Control of a 3D Quadrotor

Vishal Ramadoss

June 25, 2020

1 Introduction

To control the drone in three dimensions, the following controller architecture is adopted. A total of five controllers are implemented.

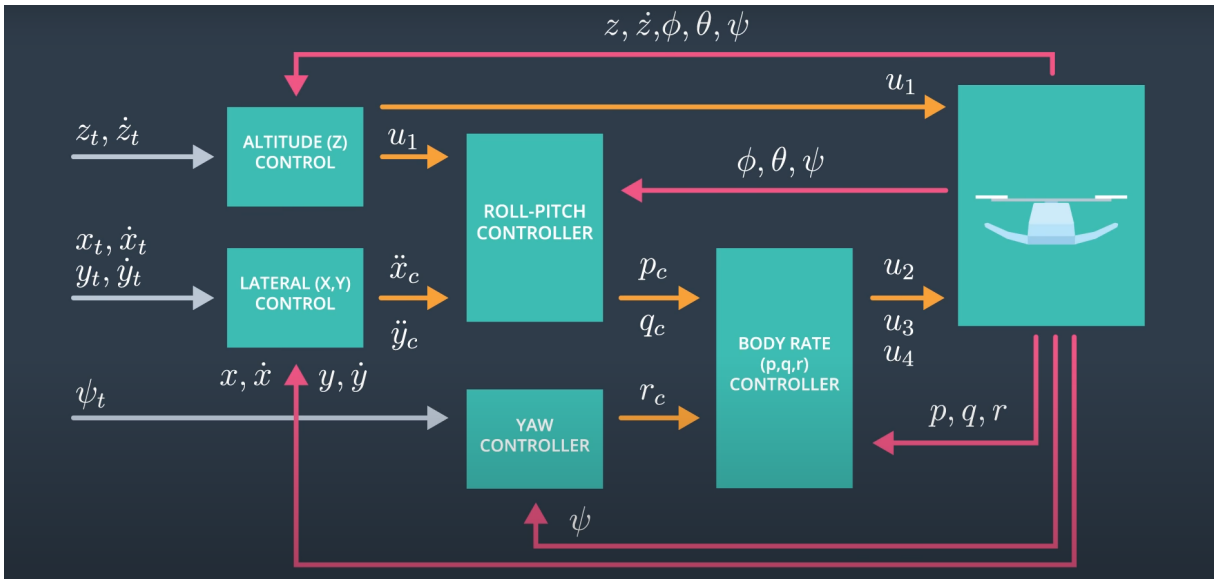

Figure 1: Block Diagram (c)Udacity.

1.1 Body rate control

Body rate controller is implemented as a P (proportional) controller. Commanded roll, pitch, and yaw rates (p, q, r), are translated into the desired commanded moments along the axis.

The control equations have the following form:

$$\dot{\omega}_p = k_p^P (p_c - p)$$

$$\dot{\omega}_q = k_p^Q (q_c - q)$$

$$\dot{\omega}_r = k_p^R (r_c - r)$$

The commanded moments M is the product of moment of inertia (I, $[kg \cdot m^2]$) and angular acceleration ($\dot{\omega}$, $[radians/s^2]$).

$$M = I\dot{\omega}$$

The implemented body rate controller is:

```
V3F momentCmd;
V3F error_rate = pqrCmd - pqr;
V3F omega_dot_desired = error_rate * kpPQR;
momentCmd = omega_dot_desired * V3F(Ixx, Iyy, Izz);
```

1.2 Roll pitch control

The roll-pitch controller a P controller responsible for commanding the roll and pitch rates in the body frame. It uses the lateral acceleration and thrust commands, in addition to the vehicle attitude to output a body rate command.

The controller first calculates the acceleration in the thrust direction in the body frame *colldes* from dividing the collective thrust by drone's mass. Then the target angles are calculated from local accelerations. Note both target angles need to be constraint within maximum title angle. Negative sign in NED coordinate system.

$$R_{13c} = b_c^x = -\ddot{x}/colldes$$
$$R_{23c} = b_c^y = -\ddot{y}/colldes$$

The desired body rates are then set using a P controller,

$$\dot{b}_c^x = k_p^{bank}(b_c^x - b^x)$$

$$\dot{b}_c^y = k_p^{bank}(b_c^y - b^y)$$

where $b^x = R_{13}$ and $b^y = R_{23}$. The given values can be converted into the angular velocities in the body frame by the matrix multiplication.

