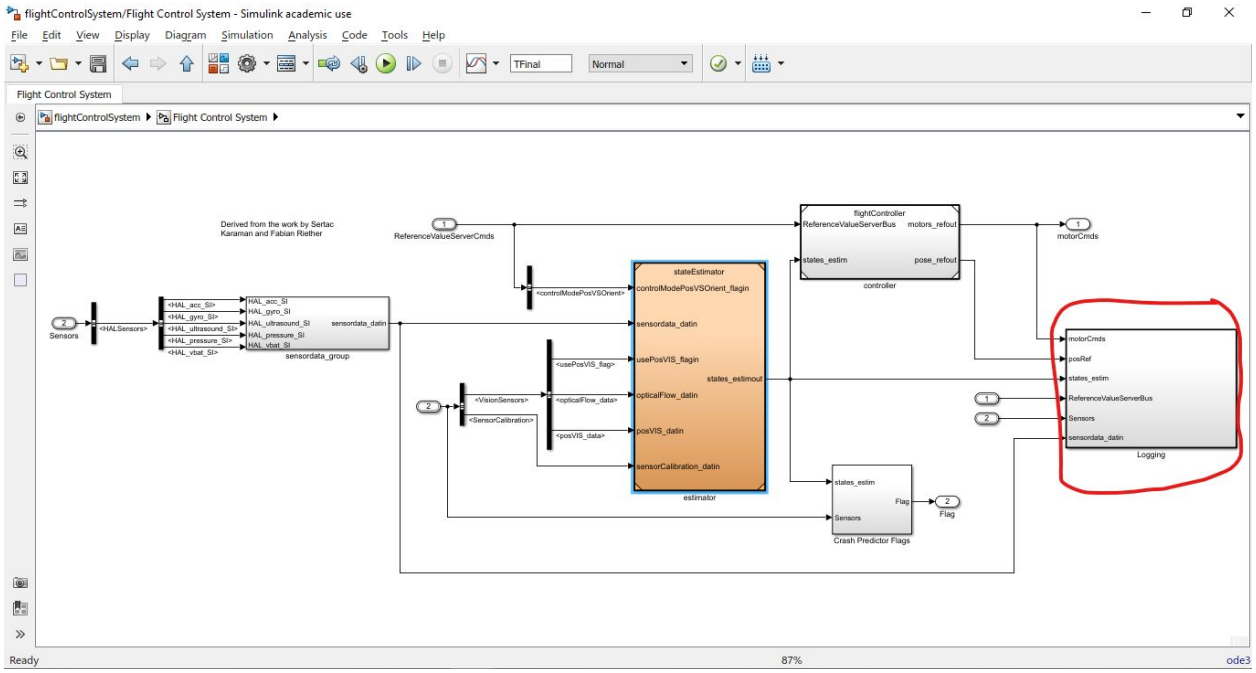
- Acceleration and gyro data are calibrated by subtracting the bias that had been previously determined, and by removing this bias zero acceleration and zero angular rate should result in zero measurement.
- Next, rotate the measurements from the sensor reference frame to the body reference frame
- Filter the data through a low pass filter to remove the high frequency noise
- Similarly the ultrasound sensor has its own bias removed
- Optical flow data has just *pass* or *fail* criteria. If the optical flow sensor has a valid position estimate and we want to use that estimate, then the AND block sets the validity flag to true

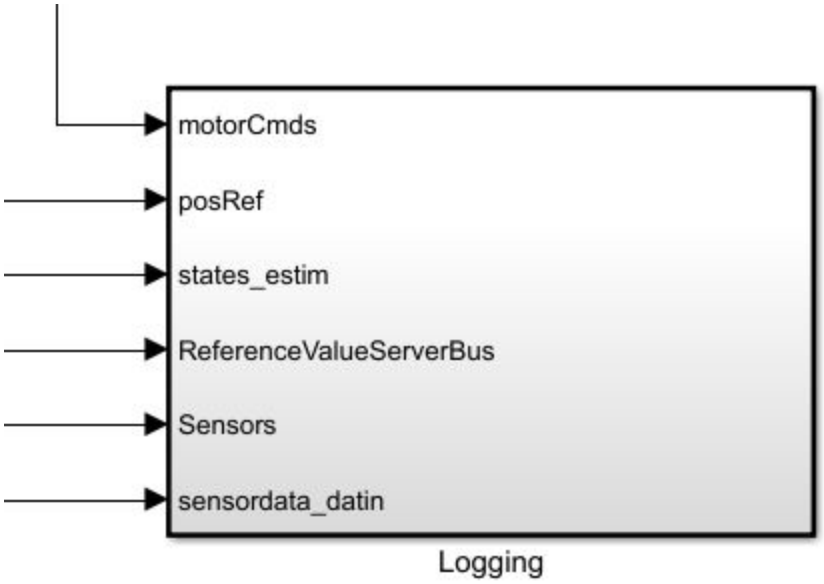
Now, since we have the filtered and calibrated data, we can begin the task of combining measurements to estimate the states we need for the controller .

The two blocks **EstimatorXYPosition** and **EstimatorAltitude** are used to estimate x-y position and altitude respectively.

