RBE- 502
ROBOT CONTROL

FINAL PROJECT - REPORT

HITANSHU SHAH

SHREYAS CHIGURUPATI

Problem Statement:

Design a sliding mode controller for altitude and attitude control of the Crazyflie 2.0 to enable the quad rotor to track desired trajectories and visit a set of desired waypoints.

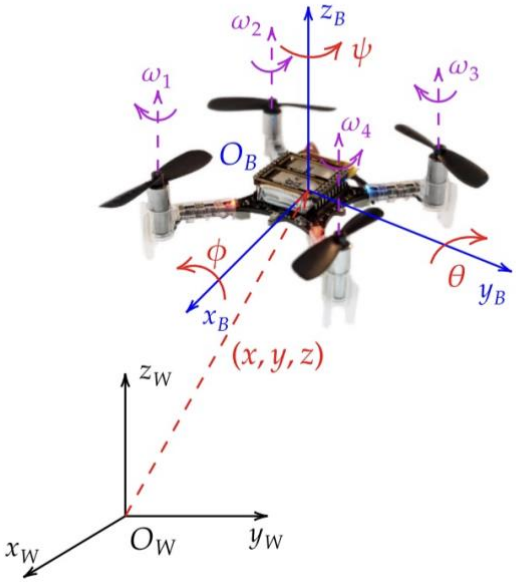


Table 1: Physical parameters of the Crazyflie 2.0 hardware platform.

Parameter	*Symbol*	*Value*
Quadrotor mass	m	27 g
Quadrotor arm length	l	46 mm
Quadrotor inertia along x-axis	I_x	16.571710×10^{-6} kg $\cdot$ m^2
Quadrotor inertia along y-axis	I_y	16.571710×10^{-6} kg $\cdot$ m^2
Quadrotor inertia along z-axis	I_z	29.261652×10^{-6} kg $\cdot$ m^2
Propeller moment of inertia	I_p	12.65625×10^{-8} kg $\cdot$ m^2
Propeller thrust factor	k_F	1.28192×10^{-8} N $\cdot$ s^2
Propeller moment factor	k_M	5.964552×10^{-3} m
Rotor maximum speed	ω_{max}	2618 rad/s
Rotor minimum speed	ω_{min}	0 rad/s

Solution:

For the final project, we were to develop a sliding mode controller for the crazyflie 2.0 quadrotor such that it tracks the desired trajectory, in the presence of external disturbances.

Part 1:

A polynomial trajectory of magnitude 5 had to be generated for the x, y, and z direction of the trajectory to be followed by the drone. The following points were given for which the drone had to reach each of these points in the given time. The robot must start from the origin being its p0 point (0,0,0) after which in 5 seconds the drone must reach the next p1 point with coordinates (0,0,1). After reaching the p1 point it had to follow the trajectory shown below to reach the other points namely p2(1,0,1), p3(1,1,1), p4(0,1,1), and p5(0,0,1).

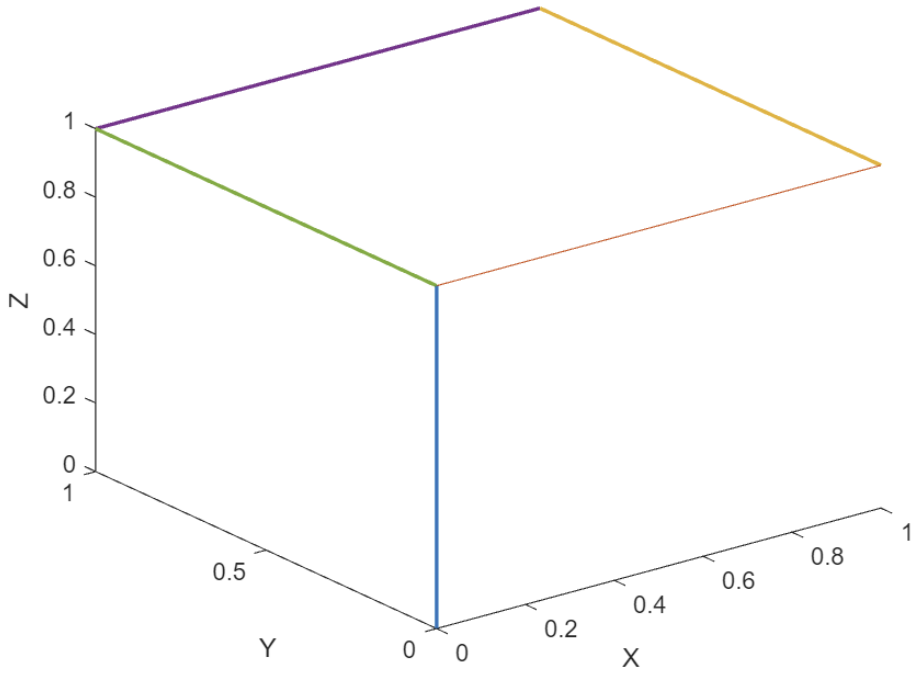


Figure 1. Desired trajectory.

