CONTROLS

README

I. Implemented Controller

➢ The body rate controller

It is responsible for commanding the desired moments about the three axis of the body frame.

```
V3F momentCmd;


//////////////////////////// BEGIN STUDENT CODE //////////////////////////
V3F u_bar, I(Ixx, Iyy, Izz);

u_bar = kpPQR*(pqrCmd - pqr);
momentCmd = u_bar * I;


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

➢ The Roll/Pitch controller

It outputs commanded roll and pitch rates in the body frame.

```
  V3F pqrCmd;
  Mat3x3F R = attitude.RotationMatrix_IwrtB();
//////////////////////////// BEGIN STUDENT CODE //////////////////////////

  float ac = -collThrustCmd / mass;
  V3F b_c = accelCmd / ac;
  V3F b_dot, b_x_y(R(0, 2), R(1, 2), 0.0);

  //cout << "acceleration cmded :" << accelCmd.x << ","<<accelCmd.y<< endl;
  //cout << "b c:" << b_c.x <<","<<b_c.y << endl;

  b_c.x = CONSTRAIN(b_c.x, -maxTiltAngle, maxTiltAngle);
  b_c.y = CONSTRAIN(b_c.y, -maxTiltAngle, maxTiltAngle);
  b_dot = (b_x_y - b_c) * kpBank;
  //cout << "b dot:" << b_dot.x << "," << b_dot.y << endl;

  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);

  //cout << pqrCmd.x << " " << pqrCmd.y << endl;

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

return pqrCmd;
```

- **The altitude controller**

  It outputs the commanded thrust to be distributed amongst the 4 motors to get the right roll, pitch and yaw.

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

float z_dot, z_dot_dot_cmd, c, z_err;


z_err = posZCmd - posZ;
z_dot = kpPosZ * z_err + velZCmd;
z_dot = CONSTRAIN(z_dot, -maxDescentRate, maxAscentRate);

integratedAltitudeError += z_err * dt;
z_dot_dot_cmd = kpVelZ * (z_dot - velZ) + accelZCmd + KiPosZ * integratedAltitudeError;
c = (z_dot_dot_cmd - 9.81f) / R(2, 2);

thrust = -c * mass;

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

return thrust;
```

- **The lateral controller**

  It outputs the commanded horizontal accelerations both in the x and y directions.

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

float x_dot, x_dot_dot_cmd, y_dot, y_dot_dot_cmd, hor_vel_norm, hor_acc_norm;

x_dot = kpPosXY * (posCmd.x - pos.x) + velCmd.x;
y_dot = kpPosXY * (posCmd.y - pos.y) + velCmd.y;

hor_vel_norm = sqrtf(x_dot * x_dot + y_dot * y_dot);

//limit the horizontal velocity to maxSpeedXY
if (hor_vel_norm > maxSpeedXY)
{
    x_dot = x_dot * maxSpeedXY / hor_vel_norm;
    y_dot = y_dot * maxSpeedXY / hor_vel_norm;
}

x_dot_dot_cmd = kpVelXY * (x_dot - vel.x)  + accelCmdFF.x;
y_dot_dot_cmd = kpVelXY * (y_dot - vel.y)  + accelCmdFF.y;
hor_acc_norm = sqrtf(x_dot_dot_cmd * x_dot_dot_cmd + y_dot_dot_cmd * y_dot_dot_cmd);

//limit the horizontal acceleration to maxAccelXY
if (hor_acc_norm > maxAccelXY)
{
    x_dot_dot_cmd = x_dot_dot_cmd * maxAccelXY / hor_acc_norm;
    y_dot_dot_cmd = y_dot_dot_cmd * maxAccelXY / hor_acc_norm;
}

accelCmd.x = x_dot_dot_cmd;
accelCmd.y = y_dot_dot_cmd;

