University of Stavanger

Faculty of Science and Technology

BACHELOR'S THESIS

Study program/ Specialization: Control Engineering and Circuit Design - Bachelor's Degree Programme	Spring semester, 2023 Open access
Writers: Nickolas Phan Børge Olav Haug Tor Ivar Haughom	Nickolas H.P Phan Børge Olav Haug Tor Ivar Haughom (Writer's signature)
Faculty supervisor: Damiano Rotondo	
Thesis title: *Sliding Mode Control and Vision-Based Line Tracking for Quadrotors*	
Credits (ECTS): 20	
Key words: Sliding Mode Control PID Control LQR Control Line Tracking	Pages: 139 + enclosure: 112 Stavanger, 15. May 2023 Date/year

Acknowledgements

The three of us would like to thank our supervisor Damiano Rotondo for his enthusiasm and valuable input on control engineering, as well as presenting an interesting and exciting thesis topic. We would also like to thank Didrik Efjestad Fjereide for helping us with the Quanser 3 DOF Hover system.

Jeg vil takke Børge og Tor Ivar for deres innsats rundt denne bacheloren. Jeg ville ikke fått et godt resultat uten dere. Samtidig vil jeg takke venner for hjelpen i prosjektet.

- **Nickolas Phan**

Jeg vil rette en stor takk til Tor Ivar og Nickolas for deres iherdige arbeid med denne bacheloroppgaven. Jeg vil også takke min bedre halvdel Anne Marthe for hennes støtte og oppmuntring gjennom semesteret.

- **Børge Olav Haug**

Jeg ønsker å takke Børge og Nickolas for deres gode samarbeid med denne bacheloroppgaven. I tillegg vil jeg takke familie og venner for deres støtte gjennom prosjektet.

- **Tor Ivar Haughom**

Abstract

This thesis describes the design of Sliding Mode Control applied to quadrotor UAV flight. This is a nonlinear control technique in which a discontinuous control signal is applied to drive the so-called sliding variable to zero, which defines the sliding surface. The sliding variable should be designed in such a way that approaching the sliding surface is beneficial to tracking the reference signals. The advantages of Sliding Mode Control are that the need for simplifying the underlying dynamical model through linearization is avoided, it is robust and adaptive, and works even if the system to be controlled is highly nonlinear or has model uncertainties. Sliding Mode Control has one major issue associated with it, namely the chattering phenomena in the control inputs, which is undesirable. This can be tackled by approximating the discontinuous sign function in the control input with a approximated continuous function, or by applying techniques such as adaptive fuzzy gain scheduling. As with other control methods, Sliding Mode Control requires tuning of the control parameters to obtain an optimal performance. In this work, genetic algorithms were investigated as a way to tune the controller parameters. The findings of this thesis were combined with the design of a line tracking algorithm in order to enter the MathWorks Minidrone Competition.

Contents

List of Figures

List of Tables

List of Abbreviations

Symbol	Description
UAV	**U**nmanned **A**erial **V**ehicle
LQR	**L**inear **Q**uadratic **R**egulator
PD	**P**orportional **D**erivative
PID	**P**orportional **I**ntegral **D**erivative
SMC	**S**liding **M**ode **C**ontroller
ISMC	**I**ntegral **S**liding **M**ode **C**ontroller
AFGS	**A**daptive **Fuzzy** **G**ain **S**cheduling
FLS	**F**uzzy **L**ogic **S**ystem
NL	**N**egative **L**arge
NS	**N**egative **S**mall
ZE	**ZE**ro
PS	**P**ositive **S**mall
PL	**P**ositive **L**arge
VL	**V**ery **L**ow
L	**L**ow
M	**M**edium
H	**H**igh
VH	**V**ery **H**igh
ISE	**I**ntegral **S**quared **E**rror
IAE	**I**ntergral **A**bsolute **E**rror
ITSE	**I**ntegral **T**ime **S**quared **E**rror
ITAE	**I**ntegral **T**ime **A**bsolute **E**rror
GA	**G**enetic **A**lgorithm
DOF	**D**egrees **O**f **F**reedom
BW	**B**lack and **W**hite
RGB	**R**ed **G**reen **B**lue
IFAC	**I**nternational **F**ederation of **A**utomatic **C**ontrol
LTI	**L**inear **T**ime-**I**nvariant

Chapter 1

Introduction

1.1 Motivation

Unmanned Aerial Vehicles (UAVs), or drones, are becoming increasingly popular and have many applications: military and security purposes, search and rescue, agriculture, forest restoration, building inspection and road traffic monitoring, just to name a few [7].

UAVs face many challenges as they are maneuvering through the air: adapting to disturbances such as wind and other weather effects, rapidly changing directions and following complex paths. The quadrotor is considered a highly nonlinear system, with complex aerodynamics. Obtaining a good mathematical model for a drone flight can be a challenging task, where the obtained model can suffer from system uncertainties, as no mathematical model is able to fully capture every aspect of the concept that is being modeled. Linearization is often required for linear control methods to be applied, which can lead to further model inaccuracies. This thesis investigates a control method called Sliding Mode Control (SMC) that shows promise in dealing with the difficulties mentioned. Some of the main strengths of this method are removal of the need for linearization, adaptability, good disturbance rejection, precision and fast response [8].

The Parrot Minidrone Competition was seen as a good opportunity for applying the SMC controller to a practical problem and served as extra motivation for the project.

1.2 Objective

The main objectives of this thesis have been to:

- Perform a literature study of various controller designs for UAVs.

- Derive a nonlinear mathematical model for quadrotor UAVs.
- Develop a Sliding Mode Controller (SMC) for quadrotor UAVs.
- Develop PID- and LQR-controllers for quadrotor UAVs for use as benchmarks to compare SMC against.
- Tune controller parameters by Genetic Algorithms.
- Compare the performance of the nonlinear SMC against that of the linear PID- and LQR-controllers.
- Design and develop a line tracking algorithm for the MathWorks Minidrone Competition.
- Compare line tracking performance using SMC against the PID-controller developed by MathWorks.

These goals will be achieved through development and simulation in MATLAB and Simulink. Simulations done in relation to the MathWorks Minidrone Competition are carried out using the *Simulink Support Package for Parrot Minidrones* MATLAB add-on. The designed controllers are also tested by using the Quanser 3 DOF Hover system available in the laboratory room E-457 at the University of Stavanger.

1.3 MathWorks Minidrone Competition

One of the goals of this thesis' project was to apply the sliding mode controller to a practical problem. The MathWorks Minidrone Competition was seen as a good opportunity for this.

The MathWorks Minidrone Competition is a competition where the participating teams have to develop an autonomous minidrone line follower. It is hosted by MathWorks, the creators of MATLAB, and hence all of the development, simulations, etc. is done using MATLAB and Simulink [9].

The competition usually consists of two rounds:

Round 1 | Simulation: A qualifying round where the line follower's performance is judged only through simulation in Simulink.

Round 2 | Deployment: A final round where the teams that qualified during the first round meet in person and deploy their algorithms on a Parrot Mambo minidrone.

In both cases, the line follower is judged by its ability to follow a track that consists of one to ten line segments that are 10 cm wide. At the end of the track there is a circular marker with a diameter of 20 cm that the minidrone must land on. The accuracy of the path taken, the completion time and the number of completed tracks is of importance [10].

A Simulink project with a Quadcopter Flight Simulation Model for Parrot Minidrones is provided by MathWorks. This project includes everything needed to setup and run the simulation and visualization of the drone flight, allowing the teams to focus only on the drone's control system. The model comes with a PID-based controller. In the case of this project, this controller will be replaced by the sliding mode controller, the main focus of the thesis. Additionally, a line tracking algorithm will be developed.

At the time of the writing of this thesis, the closest upcoming competition is the IFAC 2023 Competition. The final round of this competition is held during the IFAC World Congress in Yokohoma, Japan. This is an event hosted by the International Federation of Automatic Control to promote scientific activities and technological developments in the field of automatic control [11]. The submission deadline for the first round is 19th of April 2023. The qualifying teams then get to participate in the second round which is held from 10th to 11th of July 2023 [9].

Chapter 2

Quadrotor Dynamics

2.1 Main Components

A quadrotor is an underactuated aircraft with six degrees of freedom (6 DOF), which means that it can move longitudinally (forward and backward), vertically (upward and downward), and laterally (right and left), as well as rotationally around each axis to produce roll, pitch, and yaw movements. It is made up of four engines, with four rotors as shown in Figure 2.1. Since the four rotors have fixed angles, the quadrotor has four input forces, which are the thrusts provided by each propeller. Each propeller creates a thrust that is perpendicular to their rotation plane. They also produce a torque on the quadrotor frame, which is in the opposite direction of the propellers rotation. To cancel out this torque, there are two sets of propellers. One pair creates an upward thrust when spinning anti-clockwise, while the second pair creates an upward thrust when spinning clockwise. The torque from each propeller on the quadrotor frame can then be canceled out.

Figure 2.1: Parrot Mambo minidrone [1].

2.2 Euler Angles

Before describing the mathematical model, it is important to introduce the reference coordinates. We can define two reference systems, in which the first is a fixed coordinate system called the inertial frame. This is an earth fixed coordinate system, with the x-axis, y-axis and z-axis directed North, East and down respectively. The positive x-axis will represent the forward direction, while the positive y-axis will represent the right direction.

The second is a mobile coordinate system called the body fixed frame, which has it origin at the center of mass of the quadrotor. Figure 2.2 illustrates the two coordinate systems.

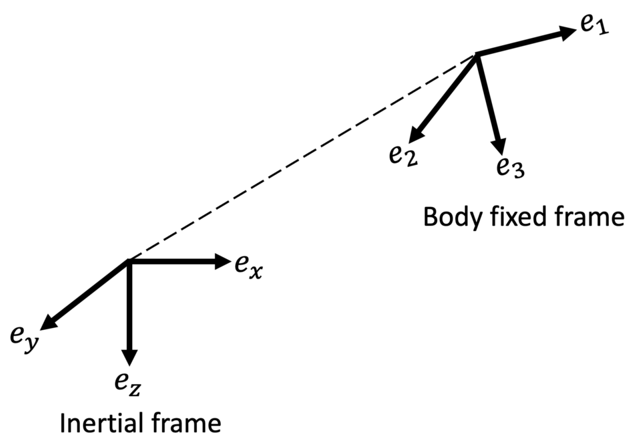


Figure 2.2: Mobile and fixed reference systems.

To describe the orientation of the quadrotor body, the ZYX Euler angles [12]: roll, pitch and yaw, are introduced, which are denoted as ϕ, θ and ψ respectively. Roll is rotation around the x-axis, pitch is rotation around the y-axis, and yaw is rotation around the z-axis, all in the inertial frame as depicted in Figure 2.3.

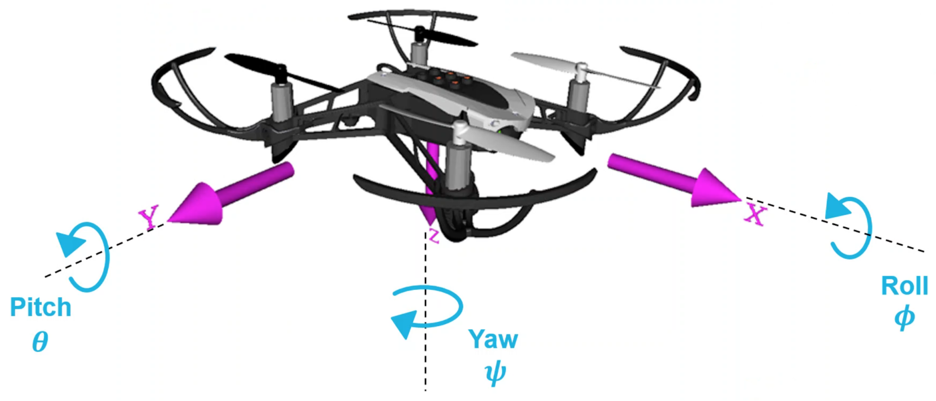


Figure 2.3: Euler Angles [2].

The Euler angles can be used to describe the orientation of a frame of reference relative to another, and they can transform the coordinates of a point in a reference frame to the coordinates of the same point in another reference frame. The Euler angles can then be used to describe the rotation from the body fixed frame to the inertial frame. The rotation matrices for each axis x, y and z is described as

$$\boldsymbol{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi) & c(\phi) \end{bmatrix} \tag{2.1}$$

$$\boldsymbol{R}_y(\theta) = \begin{bmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{bmatrix} \tag{2.2}$$

$$\boldsymbol{R}_z(\psi) = \begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

where $c() = \cos()$ and $s() = \sin()$. A rotation matrix is a square matrix that is used to perform a rotation in a certain number of dimensions. By multiplying these matrices in the desired sequence, one can obtain a combined rotation matrix that represents the orientation of an object in 3D space. The position in the inertial frame and in the body fixed frame are related by the rotation matrix $\boldsymbol{R}_{zyx}(\phi, \theta, \psi)$:

$$\begin{gathered}\boldsymbol{R}_{zyx}(\phi, \theta, \psi) = \boldsymbol{R}_z(\psi) \cdot \boldsymbol{R}_y(\theta) \cdot \boldsymbol{R}_x(\phi) \\ = \begin{bmatrix} c(\theta)c(\psi) & s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi) & c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) \\ c(\theta)s(\psi) & s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{bmatrix}\end{gathered} \tag{2.4}$$

which is used to describe the rotation from the body fixed frame to the inertial frame.

2.3 Working Principles

In terms of the orientation of the rotors relative to the body fixed frame, there are two basic types of configurations: the plus- and cross-configuration which are shown in Figure 2.4. The 3 DOF Hover utilizes the plus-configuration, whereas the cross-configuration is employed in the Parrot Mambo minidrone competition. In this section, we will delve into the details of the cross-configuration, while the plus-configuration and the 3 DOF Hover will be discussed later on. With the cross-configuration, rotor 1 is located parallel to the xy-plane, $-45°$ from the x-axis viewed from above. Rotor 2, 3, and

4 also lay in the xy-plane with 90° between each rotor in the clockwise direction viewed from above. Rotor 1 and 3 rotate positively with respect to the z-axis, and rotor 2 and 4 rotate negatively with respect to the z-axis. This is summed up in Figure 2.5.

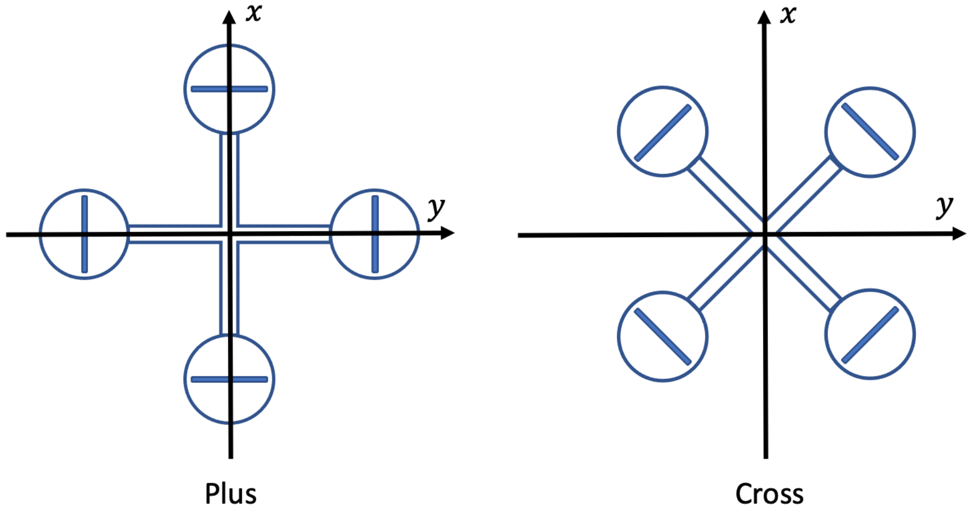


Figure 2.4: Plus- and cross-configuration.

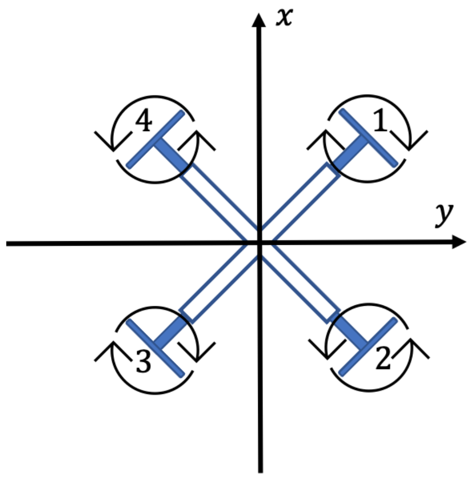


Figure 2.5: Rotor placements and rotations in the cross-configuration.

The attitude and position of the quadrotor can be controlled via the four rotor speeds, which can introduce the following four forces and torques on the quadrotor: total thrust, roll moment, pitch moment and yaw moment.

Collectively increasing all the rotor speeds will increase the total thrust of the quadrotor, making it move up and down according to if the total thrust is bigger or smaller than the weight caused by gravity.

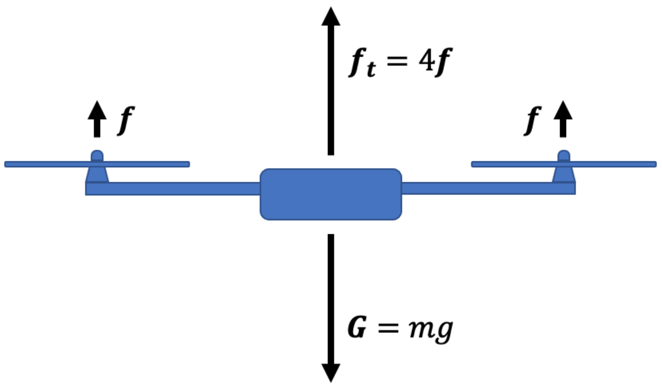


Figure 2.6: Force balance while hovering.

Roll moment is achieved by creating an unbalance in the left and right side forces. Increasing the speed of rotor 3 and 4, and decreasing the speed of rotor 1 and 2 will create a positive roll angle. A positive roll angle will make the quadrotor accelerate in the right direction, and a negative roll angle will make it accelerate in the left direction.

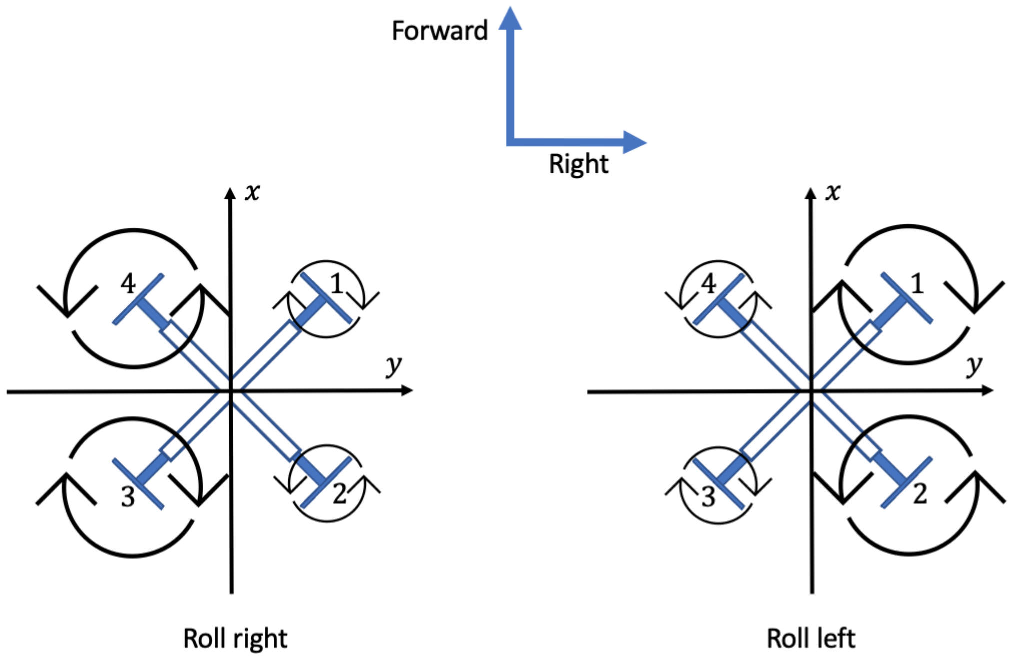


Figure 2.7: Roll in cross-configuration.

Similarly, pitch moment is achieved by creating an unbalance in the front and back forces. Thus, increasing the speed of rotor 1 and 4, and decreasing the speed of rotor 2 and 3 will result in a positive pitch angle. A positive pitch angle will make the quadrotor accelerate in the backwards direction, while a negative pitch angle will make it accelerate in the forward direction.

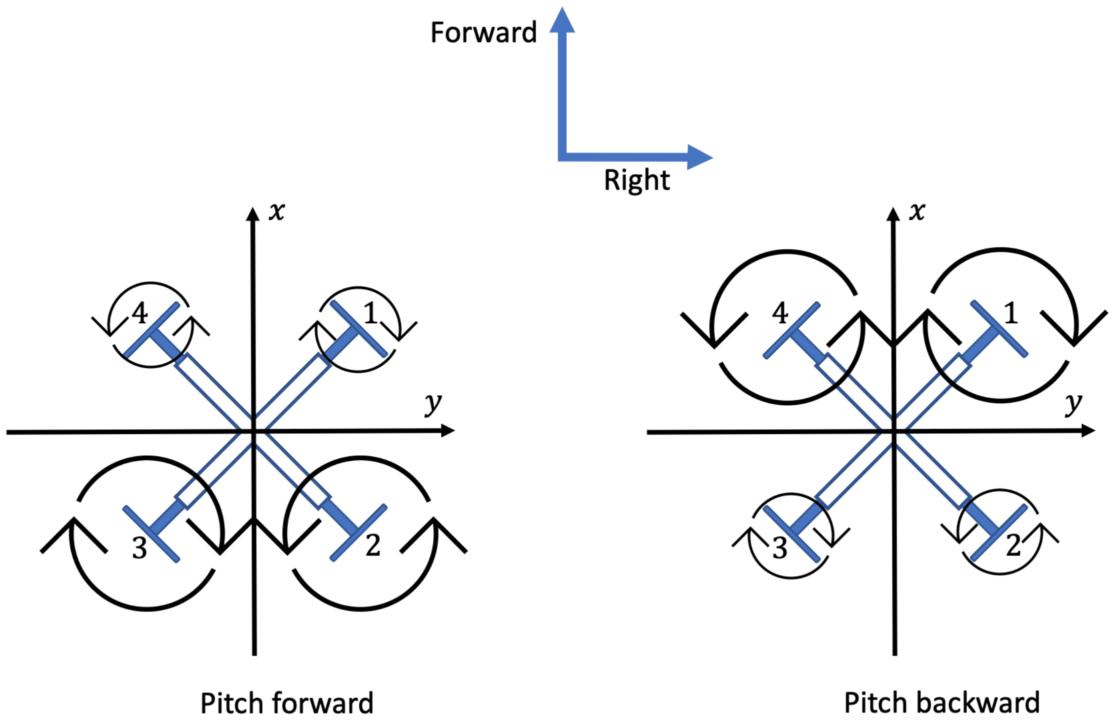


Figure 2.8: Pitch in cross-configuration.

To create a yawing moment, the pairwise anti-clockwise and clockwise rotating rotors are adjusted to cause an unbalance in the torques acting on the quadrotor frame. Increasing the speed of positively rotating rotors 1 and 3, and decreasing the speed of negatively rotating rotors 2 and 4 will decrease the yaw angle. Similarly, increasing the speed of negatively rotating rotors 2 and 4, and decreasing the speed of positively rotating rotors 1 and 3 will increase the yaw angle.

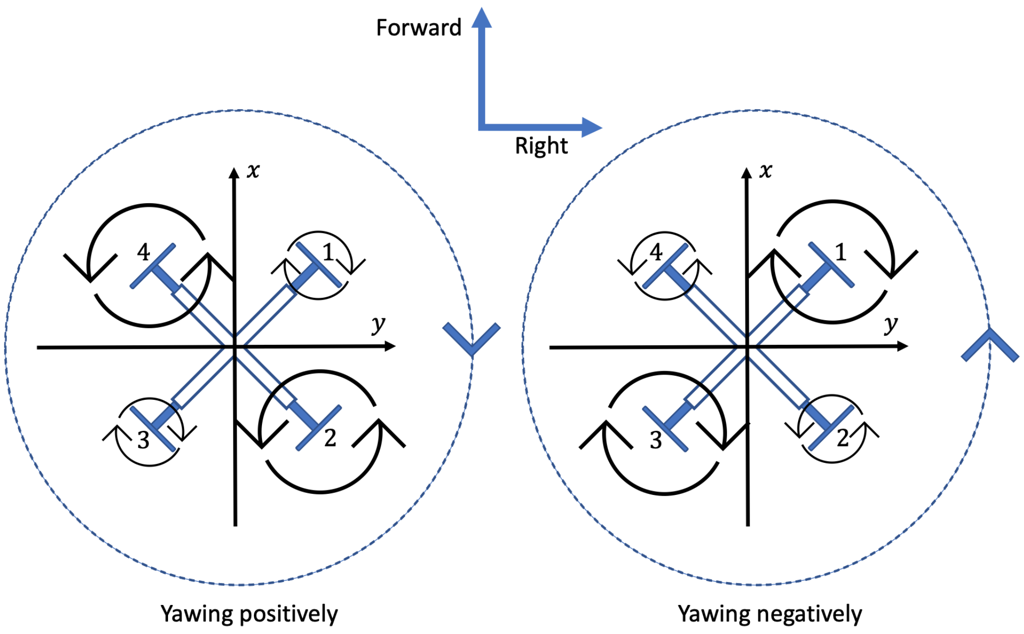


Figure 2.9: Yaw in cross-configuration.

2.4 Mathematical Model

When obtaining the mathematical model of the quadrotor, the following assumptions are made:

1. The quadrotor's structure is rigid and symmetrical.
2. The quadrotor's propellers are rigid.
3. The thrust and drag forces are proportional to the square of the rotors' rotational speed.

A mathematical model exploiting Newton and Euler equations for the 3D motion of a rigid body will now be provided. Consider the linear position $\begin{bmatrix} x & y & z \end{bmatrix}$ and the angular position $\begin{bmatrix} \phi & \theta & \psi \end{bmatrix}$ of the quadrotor in the inertial frame, as well as the linear velocities $\begin{bmatrix} u & v & w \end{bmatrix}$ and the angular velocities $\begin{bmatrix} p & q & r \end{bmatrix}$ of the quadrotor in the body fixed frame. The two reference frames are related by

$$\begin{aligned} \dot{\boldsymbol{\xi}} &= \boldsymbol{R} \cdot \boldsymbol{v_B} \ , \\ \dot{\boldsymbol{\eta}} &= \boldsymbol{T} \cdot \boldsymbol{\omega_B} \end{aligned} \tag{2.5}$$

where $\boldsymbol{\xi} = \begin{bmatrix} x & y & z \end{bmatrix}^T$, $\boldsymbol{v_B} = \begin{bmatrix} u & v & w \end{bmatrix}^T$, $\boldsymbol{\eta} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$, $\boldsymbol{\omega_B} = \begin{bmatrix} p & q & r \end{bmatrix}^T$, $\boldsymbol{R}$ is the rotation matrix from Equation (2.2), and $\boldsymbol{T}$ is a matrix of angular transformation given by

$$\boldsymbol{T} = \begin{bmatrix} 1 & s(\phi)t(\theta) & c(\phi)t(\theta) \\ 0 & c(\phi) & -s(\phi) \\ 0 & \frac{s(\phi)}{c(\theta)} & \frac{c(\phi)}{c(\theta)} \end{bmatrix} \tag{2.6}$$

where $t() = \tan()$.