$$\begin{pmatrix} p_c \\ q_c \end{pmatrix} = \frac{1}{R_{33}} \begin{pmatrix} R_{21} & -R_{11} \\ R_{22} & -R_{12} \end{pmatrix} \begin{pmatrix} \dot{b}_c^x \\ \dot{b}_c^y \end{pmatrix}$$

```
 V3F pqrCmd;
 Mat3x3F R = attitude.RotationMatrix_IwrtB();

float coll_des = collThrustCmd / mass;

if (collThrustCmd > 0.0) {
float bx_cmd = CONSTRAIN(-accelCmd.x / coll_des, -maxTiltAngle, maxTiltAngle);
float bx_error = bx_cmd - R(0, 2);
float bx_dot_cmd = kpBank * bx_error;

float by_cmd = CONSTRAIN(-accelCmd.y / coll_des, -maxTiltAngle, maxTiltAngle);
float by_error = by_cmd - R(1, 2);
float by_dot_cmd = kpBank * by_error;

pqrCmd.x = (R(1, 0) * bx_dot_cmd - R(0, 0) * by_dot_cmd) / R(2, 2);
pqrCmd.y = (R(1, 1) * bx_dot_cmd - R(0, 1) * by_dot_cmd) / R(2, 2);

}
else {
// command no rate
pqrCmd.x = 0.0;
pqrCmd.y = 0.0;
}
pqrCmd.z = 0.0;
```

1.3 Altitude controller

In this altitude controller implementation, velocity uses a feed forward P controller,

$$\dot{z} = \dot{z}_c + k_p^{posZ}(z_c - z)$$

constrained between maximum ascent rate and maximum descent rate.

Acceleration uses a feed forward PI controller, for non-ideal case 4

$$\ddot{z} = \ddot{z}_c + k_p^{velZ}(\dot{z}_c - \dot{z}) + k_i^{posZ} \int_0^t (z_c - z)dt'$$

In the altitude controller, the thrust is controlled through the vertical acceleration.

$$m\ddot{z} = -F_{thrust}b^z + mg$$

g is in positive direction, thrust is in negative direction in the NED system. $b^z = R_{33}$ are the the last row last column element in the rotation matrix to transform back from inertial frame.

$$F_{thrust} = m(g - \ddot{z})/b^z$$

The C++ implementation code:

```
float z_error = posZCmd - posZ;
float velocityZ = kpPosZ * z_error + velZCmd;
float b_z = R(2, 2);
// Limit the ascent/descent rate
velocityZ = CONSTRAIN(velocityZ, -maxAscentRate, maxDescentRate);

integratedAltitudeError += z_error * dt;

float accelerZ = accelZCmd
+ kpVelZ * (velocityZ - velZ)
+ KiPosZ * integratedAltitudeError;

thrust = mass * (CONST_GRAVITY - accelerZ) / b_z;
```

1.4 Lateral position control

The lateral position control uses two cascaded feedforward linear P controller.

$$\dot{x} = \dot{x}_c + k_p^{PosXY}(x_c - x),\ \dot{y} = \dot{y}_c + k_p^{PosXY}(y_c - y)$$
$$\ddot{x} = \ddot{x}_c + k_p^{VelXY}(\dot{x_c} - \dot{x}),\ \ddot{y} = \ddot{y}_c + k_p^{VelXY}(\dot{x_c} - \dot{y})$$

```
V3F velo_xy = velCmd + kpPosXY * (posCmd - pos);
accelCmd = accelCmd + kpVelXY * (velo_xy - vel);
```

1.5 Yaw control

It is a linear proportional heading controller to command yaw rate. Since heading angle has to be between 0 and 2π, fmodf() function is used to constraint the input heading angle and normalized.

$$r_c = k_p^{yaw}(\psi_c - \psi)$$

```
yawCmd = fmodf(yawCmd, M_PI * 2.0);
float yaw_error = AngleNormF(yawCmd - yaw);
yawRateCmd = kpYaw * yaw_error;
```

1.6 Calculate the motor commands

Distance from vehicle origin to motors is L, the distance from motor to x or y axis is $l = L/\sqrt{2}$. The thrust from each propeller is labeled as F_1, F_2, F_3, F_4, the collective thrust is

$$F_{total} = F_1 + F_2 + F_3 + F_4$$

For roll motion, the moments generated by the 1^{st} and 3^{rd} propellers are counteracted by the moment generated by the 2^{nd} and the 4^{th} propellers.

$$\tau_x = (F_1 - F_2 + F_3 - F_4)l$$

In the same fashion, the pitch is generated by the mismatch of the moments created by 1^{st} and 2^{nd} propellers and the moment generated by the 3^{rd} and 4^{th} propellers.

