Project : Building a Controller

Project Rubric : WRITE UP

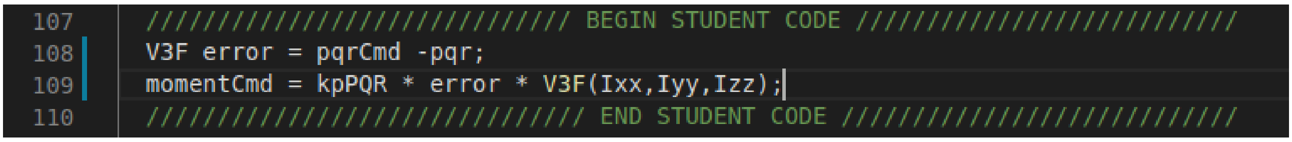
1) Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf.

This would the documewnt where I explain the rubric points

Project Rubric : Implemented Controller

1)Implemented body rate control in C++.

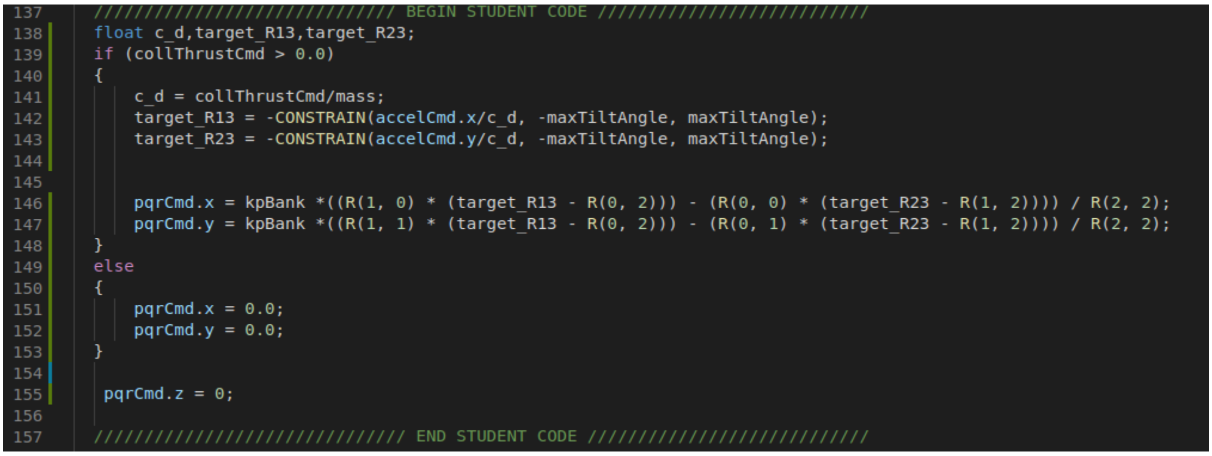
- The controller is proportional to the error in body rate command.
- The controller takes into account the moments of inertia of the drone when calculating the commanded moments.



```
////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
V3F error = pqrCmd -pqr;
momentCmd = kpPQR * error * V3F(Ixx,Iyy,Izz);
/////////////////////////////// END STUDENT CODE ///////////////////////////
```

2)Implement roll pitch control in C++.

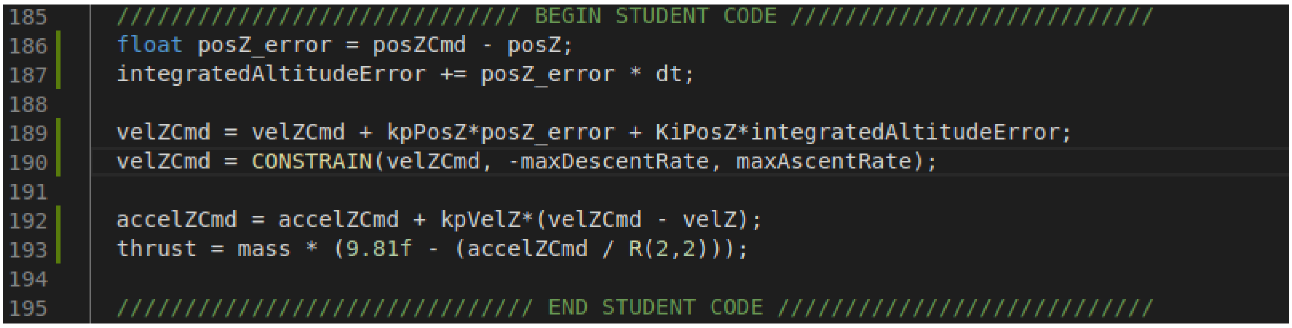
- The controller uses acceleration and thrust commands.
- The controller uses the rotation matrix to account the non-linear transformation from local accelerations to body rates.



```
/////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
float c_d,target_R13,target_R23;
if (collThrustCmd > 0.0)
{
    c_d = collThrustCmd/mass;
    target_R13 = -CONSTRAIN(accelCmd.x/c_d, -maxTiltAngle, maxTiltAngle);
    target_R23 = -CONSTRAIN(accelCmd.y/c_d, -maxTiltAngle, maxTiltAngle);


    pqrCmd.x = kpBank *((R(1, 0) * (target_R13 - R(0, 2))) - (R(0, 0) * (target_R23 - R(1, 2)))) / R(2, 2);
    pqrCmd.y = kpBank *((R(1, 1) * (target_R13 - R(0, 2))) - (R(0, 1) * (target_R23 - R(1, 2)))) / R(2, 2);
}
else
{
    pqrCmd.x = 0.0;
    pqrCmd.y = 0.0;
}

 pqrCmd.z = 0;

/////////////////////////////// END STUDENT CODE ///////////////////////////
```