The block **Complementary Filter** estimates roll, pitch and yaw and it does it using a complementary filter instead of a common filter. A complementary filter is a simple way to blend measurements from the two sensors together

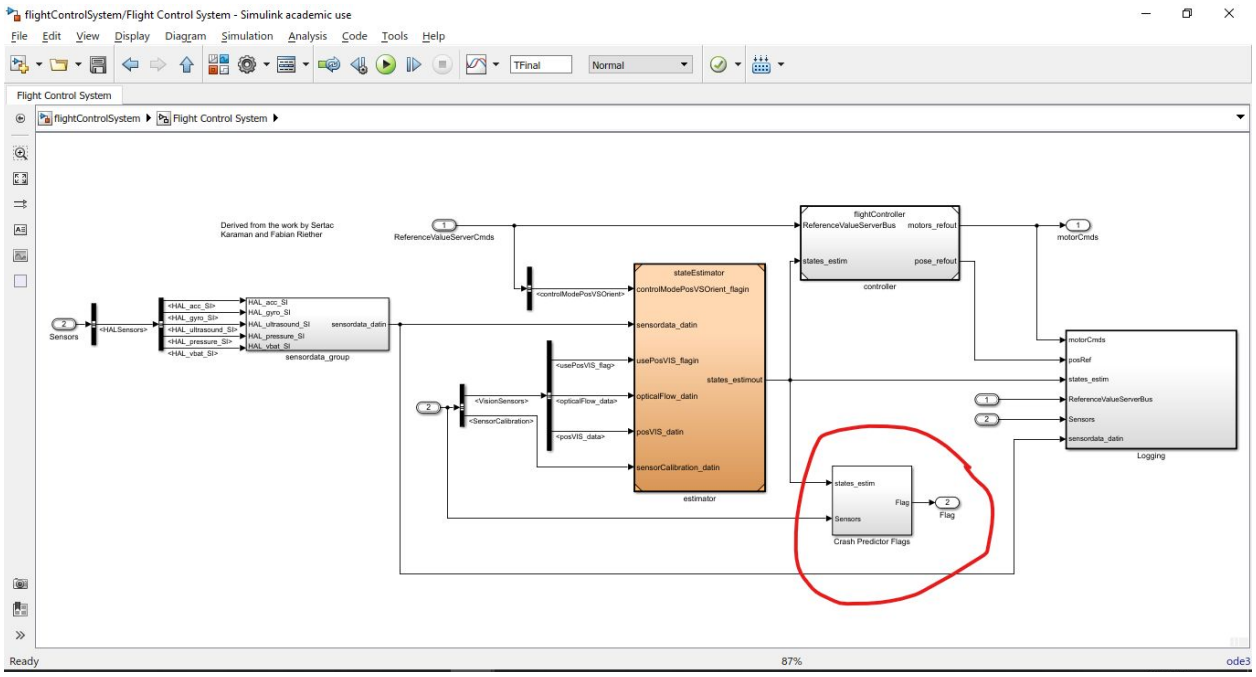
3. Data Logging



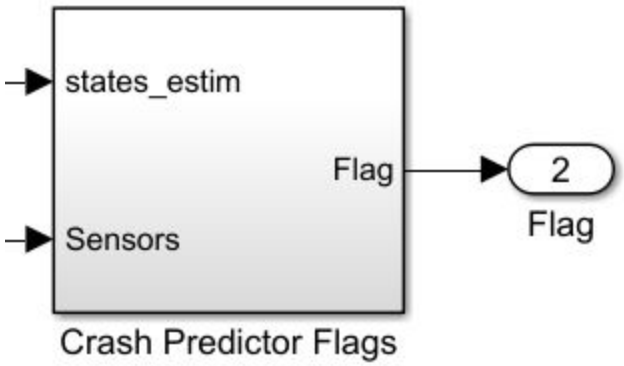


This saves a bunch of signals like motor commands, position references, etc. into .mat files. These are the values that we can download and plot after the flight.