$$\tau_y = (F_1 + F_2 - F_3 - F_4)l$$

Contrary to the roll and pitch, the yaw motion is executed by the mismatch of the moments generated by the propellers along the z axis by the reactive force. The moment generated by the propeller is directed opposite of its rotation and is proportional to the square of the angular velocities. The propellers 1 and 4 rotate in clockwise

thus producing the moment in the counterclockwise direction with negative moments. Propellers 2 and 3 rotate in counterclockwise thus the resulting moments are in opposite and have the positive moments.

$$\tau_z = -k_{appa}(F_1 - F_2 - F_3 + F_4)$$

Solve 4 equations for thrust from each propeller,

$$\begin{aligned}
F_1 &= (F_{total} + \tau_x/l + \tau_y/l - \tau_z/k_{appa})/4 \\
F_2 &= (F_{total} - \tau_x/l + \tau_y/l + \tau_z/k_{appa})/4 \\
F_3 &= (F_{total} + \tau_x/l - \tau_y/l + \tau_z/k_{appa})/4 \\
F_4 &= (F_{total} - \tau_x/l - \tau_y/l - \tau_z/k_{appa})/4
\end{aligned}$$

Motor command implementation in C++:

```
float lenth = L / sqrtf(2);
float F_x     = momentCmd.x / lenth;
float F_y     = momentCmd.y / lenth;
float F_z     = - momentCmd.z / kappa;
float F_total = collThrustCmd;
cmd.desiredThrustsN[0] = (F_total + F_x + F_y + F_z) / 4.f; // front left, [N]
cmd.desiredThrustsN[1] = (F_total - F_x + F_y - F_z) / 4.f; // front right
cmd.desiredThrustsN[2] = (F_total + F_x - F_y - F_z) / 4.f; // rear left
cmd.desiredThrustsN[3] = (F_total - F_x - F_y + F_z) / 4.f; // rear right
```

2 Flight Control Gain Tuning

Tuning these control gain parameters in QuadControlParams.txt.

2.1 S1: Intro

To make drone not falling to the ground, drone's gravity need to be compensated from propeller's thrust force. All we have to do is to adjust the Mass control parameter to 0.5 in QuadControlParams.txt. Scenario 1 is shown in Figure 2.

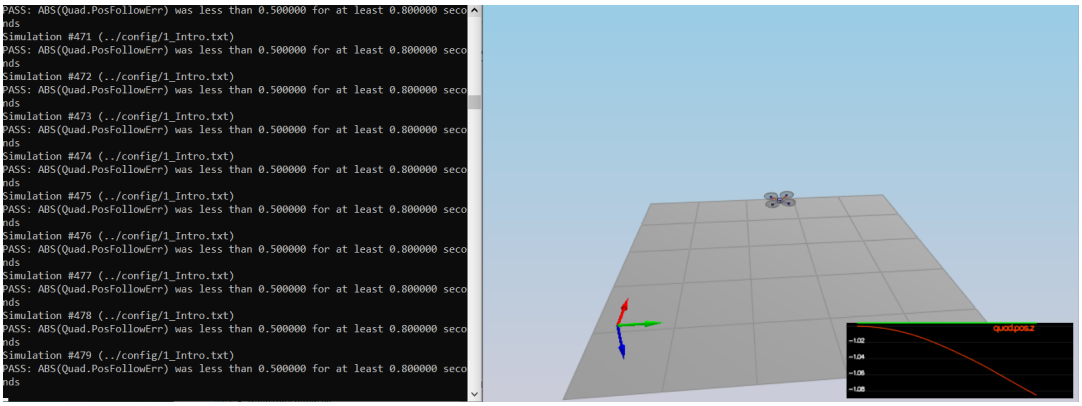

Figure 2: S1 Tuning Mass.

2.2 S2: Body rate and roll/pitch control

GenerateMotorCommands() The motor command implementation can be found in subsection 1.6. No gain tuning for this function.

