CONTROL OF 3D QUADROTOR

SCENARIO 2- BODY RATE AND PITCH/ROLL CONTROL

To complete this scenario, three controller modules were necessary to develop. These components include the *GenearteMotorCommands, BodyRateControl,* and *RollPitchControl*. The *GenearteMotorCommands* converts the desired collective thrust (T) and desired moments about quadrotor's body frame (M_x, M_y, and M_z) into the individual thrusts that must be provided by each of the four rotors (F_1, F_2, F_3, and F_4). These terms are related through the following equation where l and κ denote the moment arm length and reactive moment coefficients respectively.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ l & -l & l & -l \\ l & l & -l & -l \\ -\kappa & \kappa & \kappa & -\kappa \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{pmatrix} = \begin{pmatrix} T \\ M_x \\ M_y \\ M_z \end{pmatrix}$$

The solution to the above equations is as follows:

$$F_1 = \frac{(\kappa * M_x) + (\kappa * M_y) - (l * M_z) + (l * \kappa * T)}{4 * l * \kappa}$$

$$F_2 = \frac{-(\kappa * M_x) + (\kappa * M_y) + (l * M_z) + (l * \kappa * T)}{4 * l * \kappa}$$

$$F_3 = \frac{(\kappa * M_x) - (\kappa * M_y) + (l * M_z) + (l * \kappa * T)}{4 * l * \kappa}$$

$$F_4 = \frac{-(\kappa * M_x) - (\kappa * M_y) - (l * M_z) + (l * \kappa * T)}{4 * l * \kappa}$$

The code snippet below shows the implementation of the above equations in addition the constraining the computed thrusts (F_1, F_2, F_3, and F_4) to be within the limits *minMotorThurst* and *maxMotorThrust*.

```
VehicleCommand QuadControl::GenerateMotorCommands(float collThrustCmd, V3F momentCmd)
{
  ////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
  float l = L / (2*sqrt(2));
  cmd.desiredThrustsN[0] = (kappa*momentCmd[0] + kappa*momentCmd[1] - l*momentCmd[2] + kappa*l*collThrustCmd)/(4*kappa*l);
  cmd.desiredThrustsN[1] = (-kappa*momentCmd[0] + kappa*momentCmd[1]  + l*momentCmd[2] + kappa*l*collThrustCmd)/(4*kappa*l);
  cmd.desiredThrustsN[2] = (kappa*momentCmd[0] - kappa*momentCmd[1] + l*momentCmd[2] + kappa*l*collThrustCmd)/(4*kappa*l);
  cmd.desiredThrustsN[3] =  (-kappa*momentCmd[0] - kappa*momentCmd[1] - l*momentCmd[2] + kappa*l*collThrustCmd)/(4*kappa*l);

  for (int i=0; i < 4; ++i) {
    cmd.desiredThrustsN[i] = CONSTRAIN(cmd.desiredThrustsN[i], minMotorThrust, maxMotorThrust);
  }
  /////////////////////////////// END STUDENT CODE ///////////////////////////

  return cmd;
}
```

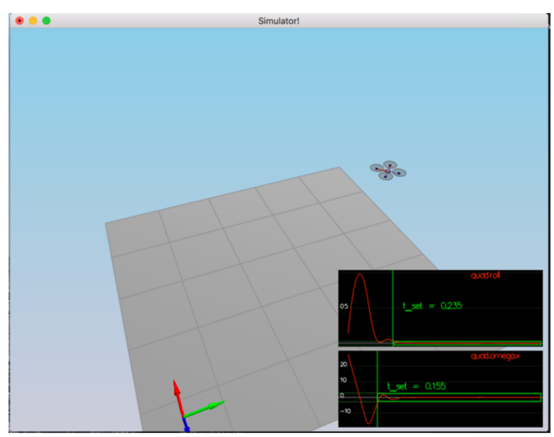
The *BodyRateControl* is a P-controller that generates the moments (M_x, M_y, and M_z) in response to error between commanded (pqrCommand) and actual (pqr) rotation rates about the quadrotor's x, y, z body-frame axis. Note how multiplication by the quadrotor's moments of inertia (I_{xx}, I_{yy}, I_{zz}) is necessary in order to convert error in rotation rates into moments.

```
V3F QuadControl::BodyRateControl(V3F pqrCmd, V3F pqr)
{
  V3F momentCmd;
  ////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
  momentCmd[0] = kpPQR.x * (pqrCmd.x - pqr.x) * Ixx;
  momentCmd[1] = kpPQR.y * (pqrCmd.y - pqr.y) * Iyy;
  momentCmd[2] = kpPQR.z * (pqrCmd.z - pqr.z) * Izz;
  /////////////////////////////// END STUDENT CODE ////////////////////////////
  return momentCmd;
}
```