4. Fault Protection

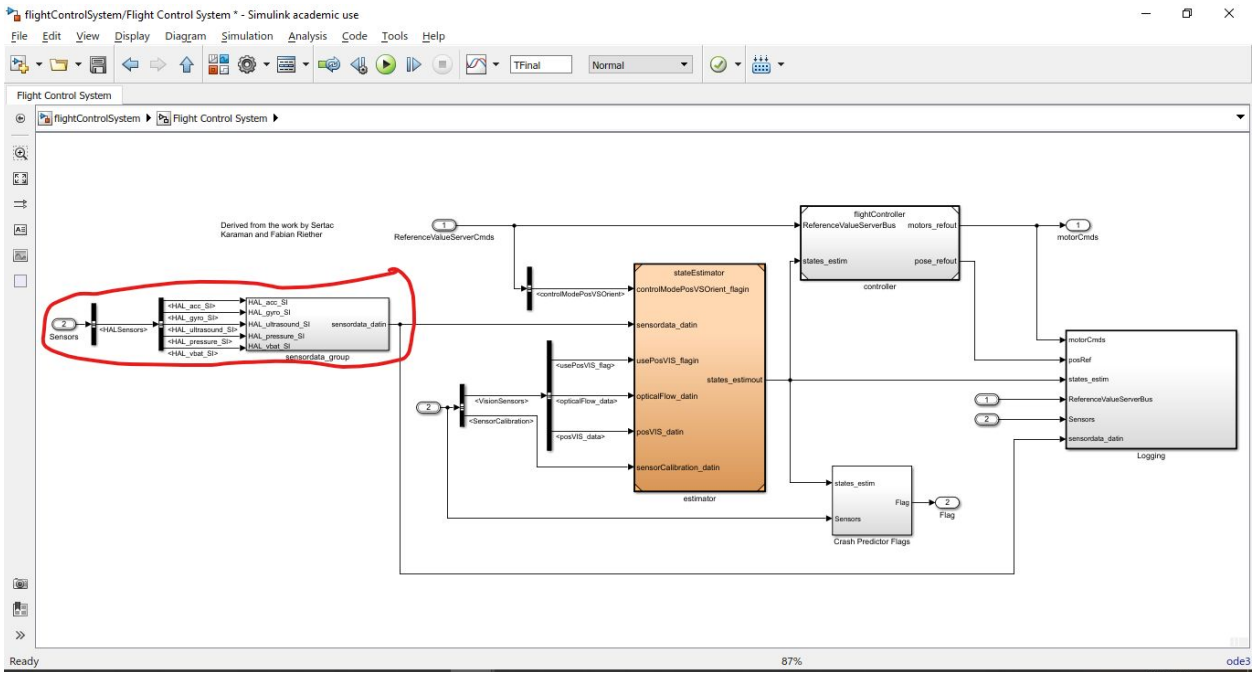


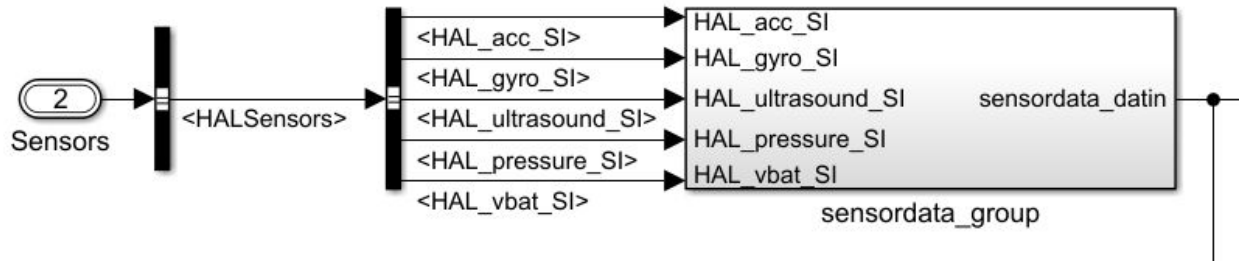
Crash Predictor Flags



This logic in this block is checking the sensitive variables like position and angular velocity for outliers and when that happens it shuts the flags that shuts down the mini drone this is where we can add extra fault protection logic if we want to.

