Robot Control (RBE502)

Final Project:

Robust Trajectory Tracking for Quadrotor UAVs using Sliding Mode Control

Project By:

Swapneel Dhananjay Wagholikar (WPI ID: 257598983)

Shounak Sheshadri Naik (WPI ID:728556739)

Part1: Desired trajectory for Generalized Coordinates

The quintic polynomial for the quadrotor's desired trajectory is defined as follows:

$$q = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

To determine the desired acceleration and velocity, this equation is differentiated:

$$\dot{q} = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4$$

$$\ddot{q} = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3$$

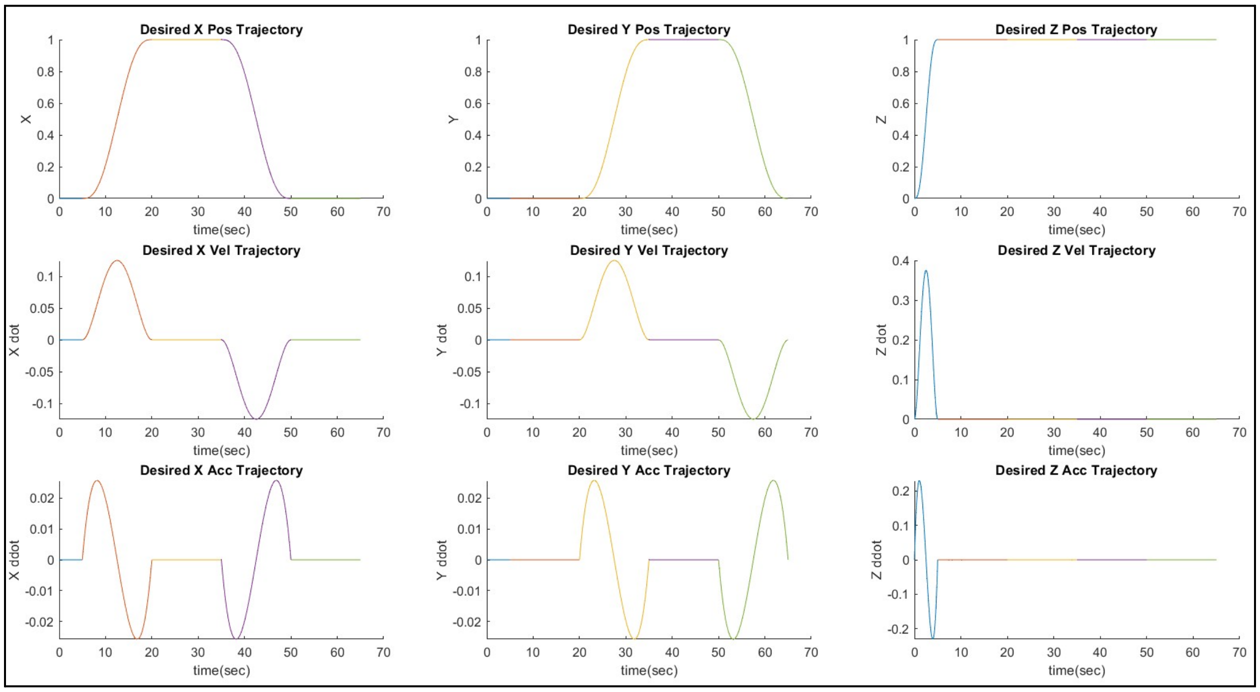
The t_initial and t_final for each of the position values are provided in this assignment. We have considered v_initial, v_final, a_initial and a_final as zero and got the resulting equation as:

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_0 \\ \dot{q}_0 \\ \ddot{q}_0 \\ q_f \\ \dot{q}_f \\ \ddot{q}_f \end{bmatrix}$$

The above coefficients are calculated using:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix}^{-1} \begin{bmatrix} q_0 \\ \dot{q}_0 \\ \ddot{q}_0 \\ q_f \\ \dot{q}_f \\ \ddot{q}_f \end{bmatrix}$$

The desired quadrotor trajectory is then discovered by solving these equations using the initial and final location, velocity, and acceleration data for the 5 trajectory equations for the 5 provided waypoints:



Part 2

In this part we describe how we formulated the **boundary layer based sliding mode control.** This is done for all the control inputs u1,u2, u3, u4.