The following equations were obtained for each of the coordinate of the quadrotor. These quintic trajectories include the position velocity and acceleration data that must be sent to the drone such that it can follow the desired trajectory as closely as possible. Each of the points that the quadrotor must reach acts as the initial and final points for the next calculation of position, time, velocity and acceleration.

qp1 =

$$\left[0 \quad 0 \quad \frac{2t^3}{25} - \frac{3t^4}{125} + \frac{6t^5}{3125}\right]$$

qp1_d =

$$\left[0 \quad 0 \quad \frac{6t^2}{25} - \frac{12t^3}{125} + \frac{6t^4}{625}\right]$$

qp1_dd =

$$\left[0 \quad 0 \quad \frac{12t}{25} - \frac{36t^2}{125} + \frac{24t^3}{625}\right]$$

qp2 =

$$\left[\frac{2t^3}{675} - \frac{t^4}{3375} + \frac{4664065662093477\,t^5}{59029581035870565 1712} \quad 0 \quad 1\right]$$

qp2_d =

$$\left[\frac{2t^2}{225} - \frac{4t^3}{3375} + \frac{23320328310467385\,t^4}{59029581035870565 1712} \quad 0 \quad 0\right]$$

qp2_dd =

$$\left[\frac{4t}{225} - \frac{4t^2}{1125} + \frac{23320328310467385\,t^3}{147573952589676412928} \quad 0 \quad 0\right]$$

qp3 =

$$\left[1 \quad \frac{2t^3}{675} - \frac{t^4}{3375} + \frac{4664065662093477\,t^5}{59029581035870565 1712} \quad 1\right]$$

qp3_d =

$$\left[0 \quad \frac{2t^2}{225} - \frac{4t^3}{3375} + \frac{23320328310467385\,t^4}{59029581035870565 1712} \quad 0\right]$$

qp3_dd =

$$\left[0 \quad \frac{4t}{225} - \frac{4t^2}{1125} + \frac{23320328310467385\,t^3}{147573952589676412928} \quad 0\right]$$

qp4 =

$$\left[1 - \frac{2t^3}{675} + \frac{t^4}{3375} - \frac{4664065662093477\,t^5}{59029581035870565 1712} \quad 1 \quad 1\right]$$

qp4_d =

$$\left[-\frac{2t^2}{225} + \frac{4t^3}{3375} - \frac{23320328310467385\,t^4}{59029581035870565 1712} \quad 0 \quad 0\right]$$

qp4_dd =

$$\left[-\frac{4t}{225} + \frac{4t^2}{1125} - \frac{23320328310467385\,t^3}{147573952589676412928} \quad 0 \quad 0\right]$$

qp5 =

$$\left[0 \quad 1 - \frac{2t^3}{675} + \frac{t^4}{3375} - \frac{4664065662093477\,t^5}{59029581035870565 1712} \quad 1\right]$$

qp5_d =

$$\left[0 \quad -\frac{2t^2}{225} + \frac{4t^3}{3375} - \frac{23320328310467385\,t^4}{59029581035870565 1712} \quad 0\right]$$

qp5_dd =

$$\left[0 \quad -\frac{4t}{225} + \frac{4t^2}{1125} - \frac{23320328310467385\,t^3}{147573952589676412928} \quad 0\right]$$

Part 2:

We had to design a controller for the quadrotor in this part of the project. The equations of motion provided were used to design a boundary layer sliding mode control law. These were used to track the desired z, ϕ, θ, ψ coordinates of the quadrotor, to track the desired trajectories, zd, ϕd, θd, and ψd. ϕd, θd are the desired roll and pitch angles of the drone respectively, which were calculated using the desired position trajectories xd, yd, zd.

Controller design for controlling z:

$$u_1 = (m * (g + \ddot{z}_d + \lambda_1 * \dot{e_1} + K_1 * sat_1))/(cos(\phi) * cos(\theta));$$

Controller design for controlling phi (ϕ):

$$u_2 = (-\dot{\theta} * \dot{\psi} * (I_y - I_z)) + (I_p * \Omega * \dot{\theta}) + (((-\lambda_2 * \dot{e_2}) - (K_2 * sat_2))$$

Controller design for controlling theta (θ):

$$u_3 = (-\dot{\phi} * \dot{\psi} * (I_z - I_x)) - (I_p * \Omega * \dot{\phi}) - (\lambda_3 * \dot{\theta} * I_y) - (K_3 * I_y * sat_3)$$

Controller design for controlling psi (ψ):

$$u_4 = (-\dot{\phi} * \dot{\theta} * (I_x - I_y)) - (\lambda_4 * \dot{\psi} * I_z) - (K_4 * I_z * sat_4);$$

Tuning Parameters for this controller were adjusted for the best results also we have designed the controller to be robust by adding the rho tuning parameter which will make the controller work even under external disturbances. Each tuning parameter was seen to have a different effect on the sliding mode controller and the convergence of the quadrotor. The tuning parameters and their adjusted values are shown below,

K(z) = This tuning parameter when increased makes the controller for z aggressive, for faster convergence to the sliding surface. Value used = 13.

K(phi) = This tuning parameter when increased makes the controller for phi (ϕ) aggressive, for faster convergence to the sliding surface. Value used = 180.

K(theta) = This tuning parameter when increased makes the controller for theta (θ) aggressive, for faster convergence to the sliding surface. Value used = 150.

K(phi) = This tuning parameter when increased makes the controller for psi (ψ) aggressive, for faster convergence to the sliding surface. Value used = 25.

Kp1 and Kd1 tuning parameters seem to have a direct impact on the error in the x direction. Values used are = [80 80].

Kp2 and Kd2 tuning parameters seem to have a direct impact on the error in the y direction. Values used are = [18 18].

