Robust Trajectory Tracking for Quadrotor UAVs using Sliding Mode Control

Akshay Jaitly, Hushmand Esmaeili

I. Introduction

II.

A. CrazyFlie

The CrazyFlie 2.0 drone is a lightweight open source drone with small propellers. Its design is publicly available for use in simulation with Gazebo. Controlling the robot involves sending desired propeller speeds, which are reached by a lower level controller.

B. Convention

m = mass (kg)
l = arm length (m)
I_p = Moment of Inertia of propeller $(kg \times m^2)$
I_n = Inertia about axis "n" $(kg \times m^2)$
K_F = Propeller thrust factor $(N \times s^2)$
K_M = Propeller moment factor (m)
Ω = Differential Propeller Speed (rad/s)

III. Dynamic Model

The robot state, X, is defined as $[x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T$ where:
x = displacement from origin in the X axis
y = displacement from origin in the Y axis
z = displacement from origin in the Z axis
ϕ = roll
θ = pitch
ψ = yaw
The equation governing the motion of the robot is
$\dot{X} = [\dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi},$
$\ddot{x}(X,u), \ddot{y}(X,u), \ddot{z}(X,u), \ddot{\phi}(X,u), \ddot{\theta}(X,u), \ddot{\psi}(X,u)]^T$

A. Equations of Motion

$$\ddot{x} = \frac{1}{m}(\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)u_1 \tag{1}$$

$$\ddot{y} = \frac{1}{m}(\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)u_1 \tag{2}$$

$$\ddot{z} = \frac{1}{m}(\cos\phi\cos\theta)u_1 - g \tag{3}$$

$$\ddot{\phi} = \dot{\theta}\dot{\psi}\frac{I_y - I_z}{I_x} - \frac{I_p}{I_x}\Omega\dot{\theta} + \frac{1}{I_x}u_2 \tag{4}$$

$$\ddot{\theta} = \dot{\phi}\dot{\psi}\frac{I_z - I_x}{I_y} + \frac{I_p}{I_y}\Omega\dot{\phi} + \frac{1}{I_y}u_3 \tag{5}$$

$$\ddot{\psi} = \dot{\phi}\dot{\theta}\frac{I_x - I_y}{I_z} + \frac{1}{I_z}u_4 \tag{6}$$

B. Control Inputs

The control inputs used in the SISO systems are
u_1, the desired upward thrust
u_2, the differential to create a moment about x
u_3, the differential to create a moment about y
u_2, the differential to create a moment about z
Since the lower level controller tries to achieve motor velocities instead of force, the "allocation matrix" is used to translate from U to $\omega^{\cdot 2}$

This transformation is given as

$$\omega^{\cdot 2} = \begin{bmatrix} \frac{1}{4kF} & -\frac{\sqrt{2}}{4kFl} & -\frac{\sqrt{2}}{4kFl} & -\frac{1}{4kMkF} \\ \frac{1}{4kF} & -\frac{\sqrt{2}}{4kFl} & \frac{\sqrt{2}}{4kFl} & \frac{1}{4kMkF} \\ \frac{1}{4kF} & \frac{\sqrt{2}}{4kFl} & \frac{\sqrt{2}}{4kFl} & -\frac{1}{4kMkF} \\ \frac{1}{4kF} & \frac{\sqrt{2}}{4kFl} & -\frac{\sqrt{2}}{4kFl} & \frac{1}{4kMkF} \end{bmatrix} U$$

Note that $\omega^{\cdot 2}$ is a hadamarad, elementwise, exponentiation.

IV. Reference Trajectories

A. Designing reference task-space trajectories

The trajectory between time t_a and t_b for each positional state (x, y , z) is formed as a polynomial, where

$$P_t = \begin{bmatrix} Cof_x \\ Cof_y \\ Cof_z \end{bmatrix} \begin{bmatrix} (t-t_a)^5 \\ (t-t_a)^4 \\ (t-t_a)^3 \\ (t-t_a)^2 \\ (t-t_a) \\ 1 \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}$$

Cof_n is a 1x6 row vector, representing the coefficients of the polynomial. For each discrete trajectory in the time span, a new coefficient needs to be derived.

To do so, the desired positions, velocities, and accelerations at t_a and t_b were used.

