Write up - **Building a Controller** - Rubric	
Criteria	Meet Specification
Implemented body rate control in python and C++.	The controller should be a proportional controller on body rates to commanded moments. The controller should take into account the moments of inertia of the drone when calculating the commanded moments. The commanded moments are calculated in the body rate control is implemented as proportional control in /python/controller.py body_rate_control method from line 220 to 227 using Python ***Used Kp (pq,r) with MOI(provided as constant) and normalize and margin by MAX_TORQUE.*** and in /cpp/src/Quad Control::BodyRateControl method from line 112 to 116 using C++. Here there is no constant MAX_TORQUE provided so the following is performed. `momentum = I * kpPQR * p_term;`
Implement roll pitch control in python and C++.	*The controller should use the acceleration and thrust commands, in addition to the vehicle attitude to output a body rate command. The controller should account for the non-linear transformation from local accelerations to body rates. Note that the drone's mass should be accounted for when calculating the target angles.* Rotation Matrix is the mechanism for changing the representation of vector in one(Body) frame to representation in another (World or Inertial)frame e.g $^{W}_{B}R_{23} = \|k_B\|\|j_W\|\cos(\beta)$ *where the subscript means the source frame (here the body frame), and the superscript means to destination frame(here the world frame).* The roll pitch control is implemented in /python/controller.py roll_pitch_controller method from line 188 to 205 using Python

	and in /cpp/src/QuadControl::RollPitchControl method from line 146 to 166 using C++. The thrust, mass of the Quad, acceleration, Kp Gains for pitch and roll and rotation matrix are put in for solving equation to get desired pitch rate and roll rate commands.
Implement altitude control in python.	The controller should use both the down position and the down velocity to command thrust. Ensure that the output value is indeed thrust (the drone's mass needs to be accounted for) and that the thrust includes the non-linear effects from non-zero roll/pitch angles. The altitude control is implemented in /python/controller.py altitude_control method from line 152 to 173 using Python. 1) Find Altitude error 2) Mutlipy by kp_gain_Alt and add velocitycmd 3) Then margin ascending and descending rate. 4) Calculate acceleration by drone mass, feedforward and integration error (with Kp_gain_i integration). 5) Normalize acceleration by rotation matrix [2,2] 6) Margin the thrust (Do that not flyway or drop down). This is painful part . But it helped in the cpp project.
Implement altitude controller in C++.	*The controller should use both the down position and the down velocity to command thrust. Ensure that the output value is indeed thrust (the drone's mass needs to be accounted for) and that the thrust includes the non-linear effects from non-zero roll/pitch angles.* Additionally, the C++ altitude controller should contain an integrator to handle the weight non-idealities presented in scenario 4. The altitude control is implemented in /cpp/src/QuadControl::AltitudeControl method from line 197 to 227 using C++.

	1) Find Altitude error 2) Mutlipy by kp_gain_Alt and add velocitycmd 3) Then margin ascending and descending rate. 4) Calculate acceleration by drone mass, feedforward and integration error (with Kp_gain_i integration). 5) Normalize acceleration by rotation matrix [2,2]. Here the Negativity of Gravity is used for Normalization. 6) Margin the thrust (Do that not flyway or drop down). This is painful part . But implementation in python helped in the cpp project.
Implement lateral position control in python and C++.	*The controller should use the local NE position and velocity to generate a commanded local acceleration.* The lateral position control is implemented in /python/controller.py lateral_position_control method from line 120 to 132 using Python 1) Using Position err, velocity err and get velocity command by product of postion gain. 2) Normalize velocity command by Euclidean distance. 3) Norm Velocity is limited by max speed. 4) Calculate local Acceleration by above all variables with kp gains. and in /cpp/src/QuadControl::LateralPositionControl method from line 257 to 280 using C++. 1) Using Postion err, velocity err and get velocity command by product of postion gain. 2) Normalize velocity command by Euclidean distance. 3) Norm Velocity is limited by max speed. 4) Calculate local Acceleration by above all variables with kp gains. 5) Normalized Euclidean Acceleration is limited by maxAccelXY. This one took from slack student help. This secret sauce controls the vehicle slip sideways.

Implement yaw control in python and C++.	*The controller can be a linear/proportional heading controller to yaw rate commands (non-linear transformation not required).* The yaw control is implemented in /python/controller.py yaw_control method from line 239 to 250 using Python and in /cpp/src/QuadControl::YawControl method from line 303 to 315 using C++. 1) Both the implementation set Margin the angle within 0 to 2*Pi 2) In Python this is achieved by np.mod function. 3) In c++ it is achieved by fmodf function.

Implement calculating the motor commands given commanded thrust and moments in C++.	The calculation implementation for the motor commands is in method from line 79 to 88.
	
	This is from slack inputs and my mentor helped me to solve this equation.
	F_tot = F0 + F1 + F2 + F3
	tau_x = (F0 - F1 + F2 - F3) * l // This is Roll
	tau_y = (F0 + F1 - F2 - F3) * l // This is Pitch
	tau_z = (-F0 + F1 + F2 - F3) * kappa // This is Yaw
	kappa = kf/km coefficient gains crucial to calculated motor commands.
	where l = L / sqrt (2) - since, unlike in lecture, L is defined as half the distance between rotors
	
	Now we have 4 equations and 4 unknowns (the F values), so we can solve them to get the F1, F2, F3 and F4 values, given that we already know the F_tot, tau_x, tau_y and tau_z inputs to the GenerateMotorCommands (float collThrustCmd, V3F momentum) function.