2.4.1 Forces

From Newton's law we have

$$m\ddot{\boldsymbol{\xi}} = \boldsymbol{R} \cdot \boldsymbol{F_B} = \boldsymbol{F_g} - \boldsymbol{F_t} \tag{2.7}$$

where m is the total mass of the quadrotor and $\boldsymbol{F_B}$ is all the forces acting on the body. $\boldsymbol{F_g}$ is the gravity force and is expressed as

$$\boldsymbol{F_g} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{2.8}$$

$\boldsymbol{F_t}$ denotes the thrust generated by the four propellers, and is expressed as

$$\boldsymbol{F_t} = f_t \boldsymbol{R} \cdot \boldsymbol{e_3} = f_t \begin{bmatrix} c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) \\ c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) \\ c(\phi)c(\theta) \end{bmatrix} \tag{2.9}$$

where $\boldsymbol{e_3}$ is the unit vector in the body fixed frames z-axis, and f_t is the total thrust force given by the sum of all the propeller thrusts

$$f_t = \sum_{i=1}^{4} f_i \tag{2.10}$$

where $f_i = b\Omega_i^2$, b is the thrust factor and Ω_i is the rotational speed of each rotor. From Equation (2.7) the following is obtained

$$\begin{cases} \ddot{x} = -\dfrac{f_t}{m} (\sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta)) \\ \ddot{y} = -\dfrac{f_t}{m} (\cos(\phi)\sin(\psi)\sin(\theta) - \cos(\psi) sin(\phi)) \\ \ddot{z} = g - \dfrac{f_t}{m} (\cos(\phi)\cos(\theta)) \end{cases} \tag{2.11}$$

2.4.2 Moments

Euler's equation gives the total torque applied to the quadrotor

$$\boldsymbol{J}\dot{\boldsymbol{\omega}}_{\boldsymbol{B}} + \boldsymbol{\omega}_{\boldsymbol{B}} \wedge \boldsymbol{J}\boldsymbol{\omega}_{\boldsymbol{B}} = \boldsymbol{\Gamma}_{\boldsymbol{f}} - \boldsymbol{\Gamma}_{\boldsymbol{g}} \tag{2.12}$$

where $\wedge$ is the cross product, $\boldsymbol{J}$ is the diagonal inertia matrix given by

$$\boldsymbol{J} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{2.13}$$

$\boldsymbol{\Gamma}_{\boldsymbol{f}}$ denotes the moment developed by the propellers in the body fixed frame

$$\boldsymbol{\Gamma}_{\boldsymbol{f}} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{2.14}$$

$\boldsymbol{\Gamma}_{\boldsymbol{g}}$ is the result of the torques due to gyroscopic effects

$$\boldsymbol{\Gamma}_{\boldsymbol{g}} = \sum_{i=1}^{4} \omega_B \wedge J_r \begin{bmatrix} 0 \\ 0 \\ (-1)^{i+1}\Omega_i \end{bmatrix} \tag{2.15}$$

where J_r represents the rotor inertia, and Ω_i represents the rotational speed of the i^{th} rotor. The simplification $\begin{bmatrix}\dot{\phi} & \dot{\theta} & \dot{\psi}\end{bmatrix}^T = \begin{bmatrix}p & q & r\end{bmatrix}^T$ can be made with the assumption of small angles of movement. From equation (2.12) we then get

$$\begin{cases} \ddot{\phi} = \dfrac{I_y - I_z}{I_x}\dot{\theta}\dot{\psi} - \dfrac{J_r}{I_x}\dot{\theta}\Omega + \dfrac{\tau_x}{I_x} \\[2ex] \ddot{\theta} = \dfrac{I_z - I_x}{I_y}\dot{\phi}\dot{\psi} + \dfrac{J_r}{I_y}\dot{\phi}\Omega + \dfrac{\tau_y}{I_y} \\[2ex] \ddot{\psi} = \dfrac{I_x - I_y}{I_z}\dot{\phi}\dot{\theta} + \dfrac{\tau_z}{I_z} \end{cases} \tag{2.16}$$

where $\Omega = -\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4$ represents the overall residual rotor angular velocity.

2.4.3 Actuator Dynamics

The forces that can be applied to the quadrotor in order to control the behavior are the following forces and torques

$$\begin{aligned} f_t &= b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ \tau_x &= bL(-\Omega_1^2 - \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ \tau_y &= bL(\Omega_1^2 - \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \\ \tau_z &= d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \end{aligned} \tag{2.17}$$

where b is the thrust factor, d is the drag factor, L is the distance between any rotor and the center of the quadrotor, f_t is the vertical thrust, and τ_x, τ_y, τ_z are the torques for each of the angular motions.

The control inputs will be defined as follows

$$\begin{cases} u_1 = f_t = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ u_2 = \tau_x = bL(-\Omega_1^2 - \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ u_3 = \tau_y = bL(\Omega_1^2 - \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \\ u_4 = \tau_z = d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \end{cases} \tag{2.18}$$

2.4.4 Dynamic Model

From Newton's law, Euler's fomula, the definitions and assumptions made, the dynamic model of the quadrotor in the inertial frame can be obtained from Equation (2.11), (2.16) and (2.18) as follows

$$\begin{cases} \ddot{x} = -\dfrac{u_1}{m}\left(\sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta)\right) \\ \ddot{y} = -\dfrac{u_1}{m}\left(\cos(\phi)\sin(\psi)\sin(\theta) - \cos(\psi) sin(\phi)\right) \\ \ddot{z} = g - \dfrac{u_1}{m}\left(\cos(\phi)\cos(\theta)\right) \\ \ddot{\phi} = \dfrac{I_y - I_z}{I_x}\dot{\theta}\dot{\psi} - \dfrac{J_r}{I_x}\dot{\theta}\Omega + \dfrac{u_2}{I_x} \\ \ddot{\theta} = \dfrac{I_z - I_x}{I_y}\dot{\phi}\dot{\psi} + \dfrac{J_r}{I_y}\dot{\phi}\Omega + \dfrac{u_3}{I_y} \\ \ddot{\psi} = \dfrac{I_x - I_y}{I_z}\dot{\phi}\dot{\theta} + \dfrac{u_4}{I_z} \end{cases} \tag{2.19}$$

2.5 Parrot Mambo Minidrone

For the simulations made in this work, the Parrot Mambo minidrone used in the MathWorks Minidrone Competition will be used. The Parrot Mambo is equipped with an ultrasonic sensor, accelerometer, gyroscope and pressure sensor. The sensors measure acceleration, angular velocity, altitude and displacement in the horizontal plane. It is also equipped with a downward facing camera with a resolution of 120x60 pixels and a refresh rate of 60 frames per second. The physical parameters of the Parrot Mambo are listed in Table 2.1.

Specification	Parameter	Value	Unit
Quadrotor mass	m	0.063	kg
Lateral moment arm	L	0.0624	m
Rotor moment of inertia	J_r	1.021×10^{-7}	kgm^2
Rolling moment of inertia	I_x	5.82857×10^{-5}	kgm^2
Pitching moment of inertia	I_y	7.16914×10^{-5}	kgm^2
Yawing moment of inertia	I_z	1×10^{-4}	kgm^2
Thrust coefficient	b	0.0107	Ns^2
Drag coefficient	d	7.8264×10^{-4}	Nms^2
Maximum total thrust	$u_1^{\max}$	1.2	N
Maximum roll/pitch moment	$u_{2,3}^{\max}$	1×10^{-3}	Nm
Maximum yaw moment	$u_4^{\max}$	1×10^{-4}	Nm

Table 2.1: Parameters of the Parrot Mambo minidrone.

To take into account that the forces and moments the quadrotor can produce are limited by the physical rotational speed of the rotors, saturated control inputs u_{si} will be implemented, where $i = \{1, 2, 3, 4\}$. The saturated control inputs can be written as follows

$$u_{si} = \text{sat}(u_i) = \begin{cases} u_i^{\max} & \text{IF } u_i \geq u_i^{\max} \\ u_i & \text{IF } -u_i^{\max} < u_i < u_i^{\max} \\ -u_i^{\max} & \text{IF } u_i \leq -u_i^{\max} \end{cases} \tag{2.20}$$

where $u_i^{\max}$ is the maximum control input value listed in Table 2.1.

2.6 3 DOF Hover

The Quanser 3 DOF Hover system [3], shown in Figure 2.10, was used in this work to test how well the developed control algorithms worked in practice. It consists of a planar round frame with four propellers. The frame is mounted on a three degrees of freedom pivot joint, which allows the body to rotate about the roll, pitch and yaw axes. The roll, pitch and yaw angles are measured using high-resolution encoders. Each encoder has a resolution of 8192 counts per revolution, thus the precision of the measured angles is accurate down to 0.0439 degrees. To obtain the rotational velocities, a derivative block and a second-order filter was used. The encoder and motor signals are transmitted through a slip ring mechanism, which allows the yaw axis to rotate continuously about 360°. The minimum and maximum pitch and roll angles are read as ±37.5°.

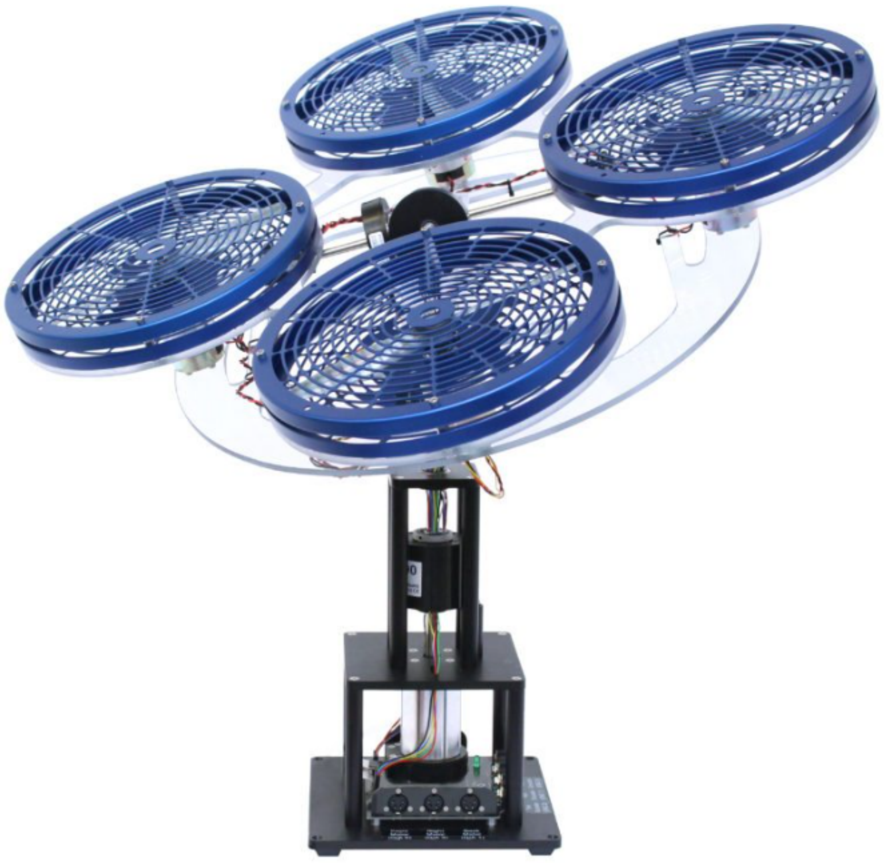


Figure 2.10: The Quanser 3 DOF Hover [3].

The physical parameters of the 3 DOF Hover are listed in Table 2.2.

Specification	Parameter	Value	Unit
Quadrotor mass	m	2.85	kg
Lateral moment arm	L	0.197	m
Rotor moment of inertia	J_r	1.91×10^{-6}	kgm^2
Rolling moment of inertia	I_x	0.0552	kgm^2
Pitching moment of inertia	I_y	0.0552	kgm^2
Yawing moment of inertia	I_z	0.110	kgm^2
Torque thrust constant of motor/propeller	K_t	0.0036	Nm/V
Force-thrust constant of motor/propeller	K_f	0.1188	N/V
Transformation constant	K_v	54.945	rad s/V
Bias voltage	V_{bias}	4	V

Table 2.2: Parameters of the 3 DOF Hover.

The 3 DOF Hover system uses a slightly different coordinate system than what was described earlier, where the z-axis is defined upwards instead of downwards. A positive yaw angle is then defined in the other direction as to what was described earlier. The 3 DOF Hover also uses a plus-configuration, where the different motors are denoted as front, back, right and left. The torques created by the rotors are given by Quanser as a combination of the motor voltages, where we have

- V_f - front motor voltage
- V_b - back motor voltage
- V_l - left motor voltage
- V_r - right motor voltage

Figure 2.11 illustrates the motor placements, propeller rotations, and the positive direction for the yaw angle.

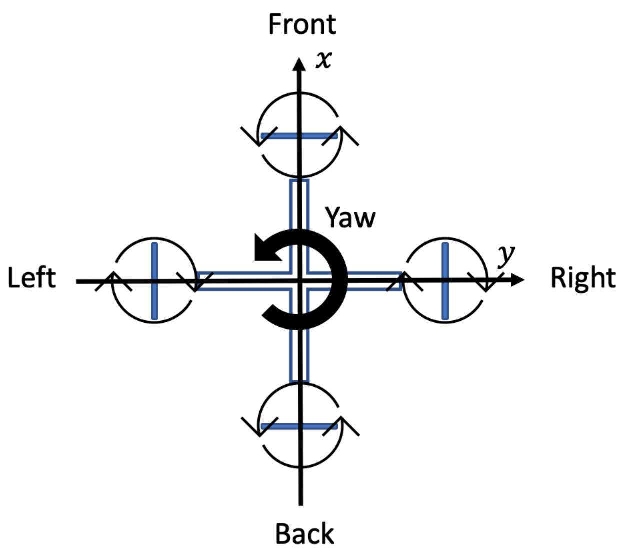


Figure 2.11: Motor placements and rotations for the 3 DOF Hover.

Roll moment is achieved by creating an unbalance in the left and right side forces. Increasing the voltage for the right motor, and decreasing the voltage for the left motor will create a positive roll angle.

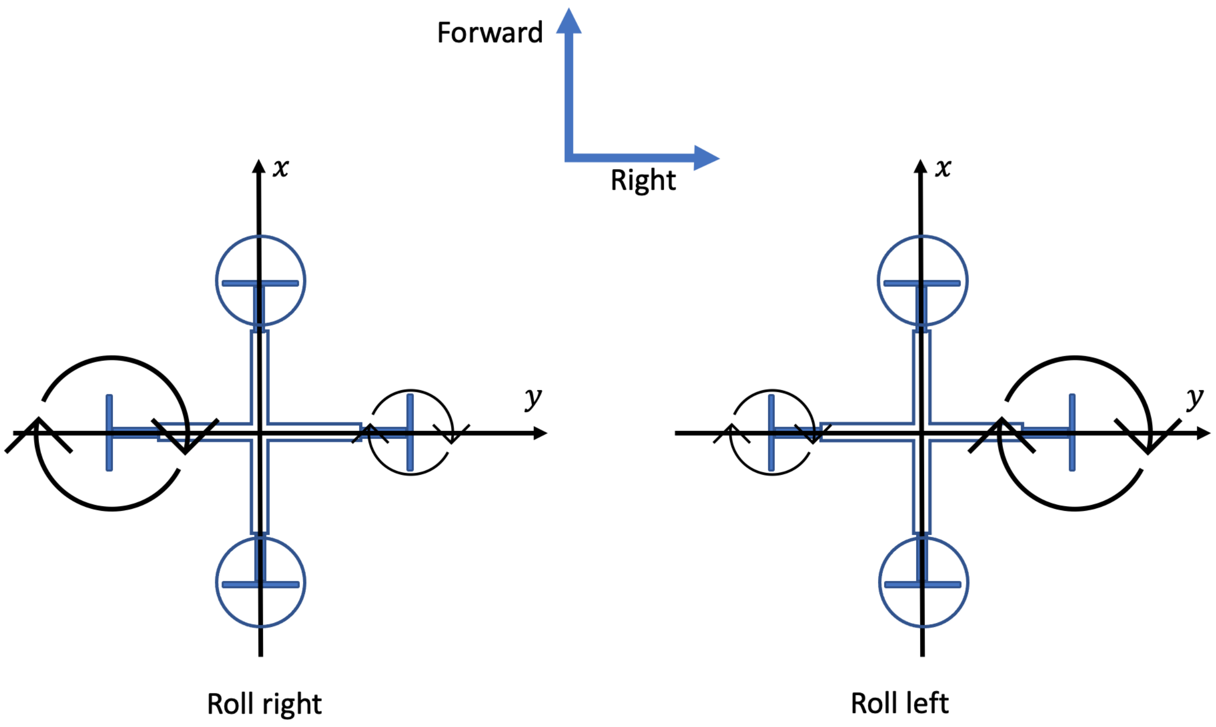


Figure 2.12: Roll in plus-configuration.

Pitch moment is achieved by creating an unbalance in the front and back forces. Thus, increasing the voltage for the front motor, and decreasing the voltage for the back motor will result in a positive pitch angle.

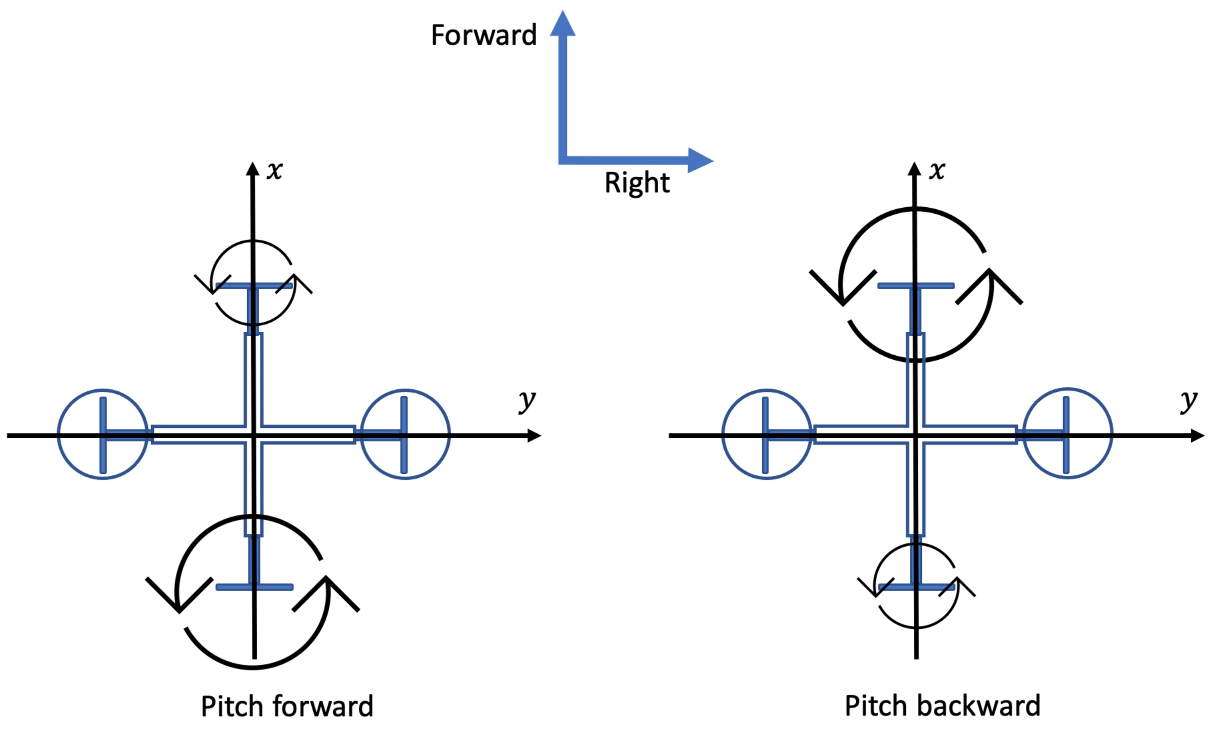


Figure 2.13: Pitch in plus-configuration.

To create a yawing moment, the pairwise anti-clockwise and clockwise rotating rotors are adjusted to cause an unbalance in the torques acting on the body frame. Increasing the voltage for the negatively rotating right and left motors, and decreasing the voltage for the positively rotating front and back motors will decrease the yaw angle. Similarly, increasing the voltage of the negatively rotating front and back motors, and decreasing the voltage of positively rotating right and left motors will increase the yaw angle.

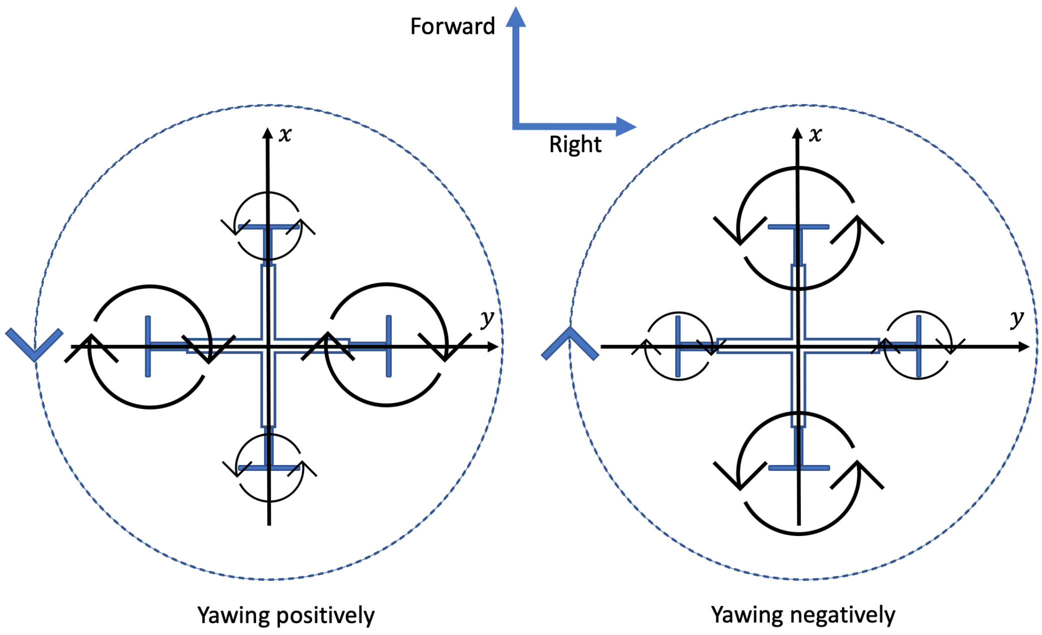


Figure 2.14: Yaw in plus-configuration.

The total thrust and torques are defined by Quanser as

$$\begin{aligned}
f_t &= u_1 = K_f(V_f + V_b + V_r + V_l) \\
\tau_x &= u_2 = LK_f(V_r - V_l) \\
\tau_y &= u_3 = LK_f(V_f - V_b) \\
\tau_z &= u_4 = K_t(V_r + V_l - V_f - V_b)
\end{aligned} \tag{2.21}$$

The overall residual rotor angular velocity Ω from the dynamic equations is defined as

$$\Omega = K_v(V_r + V_l - V_f - V_b) \tag{2.22}$$

Equation (2.21) can be solved for the different motor voltages, so that the control inputs can be transformed into voltages for the different motors as shown below

$$\begin{aligned}
V_f &= \frac{1}{4K_f}u_1 + \frac{1}{2LK_f}u_3 - \frac{1}{4K_t}u_4 \\
V_b &= \frac{1}{4K_f}u_1 - \frac{1}{2LK_f}u_3 - \frac{1}{4K_t}u_4 \\
V_r &= \frac{1}{4K_f}u_1 + \frac{1}{2LK_f}u_2 + \frac{1}{4K_t}u_4 \\
V_l &= \frac{1}{4K_f}u_1 - \frac{1}{2LK_f}u_2 + \frac{1}{4K_t}u_4
\end{aligned} \tag{2.23}$$

A bias voltage, V_{bias}, is added to prevent each propeller from going below zero and being cutoff. This will keep the rotors in motion, which can also help make the system more responsive.

Chapter 3

Control Architecture

The purpose of the control algorithm is for the quadrotor to follow the desired trajectories: $x_d(t)$, $y_d(t)$, $z_d(t)$, $\phi_d(t)$, $\theta_d(t)$, and $\psi_d(t)$. Because the quadrotor is underactuated, meaning that the number of degrees of freedom is larger than the number of control inputs, a nested loop control structure is appropriate. From Equation (2.19), it can be seen that the rotational motion is independent from the translational motion, while the opposite is not true. Therefore an inner loop control can be designed to ensure the desired attitude and altitude, while an outer loop control can be designed to ensure the desired position, by generating a reference signal fed to the inner loop control.

The dynamic model equations for $\ddot{x}$ and $\ddot{y}$ from Equation (2.19), can be rewritten as follows

$$\begin{aligned} \ddot{x} &= -\frac{u_1}{m}u_x \\ \ddot{y} &= -\frac{u_1}{m}u_y \end{aligned} \tag{3.1}$$

where two virtual inputs u_x and u_y are defined to obtain the desired x- and y-position, defined as

$$\begin{aligned} u_x &= -\frac{m}{u_1}\ddot{x} = \sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta) \\ u_y &= -\frac{m}{u_1}\ddot{y} = \cos(\phi)\sin(\psi)\sin(\theta) - \cos(\psi)\sin(\phi) \end{aligned} \tag{3.2}$$

Using the small angle assumption around the hover position, we have $\cos(\theta) \approx \cos(\phi) \approx 1$, $\sin(\theta) \approx \theta$ and $\sin(\phi) \approx \phi$. Equation (3.2) then becomes

$$\begin{aligned} u_x &= \phi\sin(\psi) + \cos(\psi)\theta \\ u_y &= \sin(\psi)\theta - \cos(\psi)\phi \end{aligned} \tag{3.3}$$

The desired pitch and roll angles with their derivatives are then calculated as

$$\begin{aligned}
\phi_d &= \sin(\psi_d)u_x - \cos(\psi_d)u_y \\
\dot{\phi}_d &= \sin(\psi_d)\dot{u}_x - \cos(\psi_d)\dot{u}_y + \cos(\psi_d)u_x\dot{\psi}_d + \sin(\psi_d)u_y\dot{\psi}_d \\
\theta_d &= \cos(\psi_d)u_x + \sin(\psi_d)u_y \\
\dot{\theta}_d &= \cos(\psi_d)\dot{u}_x + \sin(\psi_d)\dot{u}_y - \sin(\psi_d)u_x\dot{\psi}_d + \cos(\psi_d)u_y\dot{\psi}_d
\end{aligned} \tag{3.4}$$

The overall control structure is depicted in Figure 3.1.

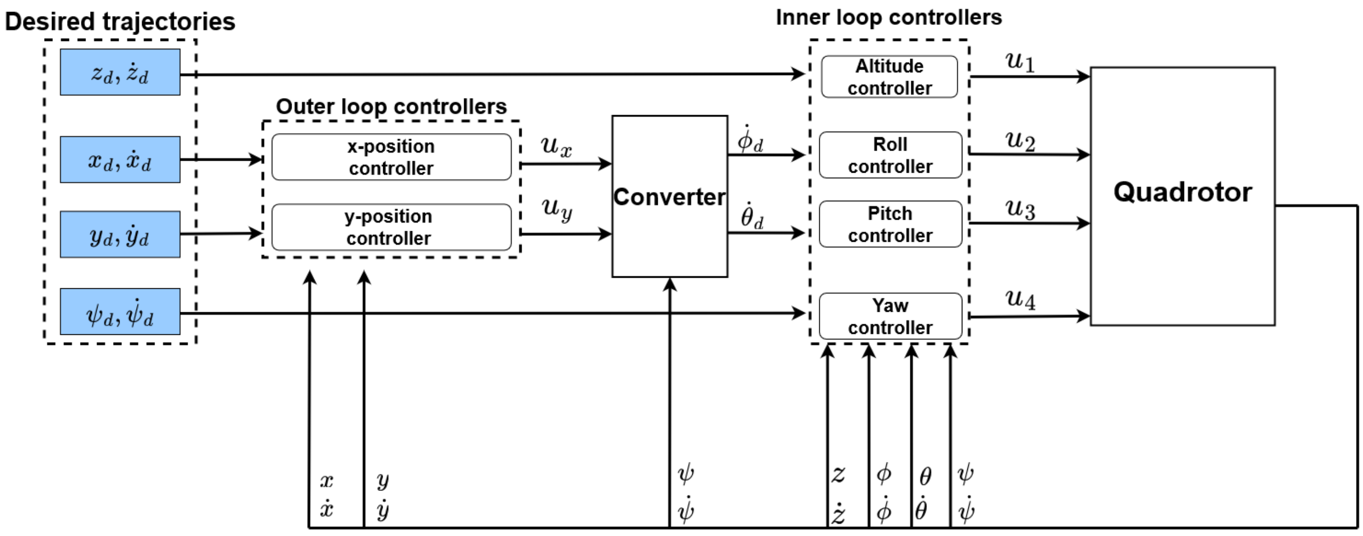


Figure 3.1: Control structure.

All the different controllers proposed in this work will use the same control architecture to ensure that the quadrotor follows the desired trajectories. By employing a consistent control architecture, the proposed controllers exhibit comparable performance characteristics while emphasizing different control strategies. This allows for a fair comparison and evaluation of their effectiveness in achieving desired trajectory tracking for the quadrotor.