sensordata_group block



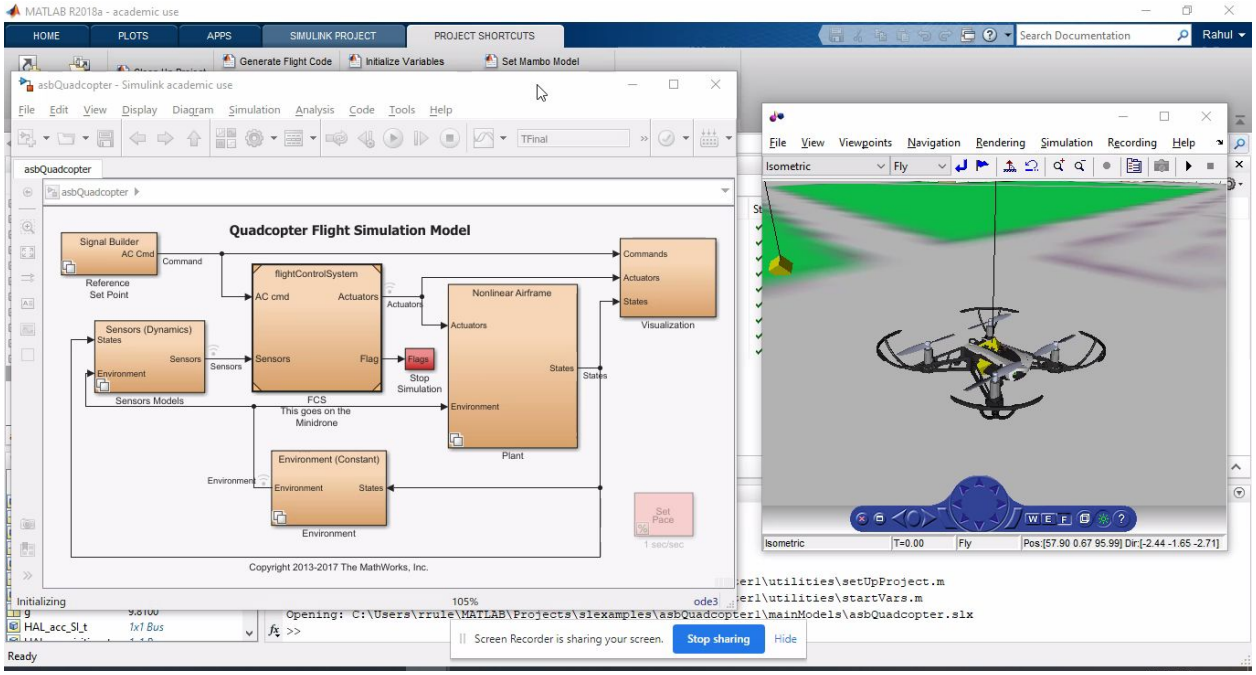


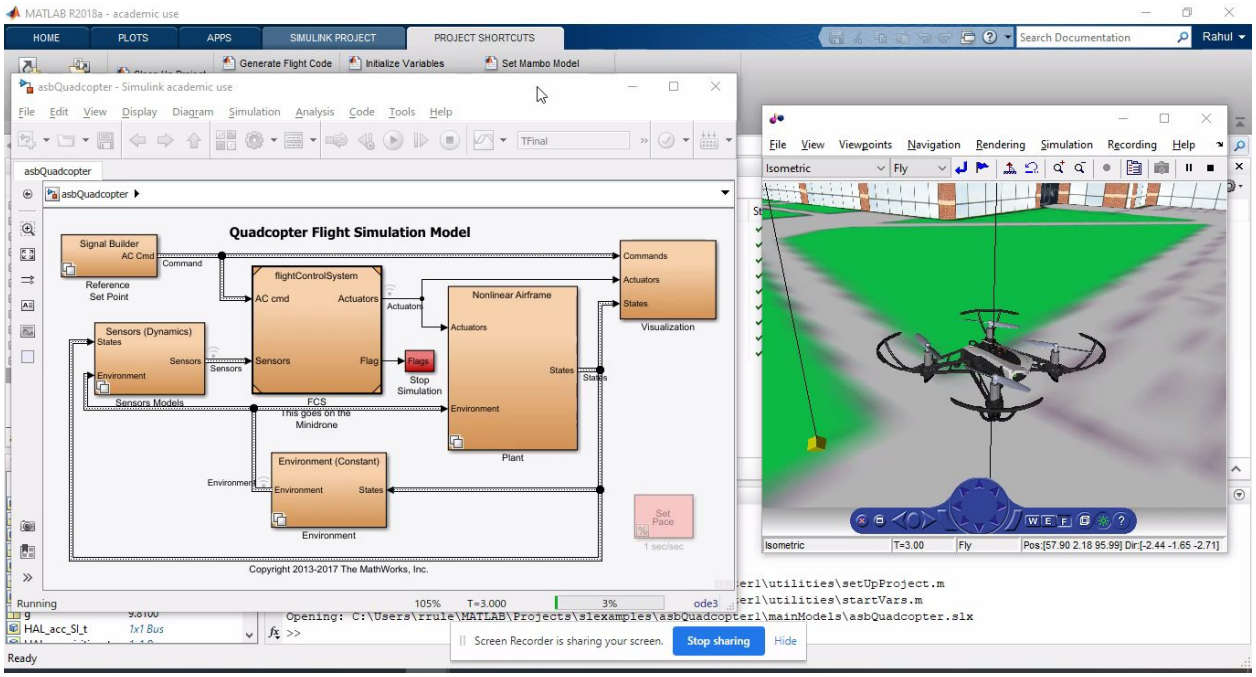
Pulling the individual sensor values off of the sensor bus so that we can have access to them elsewhere in the code.

The Simulink project
The Simulink project folder → absQuadcopter.zip

Steps to get this working:
Step 1: Install Aerospace Blockset
Step 2: Extract this zip file
Step 3: Open this folder in MATLAB and run asbQuadcopter.prj file