Using matrix A_t, where A_t =

$$\begin{bmatrix} (t-t_a)^5 & (t-t_a)^4 & (t-t_a)^3 & (t-t_a)^2 & (t-t_a) & 1 \\ 5(t-t_a)^4 & 4(t-t_a)^3 & 3(t-t_a)^2 & 2(t-t_a) & 1 & 0 \\ 20(t-t_a)^3 & 12(t-t_a)^2 & 6(t-t_a) & 2 & 0 & 0 \end{bmatrix}$$

$\begin{bmatrix} n_{t_a} \\ \dot{n}_{t_a} \\ \ddot{n}_{t_a} \\ n_{t_b} \\ \dot{n}_{t_b} \\ \ddot{n}_{t_b} \end{bmatrix}$ can be found as $\begin{bmatrix} A_{t_a} \\ A_{t_b} \end{bmatrix} Cof_n^T$ where n_t is the desired position of state n at time t

Therefore,

$$Cof_n = (inv(\begin{bmatrix} A_{t_a} \\ A_{t_b} \end{bmatrix}) \begin{bmatrix} n_{t_a} \\ \dot{n}_{t_a} \\ \ddot{n}_{t_a} \\ n_{t_b} \\ \dot{n}_{t_b} \\ \ddot{n}_{t_b} \end{bmatrix})^T$$

The trajectories are quintic. A cubic trajectory was determined to not allow for smooth following of the acceleration. Since the acceleration is non-smooth, the control input changed drastically.

B. Finding reference roll and pitch angles

To convert the desired trajectories of (x_d, y_d, z_d) to desired roll and pitch angles (ϕ_d, θ_d), the desired forces in x and y can be determined using a PD controller. These are forces $F_x(e, \dot{e})$ and $F_y(e, \dot{e})$. This serves as a higher level controller which is meant to converge to $e = \dot{e} = 0$. The resulting forces can be then converted to desired roll and pitch angles ϕ_d and θ_d, that divert the upwards thrust, u_1, to be partially in the directions x and y to achieve F_x and F_y. This serves as a lower level controller.

For the purpose of this project, the desired yaw angle ψ, as well as the desired angular velocities $\dot{\phi}_d$, $\dot{\theta}_d$, $\dot{\psi}_d$ and the desired angular accelerations $\ddot{\phi}_d$, $\ddot{\theta}_d$, $\ddot{\psi}_d$ are considered to be 0 during the whole trajectory.

V. Sliding Mode Controller

Considering the equations of motion provided in Section III, we designed boundary-based sliding mode controllers for the z, ϕ, θ and ψ coordinates of the quadrotor to track desired trajectories z_d, ϕ_d, θ_d and ψ_d obtained in Section IV.

A. Deriving Control Laws

Since the equations of motion for $\ddot{z}$, $\ddot{\phi}$, $\ddot{\theta}$, $\ddot{\psi}$ are independent from each other, four independent control laws can be designed.

To design all four control laws u_1, u_2, u_3 and u_4 using SMC, we select the general surface equation and its derivative:

$$s = \dot{e} + \lambda e \tag{7}$$

$$\dot{s} = \ddot{e} + \lambda \dot{e} \tag{8}$$

and to achieve the control objective, we must design a control law u such that the following condition is satisfied:

$$s\dot{s} \leq -k|s(t)|, \quad k > 0 \tag{9}$$

As proven through lyapunov analysis, this system converges to $e = \dot{e} = 0$.

To design u_1, the surface equation and its derivative are:

$$s = (\dot{z} - \dot{z}_d) + \lambda_1(z - z_d)$$

$$\dot{s} = (\ddot{z} - \ddot{z}_d) + \lambda_1(\dot{z} - \dot{z}_d)$$

Below is the full procedure to design u_1:

$$s\dot{s} = s[(\ddot{z} - \ddot{z}_d) + \lambda_1(\dot{z} - \dot{z}_d)]$$

$$s\dot{s} = s[-g + (\frac{1}{m}\cos\phi\cos\theta)u_1 \\ -\ddot{z}_d + \lambda_1(\dot{z} - \dot{z}_d)]$$

$$s\dot{s} = s(\frac{1}{m}\cos\phi\cos\theta)[(\frac{m}{\cos\phi\cos\theta})(-g \\ -\ddot{z}_d + \lambda_1(\dot{z} - \dot{z}_d)) + u_1]$$

Choosing u_1:

$$u_1 = (\frac{m}{\cos\phi\cos\theta})(g + \ddot{z}_d \\ -\lambda_1(\dot{z} - \dot{z}_d) + u_r)$$