Derivation for control input **u1**
We define error related terms first-

$$e = z - z\_des$$
$$e\_dot = z\_dot - z\_dot\_des$$
$$e\_ddot = z\_ddot - z\_dot\_des$$

We select a sliding surface as:

$$s = e\_dot + \lambda e$$
$$s\_dot = e\_ddot + \lambda e\_dot$$

Multiplying s by s_dot, we get,

$$ss\_dot = s\,(e\_ddot + \lambda e\_dot)$$
$$s*s\_dot = s\,(z\_ddot - z\_ddot\_des + \lambda e\_dot)$$

As per the given equation of motion:

$$z = \cos(\theta)\cos(\phi) * u1/m - g$$
$$s*s\_dot = s\,(\cos(\theta)\cos(\phi) * u1/m - g - z\_ddot\_des + \lambda e\_dot)$$

But we have this condition where

$$s*s\_dot \leq . -K\,|s|$$

Thus we select u1 such that remaining terms from the equations would be removed. Thus u1 is formulated as follows:

$$u1 = (g + z\_ddot - (\lambda e\_dot) + ur)*(m/ \cos(\theta)\cos(\phi))$$
$$ur= -K sat(s, boundary1)$$

Where

```
if (|s| < boundary1)
        sat(s, boundary1) = s/boundary1
else
        sat(s, boundary1) = sign(s)
```

Derivation for control input **u2**
We define error related terms first-

$$e = \phi - \phi\_des$$
$$\phi\_des = \sin^{-1}(-Fy/u1)$$

Here

$$Fy = m(-kp(y - y\_des) - kd(y\_dot - y\_dot\_des) + y\_ddot\_des)$$
$$e\_dot = \phi\_dot - \phi\_dot\_des$$
$$e\_ddot = \phi\_ddot - \phi\_ddot\_des$$

We select a sliding surface as:

$$s = e\_dot + \lambda e$$
$$s\_dot = e\_ddot + \lambda e\_dot$$

Multiplying s by s_dot, we get,

$$s * s\_dot = s\,(e\_ddot + \lambda e\_dot)$$
$$s * s\_dot = s\,(\phi\_ddot - \phi\_ddot\_des + \lambda e\_dot)$$

As per the given equation of motion:

$$\phi\_ddot = \theta\_dot\ \psi\_dot(Iy - Iz)/Ix - \theta\_dot * \Omega * Ip/Ix + u2/Ix$$
$$s * s\_dot = s/Ix(\ \theta\_dot * \psi\_dot * (Iy - Iz) - \theta\_dot * \Omega * Ip - \phi\_ddot\_des + Ix\lambda e\_dot + u2)$$

But we have this condition where

$$ss\_dot \leq .\ -K\,|s|$$

Thus we select u2 such that remaining terms from the equations would be removed. Thus u2 is formulated as follows:

$$u2 = (-\ \theta\_dot * \psi\_dot\ (Iy - Iz) + \theta\_dot * \Omega Ip - \lambda e\_dot * Ix) + ur$$
$$\text{As } \phi\_ddot\_des = 0$$
$$ur = -K * Ix * sat(s, boundary2)$$

Where

```
if (|s| < boundary2)
        sat(s, boundary2) = s/boundary2
else
        sat(s, boundary2) = sign(s)
```

Derivation for control input **u3**
We define error related terms first-

$$e = \theta - \theta\_des$$
$$\theta\_des = \sin^{-1}(Fx/u1)$$

Here

$$Fy = m(-kp(x - xd) - kd(x\_dot - x\_dot\_des) + x\_ddot\_des)$$
$$e\_dot = \theta\_dot - \theta\_dot\_des$$
$$e\_ddot = \theta\_ddot - \theta\_ddot\_des$$

We select a sliding surface as:

$$s = e\_dot + \lambda e$$
$$s\_dot = e\_ddot + \lambda e\_dot$$

Multiplying s by s_dot, we get,

$$s*s\_dot = s\,(e\_ddot + \lambda e\_dot)$$
$$s*s\_dot = s\,(\theta\_ddot - \theta\_ddot\_des + \lambda e\_dot)$$

As per the given equation of motion:

$$\theta\_ddot = \phi\_dot*\psi\_dot(Iz - Ix)/Iy + \phi\_dot*\Omega Ip/Iy + u3/Iy$$
$$s*s\_dot = s/Iy*(\phi\_dot*\psi\_dot(Iz - Ix) + \phi\_dot*\Omega Ip - \theta\_ddot\_des + Iy\lambda e\_dot + u3)$$

But we have this condition where

$$s*s\_dot \leq. -K\,|s|$$

Thus we select u3 such that remaining terms from the equations would be removed. Thus u3 is formulated as follows:

$$\text{As } \theta\_ddot\_des = 0$$
$$u3 = (-\phi\_dot*\psi\_dot(Iz - Ix) - \theta\_dot*\Omega Ip - \lambda e\_dot*Iy) + ur$$
$$ur = -K * Iy * sat(s, boundary3)$$

Where

```
if (|s| < boundary3)
        sat(s, boundary3) = s/boundary3
else
        sat(s, boundary3) = sign(s)
```

Derivation for control input **u4**
We define error related terms first-

$$e = \psi - \psi\_des$$
$$\text{But } \psi\_des=0$$
$$e\_dot = \psi\_dot$$
$$e\_ddot = \psi\_ddot$$

We select a sliding surface as:

$$s = e\_dot + \lambda e$$
$$s\_dot = e\_ddot + \lambda e\_dot$$

Multiplying s by s_dot, we get,

$$s*s\_dot = s\,(e\_ddot + \lambda e\_dot)$$

As per the given equation of motion:

$$\psi = \phi\_dot*\theta\_dot(Ix - Iy)/Iz + u4/Iz$$
$$s*s\_dot = s/Iz*(\phi\_dot*\theta\_dot*(Ix - Iy) + Iz\lambda*e\_dot + u4)$$

But we have this condition where

$$s*s\_dot \leq . -K\,|s|$$

Thus we select u4 such that remaining terms from the equations would be removed. Thus u4 is formulated as follows:

$$\text{As } \theta\_ddot\_des = 0$$
$$u4 = (-\phi\_dot*\theta\_dot(Ix - Iy) - \lambda e\_dot*Iy) + ur$$
$$ur = -K * Iz * sat(s, boundary3)$$

Where
```
if (|s| < boundary4)
        sat(s, boundary4) = s/boundary4
else
        sat(s, boundary4) = sign(s)
```

Part 3: Implementing the node in ROS

Using the provided Python script from the assignment, the ROS node implementation was carried out. The initial Python script was expanded with the following script:

1. Parameter Initialization:
   The Quadrotor class's init method was responsible for initializing the parameters. To record the control input values, we included quintic trajectory coefficient, system inertia characteristics, and global u_temp variables.

2. Desired Trajectory Calculation:
   The desired trajectory was calculated using the methodology explained in part 1 of the report.

3. Sliding Mode Control Implementation:
   Utilizing the control laws developed in part 2 of this report, the sliding mode control with boundary layer was put into practice. The K and gains were adjusted to ensure that the generalized coordinates followed the desired trajectories and converged on them as efficiently as possible. Using the allocation matrix, the propeller velocities were calculated from the control inputs. Constraints were used to keep the angular velocities within the specified ranges of 0 to 2618.

4. Trajectory Visualization:
   To test the performance of the developed controller on a qualitative level, the controlled trajectory was displayed in 3D and compared with the planned trajectory.

Parameter Tunings:

We tweak the K and gains in the following ways to best control the sliding mode and follow the desired trajectory:

K_1: K_1 is the gain employed in controlling the altitude "z" according to the calculations in part 2 of the report. Faster convergence to the sliding surface is brought about by an increase in K_1, which also produces an increase in the magnitude of acceleration in the X, Y, and Z directions. The ultimate K_1 value is 15 in this case.

λ_1: According to the equations in part 2 of the report, a rise in 1 leads to a quicker decline in error terms and a quicker conversion to the intended trajectory. The last value of λ_1 is set to 10.

K_p and K_d: The Kp and Kd gains are used to get Fx and Fy, which are then used to find the desired theta and phi of the quadrotor, according to the equations of motion provided in the assignment. The gain values are selected such that the Fx and Fy values are less than u1 so that the sin−1(Fx/u1) and sin−1(−Fy/u1) do not give any error due to invalid values. Also if (Fx/u1) and −Fy/u1) values go beyond 1 or -1 we cap it to 1 or -1 so that sin−1(Fx/u1) and sin−1(−Fy/u1) do not give any error. The final gains for Fx and Fy are set as follows:

K_p_x = 120
K_d_x = 5
K_p_y = 80
K_d_y = 5

k2, k3, λ2, λ3: Gains k2, k3, and λ2, λ3 are optimized for a quicker approach to the sliding surface s and a quicker convergence to the desired values of theta and phi. K2 and K3's final values are set to 100. Due to the quadrotor's sensitivity to these gain values, if k2 and k3 are adjusted too low, the quadrotor will become unstable and will crash or fall during flight. Similarly λ2 and λ3 are set to 15 and 1 respectively.