Chapter 4

Linear Control Methods

Precise control and advanced regulation techniques are essential in various industries, including production control, robotics, medical technology, and flight control. These applications often demand accurate control of complex systems, particularly in the case of drones, where stability and precise path tracking are crucial.

Among control approaches, linear control systems offer advantages due to their relative simplicity and ease of design compared to nonlinear controllers. In this study, two linear control techniques, namely Proportional Integral Derivative (PID) and Linear Quadratic Regulator (LQR), are specifically designed for the quadrotor UAV.

4.1 PID Control

Throughout the history of regulatory systems in the engineering world, PID control has been applied by many scientists and engineers to industrial applications such as regulating speed, pressure, temperature, and more. The PID controller provides a classic feedback control strategy that adjusts the control inputs based on the error between the desired and actual states [13]. It is widely used in various applications due to its simplicity and effectiveness in achieving stable and accurate control.

4.1.1 PD Controller Design

In this work, a PD controller is used in both the inner and outer loop control. The choice of a PD controller is motivated by the fact that to make a quadrotor hover in the air, two of the Euler angles, roll and pitch, must remain zero. There is therefore no need to eliminate any static errors. The integral part has several drawbacks, including increased settling time, which is undesirable because of the need of a fast response for quadrotor control. The integral term, with it's tendency to accumulate errors over time, can potentially introduce overshoot and instability if not appropriately tuned, making it less suitable for these rapid and dynamic maneuvers.

Inner Loop

The purpose of the inner loop controller is to track and stabilize the quadrotor's altitude and attitude. The desired altitude is given by z_d, while the desired attitude is given by ϕ_d, θ_d, and ψ_d. The inner loop control inputs are given by

$$\begin{aligned} u_1 &= -\left(K_{pz}(z_d - z) + K_{dz}(\dot{z}_d - \dot{z})\right) + mg \\ u_2 &= K_{p\phi}(\phi_d - \phi) + K_{d\phi}(\dot{\phi}_d - \dot{\phi}) \\ u_3 &= K_{p\theta}(\theta_d - \theta) + K_{d\theta}(\dot{\theta}_d - \dot{\theta}) \\ u_4 &= K_{p\psi}(\psi_d - \psi) + K_{d\psi}(\dot{\psi}_d - \dot{\psi}) \end{aligned} \tag{4.1}$$

where K_{pz}, $K_{p\phi}$, $K_{p\theta}$ and $K_{p\psi}$ are the proportional gains, and K_{dz}, $K_{d\phi}$, $K_{d\theta}$ and $K_{d\psi}$ are the derivative gains. The negative sign in the control input u_1 is introduced to account for the fact that the z-axis is defined downwards in this work. A feedforward term, mg, equal to the weight caused by gravity is added to u_1. The feedforward term provides a baseline force that counteracts the effect of gravity. By incorporating the feedforward term, the controller can compensate for the gravitational force without relying solely on the feedback control actions.

Outer Loop

The purpose of the outer loop control is to ensure the desired position given by x_d and y_d, by feeding reference signals to the inner loop control. The outer loop control inputs are given by

$$\begin{aligned} u_x &= -\left(k_{px}(x_d - x) + k_{dx}(\dot{x}_d - \dot{x})\right) \\ u_y &= -\left(k_{py}(y_d - y) + k_{dy}(\dot{y}_d - \dot{y})\right) \end{aligned} \tag{4.2}$$

where K_{px}, K_{py} are the proportional gains, and K_{dx}, K_{dy} are the derivative gains. The negative signs in the control inputs comes from how the virtual inputs u_x and u_y are defined in chapter 3.

4.2 LQR Control

The Linear Quadratic Regulator technique has been widely implemented on the industrial applications, such as the wheeled inverted pendulum vehicle, the voltage-source inverter and seated balance [14]. Compared to PID, an LQR system is considered better for stability and robustness. This means that LQR can handle disturbances and uncertainties in the system better than PID (such as noise and modeling uncertainties). In the LQR, the system dynamics are typically represented in state-space form, which is a mathematical model that describes the behavior of a system using a set of first-order differential equations. There is done relatively little research on the implementation of the LQR technique on the tracking problem of the quadrotor. The realization of the LQR technique on the quadrotor comes with some difficulties, such as [14]: 1) There exist high nonlinearities in the quadrotor dynamics, and the aerodynamic effects. 2) The system dynamics of the quadrotor are subject to several uncertainties, such as the model inaccuracy due to the linearity approximation and parameter perturbations.

4.2.1 State-Space Model of the Quadrotor

The LQR control is designed by using state-space analysis of the quadrotor to achieve stability and trajectory tracking. This is done by defining state variables, input variables, and the system matrices. The state-space representation of a Linear Time-Invariant (LTI) system can be defined as follows

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \\ \boldsymbol{y} &= \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u} \end{aligned} \tag{4.3}$$

where $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ represent the state matrix, input matrix, output matrix, and feedthrough matrix, respectively. The state vector is denoted by $\boldsymbol{x}$, the input/control vector is denoted by $\boldsymbol{u}$, and $\boldsymbol{y}$ is defined as the output vector.

Based to the quadrotor dynamic model from Equation (2.19) and the rewritten form of the x- and y-dynamics including the virtual inputs from Equation (3.1), the decoupled quadrotor dynamics can be represented as the state-space models as shown below

$$\begin{aligned} \dot{\boldsymbol{x}}_{\boldsymbol{x}} &= \boldsymbol{A}_{\boldsymbol{x}}\boldsymbol{x}_{\boldsymbol{x}} + \boldsymbol{B}_{\boldsymbol{x}}u_x \\ \dot{\boldsymbol{x}}_{\boldsymbol{y}} &= \boldsymbol{A}_{\boldsymbol{y}}\boldsymbol{x}_{\boldsymbol{y}} + \boldsymbol{B}_{\boldsymbol{y}}u_y \\ \dot{\boldsymbol{x}}_{\boldsymbol{z}} &= \boldsymbol{A}_{\boldsymbol{z}}\boldsymbol{x}_{\boldsymbol{z}} + \boldsymbol{B}_{\boldsymbol{z}}u_1 \\ \dot{\boldsymbol{x}}_{\boldsymbol{\phi}} &= \boldsymbol{A}_{\boldsymbol{\phi}}\boldsymbol{x}_{\boldsymbol{\phi}} + \boldsymbol{B}_{\boldsymbol{\phi}}u_2 \\ \dot{\boldsymbol{x}}_{\boldsymbol{\theta}} &= \boldsymbol{A}_{\boldsymbol{\theta}}\boldsymbol{x}_{\boldsymbol{\theta}} + \boldsymbol{B}_{\boldsymbol{\theta}}u_3 \\ \dot{\boldsymbol{x}}_{\boldsymbol{\psi}} &= \boldsymbol{A}_{\boldsymbol{\psi}}\boldsymbol{x}_{\boldsymbol{\psi}} + \boldsymbol{B}_{\boldsymbol{\psi}}u_4 \end{aligned} \tag{4.4}$$

where

$$\begin{aligned}
\boldsymbol{x_x} &= \begin{bmatrix} x \\ \dot{x} \end{bmatrix}, \quad \boldsymbol{\dot{x}_x} = \begin{bmatrix} \dot{x} \\ -\frac{u_1}{m} u_x \end{bmatrix} \\
\boldsymbol{x_y} &= \begin{bmatrix} y \\ \dot{y} \end{bmatrix}, \quad \boldsymbol{\dot{x}_y} = \begin{bmatrix} \dot{y} \\ -\frac{u_1}{m} u_y \end{bmatrix} \\
\boldsymbol{x_z} &= \begin{bmatrix} z \\ \dot{z} \end{bmatrix}, \quad \boldsymbol{\dot{x}_z} = \begin{bmatrix} \dot{z} \\ g - \frac{u_1}{m} (\cos\phi \cos\theta) \end{bmatrix} \\
\boldsymbol{x_\phi} &= \begin{bmatrix} \phi \\ \dot{\phi} \end{bmatrix}, \quad \boldsymbol{\dot{x}_\phi} = \begin{bmatrix} \dot{\phi} \\ \frac{I_y - I_z}{I_x} \dot{\theta}\dot{\psi} - \frac{J_r}{I_x} \dot{\theta}\Omega + \frac{u_2}{I_x} \end{bmatrix} \\
\boldsymbol{x_\theta} &= \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}, \quad \boldsymbol{\dot{x}_\theta} = \begin{bmatrix} \dot{\theta} \\ \frac{I_z - I_x}{I_y} \dot{\phi}\dot{\psi} + \frac{J_r}{I_y} \dot{\phi}\Omega + \frac{u_3}{I_y} \end{bmatrix} \\
\boldsymbol{x_\psi} &= \begin{bmatrix} \psi \\ \dot{\psi} \end{bmatrix}, \quad \boldsymbol{\dot{x}_\psi} = \begin{bmatrix} \dot{\psi} \\ \frac{I_x - I_y}{I_z} \dot{\phi}\dot{\theta} + \frac{u_4}{I_z} \end{bmatrix}
\end{aligned} \tag{4.5}$$

Based on the nonlinear equations from Equation (4.5), the state and input matrices can be obtained. These matrices can be linearized considering the equilibrium point when hovering $\boldsymbol{\bar{x}} = 0$, where $\boldsymbol{x}_x = \boldsymbol{x}_y = \boldsymbol{x}_z = \boldsymbol{x}_\phi = \boldsymbol{x}_\theta = \boldsymbol{x}_\psi = 0$, $u_1 = mg$ and $u_x = u_y = u_2 = u_3 = u_4 = 0$. The Jacobian matrices can be obtained by following the general procedure

$$\boldsymbol{A} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix}_{\substack{\boldsymbol{x} = \boldsymbol{\bar{x}} \\ \boldsymbol{u} = \boldsymbol{\bar{u}}}}, \quad \boldsymbol{B} = \begin{bmatrix} \frac{\partial f_1}{\partial u} \\ \frac{\partial f_2}{\partial u} \end{bmatrix}_{\substack{\boldsymbol{x} = \boldsymbol{\bar{x}} \\ \boldsymbol{u} = \boldsymbol{\bar{u}}}} \tag{4.6}$$

The state and input matrices can be obtained as

$$\begin{aligned}
\boldsymbol{A_x} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{B_x} = \begin{bmatrix} 0 \\ -g \end{bmatrix} \\
\boldsymbol{A_y} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{B_y} = \begin{bmatrix} 0 \\ -g \end{bmatrix} \\
\boldsymbol{A_z} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{B_z} = \begin{bmatrix} 0 \\ -\frac{1}{m} \end{bmatrix} \\
\boldsymbol{A_\phi} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{B_\phi} = \begin{bmatrix} 0 \\ \frac{1}{I_x} \end{bmatrix} \\
\boldsymbol{A_\theta} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{B_\theta} = \begin{bmatrix} 0 \\ \frac{1}{I_y} \end{bmatrix} \\
\boldsymbol{A_\psi} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{B_\psi} = \begin{bmatrix} 0 \\ \frac{1}{I_z} \end{bmatrix}
\end{aligned} \tag{4.7}$$

The $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices will be used in designing the LQR controllers.

4.2.2 LQR Controller Design

When implementing the LQR control given the Linear Time-Invariant (LTI) systems obtained in the previous Subsection 4.2.1, the LQR approach allows obtaining an optimal control gain $\boldsymbol{K}$ [14]

$$\boldsymbol{u}(t) = -\boldsymbol{K}\boldsymbol{x}(t) \tag{4.8}$$

which minimizes the cost function

$$J = \int_0^\infty \left(\boldsymbol{x}(t)^T\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}(t)^T\boldsymbol{R}\boldsymbol{u}(t)\right) dt \tag{4.9}$$

where $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{R} \in \mathbb{R}^{m\times m}$, where n is the number of state variables and m is the number of input variables. $\boldsymbol{Q}$ and $\boldsymbol{R}$ are symmetric positive definite weighting matrices for the states and inputs, respectively. A common method of choosing $\boldsymbol{Q}$ and $\boldsymbol{R}$ is to set them as diagonal matrices. The weighting matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ represent the relative importance or cost associated with the state variables and input variables respectively, where in the case of diagonal matrices, the i-th element along the diagonal represents the weight of the i-th state or input. These matrices are fundamental in defining the cost function that the LQR controller seeks to minimize.

Higher values in the diagonal elements of $\boldsymbol{Q}$ indicate that the corresponding state variables are more critical or have higher penalties in the cost function. By adjusting the values of $\boldsymbol{Q}$, the control engineer can prioritize certain state variables over others in terms of their control performance or stability requirements.

Higher values in the diagonal elements of $\boldsymbol{R}$ indicate that the corresponding input variables have larger penalties or are more expensive to use in the control action. By adjusting the values of $\boldsymbol{R}$, the control engineer can control the aggressiveness or effort exerted by each input variable in the control action.

The choice of the $\boldsymbol{R}$ matrix and the $\boldsymbol{Q}$ matrix comes with the trade-off between quick performance and expended energy.

The gain matrix $\boldsymbol{K}$ is computed as

$$\boldsymbol{K} = \boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} \tag{4.10}$$

The value of $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ is initially unknown and can be obtained by solving the algebraic Riccati function as shown below

$$\boldsymbol{P}\boldsymbol{A} + \boldsymbol{A}^T\boldsymbol{P} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} + \boldsymbol{Q} = 0 \tag{4.11}$$

Once $\boldsymbol{P}$ is determined, the feedback gain matrix $\boldsymbol{K}$ can be computed using (4.10).

MATLAB offers a straightforward approach to determine the feedback gain by utilizing a specific command shown in (4.12). This process involves obtaining the state matrix $\boldsymbol{A}$, input matrix $\boldsymbol{B}$ and the $\boldsymbol{Q}$ and $\boldsymbol{R}$ matrices.

$$\boldsymbol{K} = lqr(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{Q}, \boldsymbol{R}) \tag{4.12}$$

The design of the LQR controller is illustrated in Figure 4.1.

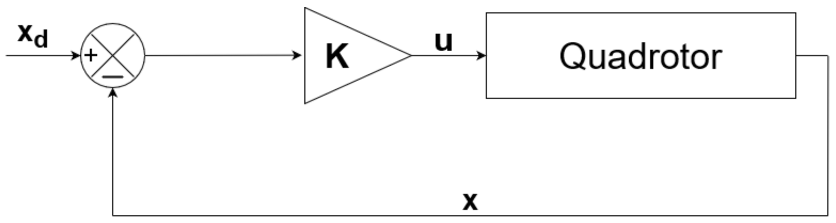


Figure 4.1: Block diagram of LQR controller.

4.3 Summary

In this chapter, we introduced two benchmark controllers for quadrotor control: the Proportional-Derivative (PD) controller and the Linear Quadratic Regulator (LQR). We discussed their design principles, implementation, and performance characteristics. These controllers provide a solid foundation for evaluating the performance of alternative control methods. In the next chapter, we will explore a sliding mode control approach. By leveraging the insights gained from the benchmark controllers, we anticipate that the sliding mode control will offer enhanced robustness, disturbance rejection, and tracking performance for the quadrotor.

Chapter 5

Silding Mode Control

5.1 Introduction to Sliding Mode Control

Sliding mode control (SMC) is a nonlinear control technique that was first developed in the 1950s and later popularized by the Russian mathematician V. Utkin. It was developed as a robust and adaptive control strategy that could work even if the system to be controlled was highly nonlinear or had model uncertainties.

The basic idea of SMC is based on defining a function named the sliding variable. When a properly designed sliding variable becomes equal to zero, it defines the sliding surface. The idea is to steer the state variables of the system to a properly chosen sliding surface, and then keep it there by means of a high frequency switching control signal.

SMC is robust with respect to internal and external disturbances. However, the chattering produced by the high frequency switching of the control can cause problems. Chattering is an undesirable phenomenon of oscillations with finite frequency. Chattering can be harmful because it leads to low control accuracy, high heat loss in electrical power circuits, and high wear of moving mechanical parts. Thus it is one of the main problems when implementing SMC. We will later present some methods that will reduce the chattering phenomena.

5.2 Lyapunov Stability

In order to achieve asymptotic convergence of the sliding variable to zero by means of the control input, Lyapunov function techniques can be applied to the sliding variable dynamics [15]. Lyapunov stability is a type of stability analysis used in dynamical systems theory, which involves searching for a function $V(\boldsymbol{x})$ of the system states called a Lyapunov function. It is named after the Russian mathematician, Aleksandr Lyapunov, who developed the concept in the late 19th century.

Let's first consider the definition of stability [15], where we consider the autonomous system

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}), \;\; f : \mathbb{D} \subset \mathbb{R}^n \to \mathbb{R}^n \tag{5.1}$$

An equilibrium point in the state space is given by vectors $\boldsymbol{x_e}$ satisfying

$$f(\boldsymbol{x_e}) = 0, \;\; \forall \; t > 0 \tag{5.2}$$

For convenience, we state that all the definitions and theorems for the case when the equilibrium point is at the origin $\boldsymbol{x} = 0$. This can be done without the loss of generality because any equilibrium point $\boldsymbol{x_e}$ can be shifted to the origin with the change of variables $\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{x_e}$.

Definition 1 The equilibrium point $\boldsymbol{x} = 0$ of (5.1) is stable if, for each $\epsilon > 0$ there exists a $\delta > 0$ such that if $||\boldsymbol{x}(0)|| < \delta$ then $||\boldsymbol{x}(t)|| < \epsilon$ for all $t > 0$. It is unstable if it is not stable.

Less formally, Definition 1 states that by starting close enough to the equilibrium point, the solution will always remain arbitrarily close to it.

Definition 2 The equilibrium point $\boldsymbol{x} = 0$ of (5.1) is said to be asymptotically stable if it is stable and δ can be chosen such that $||\boldsymbol{x}(0)|| < \delta \Rightarrow \lim\limits_{t\to\infty} \boldsymbol{x}(t) = 0$.

Less formally, Definition 2 states that by starting close enough to the equilibrium point, the solution will always remain arbitrarily close to it, in addition, the trajectory will also diverge to the equilibrium point.

Let's now consider the second method of Lyapunov [16], also referred to as the Lyapunov stability criterion or the Direct Method. The method makes use of a Lyapunov function $V(\boldsymbol{x})$, which is a scalar function and a function of the system states. It is positive and has a negative time-derivative. Conversely, the existence of such a function for a given system implies asymptotic stability.

Theorem 1 Let's consider $\boldsymbol{x_e} = 0$ as an equilibrium point for the autonomous system in (5.1), and $\mathbb{D} \subset \mathbb{R}^n$ be a domain containing $\boldsymbol{x} = 0$. If there exists a function $V(\boldsymbol{x})$ such that

1. $V(\boldsymbol{x}) > 0, \quad$ for $\boldsymbol{x} \neq \boldsymbol{x_e}$
2. $\dot{V}(\boldsymbol{x}) < 0, \quad$ for $\boldsymbol{x} \neq \boldsymbol{x_e}$
3. $V(\boldsymbol{x}) = 0, \quad$ for $\boldsymbol{x} = \boldsymbol{x_e}$
4. $V(\boldsymbol{x}) \to \infty, \quad$ for $||\boldsymbol{x}|| \to \infty$

then $\boldsymbol{x_e}$ is a asymptotically stable equilibrium point. The requirements in Theorem 1 states that the Lyapunov function is required to be a decreasing function, with a minimum point at the origin of the state space. Conceptually, this function can be likened to an energy function in mechanical systems, signifying that the system consistently progresses towards regions of lower energy. As a result, the system can be considered stable.

5.3 Designing Sliding Mode Control

We will first define a sliding variable s in such a way that $s = 0$ gives rise to a differential equation whose solution tends to zero eventually. The sliding surface is described by $s = 0$, which is a particular surface in the state space. After the initial reaching phase, the system states "slides" along the line $s = 0$ by the sliding mode control. The advantage of this is that the dynamic behavior of the system may be tailored by the particular choice of the sliding variable.
The most typical structure for the sliding variable is a linear combination of the form [17]:

$$s = e^{(k)} + \sum_{i=0}^{k-1} \lambda_i e^{(i)} \tag{5.3}$$

where e is the the error signal to be converged to zero, and the k coefficient should be $k = r - 1$, where r is the equal to relative degree of the system, which is the highest derivative of the output variable that appears in the system's dynamic equations.
For a system with a relative degree of $r = 2$, the sliding variable could then be chosen as

$$s = \dot{e} + \lambda e \tag{5.4}$$

When the sliding variable in Equation (5.4) reaches a value of zero, it gives rise to a differential equation with the solution and its derivative

$$\begin{aligned} e &= e(0) \exp(-\lambda t) \\ \dot{e} &= -\lambda e(0) \exp(-\lambda t) \end{aligned} \tag{5.5}$$

which with the choice of $\lambda > 0$ converge to zero asymptotically. In order to achieve asymptotic convergence of the error signal e as in Equation (5.5), the sliding variable s has to be driven to zero in finite time by means of a control input u. Asymptotic stability can be achived by applying Lyapunov function techniques to the sliding variable dynamics, which from Equation (5.4) becomes

$$\dot{s} = \ddot{e} + \lambda \dot{e} \tag{5.6}$$

A Lyapunov function candidate for the sliding variable dynamics can be chosen as [15]

$$V = \frac{1}{2}s^2 \tag{5.7}$$

In order to provide asymptotic stability of Equation (5.6) about the equilibrium point $s = 0$, the following conditions must be satisfied

$$\begin{aligned} &(a) \quad \dot{V} = s\dot{s} < 0 \qquad \text{for } s \neq 0 \\ &(b) \quad \lim_{|s|\to\infty} V = \infty \end{aligned} \tag{5.8}$$

With the Lyapunov function in Equation (5.7), condition (b) will be satisfied. In order to achieve finite time convergence of the sliding variable to zero, condition (a) is not enough. This is because condition (a) only ensures asymptotic convergence, which slows down when close to zero. The so-called reaching law can be implemented to ensure finite time convergence [18]. The reaching law is a differential equation that describes the dynamics of the sliding variable s. It can have the general structure

$$\begin{aligned} &\dot{s} = -\rho\,\mathrm{sign}(s) - \zeta f(s) \\ &\rho, \zeta \geq 0 \\ &s f(s) > 0, \quad \forall s \neq 0 \end{aligned} \tag{5.9}$$

where $f(s)$ is a function of s and the function sign(s) is defined as

$$\mathrm{sign}(s) = \begin{cases} 1 & s > 0 \\ -1 & s < 0 \end{cases} \tag{5.10}$$

and

$$\mathrm{sign}(0) \in [\ -1\ , \quad 1\ ] \tag{5.11}$$

as depicted in Figure 5.1.

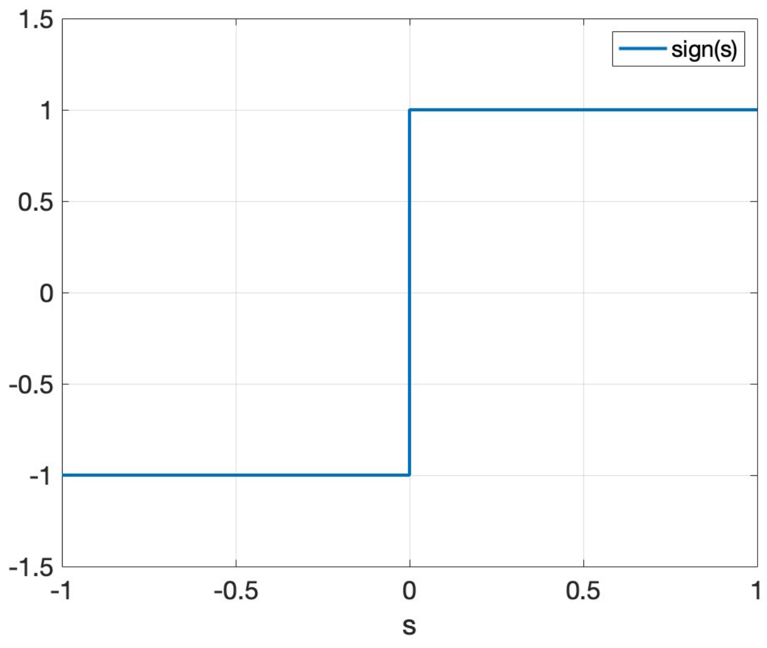


Figure 5.1: Sign function.

A special case of Equation (5.9) is given by the constant reaching law

$$\dot{s} = -\rho \, \text{sign}(s) \tag{5.12}$$

This law forces the the sliding variable to reach the sliding surface, i.e. $s = 0$, at a constant rate where the reaching time is given by $T_r = \frac{|s(t_0)|}{\rho}$, where $s(t_0)$ is the initial value of s [19]. A greater value of ρ leads a faster convergence, but as we will discuss later, also leads to higher chattering intensity.
Another example which we will use in this work, is the constant plus proportional reaching law [18] given by

$$\dot{s} = -\rho \, \text{sign}(s) - \zeta s \tag{5.13}$$

Adding the proportional rate term $-\zeta s$ forces the sliding variable to reach the sliding surface at a faster rate when s is large. If the initial state of the sliding variable satisfies $s(t_0) > 0$, then Equation (5.13) can be represented as

$$\dot{s} = -\rho - \zeta s \tag{5.14}$$

which has the solution [20]

$$s = \left(s(t_0) + \frac{\rho}{\zeta} \right) e^{-\zeta(t - t_0)} - \frac{\zeta}{\rho} \tag{5.15}$$

When the time

$$t = t_0 - \frac{1}{\zeta} \ln \frac{\rho}{\zeta s(t_0) + \rho} \tag{5.16}$$

is satisfied, the sliding variable will have reached the sliding surface. Similarly, if the initial state of the sliding variable satisfies $s(t_0) < 0$, when the time

$$t = t_0 - \frac{1}{\zeta} \ln \frac{\rho}{\rho - \zeta s(t_0)} \tag{5.17}$$

is satisfied, the sliding variable will have reached the sliding surface.
Thus, if the following inequality is satisfied

$$t \geq t_0 - \frac{1}{\zeta} \ln \frac{\rho}{\zeta |s(t_0)| + \rho} \tag{5.18}$$

the sliding variable will have reached the sliding surface at any initial state. Thus the reaching law ensures that the sliding variable reaches the sliding surface in finite time.

By using the constant plus proportional reaching law, condition (a) from Equation (5.8) is satisfied as shown below

$$\dot{V} = s\dot{s} = s(-\rho\,\mathrm{sign}(s) - \zeta s) = -\rho|s| - \zeta s^2 < 0 \tag{5.19}$$

and guarantees Lyapunov stability.

Example: Consider the nonlinear single input system

$$\ddot{x} = f(x,t) + g(x,t)u \tag{5.20}$$

where the control aim is to make the state x track a desired profile x_d. Thus we would want to drive the error $e = x - x_d$ to some small vicinity of zero. The relative degree of the system is $r = 2$, because it is the highest derivative of the system's dynamic equations. The sliding variable can then be chosen according to Equation (5.3) as

$$s = \dot{e} + \lambda e \tag{5.21}$$

In order to drive the sliding variable s to zero, and keep it thereafter, the constant plus proportional reaching law can be used. The derivative of the sliding variable is given by

$$\begin{aligned} \dot{s} &= \ddot{e} + \lambda\dot{e} = \ddot{x} - \ddot{x}_d + \lambda(\dot{x} - \dot{x}_d) \\ &= f(x,t) + g(x,t)u - \ddot{x}_d + \lambda(\dot{x} - \dot{x}_d) \end{aligned} \tag{5.22}$$

The control law can then be chosen as

$$u = \frac{1}{g(x,t)} \left(-f(x,t) + \ddot{x}_d - \lambda(\dot{x} - \dot{x}_d) - \rho\,\mathrm{sign}(s) - \zeta s\right) \tag{5.23}$$

Substituting the control law in Equation (5.23) into Equation (5.6) leads to

$$\dot{s} = -\rho - \zeta s \tag{5.24}$$

and the reaching law is satisfied, which with the choice of $\rho > 0$ and $\zeta > 0$, and using the Lyapunov function candidate

$$V = \frac{1}{2}s^2 \tag{5.25}$$

will satisfy condition (a) from Equation (5.8) as shown in Equation (5.19). Consequently a control law u that drives the sliding variable s to zero in finite time is given by

$$u = \frac{1}{g(x,t)}\left(-f(x,t) + \ddot{x}_d - \lambda(\dot{x} - \dot{x}_d) - \rho\,\mathrm{sign}(s) - \zeta s\right) \tag{5.26}$$

From the example it is clear that $\dot{s}$ must be a function of the control input u in order to successfully design a control law that satisfies the reaching law and guarantees Lyapunov stability. This must be taken into account when designing the sliding variable.

5.4 Integral Sliding Mode Control Design

In this work both the inner and outer loop will consist of the robust Integral Sliding Mode Control (ISMC). The advantage of ISMC control over the conventional SMC control is the steady-state error against disturbances. In ISMC an integral action is added to the sliding variable. For a system with a relative degree of 2, the sliding variable could be designed as

$$s = \dot{e} + \lambda e + k\int_0^t e \, d\tau \tag{5.27}$$

where e is the tracking error and λ, $k > 0$ are constants.

5.4.1 Inner Loop ISMC

The purpose of the inner loop controller is to track and stabilize the quadrotor's altitude and attitude. The desired altitude is given by z_d, while the desired attitude is given by ϕ_d, θ_d and ψ_d. The tracking errors is then defined as

$$\begin{aligned} e_z &= z - z_d \\ e_\phi &= \phi - \phi_d \\ e_\theta &= \theta - \theta_d \\ e_\psi &= \psi - \psi_d \end{aligned} \tag{5.28}$$

where z is the actual altitude and ϕ, θ and ψ is the actual attitude. The quadrotor dynamics is a set of second-order differential equations, thus the sliding variables can be designed as

$$\begin{aligned} s_z &= \dot{e}_z + \lambda_z e_z + k_z \int_0^t e_z \, d\tau \\ s_\phi &= \dot{e}_\phi + \lambda_\phi e_\phi + k_\phi \int_0^t e_\phi \, d\tau \\ s_\theta &= \dot{e}_\theta + \lambda_\theta e_\theta + k_\theta \int_0^t e_\theta \, d\tau \\ s_\psi &= \dot{e}_\psi + \lambda_\psi e_\psi + k_\psi \int_0^t e_\psi \, d\tau \end{aligned} \tag{5.29}$$

where λ_z, λ_ϕ, λ_θ, $\lambda_\psi > 0$ and k_z, k_ϕ, k_θ, $k_\psi > 0$ are constants and part of the ISMC controller gains. In order to drive the sliding variables to zero, and keep it thereafter, the constant plus proportional reaching law is used, which is given by

$$\begin{aligned} &\dot{s} = -\rho \operatorname{sign}(s) - \zeta s \\ &\rho, \zeta > 0 \end{aligned} \tag{5.30}$$

which means

$$\begin{aligned} \dot{s}_z &= -\rho_z \text{sign}(s_z) - \zeta_z s_z \\ \dot{s}_\phi &= -\rho_\phi \text{sign}(s_\phi) - \zeta_\phi s_\phi \\ \dot{s}_\theta &= -\rho_\theta \text{sign}(s_\theta) - \zeta_\theta s_\theta \\ \dot{s}_\psi &= -\rho_\psi \text{sign}(s_\psi) - \zeta_\psi s_\psi \end{aligned} \tag{5.31}$$

where ρ_z, ρ_ϕ, ρ_θ, $\rho_\psi > 0$ and ζ_z, ζ_ϕ, ζ_θ, $\zeta_\psi > 0$ are constants and part of the ISMC controller gains. The derivatives of the defined sliding variables are given by

$$\begin{aligned} \dot{s}_z &= \ddot{e}_z + \lambda_z \dot{e}_z + k_z e_z \\ &= \ddot{z} - \ddot{z}_d + \lambda_z(\dot{z} - \dot{z}_d) + k_z(z - z_d) \\ \dot{s}_\phi &= \ddot{e}_\phi + \lambda_\phi \dot{e}_\phi + k_\phi e_\phi \\ &= \ddot{\phi} - \ddot{\phi}_d + \lambda_\phi(\dot{\phi} - \dot{\phi}_d) + k_\phi(\phi - \phi_d) \\ \dot{s}_\theta &= \ddot{e}_\theta + \lambda_\theta \dot{e}_\theta + k_\theta e_\theta \\ &= \ddot{\theta} - \ddot{\theta}_d + \lambda_\theta(\dot{\theta} - \dot{\theta}_d) + k_\theta(\theta - \theta_d) \\ \dot{s}_\psi &= \ddot{e}_\psi + \lambda_\psi \dot{e}_\psi + k_\psi e_\psi \\ &= \ddot{\psi} - \ddot{\psi}_d + \lambda_\psi(\dot{\psi} - \dot{\psi}_d) + k_\psi(\psi - \psi_d) \end{aligned} \tag{5.32}$$

Then using the dynamic model equations from Equation (2.19) for $\ddot{z}$, $\ddot{\phi}$, $\ddot{\theta}$ and $\ddot{\psi}$, the control inputs for the inner loop which satisfies the reaching laws from Equation (5.31) become

$$\begin{aligned}
u_1 &= \frac{m}{\cos(\phi)\cos(\theta)}\left(\lambda_z(\dot{z}-\dot{z}_d)+g-\ddot{z}_d+k_z(z-z_d)+\rho_z\text{sign}(s_z)+\zeta_z s_z\right)\\
u_2 &= I_x\left(-\lambda_\phi(\dot{\phi}-\dot{\phi}_d)-\frac{I_y-I_z}{I_x}\dot{\theta}\dot{\psi}+\frac{J_r}{I_x}\dot{\theta}\Omega+\ddot{\phi}_d-k_\phi(\phi-\phi_d)-\rho_\phi\text{sign}(s_\phi)-\zeta_\phi s_\phi\right)\\
u_3 &= I_y\left(-\lambda_\theta(\dot{\theta}+\dot{\theta}_d)-\frac{I_z-I_x}{I_y}\dot{\phi}\dot{\psi}-\frac{J_r}{I_y}\dot{\phi}\Omega+\ddot{\theta}_d-k_\theta(\theta-\theta_d)-\rho_\theta\text{sign}(s_\theta)-\zeta_\theta s_\theta\right)\\
u_4 &= I_z\left(-\lambda_\psi(\dot{\psi}+\dot{\psi}_d)-\frac{I_x-I_y}{I_z}\dot{\phi}\dot{\theta}+\ddot{\psi}_d-k_\psi(\psi-\psi_d)-\rho_\psi\text{sign}(s_\psi)-\zeta_\psi s_\psi\right)
\end{aligned} \tag{5.33}$$

5.4.2 Outer Loop ISMC

The purpose of the outer loop control is to ensure the desired position given by x_d and y_d, by feeding reference signals to the inner loop control. The tracking errors for the position are defined as follows

$$\begin{aligned}
e_x &= x - x_d\\
e_y &= y - y_d
\end{aligned} \tag{5.34}$$

where x and y represent the actual position of the quadrotor. Then, following the exact same procedure as with the inner loop ISMC, but now using the virtual inputs u_x and u_y and the following relationship with the virtual control inputs defined in Chapter 3

$$\begin{aligned}
\ddot{x} &= -\frac{u_1}{m}u_x\\
\ddot{y} &= -\frac{u_1}{m}u_y
\end{aligned} \tag{5.35}$$

the virtual control inputs become

$$\begin{aligned}
u_x &= \frac{m}{u_1}\left(\lambda_x(\dot{x}+\dot{x}_d)-\ddot{x}_d+k_x(x-x_d)+\rho_x\text{sign}(s_x)+\zeta_x s_x\right)\\
u_y &= \frac{m}{u_1}\left(\lambda_y(\dot{y}+\dot{y}_d)-\ddot{y}_d+k_y(y-y_d)+\rho_y\text{sign}(s_y)+\zeta_y s_y\right)
\end{aligned} \tag{5.36}$$

where λ_x, $\lambda_y > 0$, k_x, $k_y > 0$, ρ_x, $\rho_y > 0$ and ζ_x, $\zeta_y > 0$ are constants and part of the ISMC controller gains. The sliding variables s_x and s_y have the same structure as Equation (5.27), and are given by

$$\begin{aligned} s_x &= \dot{e}_x + \lambda_x e_x + k_x \int_0^t e_x \, d\tau \\ s_y &= \dot{e}_y + \lambda_y e_y + k_y \int_0^t e_y \, d\tau \end{aligned} \tag{5.37}$$

5.5 Chattering Attenuation

The goal of the controller is to ensure that the system remains on the sliding surface by applying a discontinuous control input that switches the system from one side of the sliding surface to the other. In practice the switching is not perfect, for instance the switching is not instantaneous and the model uncertainties and disturbances means that the sliding surface is only known with a finite precision. The switching imperfection leads to chattering, which is an undesirable phenomena in practice because it leads to high control activity. We will present some methods to reduce chattering.

5.5.1 Quasi-Sliding Mode

One way of reducing chattering is to make the control input continuous, where the discontinuous function in the controller inputs, which is the function $\text{sign}(s)$, is approximated with a continuous function. There are several ways to approximate the sign function, but we will look at the following continuous approximation

$$\text{sign}(s) \approx \frac{s}{|s| + \epsilon} \tag{5.38}$$

where ϵ is a small scalar constant. The function is shown in Figure 5.2 for different values of ϵ. When ϵ approaches a value of zero the approximation becomes more accurate, and we have

$$\lim_{\epsilon \to 0} \frac{s}{|s| + \epsilon} = \text{sign}(s) \tag{5.39}$$

The value of ϵ should be chosen according to the trade off of the requirement of an ideal performance to that of ensuring a smooth control input. The smooth control input cannot ensure a finite time convergence of the sliding variable to zero, but a vicinity around zero. Thus obtaining a smoother control input comes at the price of a loss of robustness and a loss of accuracy.

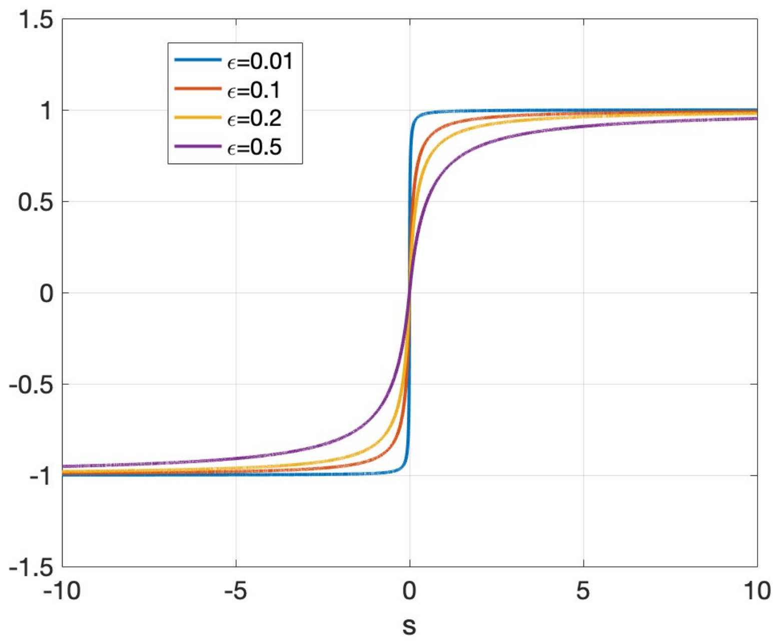


Figure 5.2: Continuous sign approximation function.

In sliding mode control, the sliding variable is driven to zero in finite time. When the approximation of the sign function is implemented, the sliding variable is only driven to a vicinity around zero, thus the smooth control is then called a quasi-sliding mode control.

5.5.2 Adaptive Fuzzy Gain Scheduling

Another way to improve chattering is to implement Adaptive Fuzzy Gain Scheduling (AFGS). When the system state has reached the sliding surface, the repeated switching of the controller induces chattering where the gain of the sign function ρ determines the chattering intensity. A fuzzy logic system can be implemented to construct AFGS where the control gain ρ is scheduled adaptively with the sliding variable to reduce the chattering intensity.

Fuzzy Logic System

A Fuzzy Logic System (FLS) is a way of handling numerical data with linguistic knowledge. It is a nonlinear mapping of an input vector into a scalar output, and it allows for a degree of truth instead of the binary true/false used in classical logic. For example, instead of describing the input value with the values High(1) or Low(0), the input variable could be described using more categories, as for example very low, low, medium, high and very high. Figure 5.3 shows a typical Takagi-Sugeno-Kang fuzzy logic architecture, which is used in this work. There are two inputs

x and y, as well as two output values w and z. Another common fuzzy logic architecture is the Mamdani system.

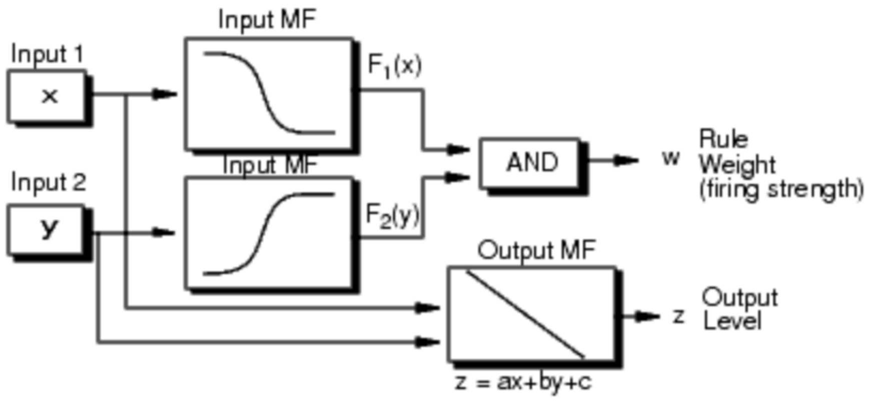


Figure 5.3: Typical Takagi-Sugeno-Kang fuzzy logic architecture [4].

The Takagi-Sugeno-Kang fuzzy architecture consists of:

Fuzzification: It is used to convert crisp inputs into fuzzy sets. Crisp inputs are the numerical values, while fuzzy sets are a vector consisting of the degree the input values are a member of each membership function. A membership function is a graph that defines how each point in the input is mapped to a membership value between 0 and 1. The membership functions are often of the types singleton, Gaussian or triangular. For example, Figure 5.4 shows unified Gaussian membership functions. The value of the input are on the x-axis while the degrees of membership are on the y-axis. The membership functions are usually given names that allow for interpretation in a linguistic manner. The fuzzy set in Figure 5.4 is defined in the following way: {NL, NS, ZE, PS, PL}, where "NL" indicates negative large, "NS" indicates negative small, "ZE" indicates zero, "PS" indicates positive small, and "PL" indicates positive large.

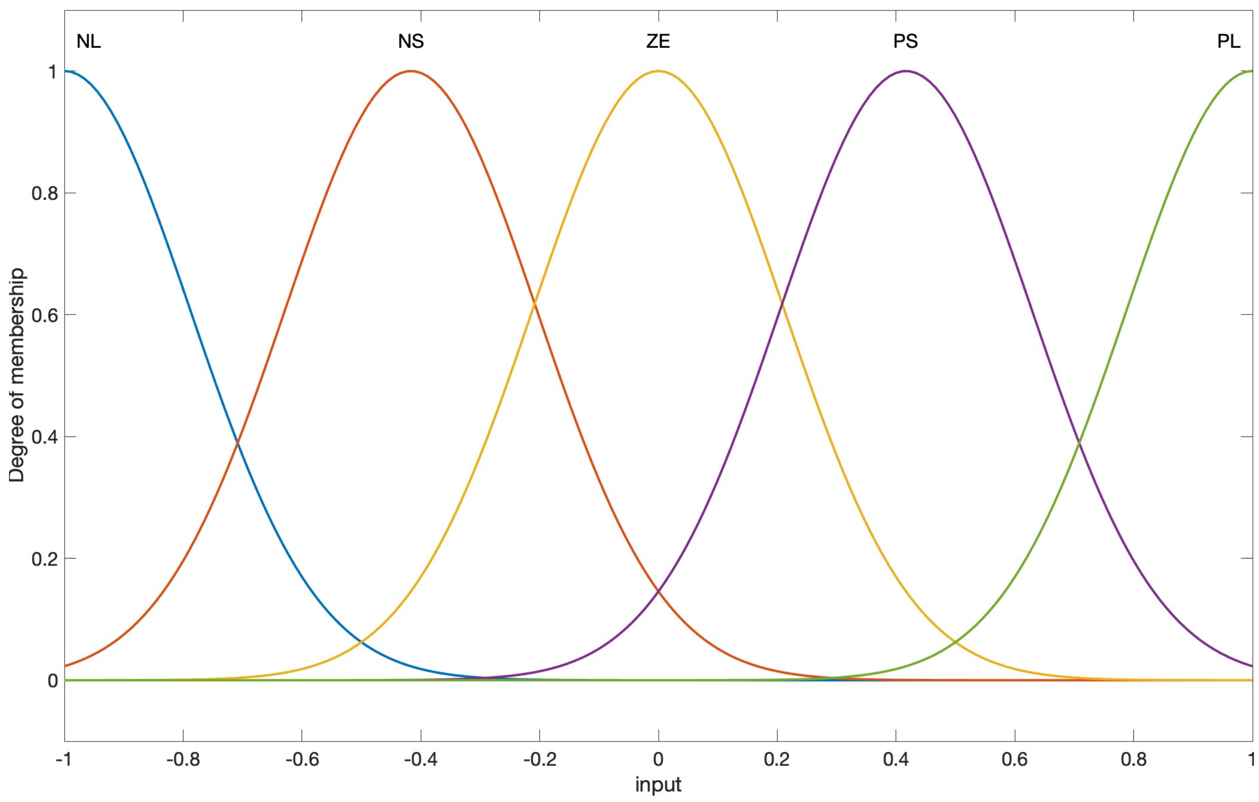


Figure 5.4: FLS membership functions.

Rules structure and defuzzification process: It contains the set of rules which is usually in the form of IF-THEN statements, which is often provided by experts. Each rule generates two values, the output level z and the firing strength w. A typical rule in a Takagi-Sugeno-Kang fuzzy system with two input variables has the form

$$\text{IF } x = F_1(x) \text{ AND } y = F_2(y) \text{ THEN } z = ax + by + c \tag{5.40}$$

where x and y are the two input values, $F_1(x)$ and $F_2(x)$ are membership functions for each of the input values and z is the output level. For a zero order Takagi-Sugeno-Kang fuzzy system the output level z is a constant, which means $a = b = 0$ and $z = c$.

The output level z_i of each rule is weighted by the firing strength w of the rule. The firing strength can be derived as

$$w_i = \min(F_1(x), F_2(y)) \tag{5.41}$$

where $F_1(...)$ and $F_2(...)$ are the membership functions for each input x and y respectively. The final output of the fuzzy logic system is the weighted average of all rule outputs, computed as

$$\text{output} = \frac{\sum_{i=1}^{N} w_i z_i}{\sum_{i=1}^{N} w_i} \tag{5.42}$$

where N is the number of rules.

Design of AFGS for Chattering Attenuation

A Fuzzy Logic System (FLS) will be designed to adaptively change the control gain ρ with the sliding varible. The sliding variable s and its derivative $\dot{s}$ will be the two inputs to the FLS, while the output will be the control parameter ρ. Unified Gaussian membership functions will be used to represent the FLS input membership functions, while the output is represented by a constant membership function. The input sets are defined as {NL, NS, ZE, PS, PL}, which indicates negative large, negative small, zero, positive small and positive large respectively. The output sets are defined as {VL, L, M, H, VH}, which indicates very low, low, medium, high and very high respectively. The input membership functions are shown in Figure 5.5, while the FLS surface is shown in Figure 5.6.

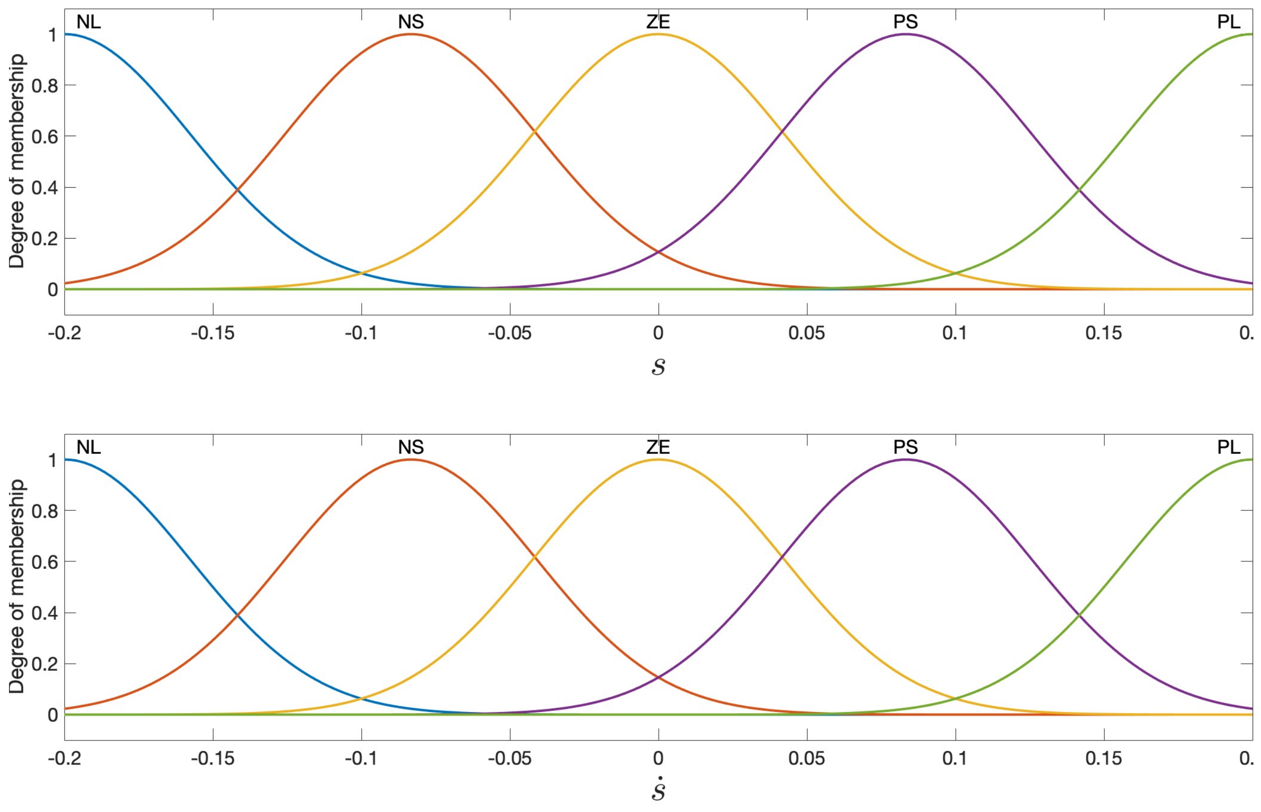


Figure 5.5: FLS inputs membership functions.

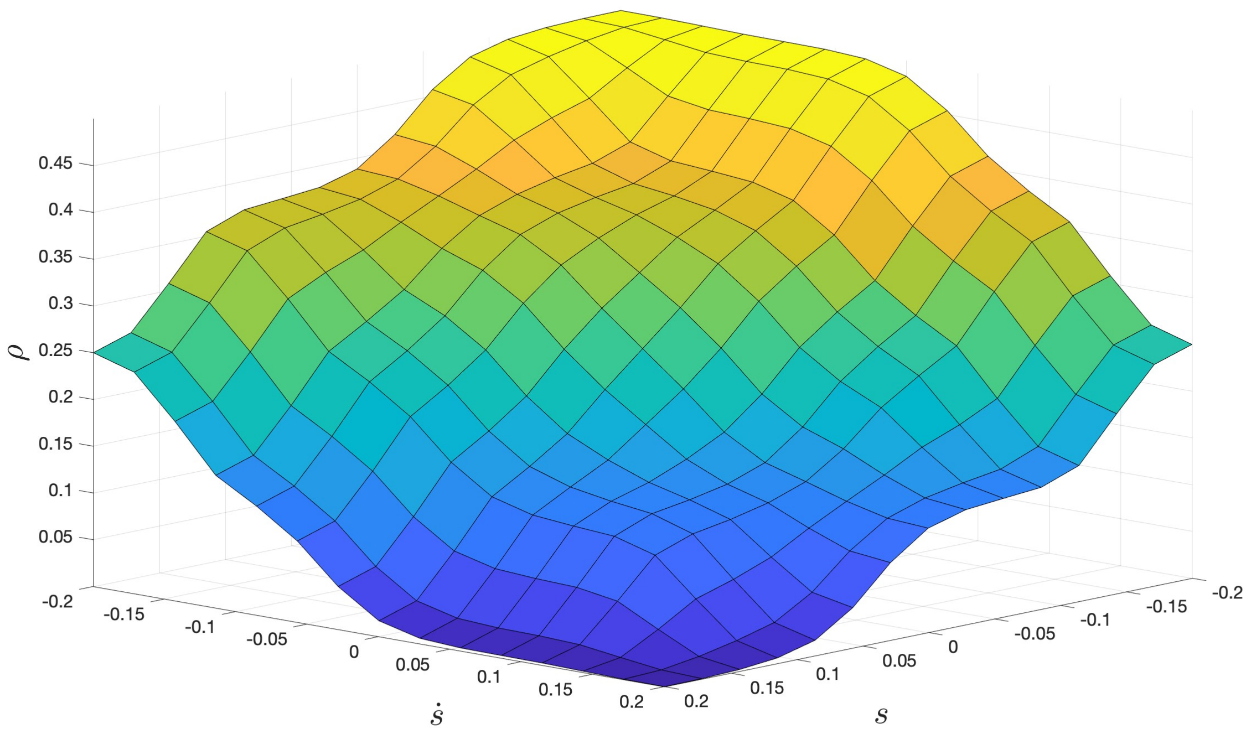


Figure 5.6: FLS surface.

The rules are designed such that when the state trajectory deviates from the sliding surface, the control gain is larger. When the state trajectory approaches the sliding surface the control gain is reduced to reduce chattering. Table 5.1 shows the fuzzy rules [21].

k		$\dot{\mathbf{s}}$				
		NL	**NS**	**ZE**	**PS**	**PL**
s	**PL**	H	L	VL	VL	VL
	PS	H	M	L	L	VL
	ZE	H	H	M	L	L
	NS	VH	H	H	M	L
	NL	VH	VH	VH	H	M

Table 5.1: Fuzzy rules.

The fuzzy logic system is implemented using the Fuzzy Logic Toolbox in MATLAB, and the Fuzzy Logic Designer app which lets you design, test, and tune a fuzzy inference system.

Chapter 6

Tuning Controllers by Genetic Algorithm

Finding good control parameters is essential in the designing of a control that can effectively regulate a system's behavior. The controllers parameters determine how the controller responds to changes in the system's behavior, and if they are not appropriately chosen, the controller may either fail to regulate the system or cause instability. An optimization algorithm called genetic algorithm is used in this work to find optimal control parameters.

6.1 Genetic Algorithm

Genetic algorithm (GA) is an optimization algorithm based on the Darwinian principle of survival of the fittest in nature. The algorithm repeatedly modifies a population of individual solutions called chromosomes, which consist of genes which represent each parameter. Chromosomes are considered as points in the solution space. For each step called a generation, GA selects chromosomes from the current population to be parents and uses them to produce the children for the next generation. The parents are chosen based on their fitness value, described by a fitness function which is the defined function you want to optimize. The new generation of candidate solutions is then used in the next iteration. The next paragraphs will go through the steps in genetic algorithm.

Initial population: The algorithm starts with creating a random initial population, where the objective is to spread the solutions around the search space as uniformly as possible. This will increase the chance of finding the global optima. If information is available in advance about the approximate location of the optima point, upper and lower bounds can be set for the parameters to reduce the search space.

Selection: For each step, the algorithm uses the current populations to create the children for the next generation. The fittest individuals in the population contributes more in the production of the next generation. The convergence rate of GA depends upon the selection pressure, which refers to the degree to which individuals with more desirable traits are favored in the selection process. It determines how much influence the fitter individuals have in the evolution of the population. The most well-known selection techniques are roulette wheel, rank, tournament, boltzmann, and stochastic universal sampling [5]. In this work the tournament selection technique was used. The individuals are selected according to their fitness values from a stochastic roulette wheel in pairs. After the selection, the individuals with higher fitness value are added to the next generation.