High Kp values resulted in high destabilization which in turn called for higher control efforts to stabilize the roll and pitch. High Kd values resulted in a sluggish response that caused the drone to deviate from its path.

Higher value of lambda (λ) in sliding mode controller would result in overshoots while trying to trace the trajectory. Values used are = [5, 15, 13, 5].

rho (γ) helped reduce the chattering and make the controller more robust. Rho is known as the uncertainty parameter, A higher rho value resulted in error in the controller where a lower rho value resulted in chattering.

Part 3:

For this part of the project, we had to write a python code which would help us evaluate the performance of the quadrotor in Gazebo as compared to our desired trajectory generated in Part1.

Code Explanation:

Firstly, a ROS node "**quadrotor_control**" is initiated. Then a Publisher and a subscriber are initialized to the topics **"/crazyflie2/command/motor_speed"** and **"/crazyflie2/ground_truth/odometry"** respectively.

After this we define a function "**rpy_limits"** which helps us to limit the rpy angles within the range of [-pi to pi].

The formula for it is implemented as:

```
def rpy_limits(self,angle):
        return (angle+np.pi)%(2*np.pi)-np.pi
```

Then we define the trajectory generation function "**traj_evaluate"** which takes time as the input argument and returns the corresponding trajectory with the desired values of **X, Y, Z** coordinates and their respective velocities and accelerations (Note: The coefficients of the trajectories are generated in part 1 using the **Quintic_traj** function). As mentioned in the problem statement, there are different trajectories to be tracked for different time intervals. So, the code is written accordingly:

```
def traj_evaluate(self):
 t = self.t
 # P0 to P1
        if (t < 5):
          qp1_d = np.array([0, 0, (6*(t**5))/3125 - (3*(t**4))/125 + (2*(t**3))/25])
          dqp1_d = np.array([0, 0, (6*(t**4))/625 - (12*(t**3))/125 + (6*(t**2))/25])
          ddqp1_d = np.array([0, 0, (24*(t**3))/625 - (36*(t**2))/125 + (12*t)/25])
          x_d = qp1_d[0]
          y_d = qp1_d[1]
          z_d = qp1_d[2]
          dx_d = dqp1_d[0]
          dy_d = dqp1_d[1]
          dz_d = dqp1_d[2]
          ddx_d = ddqp1_d[0]
          ddy_d = ddqp1_d[1]
          ddz_d = ddqp1_d[2]

 # P1 to P2
        elif(t < 20):
          qp2_d = np.array([(2*(t**5))/253125-(t**4)/2025+(22*(t**3))/2025-   (8*(t**2))/81+(32*t)/81 - 47/81, 0, 1])
          dqp2_d = np.array([(2*(t**4))/50625 - (4*(t**3))/2025 + (22*(t**2))/675 - (16*t)/81 + 32/81, 0, 0])
         ddqp2_d = np.array([(8*(t**3))/50625 - (4*(t**2))/675+(44*t)/675-16/81, 0, 0])
          x_d = qp2_d[0]
          y_d = qp2_d[1]
          z_d = qp2_d[2]
          dx_d = dqp2_d[0]
          dy_d = dqp2_d[1]
          dz_d = dqp2_d[2]
          ddx_d = ddqp2_d[0]
          ddy_d = ddqp2_d[1]
          ddz_d = ddqp2_d[2]

 # P2 to P3
         elif(t < 35):
```

```
            qp3_d = np.array([1, (2*(t**5))/253125 - (11*(t**4))/10125 + (118*(t**3))/2025 - (616*(t**2))/405 + (1568*t)/81
- 7808/81, 1])
            dqp3_d = np.array([0, (2*(t**4))/50625 - (44*(t**3))/10125 + (118*(t**2))/675 - (1232*t)/405 + 1568/81, 0])
            ddqp3_d = np.array([0, (8*(t**3))/50625 - (44*(t**2))/3375 + (236*t)/675 - 1232/405, 0])
            x_d = qp3_d[0]
            y_d = qp3_d[1]
            z_d = qp3_d[2]
            dx_d = dqp3_d[0]
            dy_d = dqp3_d[1]
            dz_d = dqp3_d[2]
            ddx_d = ddqp3_d[0]
            ddy_d = ddqp3_d[1]
            ddz_d = ddqp3_d[2]

 # P3 to P4
        elif(t < 50):
            qp4_d = np.array([- (2*(t**5))/253125 + (17*(t**4))/10125 - (286*(t**3))/2025 + (476*(t**2))/81 - (9800*t)/81 +
80000/81, 1, 1])
            dqp4_d = np.array([- (2*(t**4))/50625 + (68*(t**3))/10125 - (286*(t**2))/675 + (952*t)/81 - 9800/81, 0, 0])
            ddqp4_d = np.array([- (8*(t**3))/50625 + (68*(t**2))/3375 - (572*t)/675 + 952/81, 0, 0])
            x_d = qp4_d[0]
            y_d = qp4_d[1]
            z_d = qp4_d[2]
            dx_d = dqp4_d[0]
            dy_d = dqp4_d[1]
            dz_d = dqp4_d[2]
            ddx_d = ddqp4_d[0]
            ddy_d = ddqp4_d[1]
            ddz_d = ddqp4_d[2]


# P4 to P5
        elif(t <= 65):
            qp5_d = np.array([0, -(2*(t**5))/253125 + (23*(t**4))/10125 - (526*(t**3))/2025 + (1196*(t**2))/81 -
(33800*t)/81 + 380081/81, 1])
            dqp5_d = np.array([0, -(2*(t**4))/50625 + (92*(t**3))/10125 - (526*(t**2))/675 + (2392*t)/81 - 33800/81, 0])
            ddqp5_d = np.array([0, -(8*(t**3))/50625 + (92*(t**2))/3375 - (1052*t)/675 + 2392/81, 0])
            x_d = qp5_d[0]
            y_d = qp5_d[1]
            z_d = qp5_d[2]
            dx_d = dqp5_d[0]
            dy_d = dqp5_d[1]
            dz_d = dqp5_d[2]
            ddx_d = ddqp5_d[0]
            ddy_d = ddqp5_d[1]
            ddz_d = ddqp5_d[2]


        else:
            x_d = 0
            y_d = 0
            z_d = 1
            dx_d = 0
```

```
        dy_d = 0
        dz_d = 0
        ddx_d = 0
        ddy_d = 0
        ddz_d = 0

        return x_d, y_d, z_d, dx_d, dy_d, dz_d, ddx_d, ddy_d, ddz_d
```