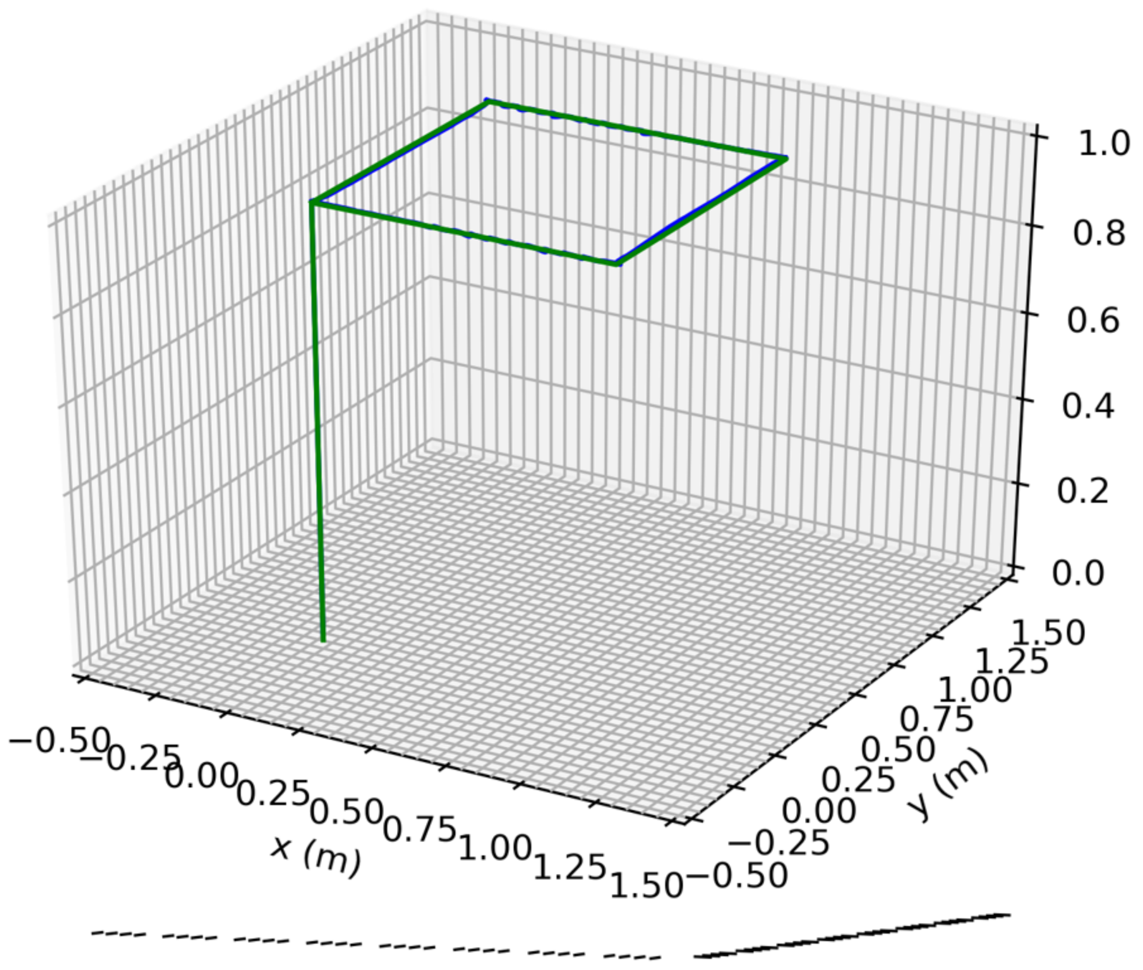
k4, λ4: The robot's rotation about the z axis (ψ) is managed via control input u4. The gain values for u4 are as follows because the ψ does not vary significantly for the simulation run for this assignment. k4 = 20, λ4 = 10. The reason for the high K2 and K3 values is that the control law of ϕ and θ contains a term that is the sum of the quadrotor's angular velocities, and we need higher K2 and K3 values to eliminate it. These gain values are used to run the simulation, and a video file of the simulation's screen is provided to the report for reference. Using the pickle module, the traced trajectory is saved in a pickle file. The trajectory, which will be discussed in the following section, is later visualized using this saved data.

Code explanation

In the Quadrotor init() function, we have calculated and stored the desired trajectory for all time instances. We have made a helper function get_trajectory to calculate the desired trajectory between two time steps. In smc_control function, we have written down the formulation for all control inputs as given in the report. We have also taken care of all the restrictions like maximum speed and range of sin inverses in this function. Finally, we multiply the calculated u1,u2,u3,u4 with the allocation matrix. Here we get the motor speeds required to trace the trajectory. We give this calculated motor speed to the actuator.

Part 4: Trajectory Visualization

In part 4, the traced path is seen and qualitatively compared with the desired trajectory. The main script's save data method is used to create a.pkl file that contains the saved data that is used for visualization. The 3D plot is shown as follows:



Discussion on the performance of the controller:

The desired trajectory is green and the traced trajectory is blue. The quadrotor follows the specified trajectory extremely accurately, as seen in the picture above, and the robust control utilizing boundary layer sliding mode control is successfully achieved. Though, there is not a lot of fluctuation, we can see some green part of the trajectory. It means it's a bit off compared with the desired trajectory. This is because chattering is prevented by using the saturation function. But even so, we can infer from the observation that the controller closely follows the desired trajectory and operates effectively despite reasonable external disturbances.