## Project: Building a Controller

[**Rubric**](https://review.udacity.com/#!/rubrics/1534/view)**Points**

**Writeup / README**

**Implementing Controller**

**1. Implemented body rate control in C++.**

Body rate controller is implemented in QuadController (lines 108-110). It is a proportional controller, which takes body rates and outputs moments:

V3F uPuQuR = (pqrCmd - pqr)\*kpPQR;

Body rate controller takes into account the moments of inertia:

momentCmd = MOI \* uPuQuR;

**2. Implement roll pitch control in C++.**

Roll pitch control is in QuadController (lines 140-157) The controller uses the acceleration and thrust commands, in addition to the vehicle attitude to output a body rate command:

float c = collThrustCmd / mass;

float r13c = -CONSTRAIN(accelCmd[0] / c, -maxTiltAngle, maxTiltAngle);

float r23c = -CONSTRAIN(accelCmd[1] / c, -maxTiltAngle, maxTiltAngle);

…

float bxDot = kpBank \* (r13 - r13c);

float byDot = kpBank \* (r23 - r23c);

The controller should account for the non-linear transformation from local accelerations to body rates:

pqrCmd[0] = (-r21\*bxDot + r11 \* byDot) / r33;

pqrCmd[1] = (-r22\*bxDot + r12 \* byDot) / r33;

Drone's mass is involved in calculation:

float c = collThrustCmd / mass;

**3. Implement altitude controller in C++.**

Altitude controller is in QuadController (lines 189-197). The controller uses both the down position and the down velocity to command thrust:

float hVelcmd = kpPosZ \* (posZCmd - posZ) + velZCmd;

The drone's mass is accounted for calculating the thrust:

thrust = -mass \* (hAccmd - 9.81f) / r33;

The thrust includes the non-linear effects from non-zero roll/pitch angles:

float r33 = R(2, 2);

The controller contains an integrator to handle the weight non-idealities presented in scenario 4:

integratedAltitudeError += (posZCmd - posZ)\*dt;

float hAccmd = kpVelZ \* (hVelcmd - velZ) + KiPosZ\*integratedAltitudeError + accelZCmd;

**4. Implement lateral position control in C++.**

Lateral position controller is in QuadController (lines 238-252). The controller uses the local position and velocity to generate a commanded local acceleration:

V3F xyVelcmd = kpPosXY \* (posCmd - pos) + velCmd;

V3F xyAccmd = kpVelXY \* (xyVelcmd - vel) + accelCmd;

**5. Implement yaw control in C++.**

Yaw controller is in QuadController (lines 275-278). The controller is a proportional heading controller to yaw rate commands:

yawRateCmd = kpYaw \* (yawCmdNormed - yawNormed);

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

Motor commands calculation is in QuadController (lines 73-83). The thrust and moments is converted to the appropriate 4 different desired thrust forces for the moments:

cmd.desiredThrustsN[0] = (l\*F\*k - l \* Mz + k \* Mx + k \* My) / (4 \* k\*l); cmd.desiredThrustsN[1] = (l\*F\*k + l \* Mz - k \* Mx + k \* My) / (4 \* k\*l);

cmd.desiredThrustsN[2] = (l\*F\*k + l \* Mz + k \* Mx - k \* My) / (4 \* k\*l);

cmd.desiredThrustsN[3] = (l\*F\*k - l \* Mz - k \* Mx - k \* My) / (4 \* k\*l);

**Flight Evaluation**

The controller is successfully able to fly the provided test trajectory and visually passes inspection of the scenarios leading up to the test trajectory.

An example. Controller in 4th scenario:

