Project 3 - Quad Controller

Submitted on 28th Sep, 2018

By Ashok Subramaniam

The project consists of implementing the following different controls and the tuning the parameters to execute the scenarios outlined.

1. General Motor Commands.
2. Body Rate control
3. Roll-Pitch Control
4. Altitude Control
5. Lateral Position Control
6. Yaw Control

- The above are coded in the C++ source, /src/QuadControl.cpp, under the “TODO” sections earmarked for the Students to write the logic and test,

- The tuning parameters are manipulated in the /config/QuadControParams.txt

1. GenerateMotorCommands() method:

The individual motor commands of quad-rotors are computed by converting the 3-axis moments and collective thrust command in this function, for front motors (left, right) and rear motors (right, left) respectively.

```
V3F ThrustVec;
ThrustVec.x = momentCmd.x / (this->L  * 2.f * sqrtf(2.f));
ThrustVec.y = momentCmd.y / (this->L  * 2.f * sqrtf(2.f));
ThrustVec.z = momentCmd.z / (this->kappa * 4.f);

/* Compute Quadrotor thrusts */
float F1   = collThrustCmd / 4.f + ThrustVec.x + ThrustVec.y - ThrustVec.z; // Front Left
float F2  = collThrustCmd / 4.f - ThrustVec.x + ThrustVec.y + ThrustVec.z; // Front Right
float F3  = collThrustCmd / 4.f + ThrustVec.x - ThrustVec.y + ThrustVec.z; // Rear Right
float F4   = collThrustCmd / 4.f - ThrustVec.x - ThrustVec.y - ThrustVec.z; //Rear Left

cmd.desiredThrustsN[0] = F1;
cmd.desiredThrustsN[1] = F2;
cmd.desiredThrustsN[2] = F3;
cmd.desiredThrustsN[3] = F4;
```

2.BodyRateControl() method:

In this 1st order proportional (P) controller, the delta of commanded and current state of PQR is multiplied body rotation proportional factor of KpPQR and returned as moment command.

```
V3F pqr_err = pqrCmd - pqr;
V3F ubar_pqr = pqr_err * this->kpPQR;
V3F i_mofi = { this->Ixx, this->Iyy, this->Izz };
momentCmd = ubar_pqr * i_mofi;
```

3.RollPitchControl() method:

This method is 1st order controller; If the thrust is +ve, this method, converts rotation matrix from world frame to body frame reference, which then to, roll and pitch rates .

```
float tgt_bx  = 0.0;
float tgt_by  = 0.0;

if ( collThrustCmd > 0.0 ) {
  float acc = collThrustCmd / this->mass;
  tgt_bx    = - CONSTRAIN( accelCmd.x / acc, -this->maxTiltAngle, this->maxTiltAngle);
  tgt_by    = - CONSTRAIN( accelCmd.y / acc, -this->maxTiltAngle, this->maxTiltAngle);
}

pqrCmd.x = (-R(1, 0) * kpBank * (R(0,2) - tgt_bx) + R(0, 0) * kpBank * (R(1,2) - tgt_by)) / R(2,2);
pqrCmd.y = (-R(1, 1) * kpBank * (R(0,2) - tgt_bx) + R(0, 1) * kpBank * (R(1,2) - tgt_by)) / R(2,2);
```

4.AltitudeControl() method:

This PI Controller returns the thrust to control the acceleration in z-direction. First the delta between the desired z-position and the current position is fed to the P-controller to get z-velocity. The delta between the target velocity and the current z-velocity is fed-back to PI controller, get the desired acceleration, which is then used to compute the thrust.

```
float posZ_err = posZCmd - posZ;

float velZ_tgt = kpPosZ * posZ_err + velZCmd;
integratedAltitudeError += posZ_err * dt;
velZ_tgt = CONSTRAIN(velZ_tgt, -maxDescentRate, maxAscentRate);

float velZ_err = velZ_tgt - velZ;
float accCmd = kpVelZ * velZ_err + KiPosZ * integratedAltitudeError +
accelZCmd - (float)CONST_GRAVITY;

thrust = - mass * accCmd / R(2,2);
```

5. LateralPositionControl() method:

- The delta of current and desired x-y position is used to compute target velocity in the x-y plane, which is constrained to maxSpeedXY.
- The delta of current and desired x-y velocities is used to computer the acceleration in x-y plane, which is again constrained to maxAccelXY.

```
V3F posErr   = posCmd - pos;
velCmd      += kpPosXY * posErr;
if (velCmd.mag() > maxSpeedXY)
  velCmd = velCmd * maxSpeedXY / velCmd.mag();

V3F velErr   = velCmd - vel;
accelCmd    += kpVelXY * velErr;
if (accelCmd.mag() > maxAccelXY)
   accelCmd = accelCmd * maxAccelXY / accelCmd.mag();
```

6. YawControl() method:

In this method, after computing the yaw error and constrained to the range of –pi to + pi, a yaw rate is computed by a P Controller.

```
float yaw_err = yawCmd - yaw;
      yaw_err = fmodf(yaw_err, 2.0f * F_PI);
if (yaw_err > F_PI)
   yaw_err = yaw_err - 2.0f * F_PI;
else if (yaw_err < -F_PI)
   yaw_err = yaw_err + 2.0f * F_PI;
yawRateCmd = kpYaw * yaw_err;
```

Parameter Tuning:

While the logic and the subsequent coding of this project were fairly simple and straight forward, with a little bit of trying, the parameter tuning alone took some 4 to 5 painful days to arrive at the reasonable outputs;

The outputs of all of them are submitted as ".png" files in the readme in "Project Write Up" directory.

# Position control gains
kpPosXY = 2.9
kpPosZ = 4.1
KiPosZ = 90

# Velocity control gains
kpVelXY = 10
kpVelZ = 37

# Angle control gains
kpBank = 17.8
kpYaw = 3

# Angle rate gains
kpPQR = 40,40,8.9

The Scenario outputs indicating the “PASS” Status for each scenario

Scenario 1:
PASS: ABS(Quad.PosFollowErr) was less than 0.500000 for at least 0.800000 seconds

Scenario 2:
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

Scenario 3:
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

Scenario 4:
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

Scenario 5:
PASS: ABS(Quad2.PosFollowErr) was less than 0.250000 for at least 3.000000 seconds