Screenshots of Simulation

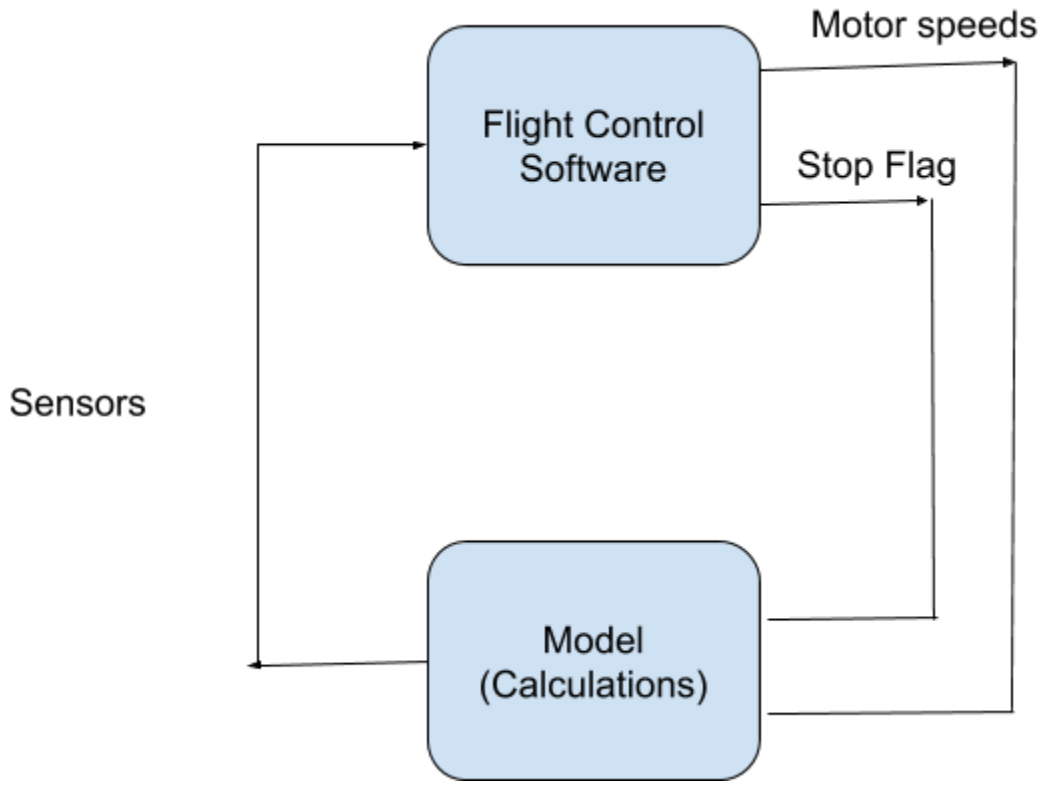




Model for Simulation

The model is basically “everything else”, anything that isn’t flight code. This includes the rest of the mini drone firmware, the hardware and the atmosphere it is flying in etc.

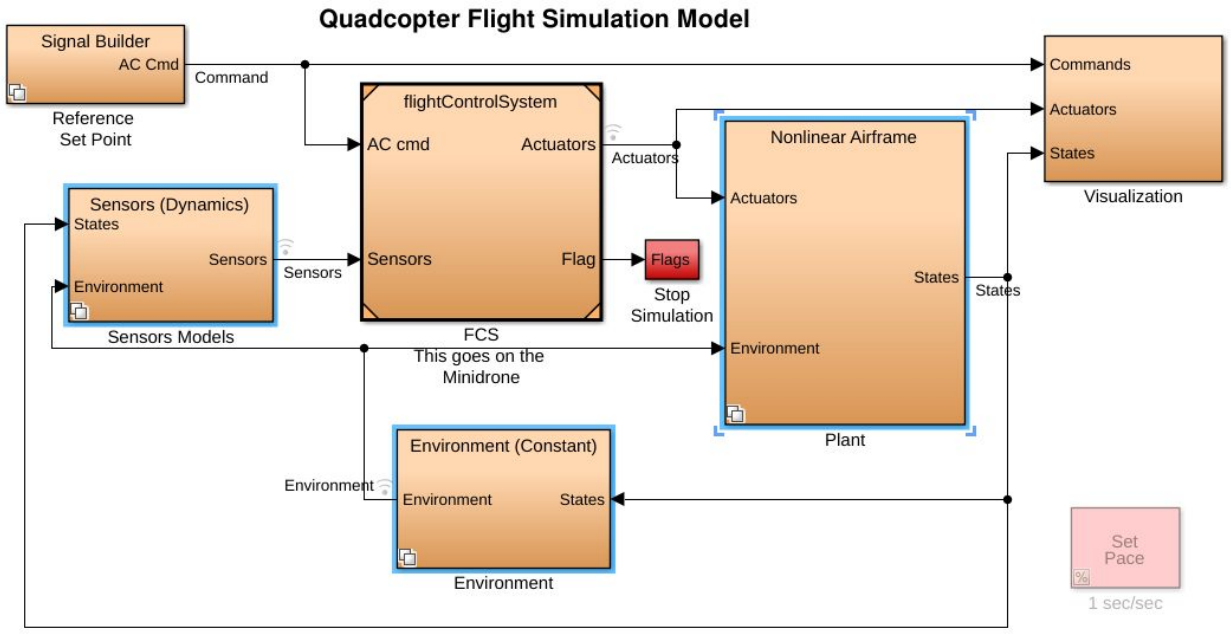
At a basic level, the model inputs motor commands and the stop flag, makes a few calculations and then outputs sensor measurements. This provides a feedback loop to the flight code.



The model can be broken down in sub-models, as in our case:

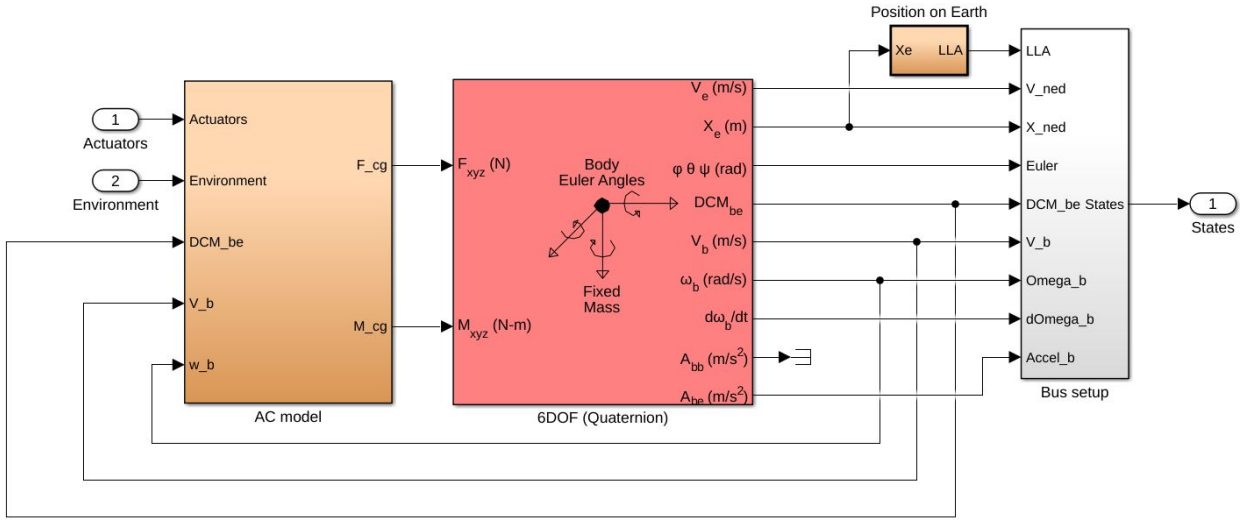
1. Actuators
2. Airframe
3. Environment
4. Sensors

As of now we have the Airframe, environment and the sensors sub models



Airframe

The non-linear airframe sub system contains two main blocks. The **AC** model on the left consists of the actuator models and a model of how the environmental disturbances impact the system. Basically anything that can create a force or torque on the mini drone is calculated in that block.

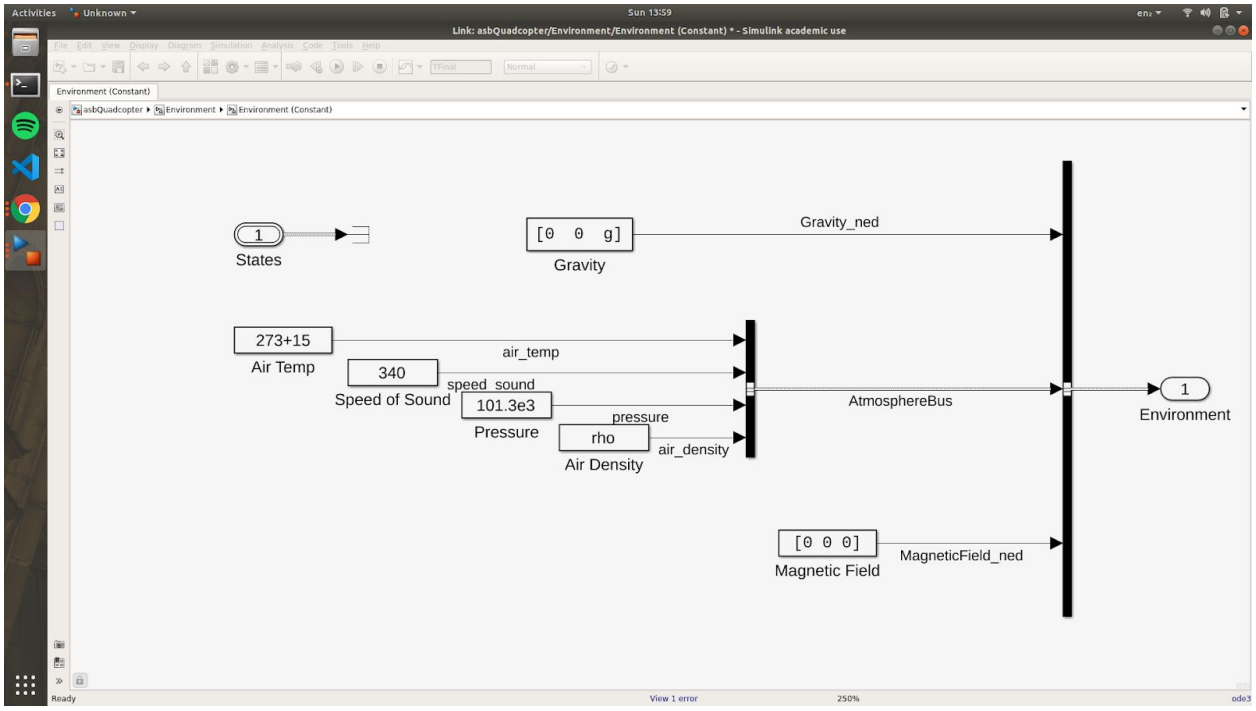


The forces and torques are then fed into the **6DOF** model. This is a rigid body model that comes with the aerospace blockset.

Environment

The models implement several *Aerospace Blockset™* environment blocks, including those for atmosphere and gravity models.

The environment variables include **Gravity, Air Temperature, Pressure, Air Density** and **magnetic field**.