Crossover: After the individuals have been chosen from the selection operator, they are employed to create a new generation. In crossover two or more individuals are chosen as parents and are combined to create a new individual for the next generation. There are different techniques for the crossover operator where the most well-known operators are single-point, two-point, k-point, uniform, partially matched, order, precedence preserving crossover, shuffle, reduced surrogate and cycle [5]. In this work, crossing of a single-point was used. In a single point crossover, a random crossover point is selected. The chromosomes of two parents solutions are swapped before and after the single point. Figure 6.1 shows the genetic information after swapping, where the line represents the single point. It replaced the tail array bits of both the parents to get the new offspring.

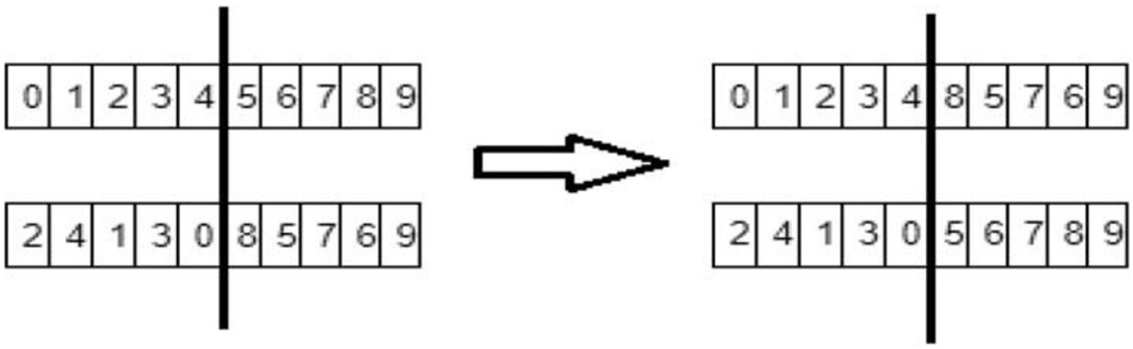


Figure 6.1: Swapping genetic information after a crossover point [5].

Mutation: In the mutation operator individuals genes are altered with, to create new individuals for the the next generation. The altering is randomly chosen such that it maintains the diversity of the population by introducing another level of randomness. The mutation operator prevents solutions from becoming similar and increases the probability of avoiding local solutions to the genetic algorithm. The mutation rate is usually set low to not let the algorithm be a primitive random search. Some of the popular mutation techniques are power mutation, uniform, non-uniform, Gaussian, shrink, supervised mutation, uniqueness mutation and Varying probability of mutation [22].

Termination: The algorithm is usually chosen to terminate when it has ran for a certain number of generations, a solution has achieved a satisfactory fitness level, or if the population has converged to a local optimum point.

6.2 Performance Indices

The fitness function is the driving force, which plays an important role in selecting the fittest individual in every iteration of the algorithm. The fitness function is similar to the inverse of a cost-function in optimization. One or a combination of the four commonly used performance indices, Integral Squared Error (ISE), Integral Absolute Error (IAE), Integral Time Squared Error (ITSE) and Integral Time Absolute Error (ITAE), can be an effective approach in the case of minimizing the errors. They are given by

$$\begin{aligned} ISE &= \int\limits_0^t e(t)^2\,dt \\ IAE &= \int\limits_0^t |e(t)|\,dt \\ ITSE &= \int\limits_0^t te(t)^2\,dt \\ ITAE &= \int\limits_0^t t|e(t)|\,dt \end{aligned} \tag{6.1}$$

where $e(t)$ is the error signal and t is time.

Integral Squared Error (ISE): ISE integrates the square of the error over time, and will penalise large errors more than small errors. This is because the square of a large error will be much bigger. Control systems that are specified to minimize ISE will tend to tolerate small errors over a longer period of time, while tending to eliminate bigger errors more quickly. This typically leads to a fast response but where you could experience some small amplitude oscillations.

Integral Absolute Error (IAE): IAE integrates the absolute error over time, as taking the absolute value makes sure that the positive and negative errors do not cancel out. It doesn't add weight to any of the errors as with ISE. Control systems that are specified to minimize IAE tend to produce a slower response than with ISE, but typically with less oscillations.

Integral Time Squared Error (ITSE): ITSE integrates the square of the error multiplied with the time over time. It penalises larger errors more than smaller errors, and it also penalises errors which exist after a longer period of time much more than those at the start of the response. Control systems that are specified to minimize ITSE will typically lead to a poor initial response as compared to ISE and IAE, but prioritizes responses with a quick settling time and smaller steady-state errors.

Integral Time Absolute Error (ITAE): ITAE integrates the absolute error multiplied with time over time. It doesn't add more weight to largers errors compared to small errors, but similar to ITSE, it penalises errors which exist after a longer period of time much more than those at the start of the response. Control systems that are specified to minimize ITSE will typically, as with ITSE, lead to a poor initial response as compared to ISE and IAE, but prioritizes response with a quick settling time and smaller steady-state errors.

6.3 Parameter Tuning by GA

The objective of the GA is to find optimal control parameters for the PD, LQR and ISMC. The values of the parameters define each individual in the GA. Each parameter can have a value according to its upper and lower bounds. The upper and lower bounds established for each parameter were determined after having previously tuned the parameters by trial and error. The other variables of the GA include the size of the population, the number of generations, the percentage of crossing between 0 and 1, the selection method, the method of crossover, and percentage of mutation. The size of the population depends on the complexity of the problem to be solved, where it is important to establish an adequate population size according to the problem.

To find good solutions, a fitness function is needed to evaluate and differentiate the good and bad results. In this work, a cost function is defined, which can be viewed as the inverse of a typical fitness function. A good solution is therefore one that minimizes the defined cost function. The integral absolute error (IAE) was chosen as an appropriate performance indicator, and the cost function was defined as the sum of all the integral absolute errors for the linear and angular positions as shown below

$$\text{Cost function} = \sum_i \int_0^t |e_i(t)|\, dt \tag{6.2}$$

where the e_i are the error signals and $i = \{x,\ y,\ z,\ \phi,\ \theta,\ \psi\}$.

The GA tuning approach was done with the use of MATLAB M-files and functions to set the control parameters, and running a Simulink file containing a mathematical model of a quadrotor from Equation (2.19) and the proposed controllers. The Simulink file was run for each solution to check the parameters' resulting performance, and obatin the cost function to search for an optimal solution. In Figure 6.2 the classical steps of the GA are shown.

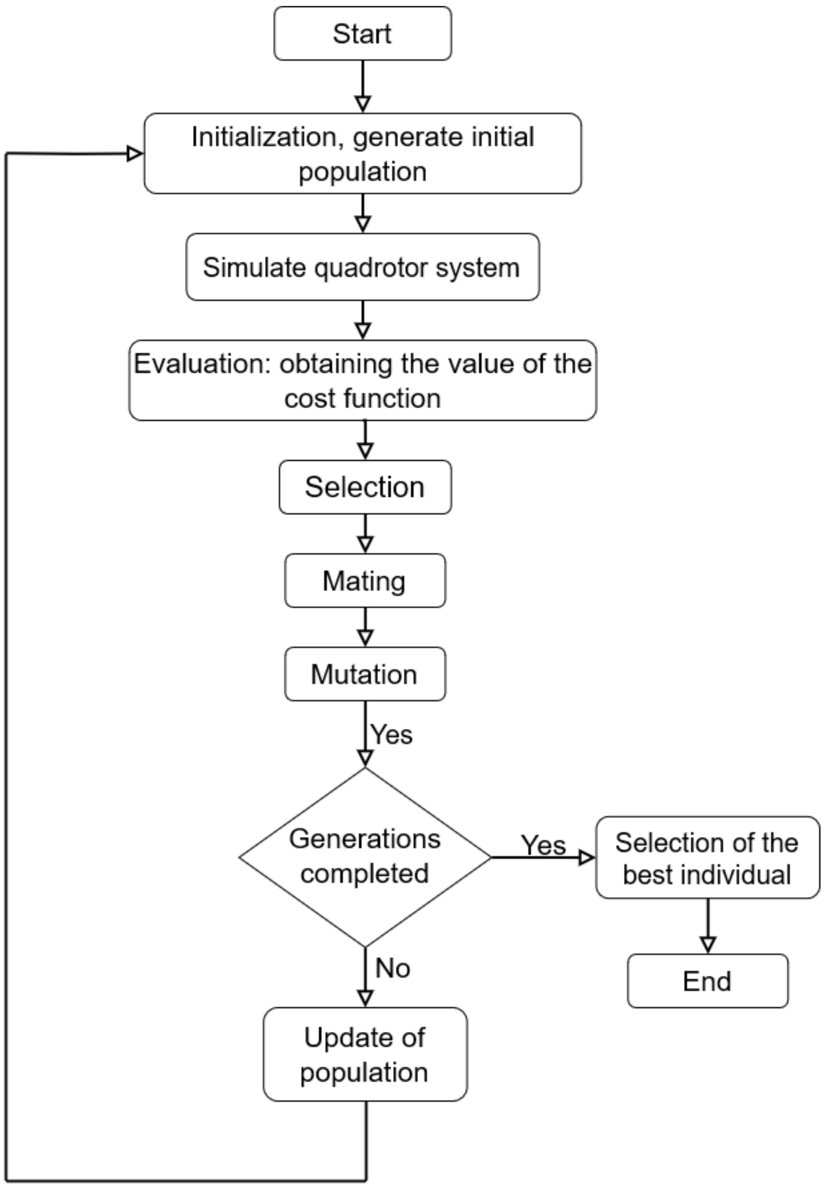

Start
Initialization, generate initial population
Simulate quadrotor system
Evaluation: obtaining the value of the cost function
Selection
Mating
Mutation
Yes
Generations completed
Yes
Selection of the best individual
End
No
Update of population

Figure 6.2: Flow diagram of the GA.

The desired trajectories used in the simulations were chosen in order to try and find an optimal solution for a variation of different 3D trajectories. The desired trajectories used in the simulations were chosen as a combination of a step signal and a ramp signal for the altitude, and a combination of a sine function and a step function for the x- and y-position. The reference signals can be seen in Appendix B. Initially, the desired yaw angle was set to zero, and the yaw controllers were manually tuned through a trial and error process to establish a satisfactory performance. The GA optimization process was then carried out to tune the outer and inner loop controllers, excluding the yaw controller. Once the GA optimization was completed, a separate GA was conducted to find the optimal control parameters specifically for the yaw controller. This optimization process

took advantage of the already optimized control parameters obtained for the other controllers. A ramp reference was set as the desired trajectory for the yaw angle, while the other references were set to zero.
The idea of tuning the yaw controller separately was to account for coupling effects. When the quadrotor rotates in the roll or pitch axes, it can induce undesired yaw motion and disturbances. By tuning the yaw controller independently, it allows for optimizing its parameters when coupling effects are minimal, ensuring better overall control performance.

In this work the tuning was done with 20 generations, crossover rate of 0.8, mutation rate of 0.01, selection per tournament, and crossing of a single point, and otherwise the default options for the genetic algorithm function in MATLAB. The population size was chosen as 10 times the number of control parameters to be tuned, where the outer loop controllers for x and y used the same control parameters. These values were established after conducting a series of tests in terms of population size and number of generations.

Chapter 7

Vision-based line tracking algorithm

7.1 The Flight Control System

The only Simulink block that needs to be changed in the competition is the Flight Control System block. As shown in Figure 7.1, this block contains two main blocks. The Image Processing System block is tasked with doing calculations based on image data coming from the down-facing camera mounted underneath the drone and then sending the results to the Control System block. This block then uses sensor data and the image processing results to calculate actuator inputs.

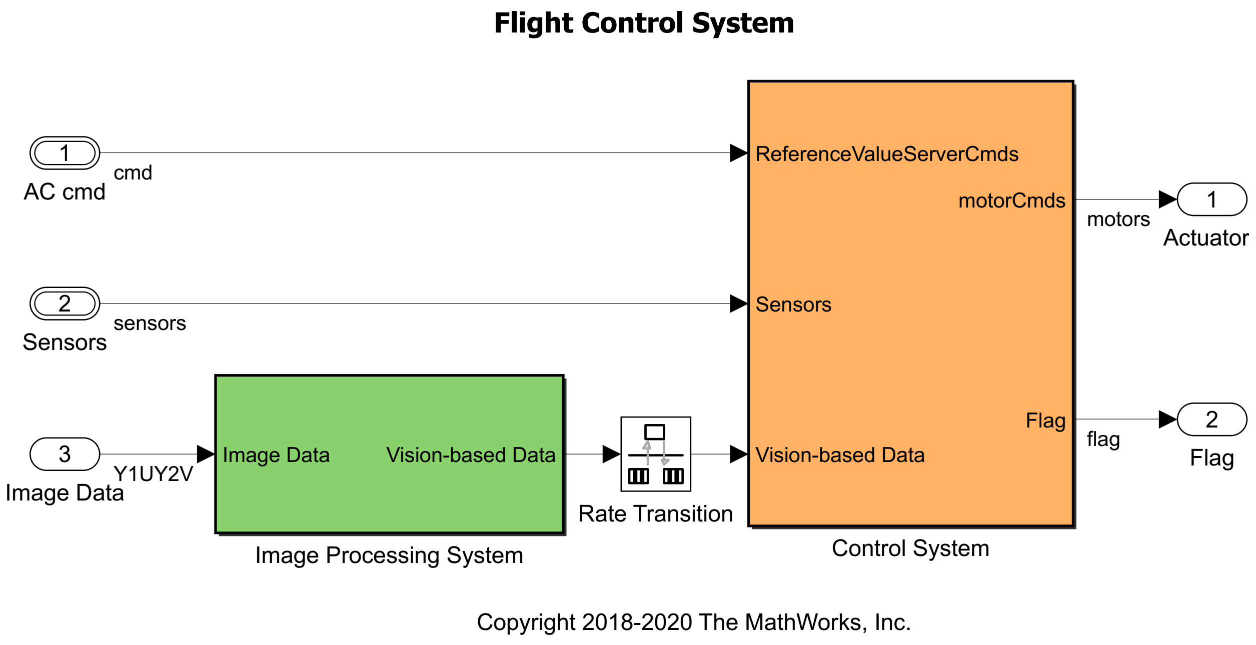


Figure 7.1: Flight Control System.

Figure 7.2 shows the inside of the Control System block. The sensor data to be used in the Controller block is sent through a block named State Estimator. Here all the necessary state variables are calculated (linear positions x, y and z, angular positions ϕ, θ and ψ, linear velocities u, v and w and angular velocities p, q and r.) This block is fully implemented by MathWorks beforehand so that all effort in the competition can be put into the Image Processing System and Path Planning blocks. In addition to changing these, the PID-controller implemented by MathWorks in the Controller block will be replaced by the Integral Sliding Mode controller.

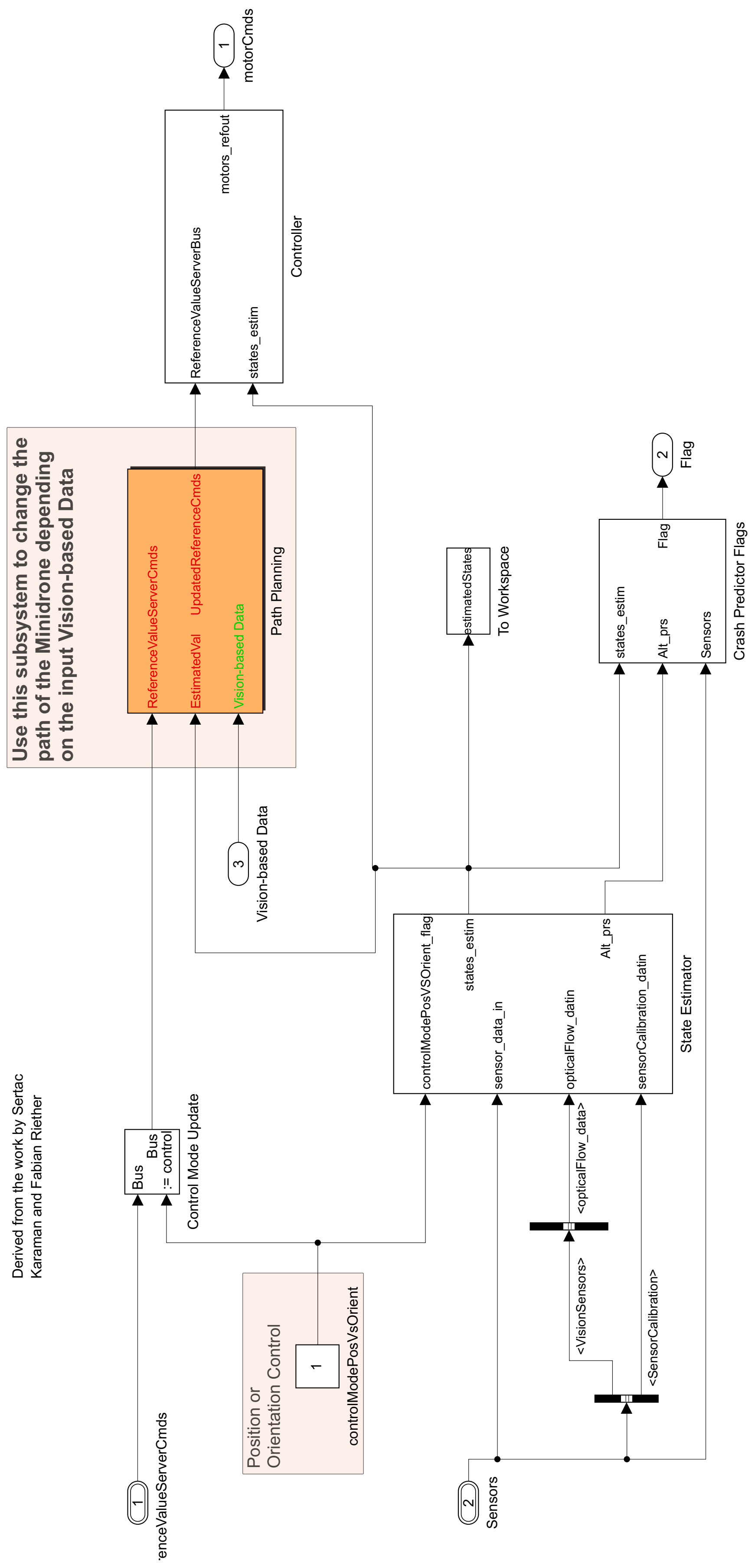


Figure 7.2: Control System.

7.2 Takeoff

During takeoff several variables will be initialized. Both the x- and y-reference will be set to 0 and the z-reference will be initialized to -1.1 so that the minidrone hovers 1.1 m above ground. The minidrone spends 3 seconds reaching this height and becoming stable before moving on to the next phases.

7.3 Pre-Processing

The image data from the camera is given in a 120x160 RGB format (after being transformed from a Y1UY2V format). The track the drone has to follow is red and therefore only the red pixels are of interest to the line tracking algorithm. Black & white (BW) images are easier to create algorithms for, so a useful first step is to convert the RGB image into a BW image where '1's/white pixels represent red pixels and '0's/black pixels represent everything that is not red.

In the simulation part of the competition the color of the track is red with hex code 0xFF0000, i.e. full red with no contributions from green and blue. The RGB image can therefore easily be converted into BW by extracting only the red value of the image and checking whether it is equal to 255 (0xFF). Figure 7.3 shows how this can be done in Simulink.

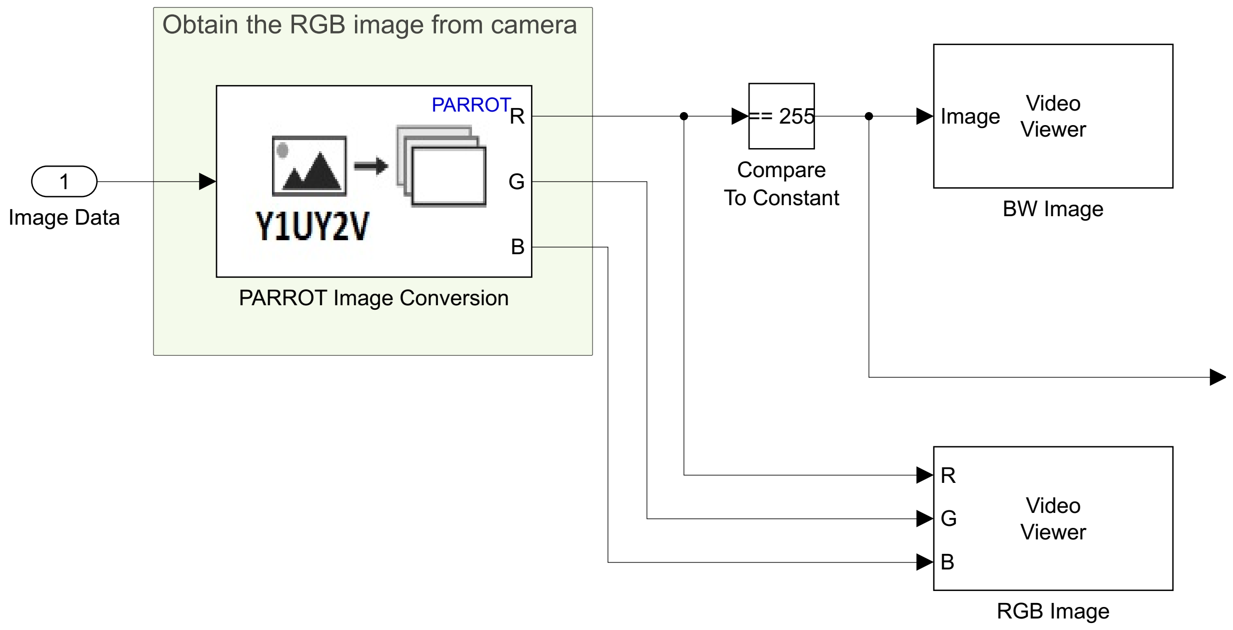


Figure 7.3: RGB-to-BW conversion in Simulink.

While this solution is good enough for the simulation round, it will not work in the in-person, deployment part of the competition. Here, shadows and variations in room lighting will produce different shades of red on the track, all of which need to be converted into white pixels in the BW image.

7.4 Calculating a Direction Vector

If a human was tasked with following the red line, they would probably align with the first line segment, walk forward until a corner is reached, turn around to align with the next line segment, walk forward, and so on until the red circle is reached. It certainly is possible to develop an algorithm where the minidrone follows the line in a similar manner by adjusting yaw and always moving forward. This, however, creates the need for a lot of stopping and rotating which unnecessarily slows down the drone. The algorithm developed here eliminates this need for adjusting yaw. This is done by using the white pixels in the BW image to calculate the direction the minidrone needs to travel towards.

The starting position of the red track is not always the same, and when placed at the start of the track, the mouth of the minidrone does not always face the same direction. Luckily, the minidrone is configured so that when the software starts running, the global reference frame coincides with the local frame, meaning that the drone considers its global starting position to be $\begin{bmatrix}0 & 0 & 0\end{bmatrix}$ and its global starting direction to be north. The coordinate system used is a right-hand coordinate system where the z-direction is facing downwards. The drone is configured so that its starting north coincides with the positive x-direction and east with the positive y-direction.

Calculating the direction is then done by "splitting" the image into two parts for both the x- and y-direction and computing sums. The number of white pixels in the north/upper 60 rows of the image gives a positive contribution to the x-direction while the opposite is true for the south/bottom 60 rows. The west/left 80 columns contribute negatively while the 80 east/right columns contribute positively to the y-direction.

This can also be done by applying a mask (in this case through elementwise multiplication) to the image and then summing all elements in the image. For the x-direction this would be a matrix with '1's in the upper half and '-1's in the lower half and for the y-direction a matrix with '1's in the right half and '-1's in the left half.

To illustrate with a simplified example, consider the following 6x10 BW image:

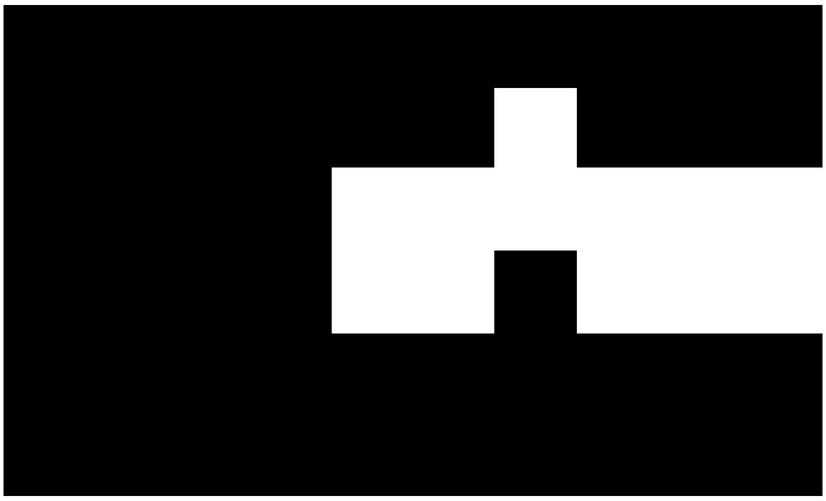


Figure 7.4: 6x10 BW image.

In matrix form:

$$\mathbf{BW} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{7.1}$$

This represents a mostly straight road that has a small “dent”.

To calculate the sum for the x-direction one would apply the following mask before calculating the sum:

$$\mathbf{Mx} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{pmatrix} \tag{7.2}$$

For the y-direction, this mask is used:

$$\mathbf{My} = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{7.3}$$

Applying those masks results in the following altered images:

$$\mathbf{BWx} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{7.4}$$

$$\mathbf{BWy} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{7.5}$$

Summing all elements of **BWx** for the x-direction and **BWy** for the y-direction gives the following direction vector:

$$\mathbf{v} = \begin{bmatrix} 2 & 8 \end{bmatrix}^{\mathrm{T}} \tag{7.6}$$

To make things simpler, this result will be normalized so that the direction vector always has a length of 1.

$$\mathbf{v}_n = \frac{\mathbf{v}}{|\mathbf{v}|} \approx \begin{bmatrix} 0.2425 & 0.9701 \end{bmatrix}^{\mathrm{T}} \tag{7.7}$$

7.4.1 A Weighted Approach

Having each pixel contribute equally to the sums can cause some instability. A more stable solution is to weight pixels far away from the center more heavily than those that are close. Using (7.1) as example, one would expect the “dent” in the road to not have such a large effect on the direction vector.

Instead of using using the masks in (7.2) and (7.3), the following masks can be used:

$$\mathbf{Mx} = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 \\ -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 \end{pmatrix} \tag{7.8}$$

$$\mathbf{My} = \begin{pmatrix} -5 & -4 & -3 & -2 & -1 & 1 & 2 & 3 & 4 & 5 \\ -5 & -4 & -3 & -2 & -1 & 1 & 2 & 3 & 4 & 5 \\ -5 & -4 & -3 & -2 & -1 & 1 & 2 & 3 & 4 & 5 \\ -5 & -4 & -3 & -2 & -1 & 1 & 2 & 3 & 4 & 5 \\ -5 & -4 & -3 & -2 & -1 & 1 & 2 & 3 & 4 & 5 \\ -5 & -4 & -3 & -2 & -1 & 1 & 2 & 3 & 4 & 5 \end{pmatrix} \tag{7.9}$$

Applying those masks results in the following altered images:

$$\mathbf{BWx} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{7.10}$$

$$\mathbf{BWy} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{7.11}$$

Summing now gives the following direction vector:

$$\mathbf{v} = \begin{bmatrix} 3 & 28 \end{bmatrix}^{\mathrm{T}} \tag{7.12}$$

Normalized:

$$\mathbf{v}_n = \frac{\mathbf{v}}{|\mathbf{v}|} \approx \begin{bmatrix} 0.1065 & 0.9943 \end{bmatrix}^{\mathrm{T}} \tag{7.13}$$

Compared to (7.7), the x-direction has now been weakened and the y-direction has been strengthened.

As the drone moves forward, many such "dents" will be introduced into the images captured by the camera. This weighted approach will reduce the effect of these.

7.5 Tunnel Vision

Calculating a direction vector directly from the freshly captured image is not enough. The tracking algorithm has to forget the white pixels in the direction it came from, otherwise the drone would just hover in place. This problem is tackled by applying tunnel vision in the direction the drone is heading in.

During the first pass of the algorithm the whole image is used to calculate the first direction vector, but from there on out tunnel vision must be applied to the image.

To create said tunnel vision some linear algebra will be applied.

Given a direction vector $\mathbf{v} = \begin{pmatrix} x_d & y_d \end{pmatrix}^{\mathrm{T}}$, a normal vector to $\mathbf{v}$ on its left side is given by $\mathbf{n}_{\mathrm{L}} = \begin{pmatrix} -y_d & x_d \end{pmatrix}^{\mathrm{T}}$, and one on its right side is given by $\mathbf{n}_{\mathrm{R}} = \begin{pmatrix} y_d & -x_d \end{pmatrix}^{\mathrm{T}}$. $\mathbf{n}_{\mathrm{L}}$ will be used from here on out and simply be referred to as $\mathbf{n}$.

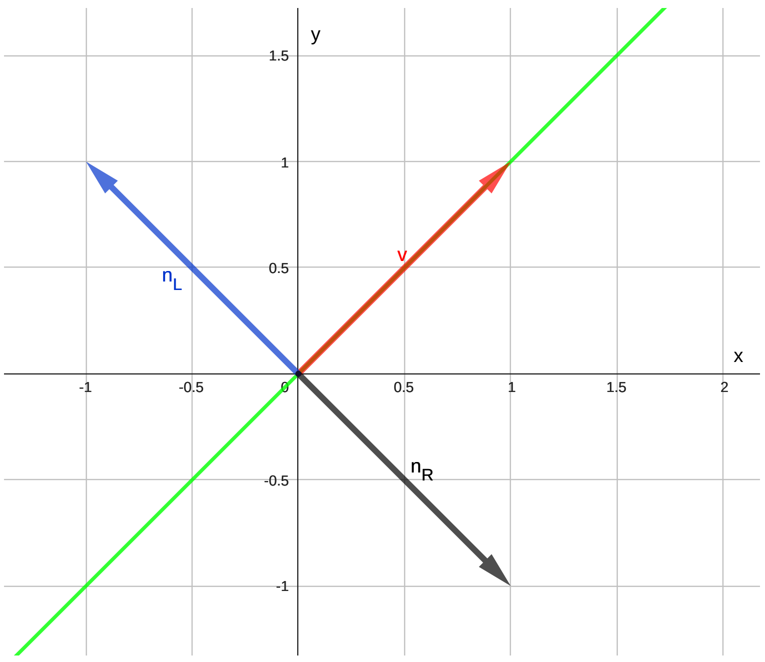


Figure 7.5: Figure illustrating what is meant by left and right side of $\mathbf{v}$. This figure was created with GeoGebra[6].

Let ℓ be a line that goes through the fixed point $\mathbf{p} = \begin{bmatrix} x_0 & y_0 \end{bmatrix}^{\mathrm{T}}$ ($\mathbf{p}$ can be selected as any point on the line) in the direction of $\mathbf{v}$. Let $\mathbf{x} = \begin{bmatrix} x & y \end{bmatrix}^{\mathrm{T}}$. For $\mathbf{x}$ to be on the line the following equation must be satisfied:

$$\mathbf{n} \cdot (\mathbf{x} - \mathbf{p}) = 0 \tag{7.14}$$

For lines that go through the origin (i.e. if $\mathbf{p} = \begin{pmatrix}0 & 0\end{pmatrix}^{\mathrm{T}}$) (7.14) reduces to:

$$\mathbf{n} \cdot \mathbf{x} = 0 \tag{7.15}$$

Similarly, the equation of a line ℓ_n that goes through $\mathbf{p}$ and is perpendicular to $\mathbf{v}$ is given by:

$$\mathbf{v} \cdot (\mathbf{x} - \mathbf{p}) = 0 \tag{7.16}$$

Now let $\mathbf{p}$ be given by:

$$\mathbf{p} = a \cdot \frac{\mathbf{n}}{||\mathbf{n}||} \tag{7.17}$$

Equation (7.17) creates a point that lies a units away from the origin in the direction of $\mathbf{n}$. This allows for parallel translation of the line ℓ by a units. $a > 0$ gives translation to the left of ℓ while $a < 0$ gives translation to the right of ℓ.

Inserting (7.17) into (7.14) yields:

$$\begin{aligned}
& \mathbf{n} \cdot \left(\mathbf{x} - a \cdot \frac{\mathbf{n}}{||\mathbf{n}||}\right) = 0 \\
\implies \quad & \left[-y_d \quad x_d)^{\mathrm{T}}\right] \cdot \left(\begin{bmatrix} x & y \end{bmatrix}^{\mathrm{T}} - a \cdot \frac{\begin{bmatrix} -y_d & x_d \end{bmatrix}^{\mathrm{T}}}{\sqrt{x_d^2 + y_d^2}} \right) = 0 \\
\implies \quad & x_d \cdot y = y_d \cdot x + a \cdot \frac{x_d^2 + y_d^2}{\sqrt{x_d^2 + y_d^2}} = y_d \cdot x + a \cdot \sqrt{x_d^2 + y_d^2}
\end{aligned} \tag{7.18}$$

If $\mathbf{v}$ is already of unit length, then $\mathbf{n}$ is too and (7.18) can be simplified into:

$$x_d \cdot y = y_d \cdot x + a \tag{7.19}$$

Replacing the equality in (7.18) or (7.19) with inequalities ($<$, $\leq$, $>$ or $\geq$) will give entire regions in the plane and intersecting such regions will create the required forward-facing tunnel vision, and other regions as needed.

Using the normal line ℓ_n and direction line ℓ shifted to the left and right by a units, tunnel vision can be created using the following intersection of regions:

$$y_d \cdot y \geq -x_d \cdot x \wedge x_d \cdot y \geq y_d \cdot x - a \cdot \sqrt{x_d^2 + y_d^2} \wedge x_d \cdot y \leq y_d \cdot x + a \cdot \sqrt{x_d^2 + y_d^2} \tag{7.20}$$

Simplified version (when $\mathbf{v}$ is of unit length):

$$y_d \cdot y \geq -x_d \cdot x \wedge x_d \cdot y \geq y_d \cdot x - a \wedge x_d \cdot y \leq y_d \cdot x + a \tag{7.21}$$

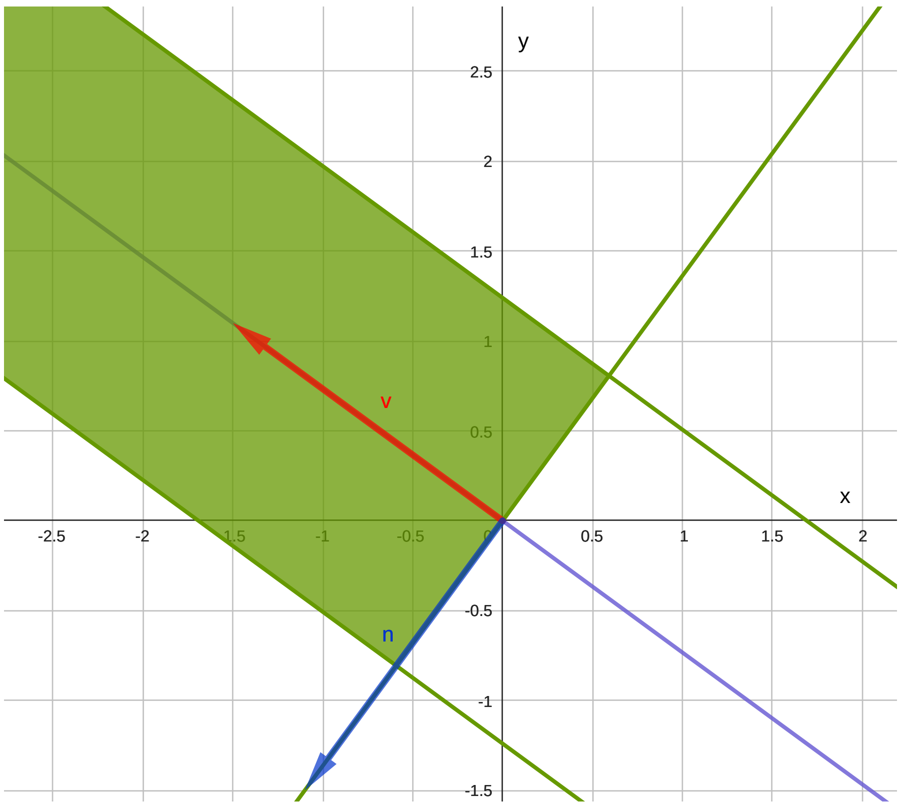


Figure 7.6: Example where $\mathbf{v} = \begin{bmatrix}-1.5 & 1.1\end{bmatrix}^{\mathrm{T}}$ and $a = 1$. This figure was created with GeoGebra[6].

In the examples shown in Figure 7.5 and 7.6 north has been the y-direction, east the x-direction and the not shown z-direction is pointing out of the image. For the drone camera the coordinate system is flipped in such a way that z is pointing into the image, x is north and y is east. Both are right-hand coordinate systems so the mathematics behind it all stays the same. It does, however, change how one interprets left and right.

The images captured by the camera are of size 120x160 and hence of discrete nature. The coordinates used will therefore be discrete with x ranging from -60 to $+60$ and y ranging from -80 to $+80$ with a step of 1 between each value. Because of an even number of pixels along each dimension, the origin $\begin{bmatrix}0 & 0\end{bmatrix}^{\mathrm{T}}$ and all other coordinates where either x and y are equal to zero are excluded from the image's coordinate system. Listing 7.1 shows an example of what using Equation (7.21) looks like in Matlab-code. The result is displayed in Figure 7.7.

```
x = nonzeros(60:-1:-60);
y = nonzeros(-80:80)';
```

```
[X,Y] = meshgrid(x,y);
X = X';
Y = Y';

a = 10;
v = [5,-7];
v = v/norm(v);
n = [v(2),-v(1)];

RegionLeft = v(1)*Y <= v(2)*X + a*(v(1)^2+v(2)^2);
RegionRight = v(1)*Y >= v(2)*X - a*(v(1)^2+v(2)^2);
RegionUp = v(2)*Y >= -v(1)*X;
RegionCombined = RegionLeft & RegionRight & RegionUp;
```

Listing 7.1: Matlab-code for creating tunnel vision.

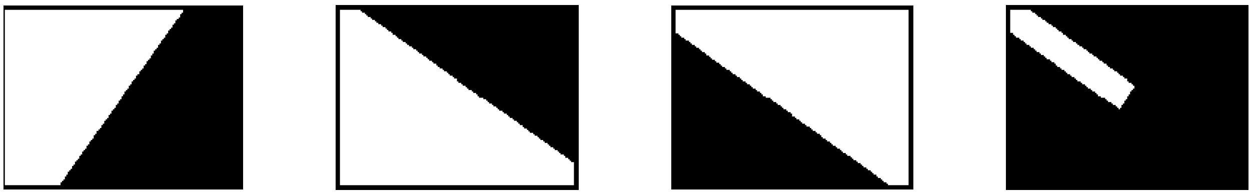


Figure 7.7: Regions created and intersected to create tunnel vision to be applied to images.

How the tunnel vision works will now be illustrated via an example. The minidrone will be flying at a height of 1.1 meters at which the track segments will be approximately 20 pixels wide when captured by the camera. In this example the tunnel vision width will be set to 40 pixels. Figure 7.8, 7.9 and 7.10 show the process.

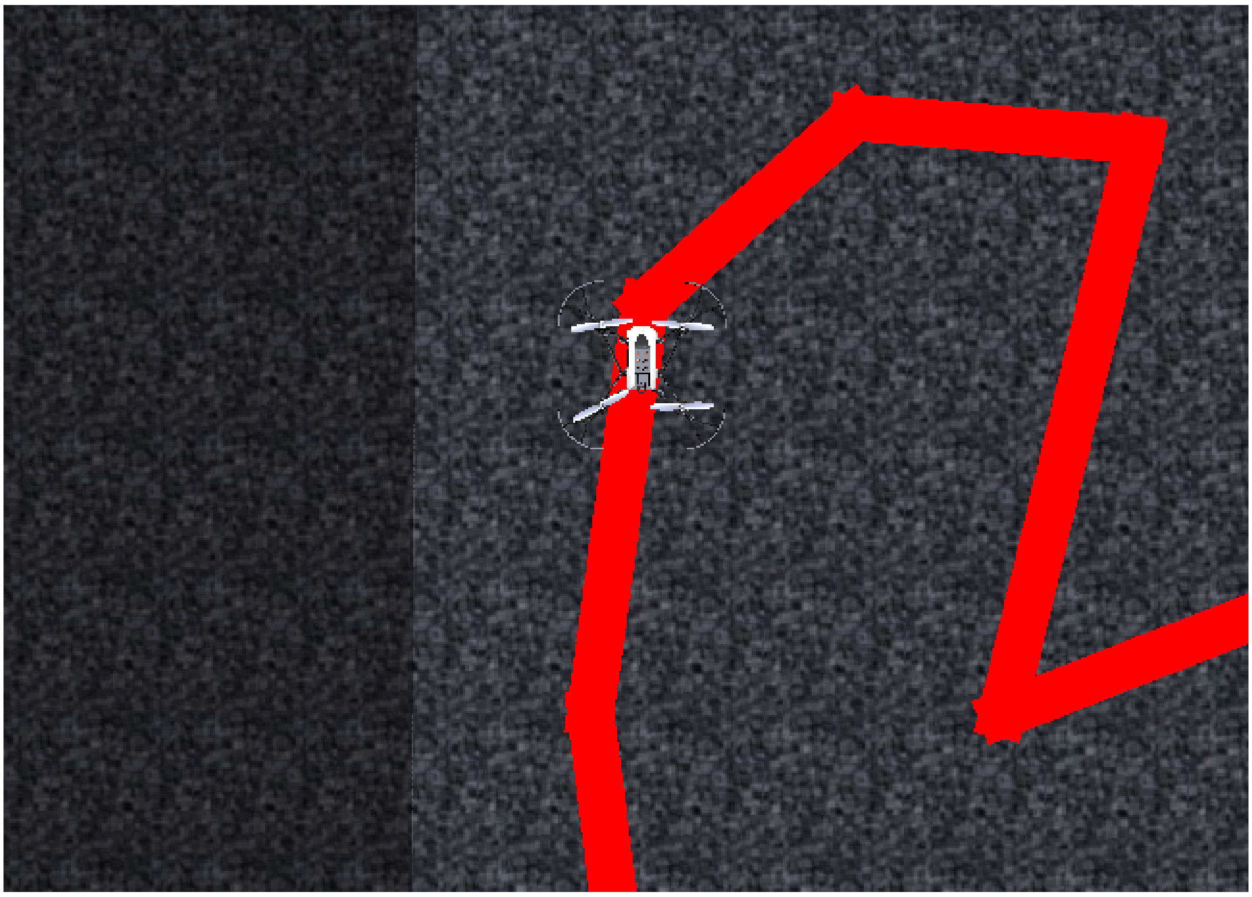
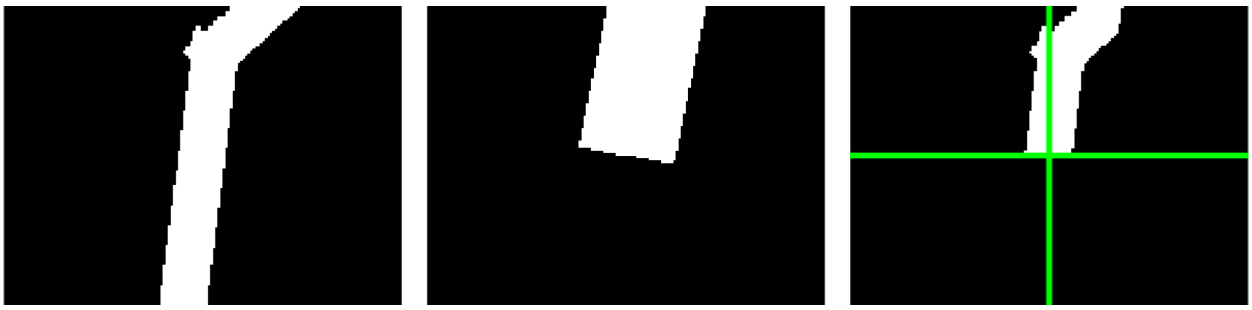


Figure 7.8: The minidrone is heading towards a turn.



(a) BW image captured by the camera.

(b) Tunnel vision to be applied to the BW image.

(c) BW image after tunnel vision has been applied. BW image after tunnel vision has been applied. Image axes have been added in.

Figure 7.9: Situation as the drone is heading towards the turn. Pixels in the opposite direction are neglected. With more pixels in the right region the drone's direction vector will start leaning even more to the right.

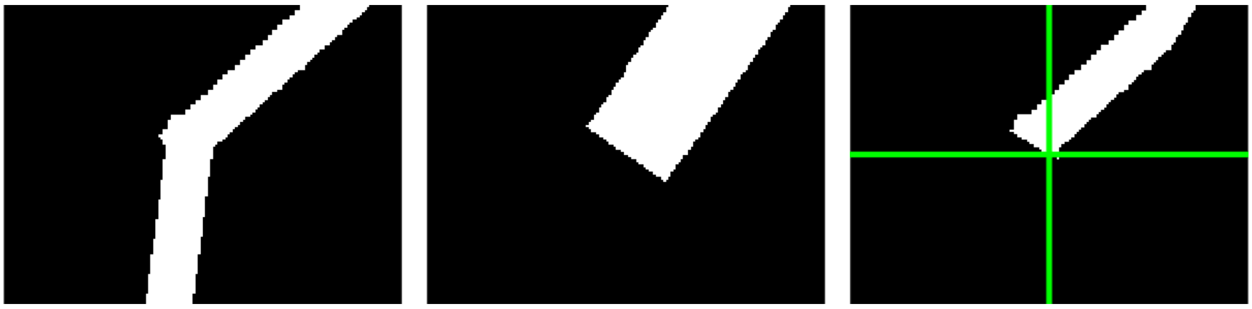

(a) BW image captured by the camera. **(b)** Tunnel vision to be applied to the BW image. **(c)** BW image after tunnel vision has been applied.

Figure 7.10: Situation shortly after heading in the new direction. The minidrone is now heading northeast.

7.6 Handling Sharp Corners

The tunnel vision approach described so far works well for many turns, especially those between 90° and 270° (where angles are measured from one track segment to the next in an anti-clockwise manner). It also works for sharper turns, but only up to a certain point, after which it starts failing. The problem that occurs is illustrated in Figure 7.11, 7.12, 7.13 and 7.14.

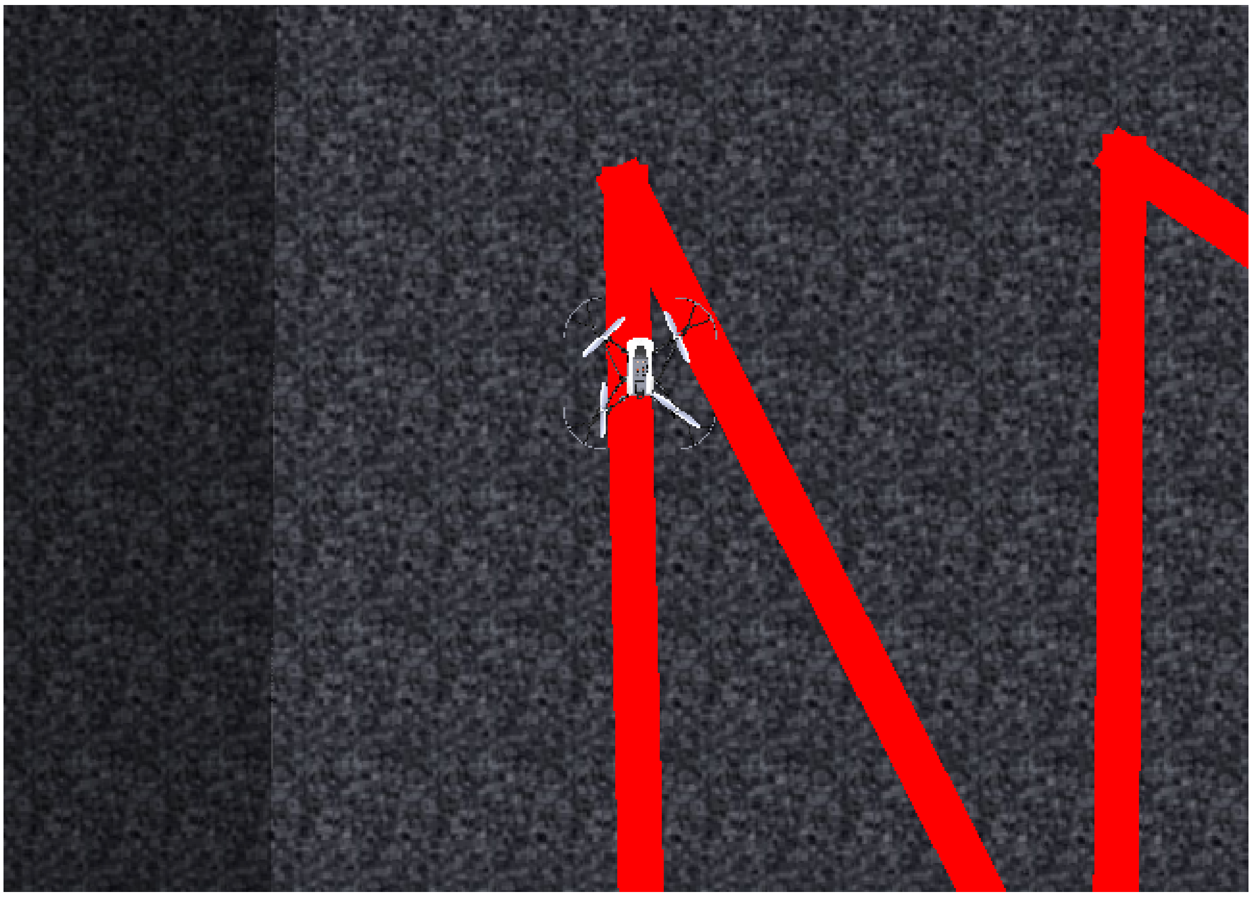
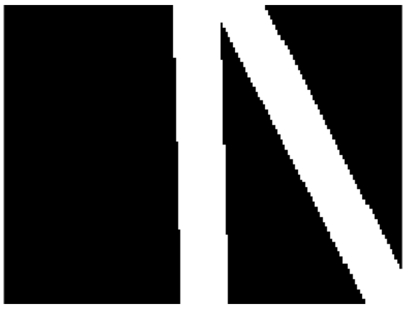
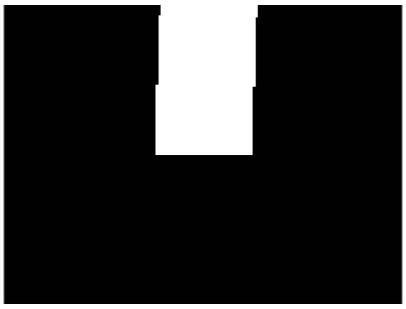


Figure 7.11: The minidrone is heading towards a relatively sharp turn.


(a) BW image captured by the camera.


(b) Tunnel vision to be applied to the BW image.

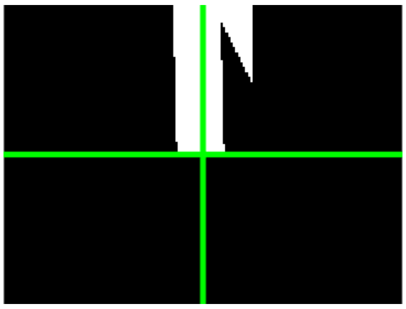
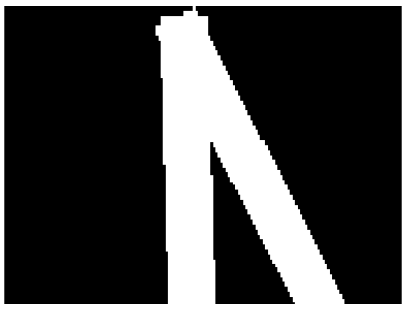
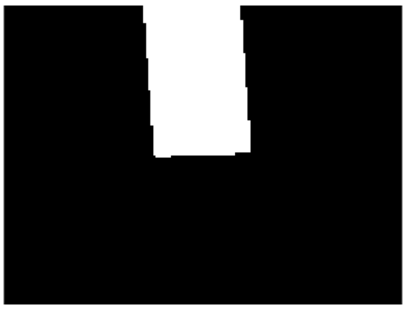

(c) BW image after tunnel vision has been applied. Image axes have been added in.

Figure 7.12: Situation as the drone is heading towards the turn. There are not enough white pixels in the right region to give a strong pull towards the right.


(a) BW image captured by the camera.


(b) Tunnel vision to be applied to the BW image.

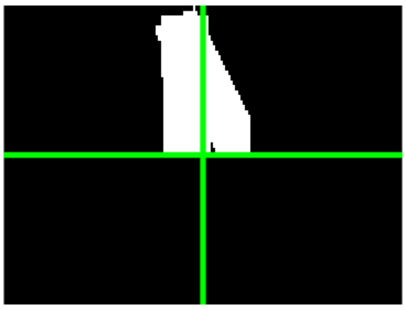
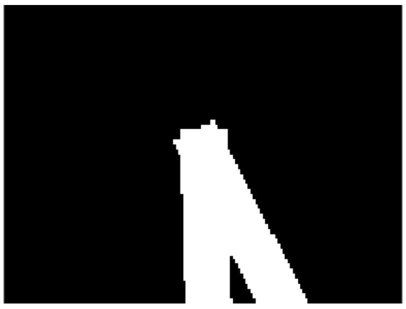
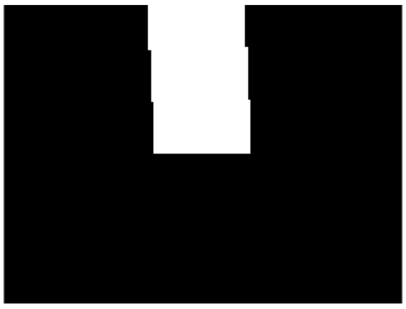

(c) BW image after tunnel vision has been applied. Image axes have been added in.

Figure 7.13: Situation as the minidrone is drawn closer to the edge of the corner. There are more white pixels in the left than in the right region.


(a) BW image captured by the camera.


(b) Tunnel vision to be applied to the BW image.

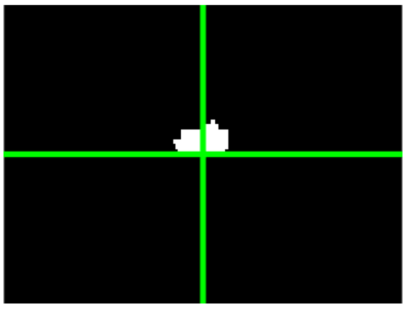
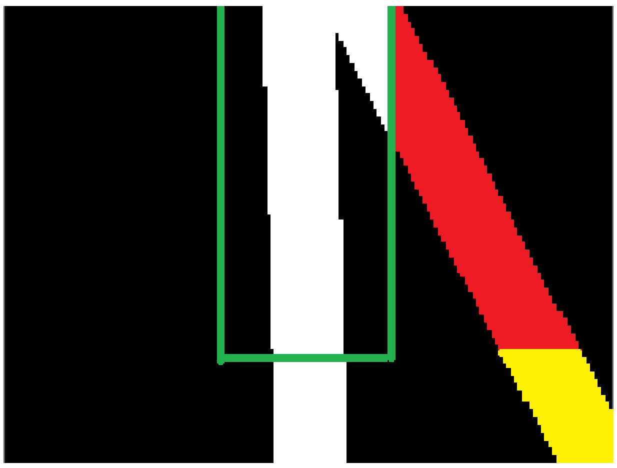

(c) BW image after tunnel vision has been applied. Image axes have been added in.

Figure 7.14: Situation as the minidrone is at the edge of the corner. Here the minidrone will either move further into the corner until there are no more white pixels left, or there will be just enough pixels to rotate the direction back onto the track. In the first case the minidrone will hover in place, in the second it sometimes locks onto the correct new path, sometimes it locks onto the path it came from.

Some of the issues of sharp turns are illustrated in Figure 7.15.


(a) A large amount of pixels (yellow) do not come into play because they lie behind the tunnel vision (green rectangle).

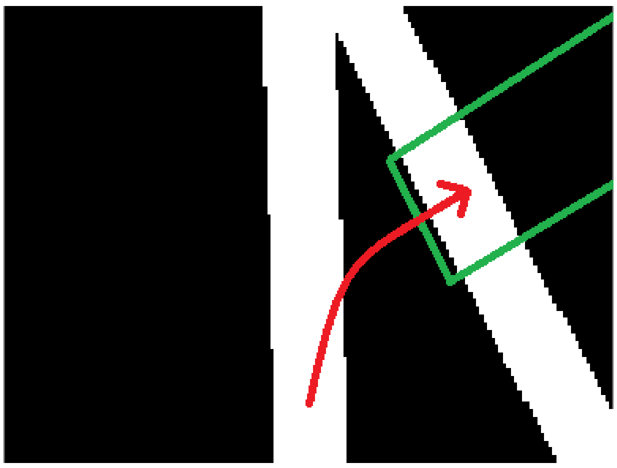

(b) Even if the minidrone would turn somewhat correctly it then runs into the next track segment in such a way that the pixels inside the tunnel vision cannot be used to determine if the drone should go left or right.