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

- **The yaw controller**

It outputs the commanded yaw rate in the body frame

```
float yawRateCmd=0.0f;
//////////////////////////// BEGIN STUDENT CODE ///////////////////////////
//cout << "Yaw Cmd B:" << yawCmd << endl;
yawCmd = fmodf(yawCmd, 2.0f * M_PI);
float y_err = (yawCmd - yaw);
//cout << "Yaw Cmd A:" << yawCmd << endl;
//cout << "Yaw :" << yaw << endl;

if (y_err > M_PI)
{
    y_err = y_err - 2.0f * M_PI;
}
else if (y_err < -M_PI)
{
    y_err = y_err + 2.0f * M_PI;
}

yawRateCmd = kpYaw * y_err;
//cout << "Yaw rate cmd :" << yawRateCmd << endl;
///////////////////////////// END STUDENT CODE ////////////////////////////

return yawRateCmd;
```

- **Computing motors thrusts**

This section computes the desired individual thrust for each of the 4 motors.

```
//////////////////////////// BEGIN STUDENT CODE ///////////////////////////
  float c_bar, p_bar, q_bar, r_bar, l;
  float omega_1, omega_2, omega_3, omega_4;

  l = L/sqrtf(2);
  c_bar = collThrustCmd / kappa;
  p_bar = momentCmd.x / (kappa * l);
  q_bar = momentCmd.y / (kappa * l);
  r_bar = momentCmd.z / kappa;

  omega_1 = (c_bar + p_bar - r_bar + q_bar) / 4.f;
  omega_2 = (c_bar + q_bar + r_bar - p_bar) / 4.f;
  omega_3 = (c_bar + r_bar + p_bar - q_bar) / 4.f;
  omega_4 = (c_bar - p_bar - q_bar - r_bar) / 4.f ;

    /*cmd.desiredThrustsN[0] = mass * 9.81f / 4.f; // front left
    cmd.desiredThrustsN[1] = mass * 9.81f / 4.f; // front right
    cmd.desiredThrustsN[2] = mass * 9.81f / 4.f; // rear left
    cmd.desiredThrustsN[3] = mass * 9.81f / 4.f; // rear right*/

  cmd.desiredThrustsN[0] = omega_1 * kappa; // front left
  cmd.desiredThrustsN[1] = omega_2 * kappa; // front right
  cmd.desiredThrustsN[2] = omega_3 * kappa; // rear left
  cmd.desiredThrustsN[3] = omega_4 * kappa; // rear right
//////////////////////////// END STUDENT CODE ////////////////////////////

return cmd;
```

II. Extra Challenge 1

1. A way to figure out a trajectory that has velocity information is either find a derivative of the position, in this case it is

$$f(x) = \begin{cases} x = A_x * \sin\left(\frac{2\pi t}{T_x} + w_x\right) + x_0 \\ y = A_y * \sin\left(\frac{2\pi t}{T_y} + w_y\right) + y_0 \\ z = A_z * \sin\left(\frac{2\pi t}{T_z} + w_z\right) + z_0 \end{cases}$$ and its derivative should be

$$f'(x) = \begin{cases} v_x = A_x * 2\pi/T_x * \cos(\frac{2\pi t}{T_x} + w_x) \\ v_y = A_y * 2\pi/T_y * \cos(\frac{2\pi t}{T_y} + w_y) \\ v_z = A_z * 2\pi/T_z * \cos(\frac{2\pi t}{T_z} + w_z) \end{cases}$$

Or given the timestep (dt), we can each time use the stored previous values of the position to get the velocity.

$$\begin{cases} v_x = {(x_{t+1} - x_t)}/{dt} \\ v_y = {(y_{t+1} - y)}/{dt} \\ v_z = {(z_{t+1} - z)}/{dt} \end{cases}$$

2. The velocity terms made a big difference. The tracking of the target position is better. This is because the velocity terms in the controller have the effect of predicting the future which lets the controller to know in advance what the target position is and adjust the collective thrust in consequence.

III. Extra Challenge 2

1. A way to provide more information for better tracking is the acceleration terms which are the derivatives of the velocity terms.
2. To fly the drone as quickly as possible while remaining within the threshold, we can increase the timestep slightly and increase the Ki gain slightly as well.