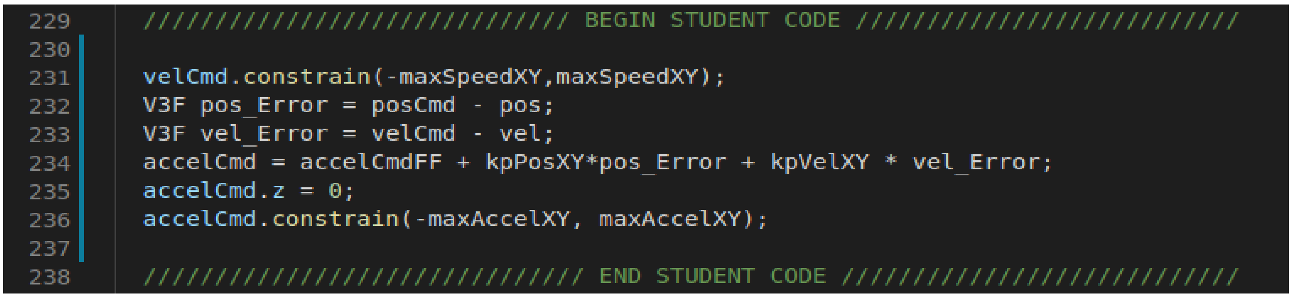
3)Implement altitude controller in C++.

- The controller uses both the down position and the down velocity to command thrust.
- The controller also contains **integratedAltitudeError** Variable which stores the integrated values over time.



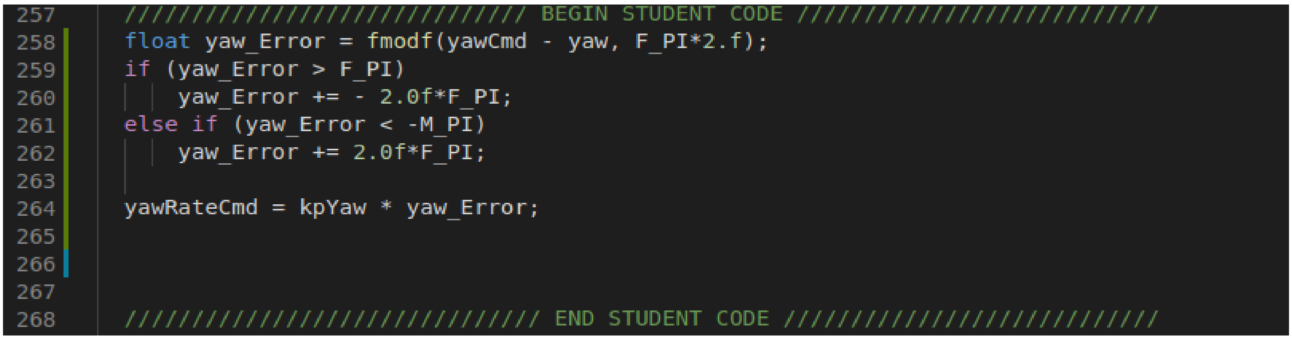
```
/////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
float posZ_error = posZCmd - posZ;
integratedAltitudeError += posZ_error * dt;

velZCmd = velZCmd + kpPosZ*posZ_error + KiPosZ*integratedAltitudeError;
velZCmd = CONSTRAIN(velZCmd, -maxDescentRate, maxAscentRate);

accelZCmd = accelZCmd + kpVelZ*(velZCmd - velZ);
thrust = mass * (9.81f - (accelZCmd / R(2,2)));

//////////////////////////////// END STUDENT CODE ////////////////////////////
```

4)Implement lateral position control in C++.



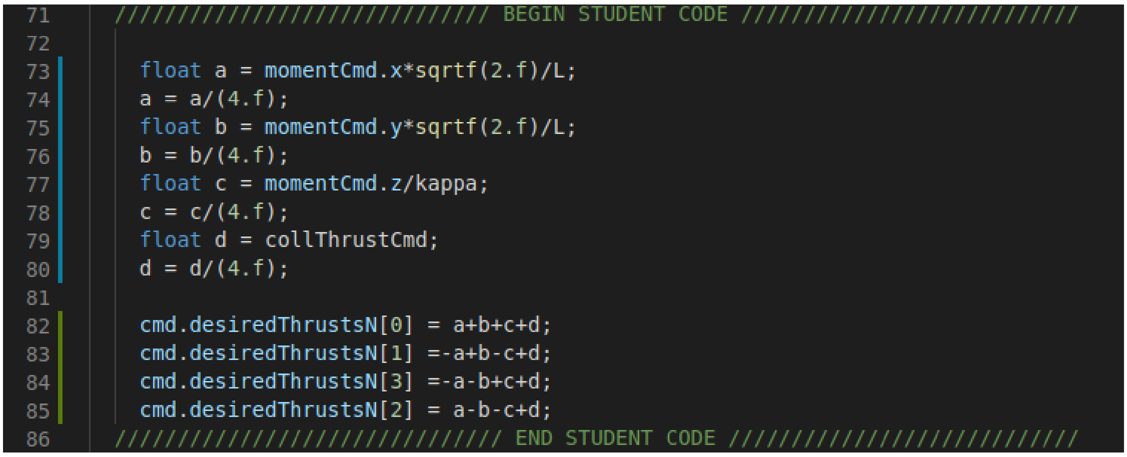
```
/////////////////////////////// BEGIN STUDENT CODE ///////////////////////////

velCmd.constrain(-maxSpeedXY,maxSpeedXY);
V3F pos_Error = posCmd - pos;
V3F vel_Error = velCmd - vel;
accelCmd = accelCmdFF + kpPosXY*pos_Error + kpVelXY * vel_Error;
accelCmd.z = 0;
accelCmd.constrain(-maxAccelXY, maxAccelXY);

//////////////////////////////// END STUDENT CODE ////////////////////////////
```

5)Implement yaw control in C++.



```
/////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
float yaw_Error = fmodf(yawCmd - yaw, F_PI*2.f);
if (yaw_Error > F_PI)
    yaw_Error += - 2.0f*F_PI;
else if (yaw_Error < -M_PI)
    yaw_Error += 2.0f*F_PI;

yawRateCmd = kpYaw * yaw_Error;



//////////////////////////////// END STUDENT CODE ////////////////////////////
```

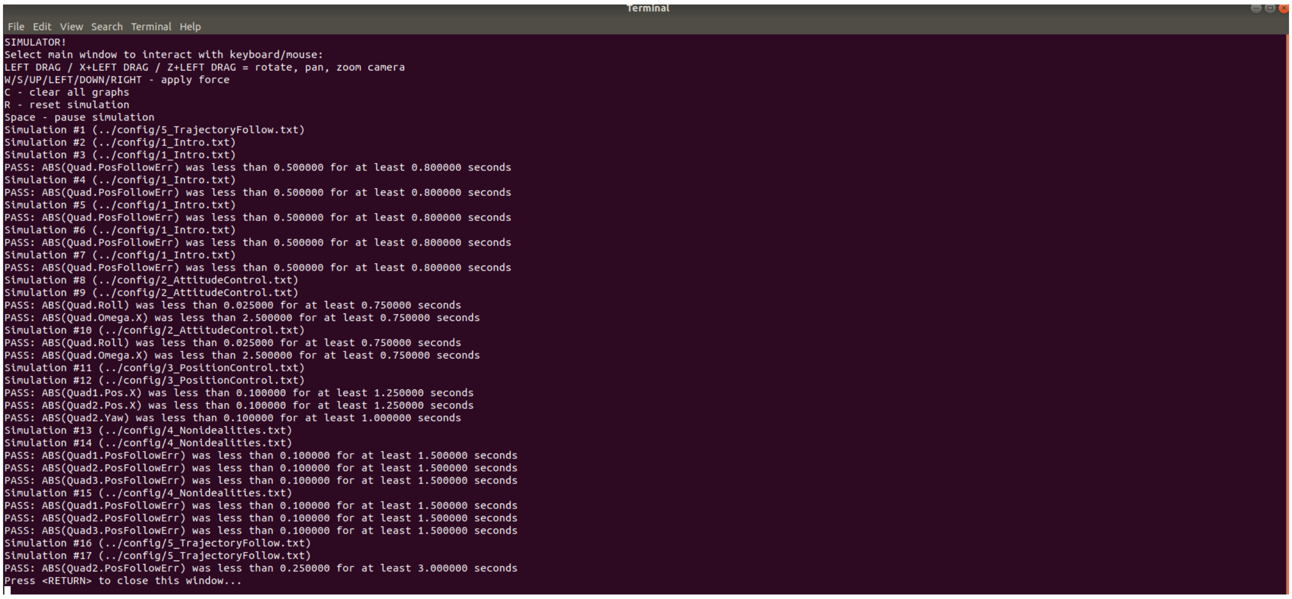
6)Implement calculating the motor commands given commanded thrust and moments in C++.



```
/////////////////////////////// BEGIN STUDENT CODE ///////////////////////////

    float a = momentCmd.x*sqrtf(2.f)/L;
    a = a/(4.f);
    float b = momentCmd.y*sqrtf(2.f)/L;
    b = b/(4.f);
    float c = momentCmd.z/kappa;
    c = c/(4.f);
    float d = collThrustCmd;
    d = d/(4.f);

    cmd.desiredThrustsN[0] = a+b+c+d;
    cmd.desiredThrustsN[1] =-a+b-c+d;
    cmd.desiredThrustsN[3] =-a-b+c+d;
    cmd.desiredThrustsN[2] = a-b-c+d;
//////////////////////////////// END STUDENT CODE ////////////////////////////
```