Figure 7.15: Problems with sharp turns.

This problem is tackled by paying special attention to certain regions of the image. The solution designed here draws inspiration from the first two line tracking algorithms in [23]. In those algorithms the images are divided into regions and decisions are made based on the number of white pixels in said regions. [23] uses a yaw-adjusting approach and the regions' placements in the image are therefore static. Here the placement of these regions will depend on the direction vector, just like how the tunnel vision moves around. The regions will be created using the same principles as discussed in Section 7.5, i.e. by intersecting regions above and below lines.

The following regions will be used:

- Far corner sensor: a thin region placed in front of the center in the direction the minidrone is heading. Placed further out than the close corner sensor. Used to detect if the end of a track segment has entered the image.
- Close corner sensor: a thin region placed in front of the center in the direction the minidrone is heading. Used to detect if the minidrone now is at the end of a track segment.
- Left sensor: A thin, but long region placed to the left of the direction the minidrone is heading. Used to decide if the minidrone should take a left turn when at a sharp corner.
- Right sensor: A thin, but long region placed to the right of the direction the minidrone is heading. Used to decide if the minidrone should take a right turn when at a sharp corner.

The so-called "sensors" will have a width of approximately 2 pixels.

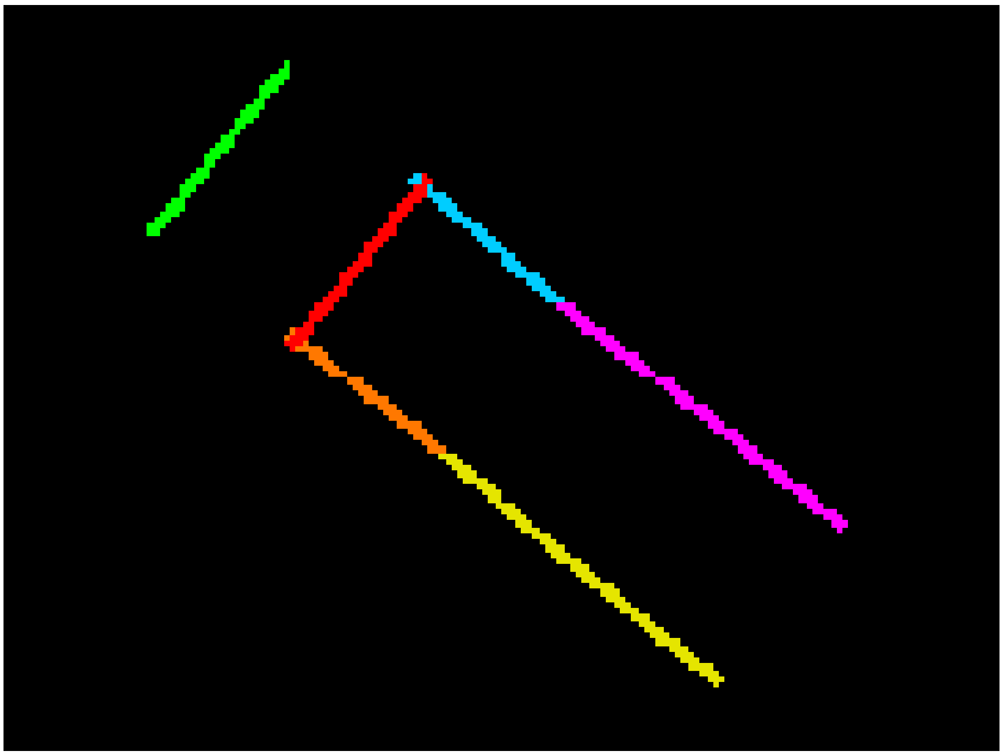


Figure 7.16: Example of where the regions will be placed when the drone is heading northwest. Colored as follows: far corner sensor in green, close corner sensor in red, front left sensor in orange, back left sensor in yellow, front right sensor in blue and back right sensor in purple.

The left and right sensors need to reach far back in order to detect left/right turns also for very sharp turns (down to 10°/up to 350° [10]). But with such long regions chances are they will overlap with the path underneath and behind the drone if the current direction has not yet been fully stabilized, potentially causing problems for detection of left/right turns. The left and right sensors have therefore been further divided into front and back sensors. The front regions will be weighted more heavily when determining whether to go in a left or right direction.

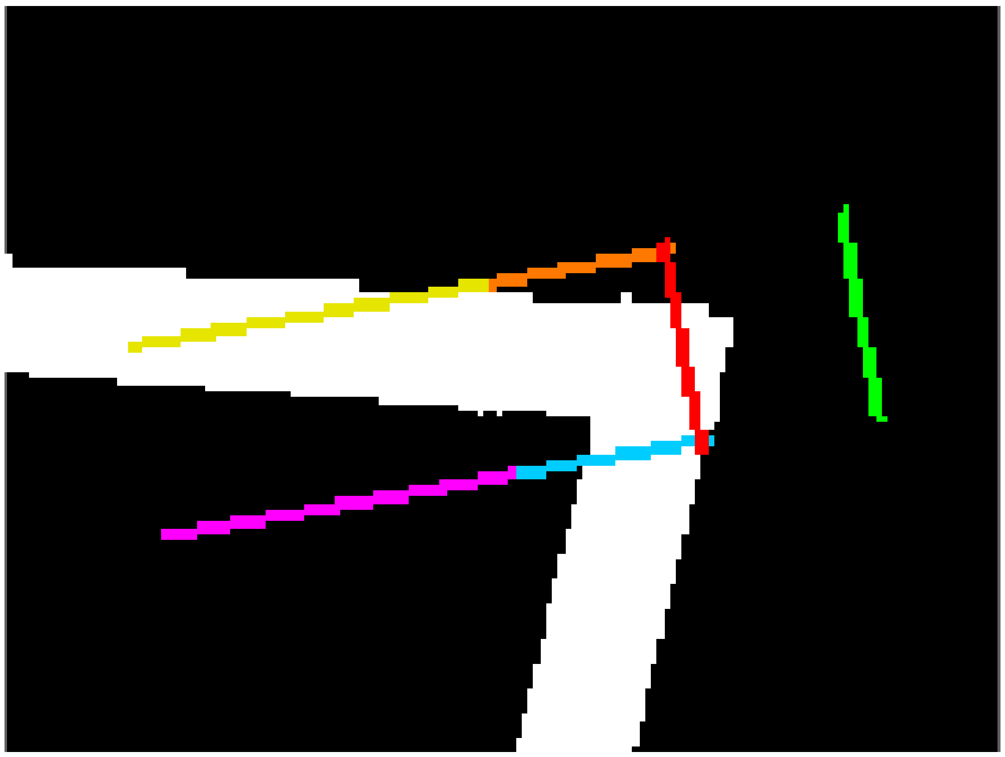
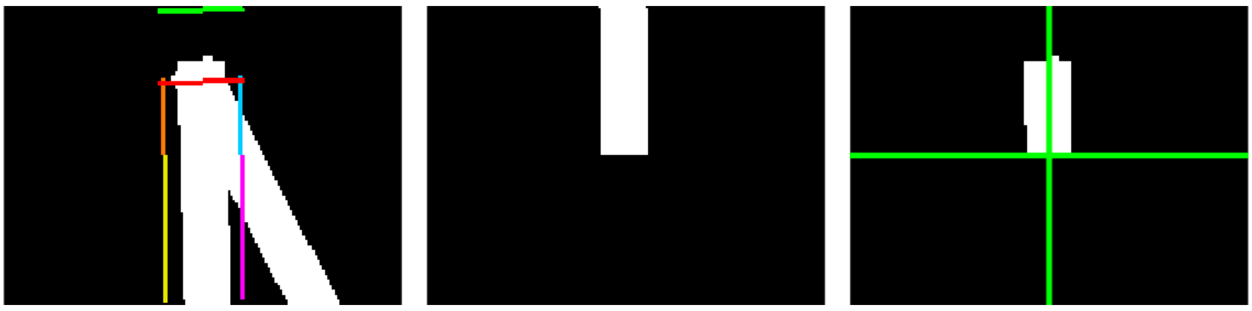


Figure 7.17: Case where the path is not yet fully stable. The back left sensor reaches deep into the track segment, but because the front right sensor is weighted more heavily in the calculations, the drone will head right.

The solution works as follows: at every step the number of white pixels in the original BW image that fall within the sensors are counted. When the number of pixels in the two corner sensors fall below a certain threshold (10 is used in the implementation), the minidrone knows that it is at the end of a sharp turn (for slack turns the amount of pixels will be high enough to not warrant leaving the tunnel vision approach). It then uses the amount of pixels in the left and right sensors to determine where to head next. For a brief moment (0.8 seconds is used in the implementation) the tunnel vision will be swapped out with the region to the left or right of the current direction. The region is pushed a few pixels (at least half the width of a track segment, 10 pixels when flying at 1.1 m) to the left/right of the direction so that the track segment behind the drone is not included in the altered BW image. A new direction will be calculated and after the short period has passed the tunnel vision is applied as before. This will be illustrated via the same example as in Figure 7.11.

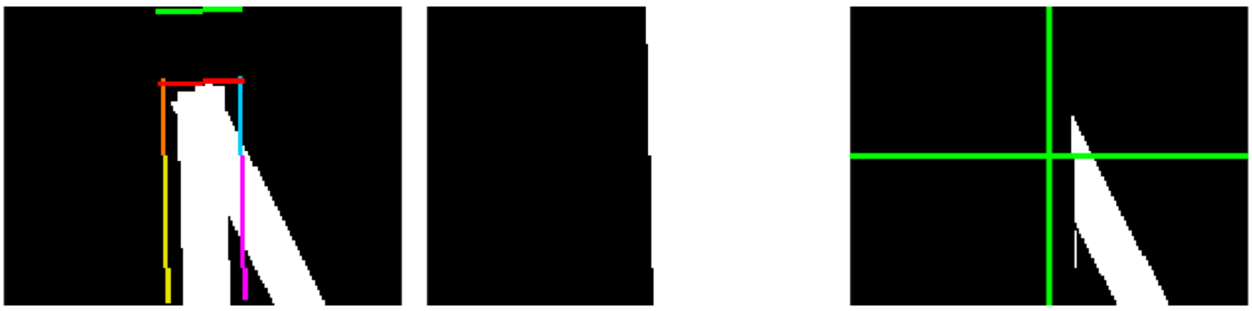


(a) BW image captured by the camera. Sensor regions have been added in.

(b) Tunnel vision to be applied to the BW image.

(c) BW image after tunnel vision has been applied. Image axes have been added in.

Figure 7.18: Situation as the drone is heading towards the turn. The BW image with tunnel vision applied is by its own not enough to get the minidrone to head in the new direction.

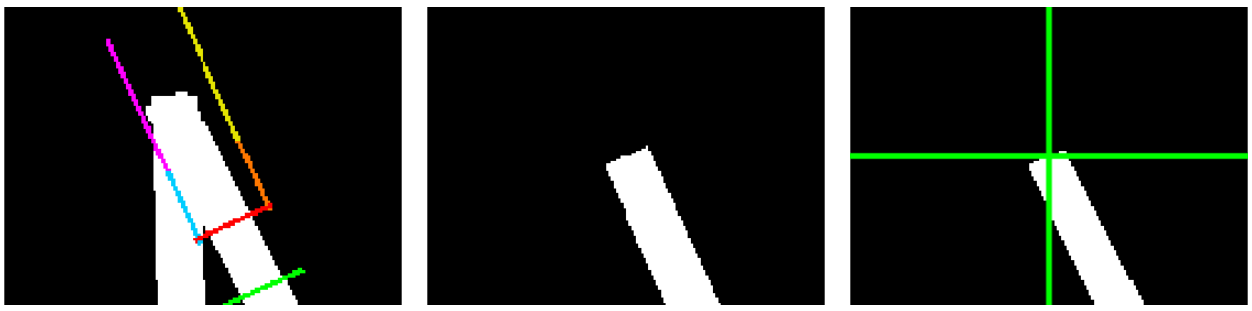


(a) BW image captured by the camera. Sensor regions have been added in.

(b) Right-side vision to be applied to the BW image.

(c) BW image after right-side vision has been applied. Image axes have been added in.

Figure 7.19: The corner sensors detect that the minidrone has reached the edge of the turn. There are now more white pixels under the right side sensors than under those on the left side, so the entire region to the right of the current direction will be used when calculating the next direction.



(a) BW image captured by the camera. Sensor regions have been added in.

(b) Tunnel vision to be applied to the BW image.

(c) BW image after tunnel vision has been applied. Image axes have been added in.

Figure 7.20: The minidrone successfully turns and continues using the narrow, forward-facing tunnel vision as before.

7.7 Landing Phase

What remains is for the minidrone to detect that its near the end of the last track segment and to detect and land on the circle. The circle always has a diameter of 20 cm and the distance from the end of the track to the center of the circle will always be 25 cm [10]. This means the distance from the end of the track to the edge of the circle will always be 10 cm. When flying 1.1 m above the ground this roughly equates to 28 pixels. With the correct amount of space between the corner sensors explained in Section 7.6, the minidrone will know that it is near the end of the track when there are white pixels inside the far corner sensor, but not in the close corner sensor, see Figure 7.21.

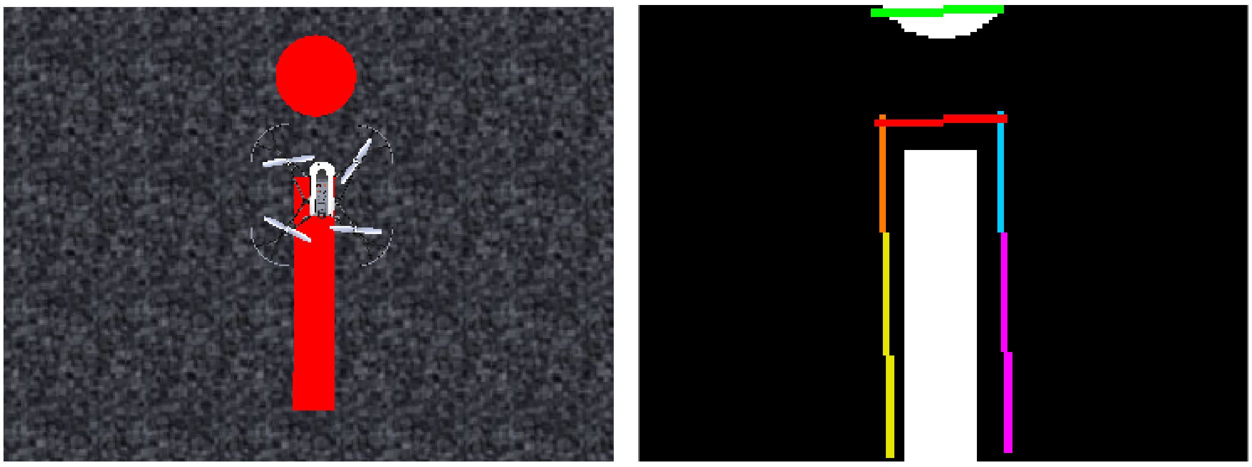

(a) The minidrone is close to the end of the track. **(b)** BW image captured by the camera. Sensor regions have been added in.

Figure 7.21: The minidrone is at the end of the track. It knows this because the far corner sensor has white pixels inside it while the close corner sensor has none.

The minidrone will then enter an "end of track" phase where it travels forward at a slower speed and where the amount of pixels directly underneath will be counted in order to detect the circle. It continues calculating the direction vector as before. As mentioned, the circle has a diameter of 20 cm, which at 1.1 m height roughly translates to 36 pixels in the BW image. The counting will therefore be done within a 40x40 square region in the middle of the BW image. The circle consists of the following approximate amount of white pixels:

$$A = \pi r^2 = \pi \cdot \left(\frac{36}{2}\right)^2 \approx 1017.36 \tag{7.22}$$

If the amount of white pixels inside the center exceed 1000 pixels the minidrone will enter the landing phase where it stops updating the x- and y-directions and gradually increases the z-direction for a smooth landing (recall that the positive z-axis is pointing downwards).

7.8 Minidrone Speed

To ensure a flight that is both quick and stable, the speed of the minidrone should depend on the amount of white pixels it uses to calculate the direction. Since there are less pixels in the x-direction than in the y-direction (120 vs. 160), the speed will be limited by the maximal amount of white pixels possible inside a tunnel vision region pointing strictly in the x-direction. The tunnel vision will be set to the same pixel width as the track segments. This gives a total of $60 \times 20 = 120$ pixels. The minimum amount of pixels inside the tunnel vision is 0. For improved stability the drone should move slower when there are pixels inside the left/right sensors (Section 7.6). The (weighted) sum of pixels within these will be subtracted from the amount of pixels inside the tunnel vision. Note that this results in a negative minimum amount of pixels.

During simulation, the path planning block runs with a sampling period of 0.005 s. If one sets the drone speed to e.g. 0.0005 m per sample, this equates to 0.0005 m/0.005 s = 0.1 m/s. This will be the minimum speed the drone will be allowed to travel at (excluding during takeoff and landing). Similarly, the drone will have a maximum speed of 0.0011 m per sample, i.e. 0.22 m/s. These values were decided through experimentation. Allowing the minidrone to fly faster can result in the camera and image processing system not being able to keep up (the image processing system block is set to run with a sampling period 40 times larger than the rest of the system).

The amount of pixels inside the tunnel vision must then be mapped to the desired minidrone speed. Consider a standard linear equation where the input is shifted and amplified:

$$y = a(x - b) \tag{7.23}$$

Let y_{min} be minimum speed, y_{max} be maximum speed, x_{min} minimum pixel amount and x_{max} maximum pixel amount.

The following two equations should hold:

$$y_{\text{min}} = a(x_{\text{min}} - b) \tag{7.24}$$

$$y_{\text{max}} = a(x_{\text{max}} - b) \tag{7.25}$$

Subtracting (7.24) from (7.25) and solving for a yields:

$$a = (y_{\text{max}} - y_{\text{min}})/(x_{\text{max}} - x_{\text{min}}); \tag{7.26}$$

Solving (7.24) for b yields:

$$b = x_{\text{min}} - \frac{y_{\text{min}}}{a} \tag{7.27}$$

Using Equation (7.23) with these values for a and b will then give the desired mapping.

Chapter 8

Results

To evaluate the performance of the developed algorithms several experiments of various types were conducted. To test how well the control algorithms track its given references, a series of numerical simulations were run, as well as physical experiments on the Quanser 3 DOF Hover system. To test how the line tracking algorithm performs with different control systems, visualized simulations were run using the Parrot Minidrone Competition Project.

Figure 8.1: QR code for videos showcasing the 3 DOF Hover and Line Tracking experiments.

8.1 Simulation

To validate the performances of the proposed controllers, numerical simulations will be presented in this section. The quadrotor mathematical model from Equation (2.19) was simulated using MATLAB/Simulink environment. The parameters of the quadrotor were set according to the Parrot Mambo minidrone values from Table 2.1. The initial attitude and position of the quadrotor was chosen as $\begin{bmatrix}0 & 0 & 0\end{bmatrix}$ rad and $\begin{bmatrix}0 & 0 & 0\end{bmatrix}$ m for all the simulations. IAE, ISE, ITAE and ITSE has been selected as the performance indicators to evaluate and compare the different controllers, where minimizing the different indices indicates a better performance. Simulations considering both the absence and presence of model uncertainties and external disturbances were conducted.

8.1.1 Chattering Attenuation

In this subsection, the objective was to compare the performance of the ISMC with the sign function, approximated continuous sign function (quasi) and with adaptive fuzzy gain scheduling (AFGS). The small scalar constant in the approximated continuous sign function was chosen as $\epsilon = 0.2$. The performance with the use of the sign function is denoted as ISMC in this subsection. The performance with the use of the approximated sign function is denoted Quasi, and the performance with the use of adaptive fuzzy gain scheduling is denoted AFGS. The comparison will be made in terms of chattering attenuation and tracking performance of the desired attitude. The desired attitude was chosen as $\phi_d = \theta_d = \sin(t)$, $\psi_d = 0.5\sin(t)$. The control parameters used in the simulations are shown in Table 8.1.

i	λ_i	ρ_i	ζ_i	k_i
ϕ	10	0.5	2	0.001
θ	10	0.5	2	0.001
ψ	5	0.5	1	0.001

Table 8.1: ISMC parameters.

The simulation results without any uncertainties or disturbances are shown in Figure 8.2-8.7. Figure 8.2 shows the desired attitude and the actual attitudes. The attitude errors are shown in Figure 8.3, and the different control inputs are shown in Figure 8.4-8.6. The Adaptive gains for the AFGS are shown in Figure 8.7, where they change their amplitude according to the deviation of the trajectories from the sliding surface.

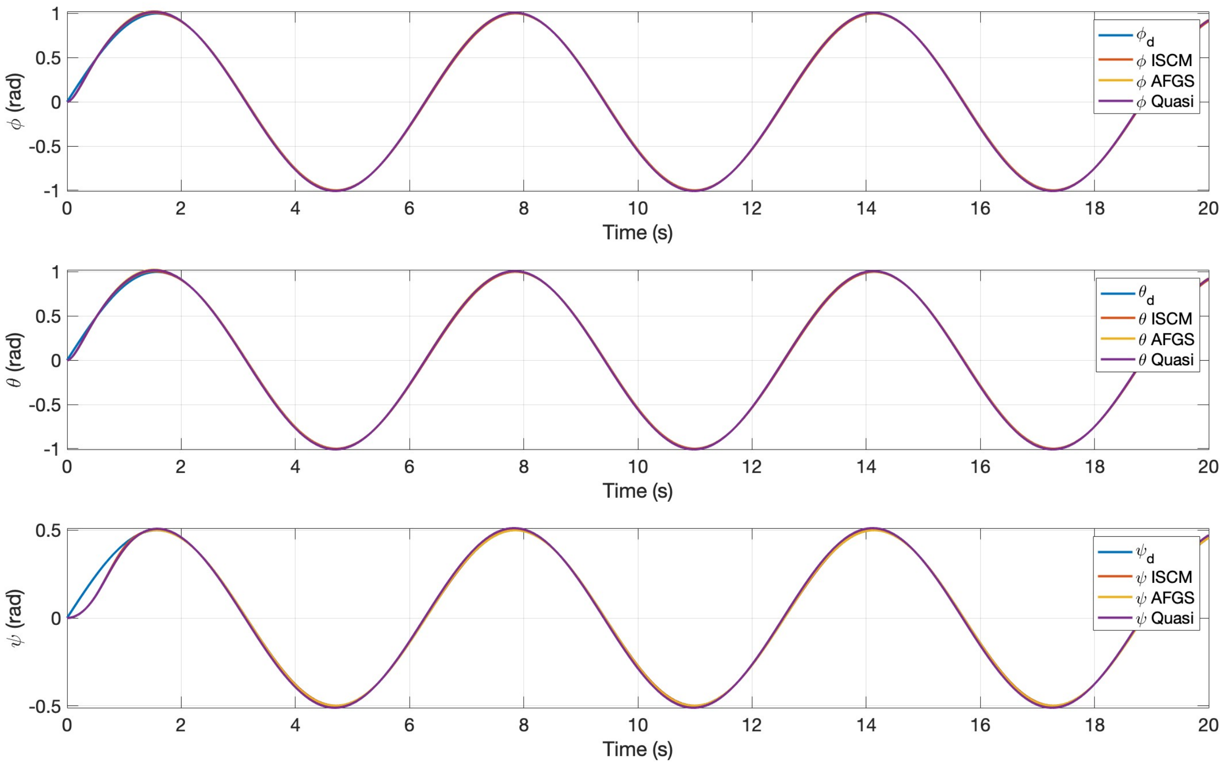


Figure 8.2: Attitude angles.

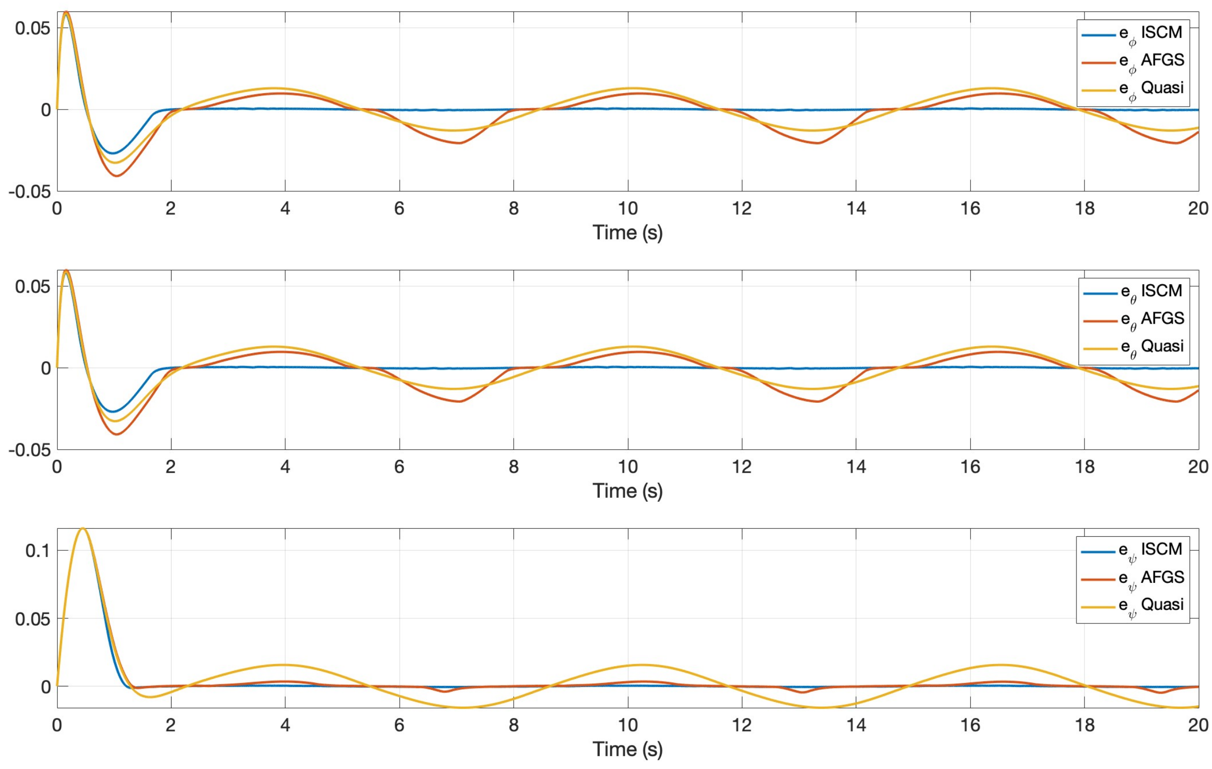


Figure 8.3: Attitude errors.