Applying u_1:

$$s\dot{s} = s[u_r]$$

Next, we select u_r such that the sliding condition is satisfied:

$$u_r = -k_1 sat(s), \quad k_1 > 0$$

where $sat(s)$ is a saturation function for the boundary-layer. The sliding condition

$$s\dot{s} = s[-k_1 sat(s)] \leq -k_1|s(t)|$$

is satisfied for all $k_1 > 0$. Therefore, the designed control law u_1 is given by:

$$u_1 = (\frac{m}{\cos\phi\cos\theta})[g + \ddot{z}_d - \lambda_1(\dot{z} - \dot{z}_d) - k_1 sat(s)] \tag{10}$$

We follow a similar procedure for ϕ, θ and ψ to find the respective control laws u_2, u_3 and u_4 The full procedures can be found in the Appendix. The resulting control laws are given by:

$$u_2 = I_x[-\dot{\theta}\dot{\psi}\frac{I_y - I_z}{I_x} + \frac{I_p}{I_x}\Omega\dot{\theta} + \ddot{\phi}_d - \lambda_2(\dot{\phi} - \dot{\phi}_d) - k_2 sat(s)] \tag{11}$$

$$u_3 = I_y[-\dot{\phi}\dot{\psi}\frac{I_z - I_x}{I_y} - \frac{I_p}{I_y}\Omega\dot{\phi} + \ddot{\theta}_d - \lambda_3(\dot{\theta} - \dot{\theta}_d) - k_3 sat(s)] \tag{12}$$

$$u_4 = I_z[-\dot{\phi}\dot{\theta}\frac{I_x - I_y}{I_z} + \ddot{\psi}_d - \lambda_4(\dot{\psi} - \dot{\psi}_d) - k_4 sat(s)] \tag{13}$$

where $k_2, k_3, k_4 > 0$

B. Tuning Parameters

Following the design of the controllers, different parameters in the control laws need to be tuned in order to obtain the desired behavior of the robot — in our case, to follow the reference trajectory. We used ROS and Gazebo as our simulation environment, as explained in Section VI, to tune the quadrotor. After extensive testing, the final tuning parameters and their values that we used for a stable simulation of the robot are listed below.

- Proportional gain for F_x and F_y in PD controller: $k_p = 40$
- Derivative gain for F_x and F_y in PD controller: $k_d = 1$

- Gain for u_1 saturation function: $k_1 = 15$
- Gain for u_2 saturation function: $k_2 = 100$
- Gain for u_3 saturation function: $k_3 = 100$
- Gain for u_4 saturation function: $k_4 = 10$
- Gain for $\dot{z}$ error in u_1: $\lambda_1 = 7$
- Gain for $\dot{\phi}$ error in u_2: $\lambda_2 = 15$
- Gain for $\dot{\theta}$ error in u_3: $\lambda_3 = 15$
- Gain for $\dot{\psi}$ error in u_4: $\lambda_4 = 5$
- Boundary layer for saturation function for all control laws: $b_1, b_2, b_3, b_4 = 1$

k_p and k_d served a higher level controller, they determined what the forces in X and Y should be. Too high a desired force meant too aggressive an angle in θ and ϕ, which were hard to maintain while being stable. k_d was especially set low, as a high k_d would mean an initial high acceleration. Deceleration was not easy to do, as it meant turning the quadrotor in the direction opposite of its motion, which (similar to high u_1) caused odd swaying, so parameters were chosen for less acceleration.

K1 had to be set such that the quadrotor would be robust in the Z direction, while also being low enough that when turned in order to accelerate, the rotor limits were not reached.

k_2 and k_3 were set to converge more aggressively, as these were lower level controllers. However, the sliding surface was reached, which caused chattering of the control input. Lambda was increased as the positional error is what was needed to be converged on.

ψ was rarely unstable, and needed little tuning to work.

The boundary layers for the saturation functions allowed to subdue chattering – which was demonstrated.

VI. Results

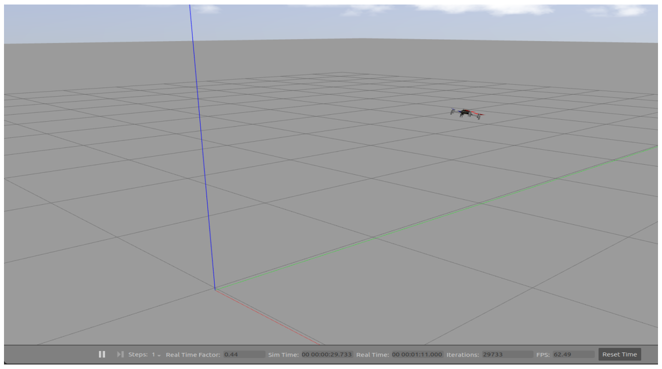

Fig. 1. **Crazyflie 2.0 quadrotor following reference trajectory in Gazebo.**

A. Experimental Setup

To evaluate the performance of the designed controllers, we used ROS and Gazebo to simulate the robot and the physics. We implemented a ROS node in Python based on the sample code provided to us by our instructor. In our script, we initialize all the physical parameters of the robot, as well as the tuning parameters for the control laws. We also initialize variables to store the data of the simulation for later visualization.

Within our script, we define a function to evaluate our trajectory over time. We implement the derived equations from Section IV and return the desired positions, velocities and accelerations.

To implement the control laws, we defined functions that return the control inputs given some parameters, which are all the variables in our designed control laws from Section V. The tuning parameters, such as λ, k and the boundary layer, are constant and defined at the beginning of the script.

Finally, the sample code given to us already included the odometry callback function to receive the state of the robot data from ROS. It also included code to save the data for later visualization. However, we expanded the data collection code to store other data that we wanted to visualize, including the actual and desired roll, pitch and yaw angles throughout the robot motion, as well as the control inputs and motor speeds.

B. Simulation

After implementing our scripts, we ran the simulation on Gazebo through ROS. The simulation of the robot following the trajectory in Gazebo is shown in Figure 1. The actual trajectory over the reference trajectory is plotted in 3D, as shown in Fig. 2. the

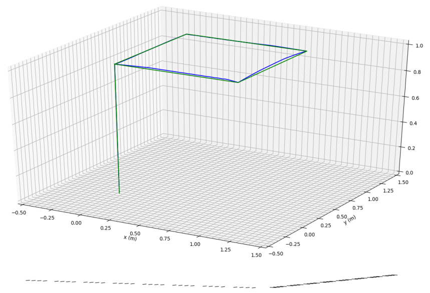

Fig. 2. **3D plot of the actual trajectory over the reference trajectory over the motion of the quadrotor.**

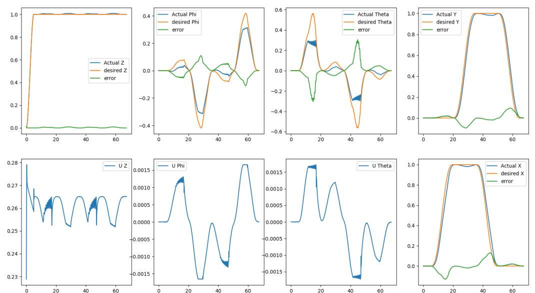

Fig. 3. **Performance of the quadrotor during simulation.**

The trajectory is followed fairly well (Fig. 2, Fig 3), with deviation visually obvious along the Z axis. k_1 – the convergence factor for the z SISO system had to be set low, as a large control input for U1 would result in over-saturation of the rotors.This caused a stabilization of the quadrotor to happen at $\phi = 0$ and $\theta = 0$. If this happened during translation in x and y, it would cause oscillations as the desired values would be non-zero.

The discussion about tuning can be found in section V B

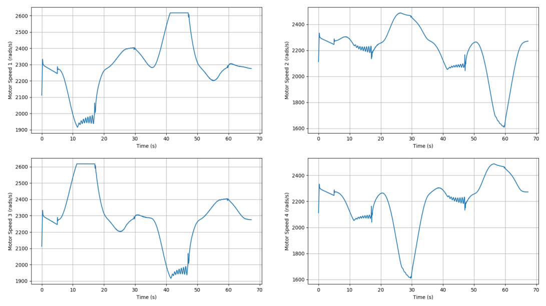

Fig. 4. **Motors speed of the quadrotor during simulation.**

The rotors occasionally meet their limit of 2600 rads/sec. At equilibrium, they converged to about 2300 rads/sec as a result of U1. This meant that the controlled efforts of u_2, u_3, and u_4 had a low margin to operate within (Fig. 4). In all simulations this was still stable, however work needs to yet be done to limit the control input.

VII. Conclusions and Future Work

The sliding mode controller converges on a set trajectory for the Crazyflie 2.0 quadrotor. The controller cannot be stable for fast movement, as the rotors go past their limit which cayuses issues.

In order to combat this issue, steps can be taken in controller design. stability is guaranteed when $\dot{s}sat(s) \leq -k$, however, the controller designed now ensures that $\dot{s}sat(s) \approx -k$. Unneccesary control effort can be removed, and better convergence can be found, by incorporating sigmoids into the control design.

For control affine functions considering the general formulation of s, and an output y

$$\dot{s} = \ddot{e} - \lambda(\dot{e}) = f(y) + g(y)U - \ddot{y}_d - \lambda(\dot{e})$$

The formulation of the control law in this paper followed the law that, for $U = \frac{\lambda\dot{e} + \ddot{y}_d - f(y) - ksat(s)}{g(y)}$, $\dot{s} = -ksat(s)$, therefore, $\dot{s} \leq -ksat(s)$ which is stable. In order to get a lower and more robust control input, each term can be multiplied by a sigmoid function, S_g.

This would result in

$$U = \frac{\lambda\dot{e}S_g(-(\lambda\dot{e}t_1 - p_1)sat(s)) + \ddot{y}_d S_g(-(y_d t_2 - p_2)sat(s))}{g(y)}$$
$$+ \frac{-f(y)S_g((f(y)t_3 - p_3)sat(s)) - ksat(s)}{g(y)}$$

where t_n and p_n are tuning parameters. If $S_g(a)$ is a function such that $S_g \approx 0$ at $a < 0$, and $S_g > 1$ at $a > 0$, this would reduce the control input and $\dot{s} \leq -ksat(s)$ would hold, resulting in a better controller. If $S_g(a) > 1 + \rho$ where ρ was the uncertainty in the parameter a, this would result in a formally robust controller, even when $k = 0$.

Implementing this may help address the issue of over saturation.

References

[1] Farzan, Siavash, "Robust Trajectory Tracking for Quadrotor UAVs using Sliding Mode Control" Final Project Document, WPI

Appendix

Derivation of control law for ϕ:

$$s = (\dot{\phi} - \dot{\phi}_d) + \lambda_2(\phi - \phi_d)$$

$$\dot{s} = (\ddot{\phi} - \ddot{\phi}_d) + \lambda_2(\dot{\phi} - \dot{\phi}_d)$$

$$s\dot{s} = s[(\ddot{\phi} - \ddot{\phi}_d) + \lambda_2(\dot{\phi} - \dot{\phi}_d)]$$

$$s\dot{s} = s[\dot{\theta}\dot{\psi}\frac{I_y - I_z}{I_x} - \frac{I_p}{I_x}\Omega\dot{\theta} + \frac{1}{I_x}u_2 - \ddot{\phi}_d + \lambda_2(\dot{\phi} - \dot{\phi}_d)]$$

$$s\dot{s} = s(\frac{1}{I_x})[\dot{\theta}\dot{\psi}(I_y - I_z) - I_p\Omega\dot{\theta} + u_2$$
$$-I_x\ddot{\phi}_d + I_x\lambda_2(\dot{\phi} - \dot{\phi}_d)]$$

Choosing u_2:

$$u_2 = -\dot{\theta}\dot{\psi}(I_y - I_z) + I_p\Omega\dot{\theta}$$
$$+I_x\ddot{\phi}_d - I_x\lambda_2(\dot{\phi} - \dot{\phi}_d) + u_r$$

Applying u_2:

$$s\dot{s} = s(\frac{1}{I_x})[u_r]$$

Next, we select u_r such that the sliding condition is satisfied:

$$u_r = -k_2 sat(s), \quad k_2 > 0$$

where $sat(s)$ is a saturation function for the boundary-layer. The sliding condition

$$s\dot{s} = s(\frac{1}{I_x})[-k_2 sat(s)] \leq -k_2|s(t)|$$

is satisfied for all $k_2 > 0$. Therefore, the designed control law u_2 is given by:

$$u_2 = -\dot{\theta}\dot{\psi}(I_y - I_z) + I_p\Omega\dot{\theta} + I_x\ddot{\phi}_d - I_x\lambda_2(\dot{\phi} - \dot{\phi}_d) - k_2 sat(s) \quad (14)$$

or

$$u_2 = I_x[-\dot{\theta}\dot{\psi}\frac{I_y - I_z}{I_x} + \frac{I_p}{I_x}\Omega\dot{\theta} + \ddot{\phi}_d - \lambda_2(\dot{\phi} - \dot{\phi}_d) - k_2 sat(s)] \quad (15)$$