The *RollPitchControl* is a P-controller that generates the commanded rotation rates (pqrCommand.x and pqrCommand.y) about the quadrotor body frame to achieve a commanded lateral acceleration. This is accomplished by determining the difference between the quadrotors required attitude components (b_x, b_y) and its actual attitude components R_{13} and R_{23} to determine the rate at which those components should change ($\dot{b}_x$,$\dot{b}_y$). Those rates are then used to derive the commanded p and q through the following relation.

```
// returns a desired roll and pitch rate
V3F QuadControl::RollPitchControl(V3F accelCmd, Quaternion<float> attitude, float collThrustCmd)
{
  V3F pqrCmd;
  Mat3x3F R = attitude.RotationMatrix_IwrtB();

  ////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
  float bx_target = CONSTRAIN(accelCmd.x * (mass / -collThrustCmd), -maxTiltAngle, maxTiltAngle);
  float by_target = CONSTRAIN(accelCmd.y * (mass / -collThrustCmd), -maxTiltAngle, maxTiltAngle);

  float bx_dot = kpBank * (bx_target - R(0, 2));
  float by_dot = kpBank * (by_target - R(1, 2));

  pqrCmd[0] =(1 / R(2,2)) * (R(1,0)*bx_dot - R(0,0)*by_dot);
  pqrCmd[1] =(1 / R(2,2)) * (R(1,1)*bx_dot - R(0,1)*by_dot);
  pqrCmd[2] = 0;
  /////////////////////////////// END STUDENT CODE ////////////////////////////

  return pqrCmd;
}
```

The image to the right shows a snapshot of the simulator executing scenario 2 immediately after the quadrotor has been stabilized for long enough to pass the performance requirements (green PASS box appears).



SCENARIO 3- POSITION, VELOCITY, AND YAW ANGLE

To complete this scenario, three controller modules were necessary to develop. These components include the *LateralPosition, AltitudeControl,* and *YawControl*. The *LateralPosition* is a PD controller with a feed-forward term determines the required lateral accelerations necessary to follow the x, y positions and velocity of the target trajectory. The “P” and “D” terms are related to the difference between the quadrotor’s position and velocity and the trajectory position and velocity. Note also how the code below constrains the commanded velocity and accelerations to be between the limits *maxSpeedXY* and *maxAccelXY*.

```
// returns a desired acceleration in global frame
  V3F QuadControl::LateralPositionControl(V3F posCmd, V3F velCmd, V3F pos, V3F vel, V3F accelCmdFF)
{
  // make sure we don't have any incoming z-component
  accelCmdFF.z = 0;
  velCmd.z = 0;
  posCmd.z = pos.z;

  // we initialize the returned desired acceleration to the feed-forward value.
  // Make sure to _add_, not simply replace, the result of your controller
  // to this variable
  V3F accelCmd;

  ////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
  // Constrain magnitude of velocity command.
  if (velCmd.magXY() > maxSpeedXY) {
    velCmd /= velCmd.magXY();
    velCmd *= maxSpeedXY;
  }

  accelCmd.x = kpPosXY * (posCmd.x - pos.x) + kpVelXY * (velCmd.x - vel.x) + accelCmdFF.x;
  accelCmd.y = kpPosXY * (posCmd.y - pos.y) + kpVelXY * (velCmd.y - vel.y) + accelCmdFF.y;
  accelCmd.z = 0;

  // Constrain magnitude of acceleration command.
  if (accelCmd.magXY() > maxAccelXY) {
    accelCmd /= accelCmd.magXY();
    accelCmd *= maxAccelXY;
  }
  /////////////////////////////// END STUDENT CODE ////////////////////////////
  return accelCmd;
}
```

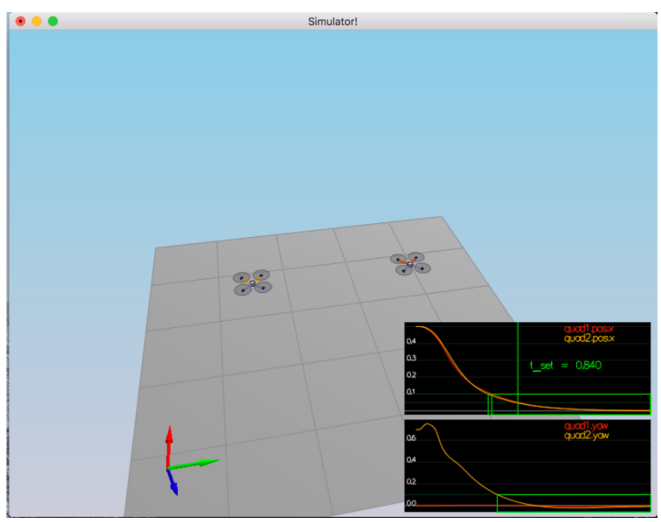
The *AltitudeControl* is a PID controller with a feed-forward term that determines the required thrust necessary to follow the z positions and velocity of the target trajectory. The “P”, “I”, and “D” terms are related to the difference between the quadrotor’s altitude and vertical velocity and the trajectory’s altitude and vertical velocity. Note also how the code below constrains the commanded vertical velocity to be between the limits *maxAscentRate* and *maxDescentRate*. The “I” term in this control is essential to the success of scenario 4.

```
float QuadControl::AltitudeControl(float posZCmd, float velZCmd, float posZ, float velZ, Quaternion<float> attitude, float accelZCmd, float dt)
{
  Mat3x3F R = attitude.RotationMatrix_IwrtB();
  float thrust = 0;

  ////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
  // Constrain velocity command to provided ascent and descent rates.
  velZCmd = CONSTRAIN(velZCmd, -maxAscentRate, maxDescentRate);
  // position error integral.
  integratedAltitudeError += (posZCmd - posZ) * dt;
  thrust = kpPosZ * (posZCmd - posZ) + kpVelZ * (velZCmd - velZ) + KiPosZ * integratedAltitudeError + accelZCmd;
  thrust = (1 / R(2,2) ) * (thrust - CONST_GRAVITY);
  thrust *= mass;
  /////////////////////////////// END STUDENT CODE ////////////////////////////
  return
```