So, now that we have the desired trajectory, we have modified the function **"smc_control"** to implement our control laws designed in part 2. The code for it is as shown below:

First, we need to initialize the physical parameters given in the problem:

```
def smc_control(self, xyz, xyz_dot, rpy, rpy_dot):
  global omega
  # Physical parameters (given)
        m = 27*1e-3
        g = 9.8
        l = 46*1e-3
        Ix = 16.5710*1e-6
        Iy = 16.5710*1e-6
        Iz = 29.261652*1e-6
        Ip = 12.65625*1e-8
        Kf = 1.28192*1e-8
        Km = 5.964552*1e-3
```

Then we need to obtain the desired values by evaluating the corresponding trajectories:

```
x_d, y_d, z_d, dx_d, dy_d, dz_d, ddx_d, ddy_d, ddz_d = self.traj_evaluate()
```

After this we need to assign the generalized coordinates that are taken from the gazebo messages to our variables.

```
        x=xyz[0,0]
        y=xyz[1,0]
        z=xyz[2,0]
        dx=xyz_dot[0,0]
        dy=xyz_dot[1,0]
        dz=xyz_dot[2,0]
        phi=rpy[0,0]
        theta=rpy[1,0]
        psi=rpy[2,0]
        dphi=rpy_dot[0,0]
        dtheta=rpy_dot[1,0]
        dpsi=rpy_dot[2,0]
```

Then we have setup the tuning variables:

```
        # Tuning Parameters
        Kp = [80, 80]
        Kd = [15, 18]
        K = np.array([13, 180, 150, 25])
       lamb = np.array([5, 15, 13, 5])
```

```
Phi = 0.9
```

Here, **Phi** corresponds to the boundary layer designed for the SMC. Now, we must implement the control laws that are designed in part 2.
Relation required to calculate theta and phi desired are given in the problem:

```
Fx = m * (-Kp[0]*(x-x_d) - Kd[0]*(dx-dx_d) + ddx_d)
Fy = m * (-Kp[1]*(y-y_d) - Kd[1]*(dy-dy_d) + ddy_d)
```

Notice that we are using the saturation function instead of a better sign to remove the chattering problem caused. The Saturation value are defined individually for each surface that we defined (S1, S2, S3, S4).

```
## Control Law
 # Control input u1
 e1 = z_d - z
 de1 = dz_d - dz
 S1 = de1 + (lamb[0]*e1)          # Sliding surface for u1
 sat1 = min(max(S1/Phi, -1), 1)   # Boundary condition for sliding surface S1
 u1 = (m * (g+ ddz_d + lamb[0]*de1 + K[0]*sat1)) /(cos(phi)*cos(theta))

 phi_d = asin(-Fy/u1)
 dphi_d = 0

 # Control input u2
 e2 =  self.rpy_limits(phi - phi_d)
 de2 = self.rpy_limits(dphi - dphi_d)
 S2 = de2 + (lamb[1]*e2)          # Sliding surface for u2
 sat2 = min(max(S2/Phi, -1), 1)   # Boundary condition for sliding surface S2
 u2 = (-dtheta*dpsi*(Iy-Iz)) + (Ip*omega*dtheta) + (((-lamb[1]*de2) - (K[1]*sat2))*Ix)

 theta_d = asin(Fx/u1)            # Given
 dtheta_d = 0                     # Given

 # Control input u3
 e3 = self.rpy_limits(theta - theta_d)
 de3 = self.rpy_limits(dtheta - dtheta_d)
 S3 = de3 + (lamb[2]*e3)          # Sliding surface for u3
 sat3 = min(max(S3/Phi, -1), 1)   # Boundary condition for sliding surface S3
 u3 = (-dphi*dpsi*(Iz-Ix)) - (Ip*omega*dphi) - (lamb[2]*dtheta*Iy) -(K[2]*Iy*sat3)

 # Control input u4
 psi_d = 0                  # Given
 dpsi_d =0                  # Given
 e4 = self.rpy_limits(psi - psi_d)
 de4 = self.rpy_limits(dpsi - dpsi_d)
```

```
S4 = de4 + (lamb[3]*e4)          # Sliding surface for u4
sat4 = min(max(S4/Phi, -1), 1)   # Boundary condition for sliding surface S4
u4 = (-dphi*dtheta*(Ix-Iy)) - (lamb[3]*dpsi*Iz) - (K[3]*Iz*sat4)
```