Project Rubric : Flight Evaluation

Test Results of my controller



```
SIMULATOR!
Select main window to interact with keyboard/mouse:
LEFT DRAG / X+LEFT DRAG / Z+LEFT DRAG = rotate, pan, zoom camera
W/S/UP/LEFT/DOWN/RIGHT - apply force
C - clear all graphs
R - reset simulation
Space - pause simulation
Simulation #1 (../config/5_TrajectoryFollow.txt)
Simulation #2 (../config/1_Intro.txt)
Simulation #3 (../config/1_Intro.txt)
PASS: ABS(Quad.PosFollowErr) was less than 0.500000 for at least 0.800000 seconds
Simulation #4 (../config/1_Intro.txt)
PASS: ABS(Quad.PosFollowErr) was less than 0.500000 for at least 0.800000 seconds
Simulation #5 (../config/1_Intro.txt)
PASS: ABS(Quad.PosFollowErr) was less than 0.500000 for at least 0.800000 seconds
Simulation #6 (../config/1_Intro.txt)
PASS: ABS(Quad.PosFollowErr) was less than 0.500000 for at least 0.800000 seconds
Simulation #7 (../config/1_Intro.txt)
PASS: ABS(Quad.PosFollowErr) was less than 0.500000 for at least 0.800000 seconds
Simulation #8 (../config/2_AttitudeControl.txt)
Simulation #9 (../config/2_AttitudeControl.txt)
PASS: ABS(Quad.Roll) was less than 0.025000 for at least 0.750000 seconds
PASS: ABS(Quad.Omega.X) was less than 2.500000 for at least 0.750000 seconds
Simulation #10 (../config/2_AttitudeControl.txt)
PASS: ABS(Quad.Roll) was less than 0.025000 for at least 0.750000 seconds
PASS: ABS(Quad.Omega.X) was less than 2.500000 for at least 0.750000 seconds
Simulation #11 (../config/3_PositionControl.txt)
Simulation #12 (../config/3_PositionControl.txt)
PASS: ABS(Quad1.Pos.X) was less than 0.100000 for at least 1.250000 seconds
PASS: ABS(Quad2.Pos.X) was less than 0.100000 for at least 1.250000 seconds
PASS: ABS(Quad2.Yaw) was less than 0.100000 for at least 1.000000 seconds
Simulation #13 (../config/4_Nonidealities.txt)
Simulation #14 (../config/4_Nonidealities.txt)
PASS: ABS(Quad1.PosFollowErr) was less than 0.100000 for at least 1.500000 seconds
PASS: ABS(Quad2.PosFollowErr) was less than 0.100000 for at least 1.500000 seconds
PASS: ABS(Quad3.PosFollowErr) was less than 0.100000 for at least 1.500000 seconds
Simulation #15 (../config/4_Nonidealities.txt)
PASS: ABS(Quad1.PosFollowErr) was less than 0.100000 for at least 1.500000 seconds
PASS: ABS(Quad2.PosFollowErr) was less than 0.100000 for at least 1.500000 seconds
PASS: ABS(Quad3.PosFollowErr) was less than 0.100000 for at least 1.500000 seconds
Simulation #16 (../config/5_TrajectoryFollow.txt)
Simulation #17 (../config/5_TrajectoryFollow.txt)
PASS: ABS(Quad2.PosFollowErr) was less than 0.250000 for at least 3.000000 seconds
Press <RETURN> to close this window...
```