The *YawControl* is a P-controller that generates the rotation rate command about the quadrotor's z body-frame axis to meet a desired yaw angle. The "P" term relates to the difference between the quadrotor's actual and desired yaw angles. Note how the difference in actual and desired yaw angles is constrained to be between -Pi and Pi.

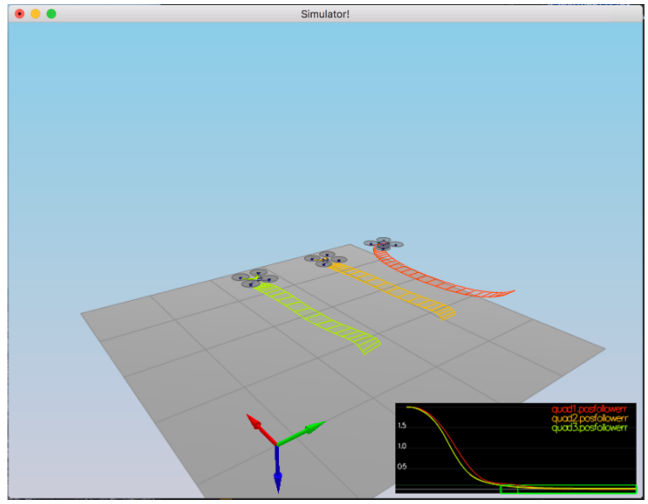
```
// returns desired yaw rate
float QuadControl::YawControl(float yawCmd, float yaw)
{
  float yawRateCmd=0;
  ////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
  float yaw_error = yawRateCmd - yaw;
  // Keep yaw_error in the interval [-pi to pi]
  if (yaw_error > F_PI) {
    yaw_error -= 2*F_PI;
  }
  if (yaw_error < -F_PI) {
    yaw_error += 2*F_PI;
  }
  yawRateCmd = kpYaw * yaw_error;
  /////////////////////////////// END STUDENT CODE ////////////////////////////

  return yawRateCmd;

}
```

The image to the right shows a snapshot of the simulator executing scenario 3 immediately after the quadrotors have been stabilized for long enough to pass the performance requirements (green PASS box appears).



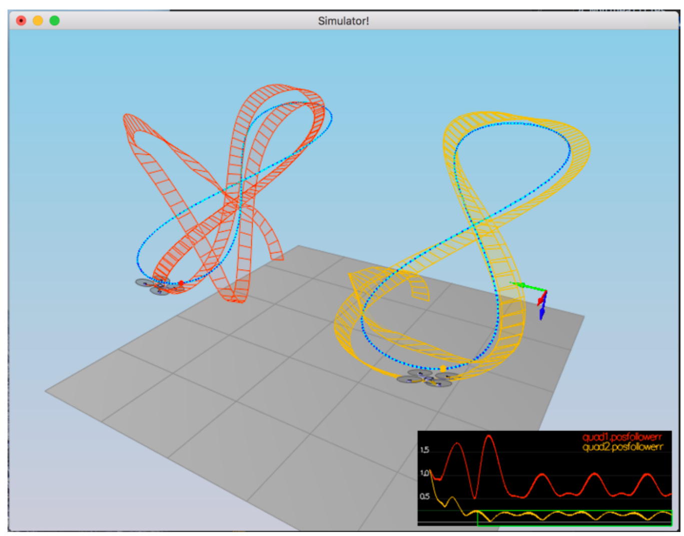
SCENARIO 4- POSITION, VELOCITY, AND YAW ANGLE

As mentioned above, the key to passing scenario 4 is the inclusion of an integral term in the altitude controller to account for nonidealities between reality and our model that leave a persistent error. The integral term eliminates persistent errors. The image to the right shows a snapshot of the simulator executing scenario 4 immediately after the quadrotors have been stabilized for long enough to pass the performance requirements (green PASS box appears).



SCENARIO 5- TRACKING TRAJECTORIES

All of the above scenarios including scenario 5 (Trajectory tracking) with the same parameter controller gains listed below. Furthermore, The image to the right shows a snapshot of the simulator executing scenario 5 immediately after the quadrotor has been tracking the Figure-8 trajectory for the required duration (green PASS box appears).




```
# Position control gains
kpPosXY = 25
kpPosZ = 35
KiPosZ = 45

# Velocity control gains
kpVelXY = 10
kpVelZ = 10

# Angle control gains
kpBank = 15
kpYaw = 2

# Angle rate gains
kpPQR = 40, 40, 5
```