Now, all the control laws are stored in a vector **u**.

```
u = np.array([[u1], [u2], [u3], [u4]])
```

After this step, we need to calculate the rotor speeds in order to send the command to gazebo simulator to move the robot. This can be done by using the **allocation matrix** given in the problem statement, which gives the relation between rotor speeds and the control inputs.

```
# Allocation Matrix
A = np.array(([1/(4*kf), -sqrt(2)/(4*kf*l), -sqrt(2)/(4*kf*l), 1/(4*km*kf)],
              [1/(4*kf), -sqrt(2)/(4*kf*l),  sqrt(2)/(4*kf*l),  1/(4*km*kf)],
              [1/(4*kf),  sqrt(2)/(4*kf*l),  sqrt(2)/(4*kf*l), -1/(4*km*kf)],
              [1/(4*kf),  sqrt(2)/(4*kf*l), -sqrt(2)/(4*kf*l),  1/(4*km*kf)]))
```

After this we use the numpy library function ‘matmul’ to multiply the **A** matrix with the control input vector **u** to obtain the **square** of rotor speeds. These are then fed into the **motor_vel** vector which sends the data to **mav_msgs** which in turn commands the robot to move with the calculated speeds (**u1**) and moments (**u2, u3, u4**).

```
w = np.matmul(A,u)
 motor_vel = np.zeros([4,1])
 motor_vel[0,0] = sqrt(w[0,0])
 motor_vel[1,0] = sqrt(w[1,0])
 motor_vel[2,0] = sqrt(w[2,0])
 motor_vel[3,0] = sqrt(w[3,0])
```

Then these velocities are then converted to twists in the **odom_callback** functions which are used to calculate the transformation matrix to determine the real time pose of the quad rotor stored in separate arrays namely **x_series, y_series, z_series, t_series** and then calling back our **smc_control** function to pass the current (run time values of the generalized coordinates).

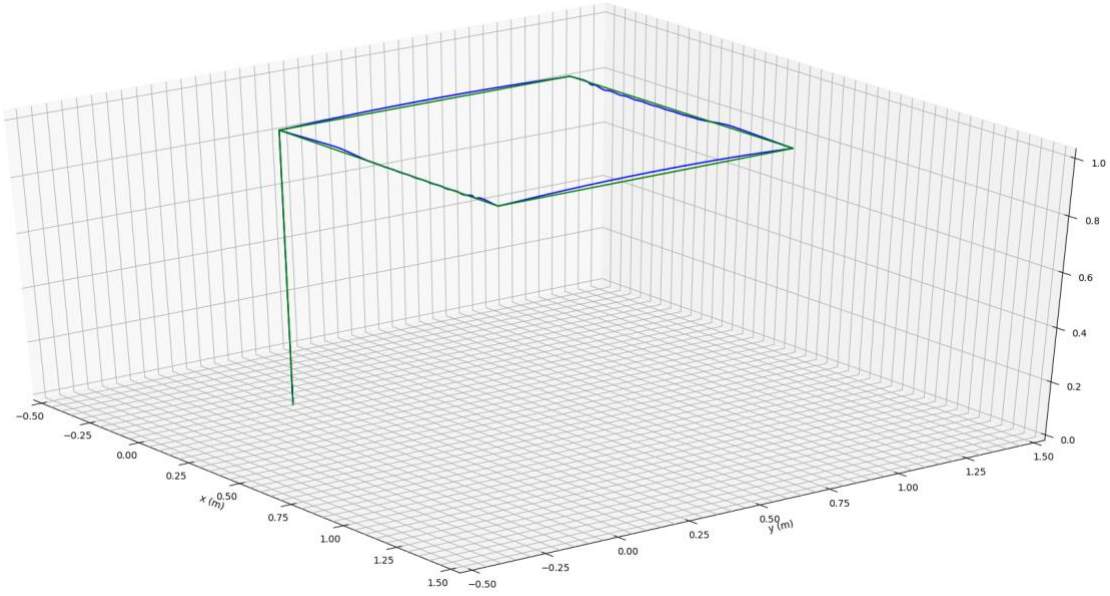
Finally, the **save_data** function collects all the X, Y, Z values that are stored in arrays and saves them in a file named **‘log.pkl’**, which are then used in part 4 to plot the desired vs actual trajectories.

