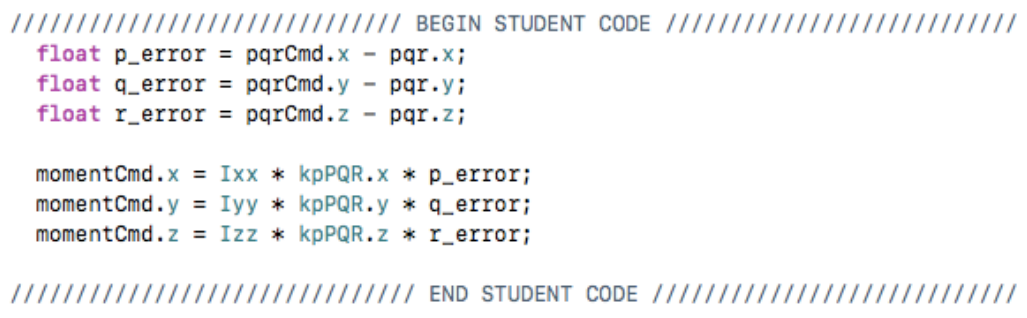
Jackson Burrus
12/9/18

Control of a 3D Quadrotor Writeup

1.

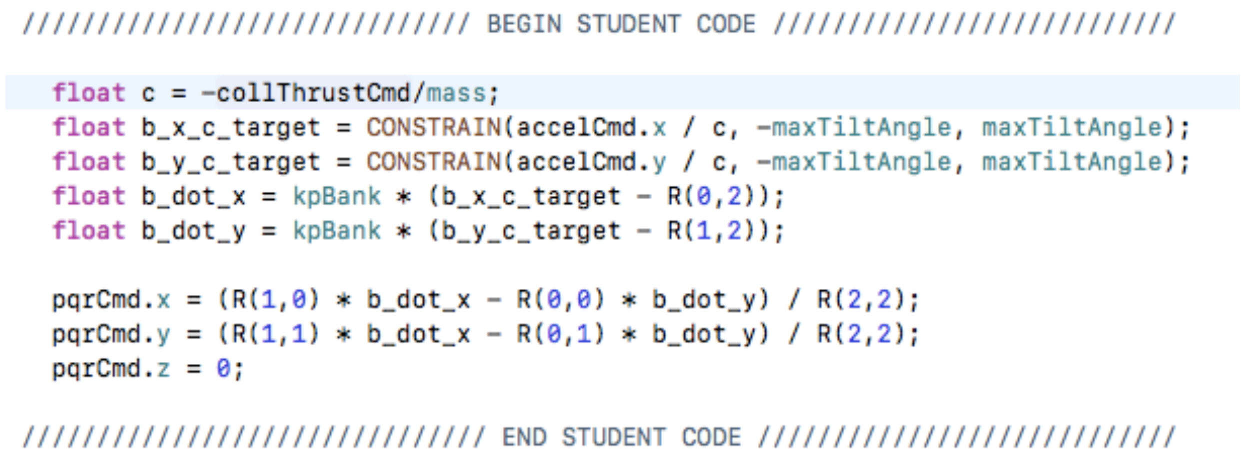
A P controller was implemented for the body rate controller using the error between the desired body rate and the actual body rate. The moment is then calculated using this error along with the gain and moment of inertia. The code can be seen below:



```
////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
  float p_error = pqrCmd.x - pqr.x;
  float q_error = pqrCmd.y - pqr.y;
  float r_error = pqrCmd.z - pqr.z;

  momentCmd.x = Ixx * kpPQR.x * p_error;
  momentCmd.y = Iyy * kpPQR.y * q_error;
  momentCmd.z = Izz * kpPQR.z * r_error;

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

2.

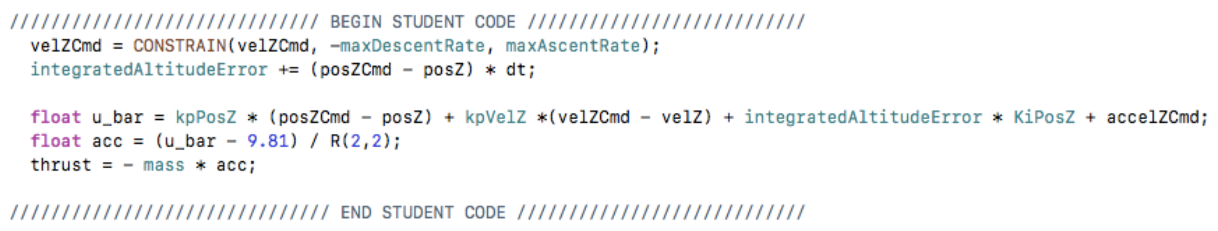
For the roll pitch controller, the acceleration is first calculated from the thrust command. Then the target angles are calculated and constrained using the max tilt angles. This target angle is then converted into a body rate command as the output. This code can be seen below:



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

  float c = -collThrustCmd/mass;
  float b_x_c_target = CONSTRAIN(accelCmd.x / c, -maxTiltAngle, maxTiltAngle);
  float b_y_c_target = CONSTRAIN(accelCmd.y / c, -maxTiltAngle, maxTiltAngle);
  float b_dot_x = kpBank * (b_x_c_target - R(0,2));
  float b_dot_y = kpBank * (b_y_c_target - R(1,2));

  pqrCmd.x = (R(1,0) * b_dot_x - R(0,0) * b_dot_y) / R(2,2);
  pqrCmd.y = (R(1,1) * b_dot_x - R(0,1) * b_dot_y) / R(2,2);
  pqrCmd.z = 0;

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

3.

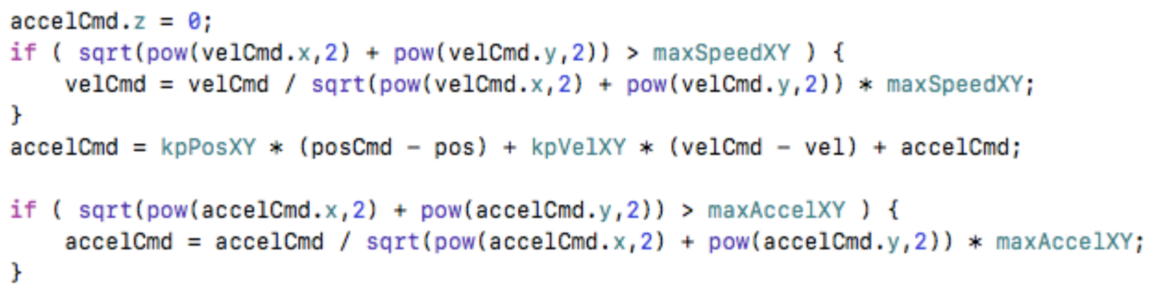
For the altitude controller, the desired velocity command is first constrained between the minimum and maximum velocities. A PID controller is then used to produce a thrust command output. The code can be seen below:



```
////////////////////////////// BEGIN STUDENT CODE //////////////////////////
  velZCmd = CONSTRAIN(velZCmd, -maxDescentRate, maxAscentRate);
  integratedAltitudeError += (posZCmd - posZ) * dt;

  float u_bar = kpPosZ * (posZCmd - posZ) + kpVelZ *(velZCmd - velZ) + integratedAltitudeError * KiPosZ + accelZCmd;
  float acc = (u_bar - 9.81) / R(2,2);
  thrust = - mass * acc;

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

4.

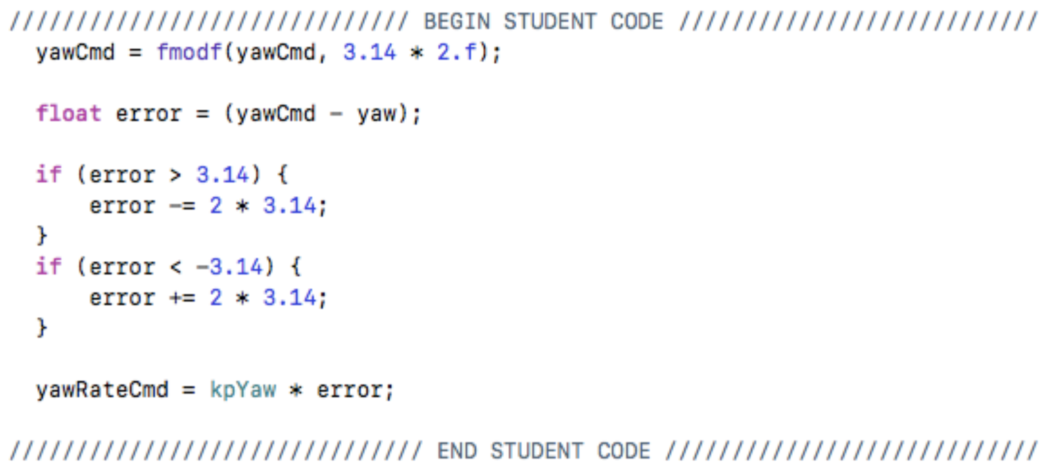
The lateral position controller uses a PD controller to output the acceleration command. The velocity and acceleration are both constrained by the maximum lateral speed and acceleration, respectfully.



```
accelCmd.z = 0;
if ( sqrt(pow(velCmd.x,2) + pow(velCmd.y,2)) > maxSpeedXY ) {
    velCmd = velCmd / sqrt(pow(velCmd.x,2) + pow(velCmd.y,2)) * maxSpeedXY;
}
accelCmd = kpPosXY * (posCmd - pos) + kpVelXY * (velCmd - vel) + accelCmd;

if ( sqrt(pow(accelCmd.x,2) + pow(accelCmd.y,2)) > maxAccelXY ) {
    accelCmd = accelCmd / sqrt(pow(accelCmd.x,2) + pow(accelCmd.y,2)) * maxAccelXY;
}
```

5.

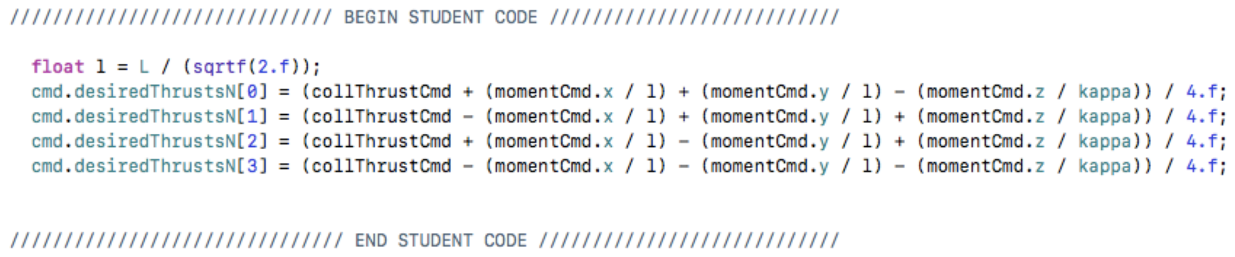
The yaw controller is a P controller that uses the error between the yaw command and actual yaw. The yaw command was limited to the range [0, 2π]. The error was modified to make the drone rotate the least amount required to match the yaw command.



```
////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
  yawCmd = fmodf(yawCmd, 3.14 * 2.f);

  float error = (yawCmd - yaw);

  if (error > 3.14) {
      error -= 2 * 3.14;
  }
  if (error < -3.14) {
      error += 2 * 3.14;
  }

  yawRateCmd = kpYaw * error;

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

6.

The thrust command for each motor is calculated using information from the collective thrust command and moment commands calculated from the controllers above. This code can be seen below:



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

  float l = L / (sqrtf(2.f));
  cmd.desiredThrustsN[0] = (collThrustCmd + (momentCmd.x / l) + (momentCmd.y / l) - (momentCmd.z / kappa)) / 4.f;
  cmd.desiredThrustsN[1] = (collThrustCmd - (momentCmd.x / l) + (momentCmd.y / l) + (momentCmd.z / kappa)) / 4.f;
  cmd.desiredThrustsN[2] = (collThrustCmd + (momentCmd.x / l) - (momentCmd.y / l) + (momentCmd.z / kappa)) / 4.f;
  cmd.desiredThrustsN[3] = (collThrustCmd - (momentCmd.x / l) - (momentCmd.y / l) - (momentCmd.z / kappa)) / 4.f;


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