BodyRateControl() The body rate control implementation can be found in subsection 1.1. Tuning kpPQR helps to get the vehicle to stop spinning quickly once the roll (omega.x) gets to zero. The default kpPQR gains works

```
# Angle rate gains
kpPQR = 23, 23, 5
```

RollPitchControl() The roll pitch control implementation can be found in subsection 1.2. Tune kpBank minimize settling time and avoid too much overshoot.

```
# Angle control gains
kpBank = 8
```

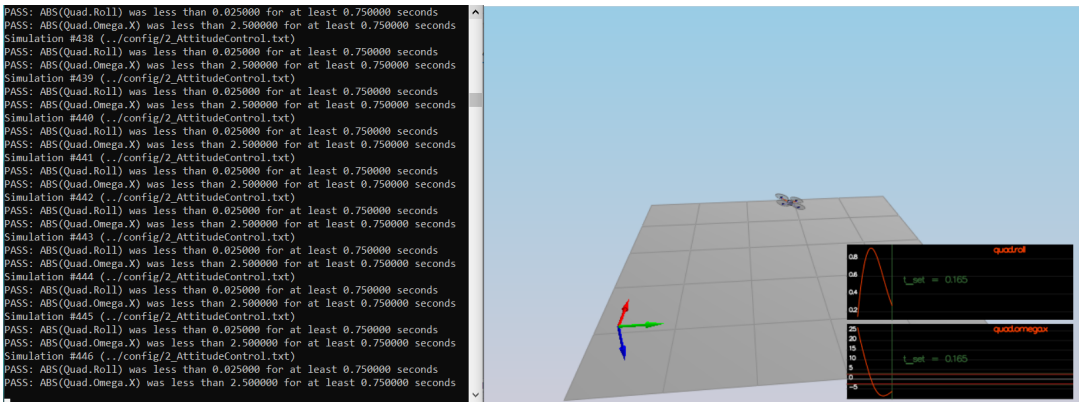

Figure 3: Scenario 2 Body rate and roll/pitch control.

2.3 S3: Position/velocity and yaw angle control

LateralPositionControl() The code for lateral position control is implemented in 1.3.

AltitudeControl() The code for altitude control is implemented in 1.4.

Tuning parameters kpPosZ, kpVelZ, kpVelXY and kpVelXY find out only kpPosXY need to be changed from 1 to 2 to make the quads go to their destination points and tracking error go down.

```
# Position control gains
kpPosZ = 1
kpPosXY = 2
# Velocity control gains
kpVelXY = 4
kpVelZ = 4
```

one quad remains rotated in yaw.

YawControl() The code for yaw control is implemented in 1.5. Tuning parameters kpYaw to 2 makes it work.

```
# Angle control gains
kpYaw = 2
```

2.4 S4: Non-idealities and robustness

Scenario 4 simulation explores the non-ideality and robustness of controllers. Three quads are configured to move and have different characteristics:

- The green quad with shifted COM

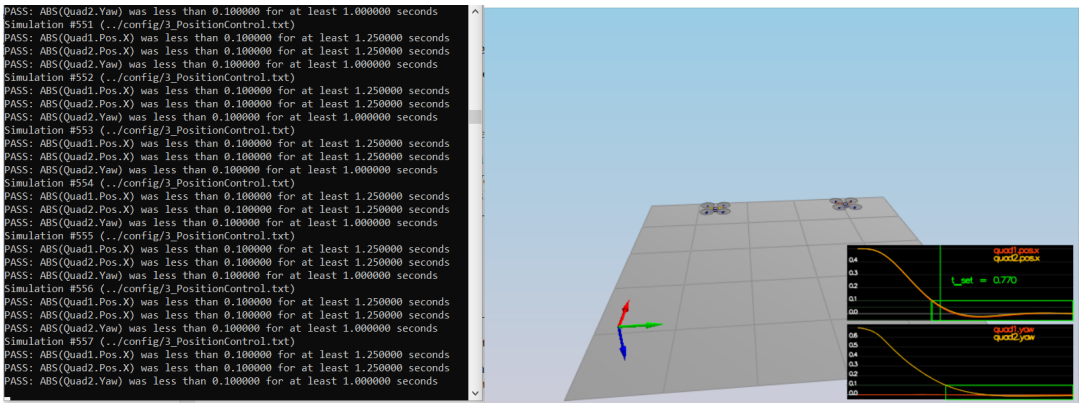

Figure 4: Scenario 3 Position/velocity and yaw angle control.

- The orange vehicle is ideal case
- The red vehicle is heavier

With controller gain parameters set from the previuos step, not all the quads moving along to the target. Adding integral control in AltitudeControl() helps with the different-mass vehicle. see Figure 5.

```
# Position control gains
kpPosXY = 3
kpPosZ = 5
KiPosZ = 10

# Velocity control gains
kpVelXY = 10
kpVelZ = 16

# Angle control gains
kpBank = 10
kpYaw = 2

# Angle rate gains
kpPQR = 61, 61, 5
```