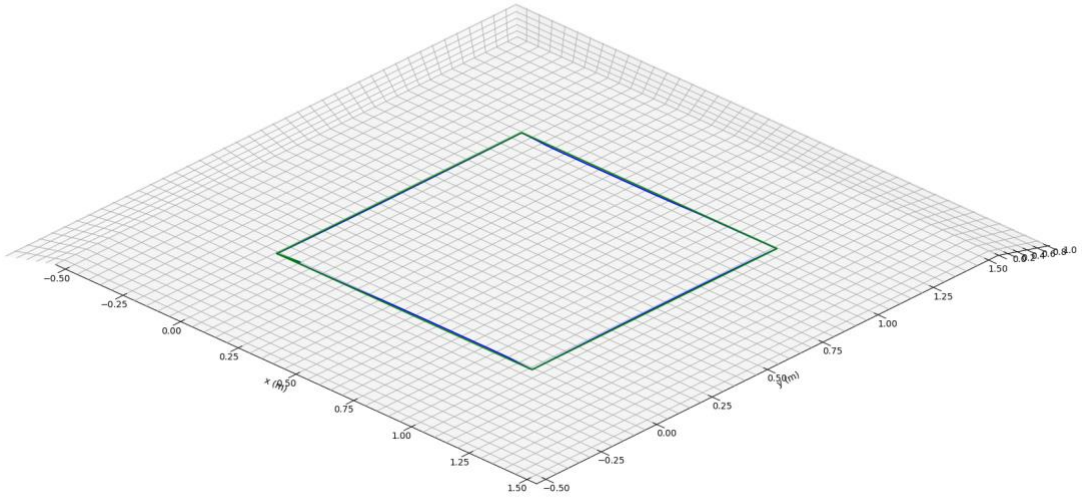
We have used the **os** library to open the visualize scripts within the main function. This line of code conveniently runs the script without us manually running the code.

```
os.system("rosrun project visualize.py")
```

Part 4:

After testing the performance in gazebo as shown in the demonstration video attached. The actual trajectory and the desired trajectory of the quadrotor is saved into a **log.pkl** file under the scripts directory. The code provided in the project was used to compare our performance. The plots generated are shown below, we can see that the quadrotor successfully follows the desired trajectory and visits all the points that we want it too. The velocities plots of the rotors are also shown below which are well within the desired limit for the same.




−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
x (m)
y (m)

Sliding Mode Controller design

generalized coordinates:-

$$q = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T$$
$$q_1\ q_2\ q_3\ q_4\ q_5\ q_6\ q_7\ q_8\ q_9\ q_{10}\ q_{11}\ q_{12}$$

For 'z' (u_1) :-

Step 1:-

selecting the sliding surface for 'z' coordinate as

$$s_1 = \dot{e} + \lambda_1 e \qquad e = z_d - z\ ,\ \dot{e} = \dot{z}_d - \dot{z}$$

$$\dot{s}_1 = \ddot{e} + \lambda_1 \dot{e} \longrightarrow \text{Contain } u_1 \text{ \& } s \to 0,\ e, \dot{e} \to 0$$

$\therefore$ Valid surface equation.

Step 2:-

design u such that $s_1 \dot{s}_1 \leq -k|s|\ ,\ k > 0$

$$s_1 \dot{s}_1 = s_1 \left[\ddot{z} - \ddot{z}_d + \lambda_1 (\dot{z} - \dot{z}_d) \right] \longrightarrow (1)$$

$$\ddot{z} = \frac{1}{m} (\cos q_4 \cos q_5) u_1 - g \longrightarrow \text{given}$$

$$\ddot{z} = -g + \frac{1}{m} (c_4 c_5) u_1$$

control affine form

$$\left(\ddot{z} = f(z, \dot{z}) + g(z, \dot{z})\, u \right)$$

(1) $\Rightarrow$

$$s_1 \dot{s}_1 = s_1 \left[f(z, \dot{z}) + g(z, \dot{z})\, u - \ddot{z}_d + \lambda_1 \dot{e} \right]$$

where,

$$f(z, \dot{z}) = -g$$

$$g(z, \dot{z}) = \frac{1}{m} \cos q_4 \cos q_5$$

$\rho = 0$

Step 3:

$$u_1 = \frac{-f(z,\dot{z}) + \ddot{z}_d - \lambda_1 \dot{e} + u_{r_1}}{g(z,\dot{z})}$$

$$u_{r_1} = -(\rho + k_1)\, \mathrm{Sgn}(s_1)$$

Apply u_1 to $s_1\dot{s}_1$

$$s_1\dot{s}_1 = s_1\left[f(z,\dot{z}) + g(z,\dot{z})\,u - \ddot{z}_d + \lambda_1(\dot{z} - \dot{z}_d)\right]$$

$$= s_1 g(z,\dot{z})\left[\frac{f(z,\dot{z}) - \ddot{z}_d + \lambda\dot{e}}{g(z,\dot{z})} + u\right]$$

$$= s_1\, g(z,\dot{z}) \cdot [u_{r_1}]$$

$$= s_1\, g(z,\dot{z})\left[-K_1\, \mathrm{Sgn}(s_1)\right]$$

$$s\dot{s}_1 \le -\underbrace{k_0\, g(z,\dot{z})}_{K}\, |s_1| \qquad \left[\because \mathrm{Sgn}(s_1)\cdot s_1 = |s_1|\right]$$

$\Rightarrow$ Sliding condition is satisfied.

$$u_1 = \frac{g + \ddot{z}_d - \lambda_1\dot{e} - k_1\, \mathrm{Sgn}(s_1)}{(1/m\cos\phi\cos\theta)}$$

For 'ϕ' (u_2) :-

Step 1:-

Selecting the sliding surface for 'ϕ' coordinate as

$$e_2 = \phi - \phi_d$$
$$\dot{e}_2 = \dot{\phi} - \dot{\phi}_d$$

$$\Omega = \omega_1 - \omega_2 + \omega_3 - \omega_4$$
$$\dot{\phi}_d = 0 \qquad \phi_d = \sin^{-1}\left[\frac{F_x}{u_1}\right]$$
$$\ddot{\phi}_d = 0$$

$$s_2 = e_2 + \lambda_2 e_2$$
$$\dot{s}_2 = \ddot{e}_2 + \lambda_2 \dot{e}_2 \longrightarrow \text{Contain } u_2 \text{ \& } s \to 0,\ e, \dot{e} \to 0$$
$$\therefore \text{Valid Surface equation}$$