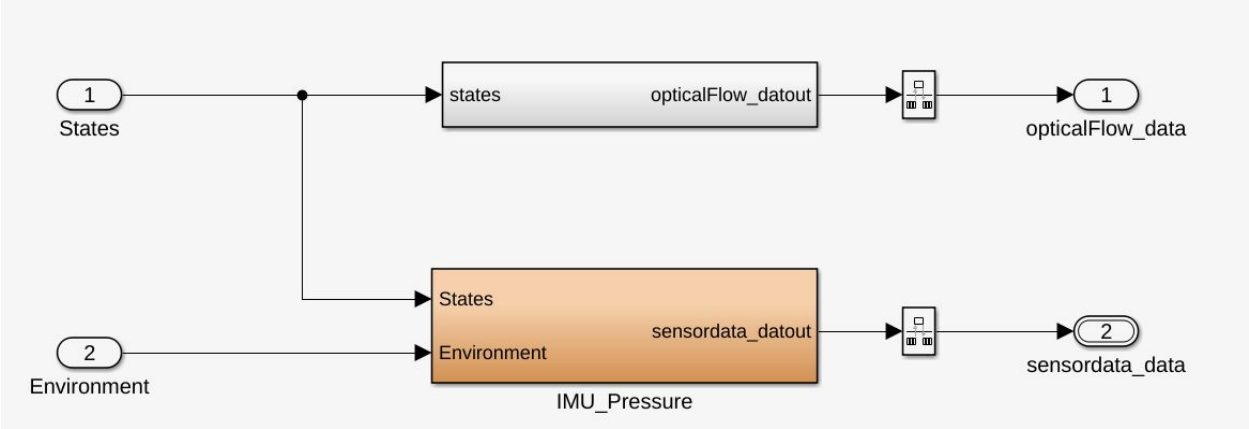


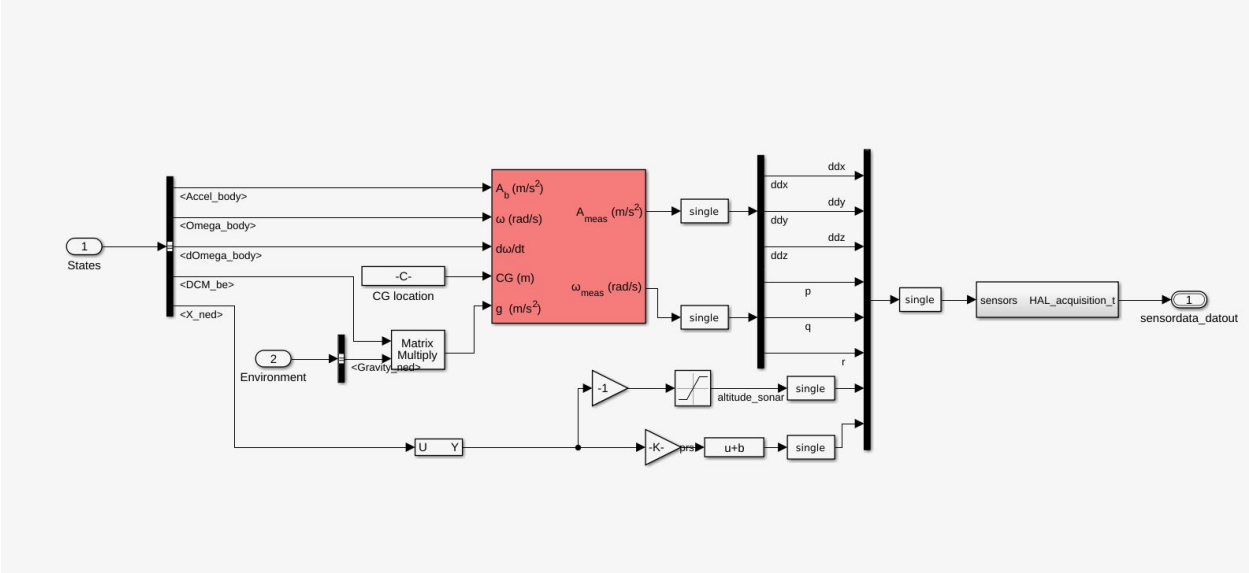
This is an example for a static environment system suitable for testing the howering of the drone.

Sensors

The simulation uses a set of sensors to determine its states:

1. An Inertial Measurement Unit (IMU) to measure the angular rates and translational accelerations.
2. A camera for optical flow estimation.
3. A sonar for altitude measurement.



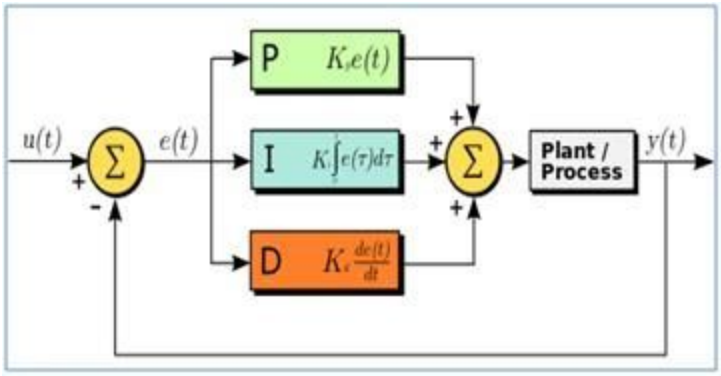


Software algorithm for controller

- We are using **PID**(proportional–integral–derivative) controller algorithms for controlling the system.

PID(proportional–integral–derivative) controller :

- It is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications requiring continuously modulated control.
- PID stands for Proportional-Integral-Derivative. These three controllers are combined in such a way that it produces a control signal.