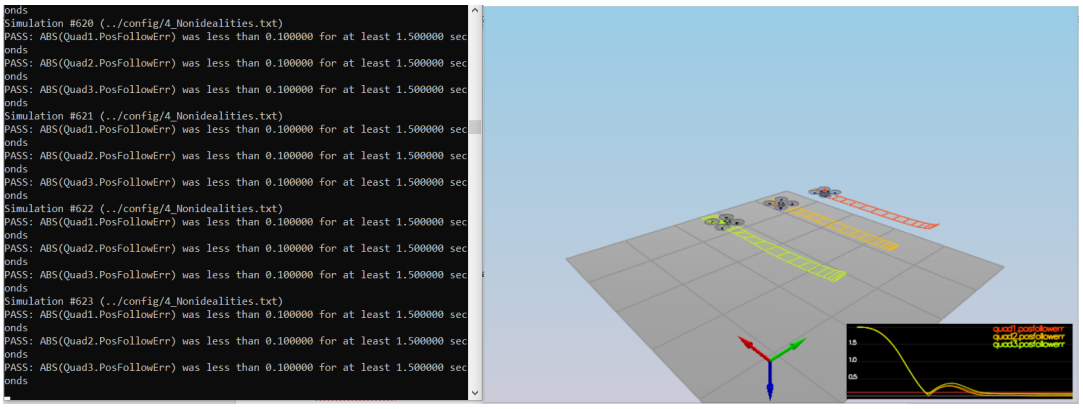

Figure 5: Scenario 4 Non-idealities and robustness.

2.5 S5: Tracking trajectories

Scenario 5 put all working parts of a controller together and test its performance on a yet another complex trajectory. Without any further gain tuning from previous scenario,it works, shown in Figure 6.

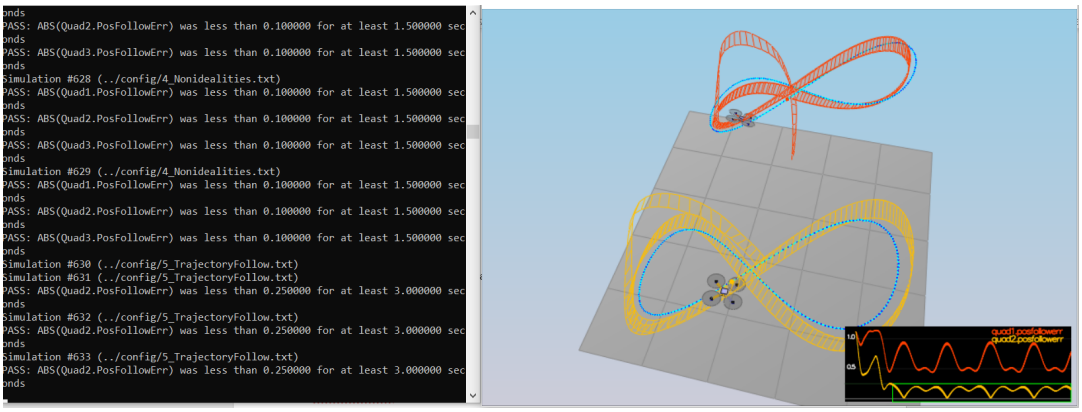

Figure 6: Scenario 5 Tracking trajectories.

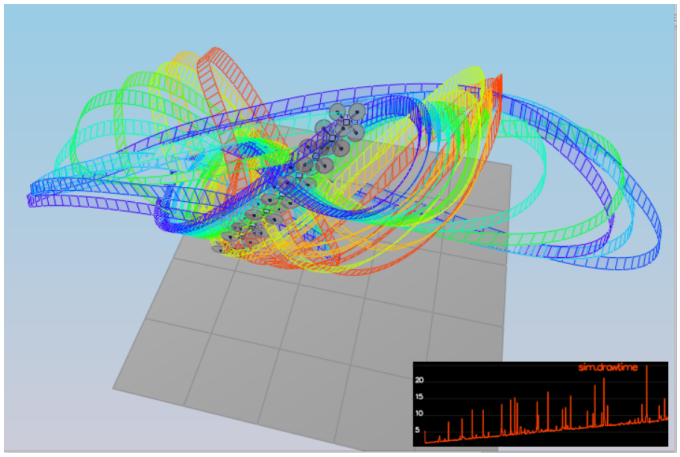

Figure 7: Bonus Scenario: Test many quards.

2.6 Bonus Scenario

Now the C++ controller is able to fly the all provided test trajectories and also bonus case. (Figure 7).

Acknowledgements

To Udacity team for this project and for the links provided
https://docs.px4.io/v1.9.0/en/config_mc/pid_tuning_guide_multicopter.html
https://github.com/darienmt/
http://www.dynsyslab.org/wp-content/papercite-data/pdf/schoellig-acc12.pdf