Derivation of control law for θ:

$$s = (\dot{\theta} - \dot{\theta}_d) + \lambda_3(\theta - \theta_d)$$

$$\dot{s} = (\ddot{\theta} - \ddot{\theta}_d) + \lambda_3(\dot{\theta} - \dot{\theta}_d)$$

$$s\dot{s} = s[(\ddot{\theta} - \ddot{\theta}_d) + \lambda_3(\dot{\theta} - \dot{\theta}_d)]$$

$$s\dot{s} = s[\dot{\phi}\dot{\psi}\frac{I_z - I_x}{I_y} - \frac{I_p}{I_y}\Omega\dot{\phi} + \frac{1}{I_y}u_3 - \ddot{\theta}_d + \lambda_3(\dot{\theta} - \dot{\theta}_d)]$$

$$s\dot{s} = s(\frac{1}{I_y})[\dot{\phi}\dot{\psi}(I_z - I_x) - I_p\Omega\dot{\phi} + u_3 \\ -I_y\ddot{\theta}_d + I_y\lambda_3(\dot{\theta} - \dot{\theta}_d)]$$

Choosing u_3:

$$u_3 = -\dot{\phi}\dot{\psi}(I_z - I_x) + I_p\Omega\dot{\phi} \\ +I_y\ddot{\theta}_d - I_y\lambda_3(\dot{\theta} - \dot{\theta}_d) + u_r$$

Applying u_3:

$$s\dot{s} = s(\frac{1}{I_y})[u_r]$$

Next, we select u_r such that the sliding condition is satisfied:

$$u_r = -k_3 sat(s), \quad k_3 > 0$$

where $sat(s)$ is a saturation function for the boundary-layer. The sliding condition

$$s\dot{s} = s(\frac{1}{I_y})[-k_3 sat(s)] \leq -k_3|s(t)|$$

is satisfied for all $k_3 > 0$. Therefore, the designed control law u_3 is given by:

$$u_3 = -\dot{\phi}\dot{\psi}(I_z - I_x) + I_p\Omega\dot{\phi} + I_y\ddot{\theta}_d - I_y\lambda_3(\dot{\theta} - \dot{\theta}_d) - k_3 sat(s) \tag{16}$$

or

$$u_3 = I_y[-\dot{\phi}\dot{\psi}\frac{I_z - I_x}{I_y} - \frac{I_p}{I_y}\Omega\dot{\phi} + \ddot{\theta}_d - \lambda_3(\dot{\theta} - \dot{\theta}_d) - k_3 sat(s)] \tag{17}$$

Derivation of control law for ψ:

$$s = (\dot{\psi} - \dot{\psi}_d) + \lambda_4(\psi - \psi_d)$$

$$\dot{s} = (\ddot{\psi} - \ddot{\psi}_d) + \lambda_4(\dot{\psi} - \dot{\psi}_d)$$

$$s\dot{s} = s[(\ddot{\psi} - \ddot{\psi}_d) + \lambda_4(\dot{\psi} - \dot{\psi}_d)]$$

$$s\dot{s} = s[\dot{\phi}\dot{\theta}\frac{I_x - I_y}{I_z} + \frac{1}{I_z}u_4 - \ddot{\psi}_d + \lambda_4(\dot{\psi} - \dot{\psi}_d)]$$

$$s\dot{s} = s(\frac{1}{I_z})[\dot{\phi}\dot{\theta}(I_x - I_y) + u_4 - I_z\ddot{\psi}_d + I_z\lambda_4(\dot{\psi} - \dot{\psi}_d)]$$

Choosing u_4:

$$u_4 = -\dot{\phi}\dot{\theta}(I_x - I_y) + I_z\ddot{\psi}_d - I_z\lambda_4(\dot{psi} - \dot{\psi}_d) + u_r$$

Applying u_4:

$$s\dot{s} = s(\frac{1}{I_z})[u_r]$$

Next, we select u_r such that the sliding condition is satisfied:

$$u_r = -k_4 sat(s), \quad k_4 > 0$$

where $sat(s)$ is a saturation function for the boundary-layer. The sliding condition

$$s\dot{s} = s(\frac{1}{I_z})[-k_4 sat(s)] \leq -k_4|s(t)|$$

is satisfied for all $k_4 > 0$. Therefore, the designed control law u_4 is given by:

$$u_4 = -\dot{\phi}\dot{\theta}(I_x - I_y) + I_z\ddot{\psi}_d - I_z\lambda_4(\dot{\psi} - \dot{\psi}_d) - k_4 sat(s) \quad (18)$$

or

$$u_4 = I_z[-\dot{\phi}\dot{\theta}\frac{I_x - I_y}{I_z} + \ddot{\psi}_d - \lambda_4(\dot{\psi} - \dot{\psi}_d) - k_4 sat(s)] \quad (19)$$