Step 2:-

design u_2 such that $s_2 \dot{s}_2 \leq -k|s_2|$, $k > 0$

$$s_2 \dot{s}_2 = s_2\left[\ddot{\phi} - \ddot{\phi}_d + \lambda_2(\dot{\phi} - \dot{\phi}_d)\right] \longrightarrow (1)$$

$$\ddot{\phi} = \left(\dot{\theta}\dot{\psi}\,\frac{i_y - i_z}{i_x} - \frac{i_p}{i_x}\,\Omega\,\dot{\theta}\right) + \frac{1}{i_x}\,u_2$$

(given)

control affine form

$$\left(\ddot{\phi} = f_2(\phi, \dot{\phi}) + g_2(\phi, \dot{\phi})\,u_2\right)$$

(1) $\Rightarrow$

$$s_1 \dot{s}_1 = s_1\left[f_2(\phi, \dot{\phi}) + g_2(\phi, \dot{\phi})\,u_2 - \ddot{\phi}_d + \lambda \dot{e}_2\right]$$

where,

$$f(\phi, \dot{\phi}) = \dot{\theta}\dot{\psi}\,\frac{i_y - i_z - i_p\,\Omega\,\dot{\theta}}{i_x}$$
$$g(\phi, \dot{\phi}) = \frac{1}{i_x}$$

$\rho = 0$

Step 3:-

$$u_2 = \frac{-f_2(\phi,\dot{\phi}) + \ddot{\phi}_d - \lambda_2 \dot{e} + u_{r_2}}{g_2(\phi,\dot{\phi})}$$

$$u_{r_2} = -(\rho + K_2 \operatorname{Sgn}(s_2))$$

Apply u_2 to $s_2\dot{s}_2$

$$s_2\dot{s}_2 = s_2\left[f(\phi,\dot{\phi}) + g(\phi,\dot{\phi})\, u_2 - \ddot{\phi}_d + \lambda_2(\dot{\phi} - \dot{\phi}_d) \right]$$

$$= s_2 g_2(\phi,\dot{\phi}) \left[\frac{f_2(\phi,\dot{\phi}) - \ddot{\phi}_d + \lambda_2 e}{g_2(\phi,\dot{\phi})} + u_2 \right]$$

$$= s_2 g_2(\phi,\dot{\phi}) \cdot [u_{r_2}]$$

$$= s_2 g_2(\phi,\dot{\phi}) \left[-K_2 \operatorname{Sgn}(s_2) \right]$$

$$s_2\dot{s}_2 \leq -\underbrace{K_0 g(\phi,\dot{\phi})}_{K} |s_2| \qquad \left[\because \operatorname{Sgn}(s_2) \cdot s_2 = |s_2| \right]$$

$\Rightarrow$ Sliding condition is satisfied.

$$u_2 = -\left[\dot{\theta}\,\dot{\psi}\,(I_y - I_z) - I_p \Omega \dot{\theta} \right] + \left(\ddot{\phi}_d - \lambda \dot{e} - K_2 \operatorname{Sgn}(s_2) \right) I_x$$

($\ddot{\phi}_d \to 0$)

For 'θ' (u_3) :-

Step 1:-

Selecting the sliding surface for 'θ' coordinate as

$$e_3 = \theta - \theta_d$$
$$\dot{e}_3 = \dot{\theta} - \dot{\theta}_d$$

$$\Omega = \omega_1 - \omega_2 + \omega_3 - \omega_4$$
$$\dot{\theta}_d = 0 \qquad \theta_d = \sin^{-1}\left[\frac{-F_y}{u_1}\right]$$
$$\ddot{\theta}_d = 0$$

$$S_3 = e_3 + \lambda_3 e_3$$
$$\dot{S}_3 = \ddot{e}_3 + \lambda_3 \dot{e}_3 \longrightarrow \text{Contain } u_3 \text{ \& } S \to 0,\ e, \dot{e} \to 0$$

$\therefore$ Valid Surface equation

Step 2:-

design u_3 such that $S_3 \dot{S}_3 \leq -K|S_3|, \quad K > 0$

$$S_3 \dot{S}_3 = S_3\left[\ddot{\theta} - \ddot{\theta}_d + \lambda_3(\dot{\theta} - \dot{\theta}_d)\right] \longrightarrow (1)$$

(given)

$$\ddot{\theta} = \frac{q_{10} \cdot q_{12} \cdot i_z - i_x + i_p \cdot \Omega q_{10}}{i_y} + \frac{u_3}{i_y}$$

Control affine form

$$\left(\ddot{\theta} = f_3(\theta, \dot{\theta}) + g_3(\theta, \dot{\theta})\, u_3\right)$$

