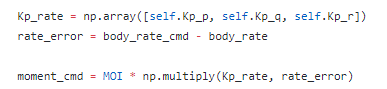
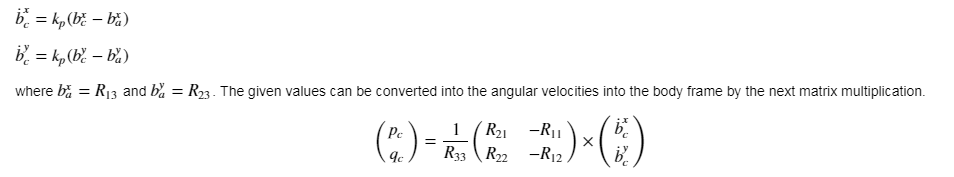
1. Implemented body rate control in C++

This code is implemented from line# 110 to 114 in QuadControl.cpp file. As per following snippet from Python solution for Controls project, we are supposed to implement a P controller for body rate and multiply it with MOI to return a moment command. And this is easily implemented in above stated lines.



1. Implement roll pitch control in C++

This function is implemented from line# 145 to 163 in QuadControl.cpp file. Initially, we need to make sure that received command is having value greater than 0. The roll-pitch controller is a P controller responsible for commanding the roll and pitch rates ( pcpc and qcqc ) in the body frame. First, it sets the desired rate of change of the given matrix elements using a P controller. We are using following equations from lecture notes to come up with x and y commands



1. Implement altitude controller in C++

This function is implemented from line# 194 to 213. In this command we are implementing a classical PID controller with feed-forward acceleration.

1. Implement lateral position control in C++

This function is implemented from line# 246 to 272. Initially we are making sure that velocity command received is within our threshold limit. And then we implement a PD controller with P term multiplied by positional error and d term with velocity error adding Feed-forward acceleration to the equation.

1. Implement yaw control in C++

This function is implemented from line# 290 to 309. Yaw control is also a simple P controller. Only precaution we need to take in this loop is to make sure Yaw’s value does not go outside of 2\*pi limit.

1. Implement calculating the motor commands given commanded thrust and moments in C++

This function is implemented from line# 73 to 82 in QuadControl.cpp file. As inputs we are given desired collective thrust, which we store in variable t4, and momentCmd around x, y and z axis. I used following lecture notes to come up with individual motor thrust. Following equations shows us how individual moments are related to motor thrust, and once we get to know individual moments (which are provided as input to function), we can solve it to get motor thrusts