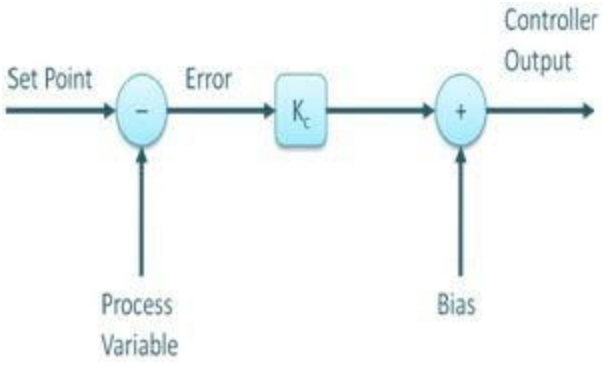
Working of PID controller

- PID controller maintains the output such that there is zero error between process variable and set point/ desired output by closed loop operations.
- It uses three basic control behaviors that are explained below.

P- Controller:

- Proportional or P- controller gives output which is proportional to current error.
- It compares desired or set point with actual value or feedback process value, and the resulting error is multiplied with proportional constant to get the output.
- If the error value is zero, then this controller output is zero.

- This controller requires biasing or manual reset when used alone. This is because it never reaches a steady state condition.
- It provides stable operation but always maintains the steady state error.
- Speed of the response is increased when the proportional constant increases.



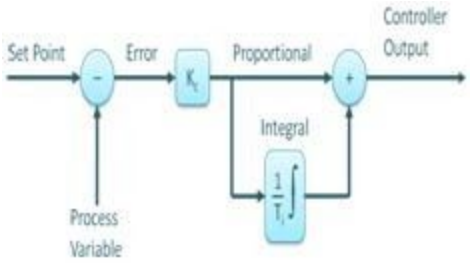
P- controller

P-controller response

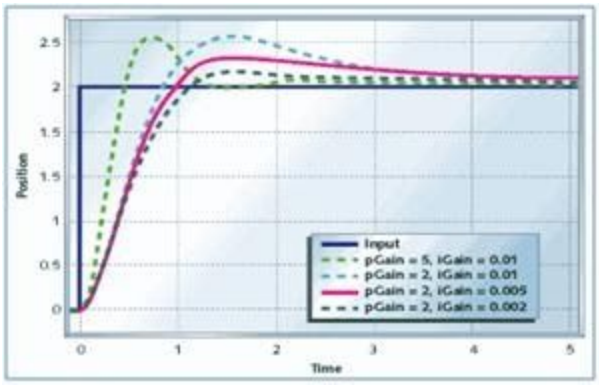
I-Controller:

- Due to the limitation of a p-controller where there always exists an offset between the process variable and set point, an I-controller is needed, which provides necessary action to eliminate the steady state error.
- It integrates the error over a period of time until the error value reaches to zero.

- It holds the value to the final control device at which error becomes zero.
- Integral control decreases its output when negative error takes place.
- It limits the speed of response and affects stability of the system.
- Speed of the response is increased by decreasing integral gain Ki.



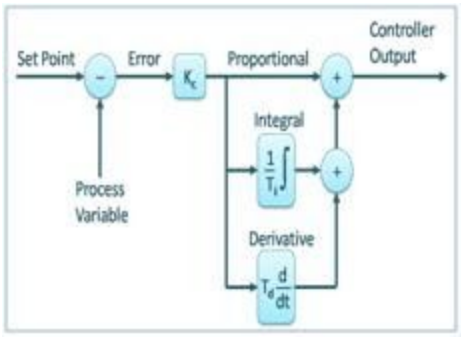
PI-controller



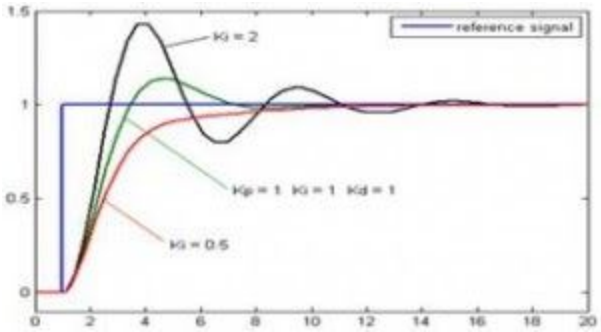
PI-controller response

D-Controller:

- The I-controller doesn't have the capability to predict the future behavior of error. So it reacts normally once the set point is changed.
- D-controller overcomes this problem by anticipating future behavior of the error.
- Its output depends on the rate of change of error with respect to time, multiplied by derivative constant.
- It improves the stability of the system by compensating phase lag caused by the I-controller.
- Increasing the derivative gain increases speed of response.



PID-controller



PID-controller response

- So finally by combining these three controllers, we can get the desired response for the system.

References

- https://in.mathworks.com/help/aeroblks/
- https://in.mathworks.com/help/rtw/
- Quadrotor control: modeling, nonlinear control design, and simulation
- https://in.mathworks.com/matlabcentral/fileexchange/63318-simulink-support-package-for-parrot-minidrones
- https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/s15/syllabus/ppp/Lec08-Control3.pdf