(1) $\Rightarrow$

$$S_3 \dot{S}_3 = S_3\left[f_3(\theta, \dot{\theta}) + g_3(\theta, \dot{\theta})\, u_3 - \ddot{\theta}_d + \lambda_3 \dot{e}_3\right]$$

where,

$$f(\theta, \dot{\theta}) = \frac{q_{10} \cdot q_{12} \cdot i_z - i_x + i_p \cdot \Omega q_{10}}{i_y}$$

$$g(\theta, \dot{\theta}) = \frac{1}{i_y}$$

$\rho = 0$

Step 3:-

$$u_3 = \frac{-f_3(\theta,\dot\theta) + \ddot\theta_d - \lambda_3 \dot e_3 + u_{r_3}}{g_3(\theta,\dot\theta)}$$

$$u_{r_3} = -(\rho + K_3 \operatorname{Sgn}(s_3))$$

Apply u_3 to $s_3\dot s_3$

$$s_3\dot s_3 = s_3\left[f_3(\theta,\dot\theta) + g_3(\theta,\dot\theta)\,u_3 - \ddot\theta_d + \lambda_3(\dot\theta - \dot\theta_d)\right]$$

$$= s_3 g_3(\theta,\dot\theta)\left[\frac{f_3(\theta,\dot\theta) - \ddot\theta_d + \lambda_3 e_3}{g_3(\theta,\dot\theta)} + u_3\right]$$

$$= s_3 g_3(\theta,\dot\theta)\,[u_{r_3}]$$

$$= s_3 g_3(\theta,\dot\theta)\left[-K_3 \operatorname{Sgn}(s_3)\right]$$

$$s_3\dot s_3 \le -\underbrace{K_0\, g_3(\theta,\dot\theta)}_{K}\,|s_3| \qquad \left[\because \operatorname{Sgn}(s_3)\cdot s_3 = |s_3|\right]$$

$\Rightarrow$ Sliding condition is satisfied.

$$u_3 = -\dot\phi\,\dot\psi\,(i_z - i_x) - i_p \Omega \dot\phi - \lambda_3 \dot\theta \cdot i_y - K_3 i_y \operatorname{Sgn}(s_3)$$

For 'ψ' (u_4) :-

Step 1:- selecting the sliding surface for 'ψ' coordinate as

$$e_4 = \psi - \psi_d$$
$$\dot{e}_4 = \dot{\psi} - \dot{\psi}_d$$

$$\Omega = \omega_1 - \omega_2 + \omega_3 - \omega_4$$
$$\dot{\psi}_d = 0 \qquad \psi_d = 0$$
$$\ddot{\psi}_d = 0$$

$$S_4 = e_4 + \lambda_4 e_4$$
$$\dot{S}_4 = \ddot{e}_4 + \lambda_4 \dot{e}_4 \longrightarrow \text{Contain } u_4 \text{ \& } S \to 0,\ e, \dot{e} \to 0$$

$\therefore$ Valid Surface equation

Step 2:-

design u_4 such that $S_4 \dot{S}_4 \leq -k|S_4|, \quad k > 0$

$$S_4 \dot{S}_4 = S_4 \left[\ddot{\psi} - \ddot{\psi}_d + \lambda_4 (\dot{\psi} - \dot{\psi}_d) \right] \longrightarrow (1)$$

$$\ddot{\psi} = \frac{q_{10} q_{11} (i_x - i_y)}{i_z} + \frac{u_4}{i_z}$$

($\frac{q_{10} q_{11} (i_x - i_y)}{i_z}$: given)

control affine form

$$\ddot{\psi} = f_4(\psi, \dot{\psi}) + g_4(\psi, \dot{\psi})\, u_4$$

(1) $\Rightarrow$

$$S_4 \dot{S}_4 = S_4 \left\{ f_4(\psi, \dot{\psi}) + g_4(\psi, \dot{\psi})\, u_4 - \ddot{\psi}_d + \lambda_4 \dot{e}_4 \right\}$$

where,

$$f(\psi, \dot{\psi}) = \frac{q_{10} q_{11} (i_x - i_y)}{i_z}$$
$$g(\psi, \dot{\psi}) = \frac{1}{i_z}$$

$\rho = 0$

Step 3:-

$$u_4 = \frac{-f_4(\varphi,\dot{\varphi}) + \ddot{\varphi}_d - \lambda_4 \dot{e}_4 + u_{r_4}}{g_4(\varphi,\dot{\varphi})}$$

$$u_{r_4} = -(\rho + k_4 \operatorname{Sgn}(s_4))$$

Apply u_4 to $s_4 \dot{s}_4$

$$s_4 \dot{s}_4 = s_4 \left[f_4(\varphi,\dot{\varphi}) + g_4(\varphi,\dot{\varphi})\, u_4 - \ddot{\varphi}_d + \lambda_4(\dot{\varphi} - \dot{\varphi}_d) \right]$$

$$= s_4 g_4(\varphi,\dot{\varphi}) \left[\frac{f_4(\varphi,\dot{\varphi}) - \ddot{\varphi}_d + \lambda_4 e_4}{g_4(\varphi,\dot{\varphi})} + u_4 \right]$$

$$= s_4 g_4(\varphi,\dot{\varphi}) \cdot [u_{r_4}]$$

$$= s_4 g_4(\varphi,\dot{\varphi}) \left[-k_4 \operatorname{Sgn}(s_4) \right]$$

$$s_4 \dot{s}_4 \le -\underbrace{k_0\, g_4(\varphi,\dot{\varphi})}_{K} |s_4| \qquad \left[\because \operatorname{Sgn}(s_4)\, s_4 = |s_4| \right]$$

$\Rightarrow$ Sliding condition is satisfied.

$$u_4 = -\dot{\phi}\,\dot{\theta}\,(i_x - i_y) - \lambda_4 \dot{\varphi}\, i_z - k_4 i_z \operatorname{Sgn}(s_4)$$