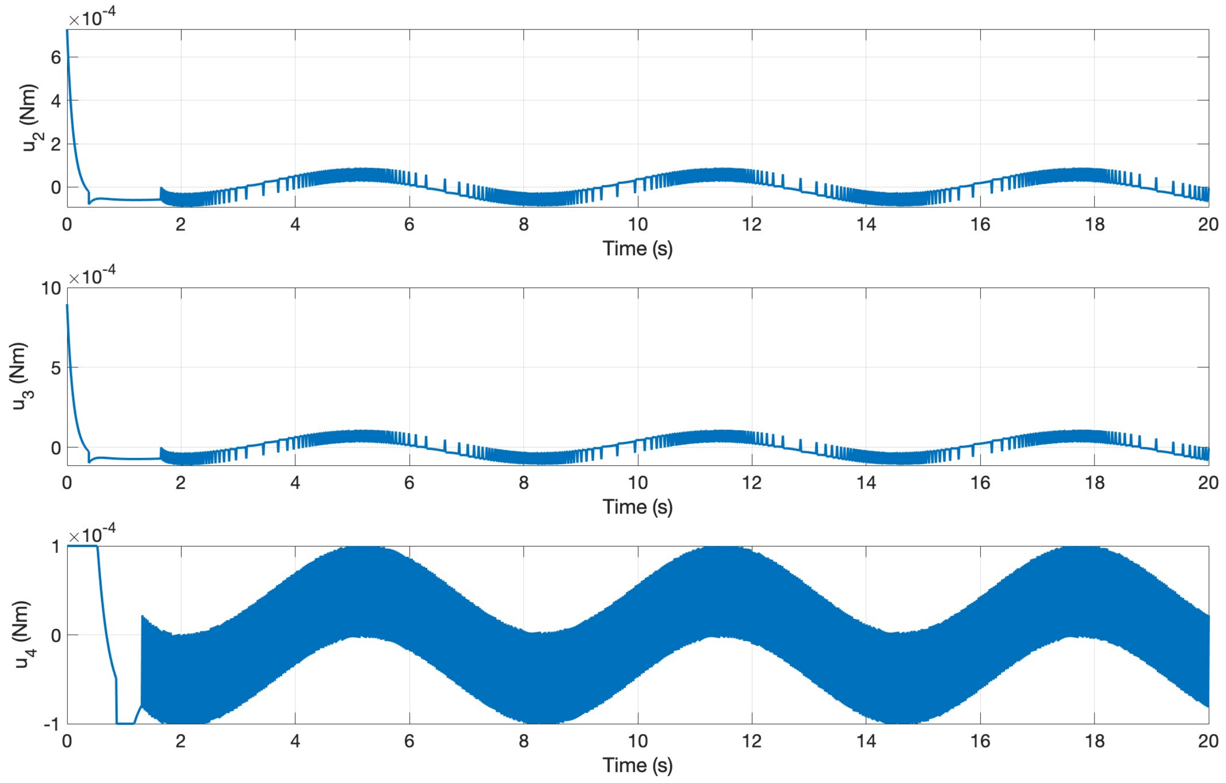


Figure 8.4: ISMC control inputs.

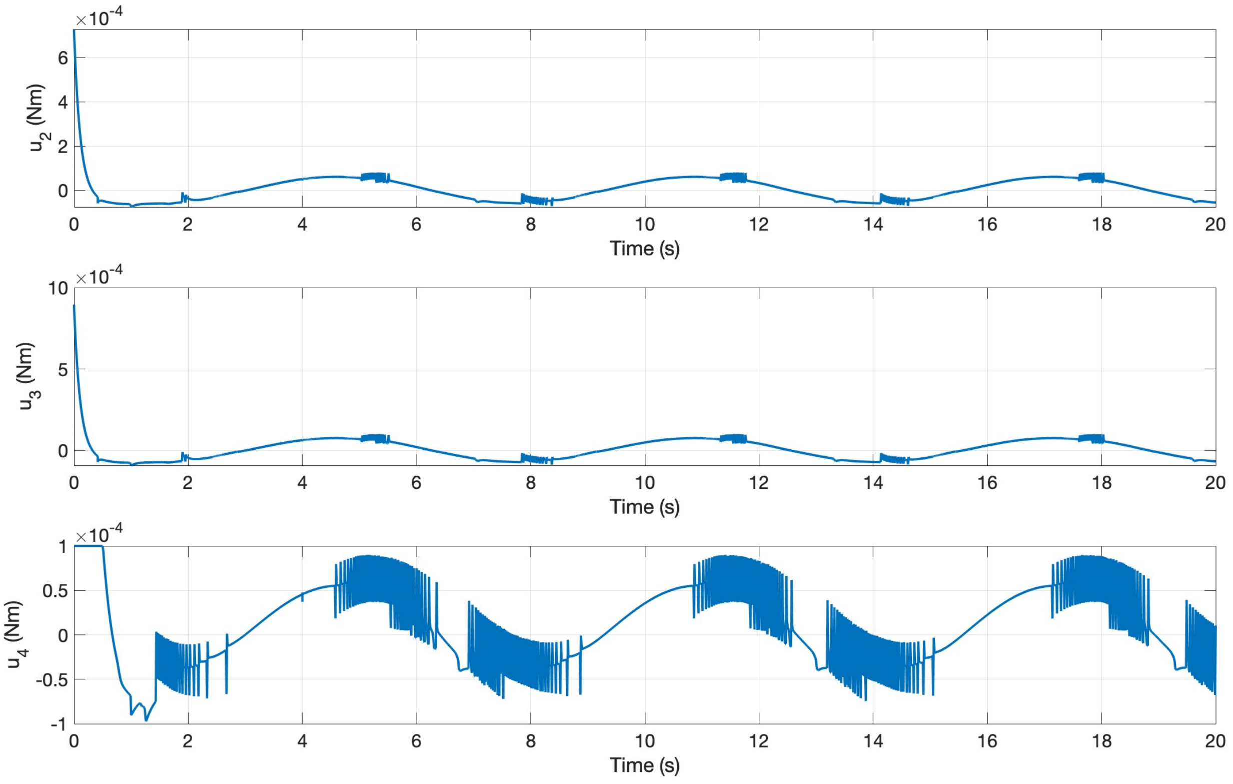


Figure 8.5: AFGS control inputs.

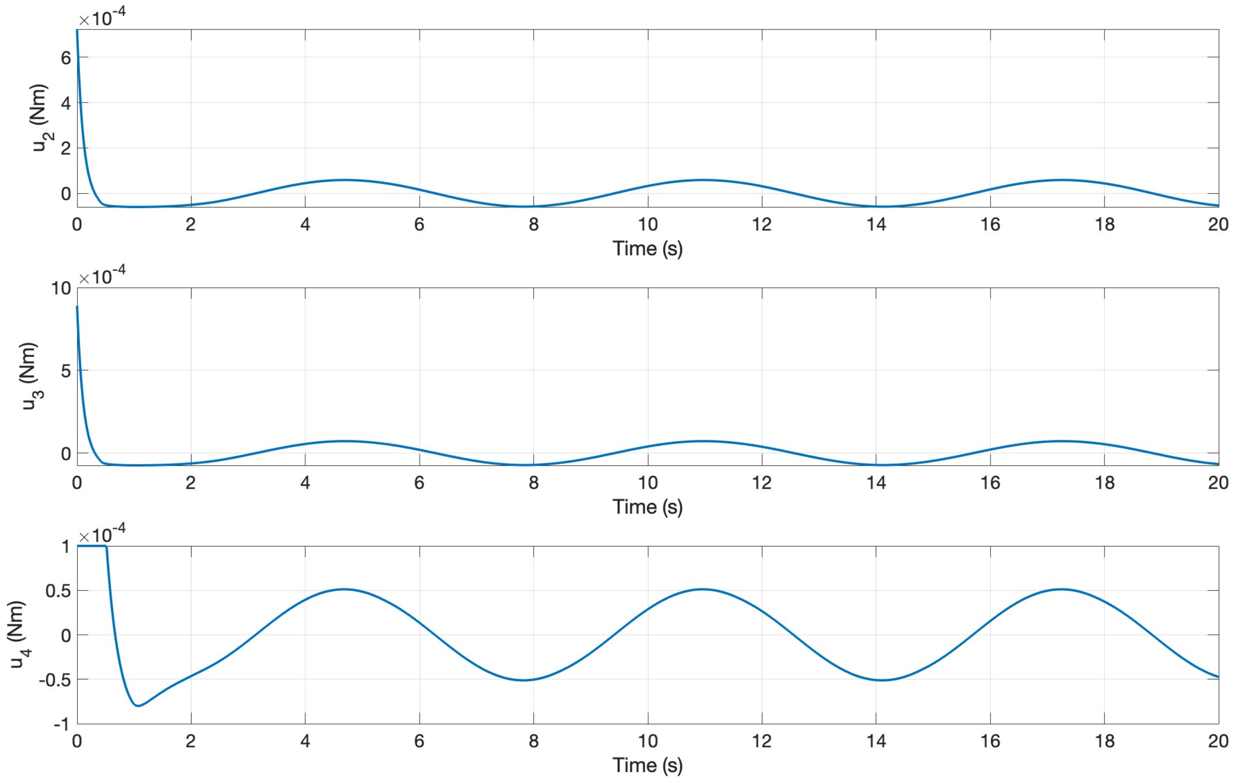


Figure 8.6: Quasi control inputs.

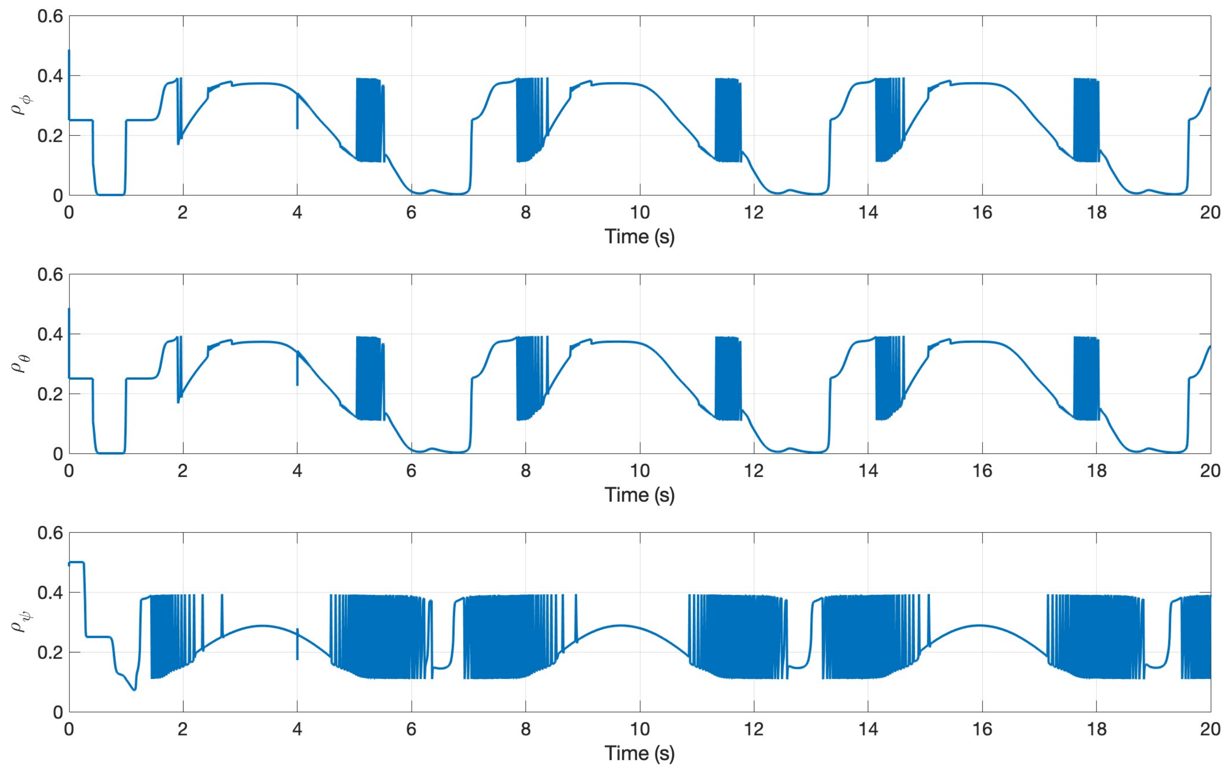


Figure 8.7: AFGS control gains.

Table 8.2 shows the performance indices comparison between the three control methods.

		ISMC	AFGS	Quasi
ϕ	IAE	0.0442	0.1876	0.1950
	ISE	0.0012	0.0037	0.0031
	ITAE	0.0848	1.6030	1.6769
	ITSE	0.00062832	0.0221	0.0178
θ	IAE	0.0442	0.1876	0.1950
	ISE	0.0012	0.0037	0.0031
	ITAE	0.0848	1.6030	1.6769
	ITSE	0.00062832	0.0221	0.0178
ψ	IAE	0.0829	0.1042	0.2616
	ISE	0.0068	0.0072	0.0092
	ITAE	0.0999	0.2903	2.0301
	ITSE	0.0032	0.0041	0.0281

Table 8.2: Performance indices without any uncertainties and disturbances.

Uncertainty (20% added) in rotary inertia and external disturbances were then implemented. In this case, we considered model uncertainty 20% added in all three moments of inertia. The disturbances were implemented as $\begin{bmatrix} d_\phi & d_\theta & d_\psi \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0.25 \end{bmatrix}$ at $t = 10$ s, and represent the unknown constant disturbances changing slowly along time, such as static wind and unmodeled dynamic errors. They were added to the quadrotor dynamic model as follows

$$\begin{aligned}
\ddot{\phi} &= \frac{I_y - I_z}{I_x}\dot{\theta}\dot{\psi} - \frac{J_r}{I_x}\dot{\theta}\Omega + \frac{u_2}{I_x} - d_\phi \\
\ddot{\theta} &= \frac{I_z - I_x}{I_y}\dot{\phi}\dot{\psi} + \frac{J_r}{I_y}\dot{\phi}\Omega + \frac{u_3}{I_y} - d_\theta \\
\ddot{\psi} &= \frac{I_x - I_y}{I_z}\dot{\phi}\dot{\theta} + \frac{u_4}{I_z} - d_\psi
\end{aligned} \tag{8.1}$$

The simulation results with uncertainties and disturbances are shown in Figure 8.8-8.13. Figure 8.8 shows the desired attitude and the actual attitudes. The attitude errors are shown in Figure 8.9, and the different control inputs are shown in Figure 8.10-8.12. The Adaptive gains for the AFGS are shown in Figure 8.13.

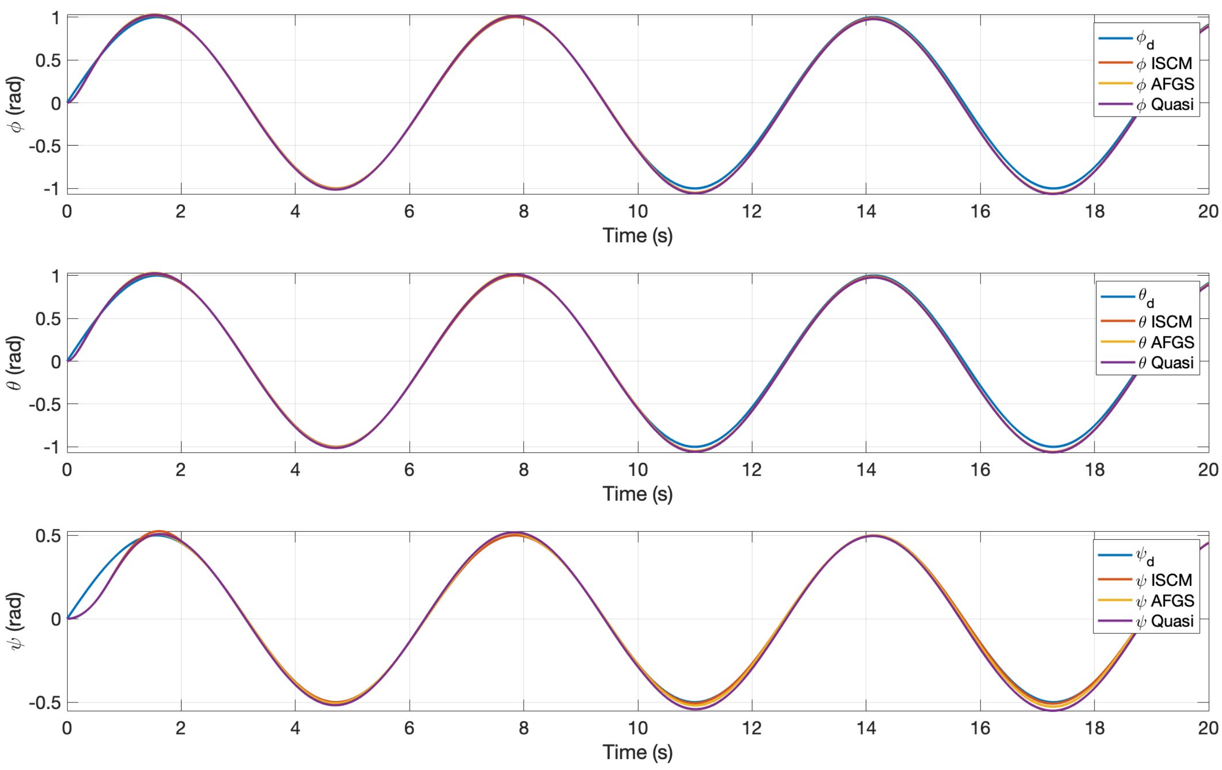


Figure 8.8: Attitude of quadrotor with uncertainties and disturbances.

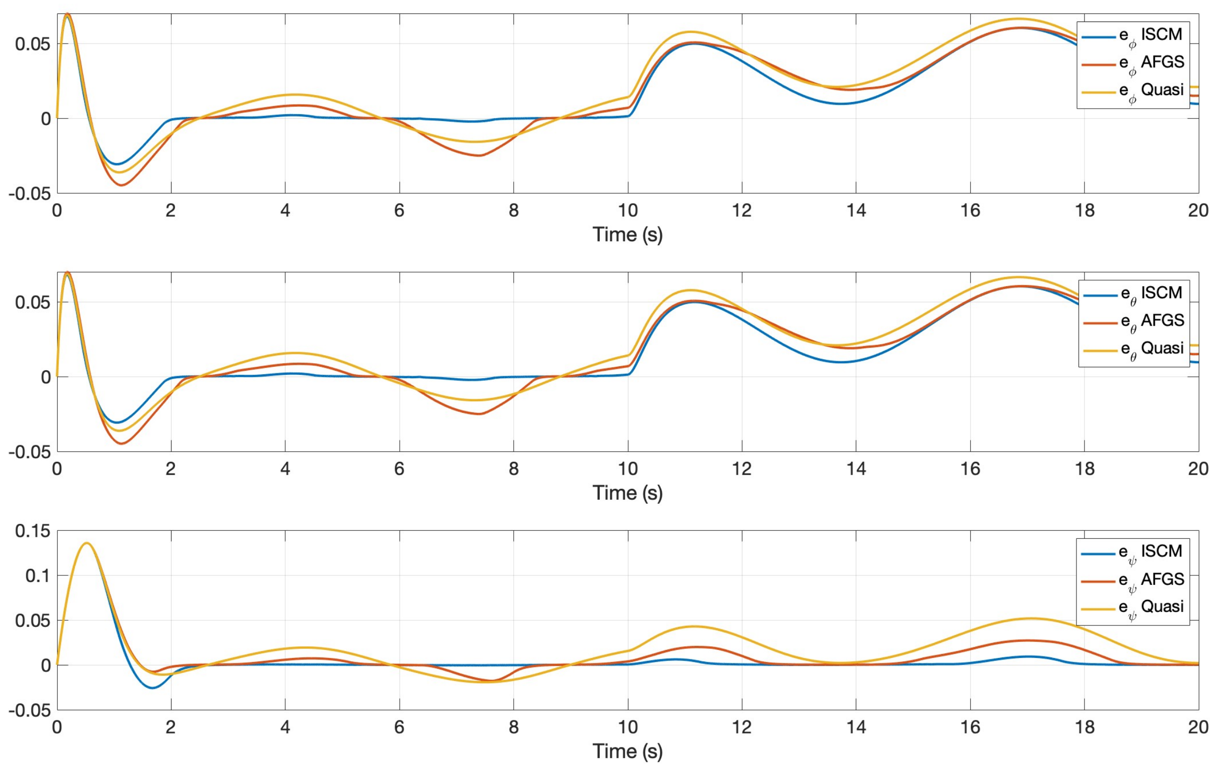


Figure 8.9: The attitude errors with uncertainties and disturbances.

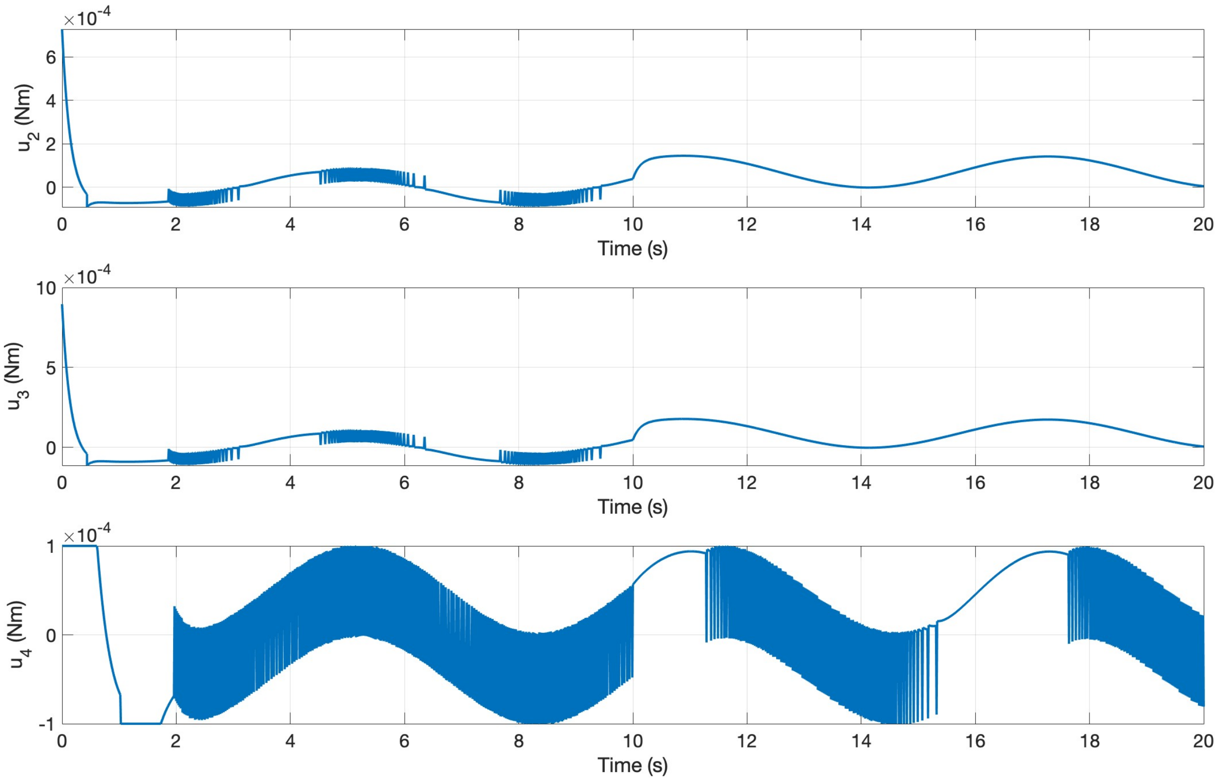


Figure 8.10: ISMC control inputs with uncertainties and disturbances.

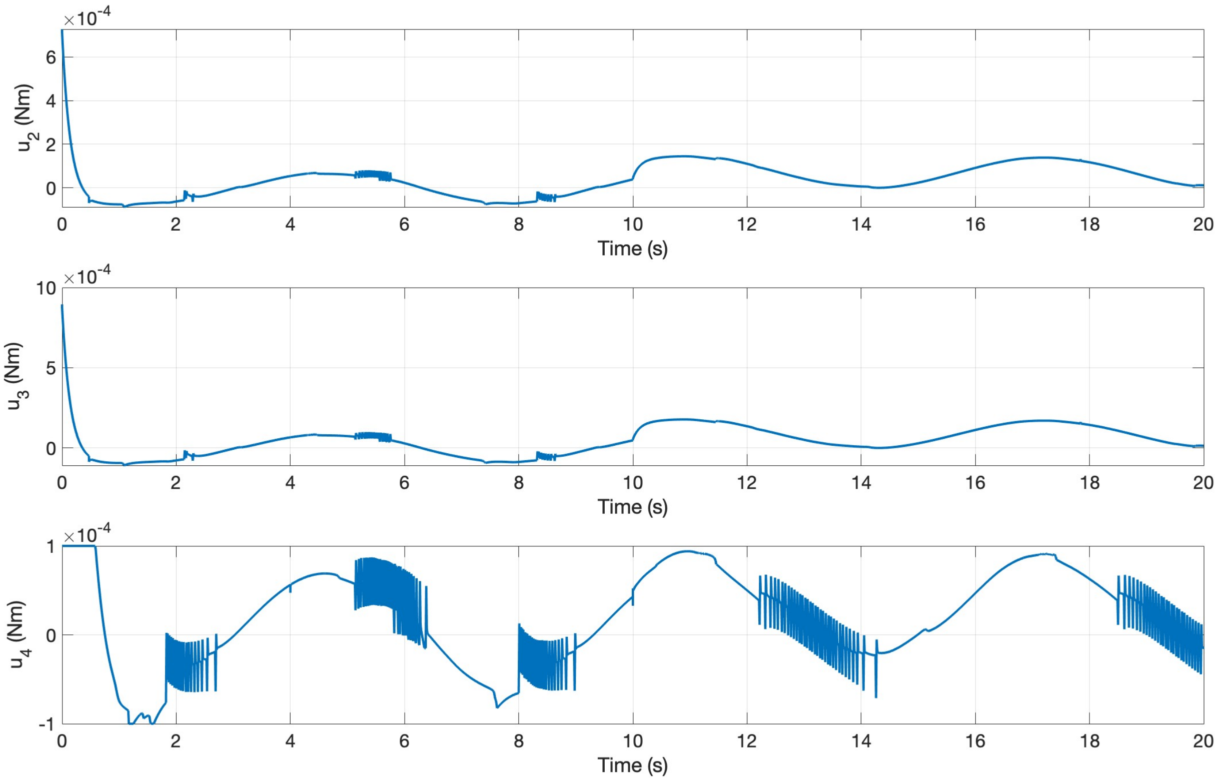


Figure 8.11: AFGS control inputs with uncertainties and disturbances.

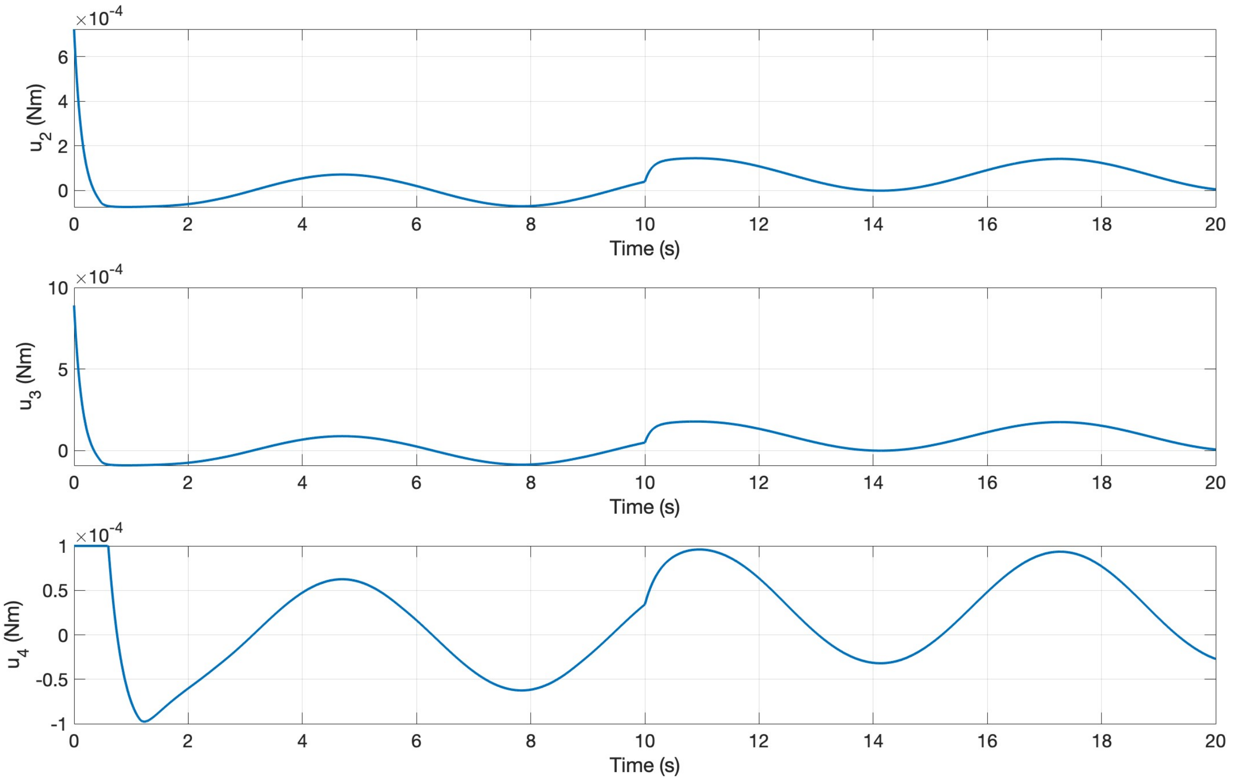


Figure 8.12: Quasi control inputs with uncertainties and disturbances.

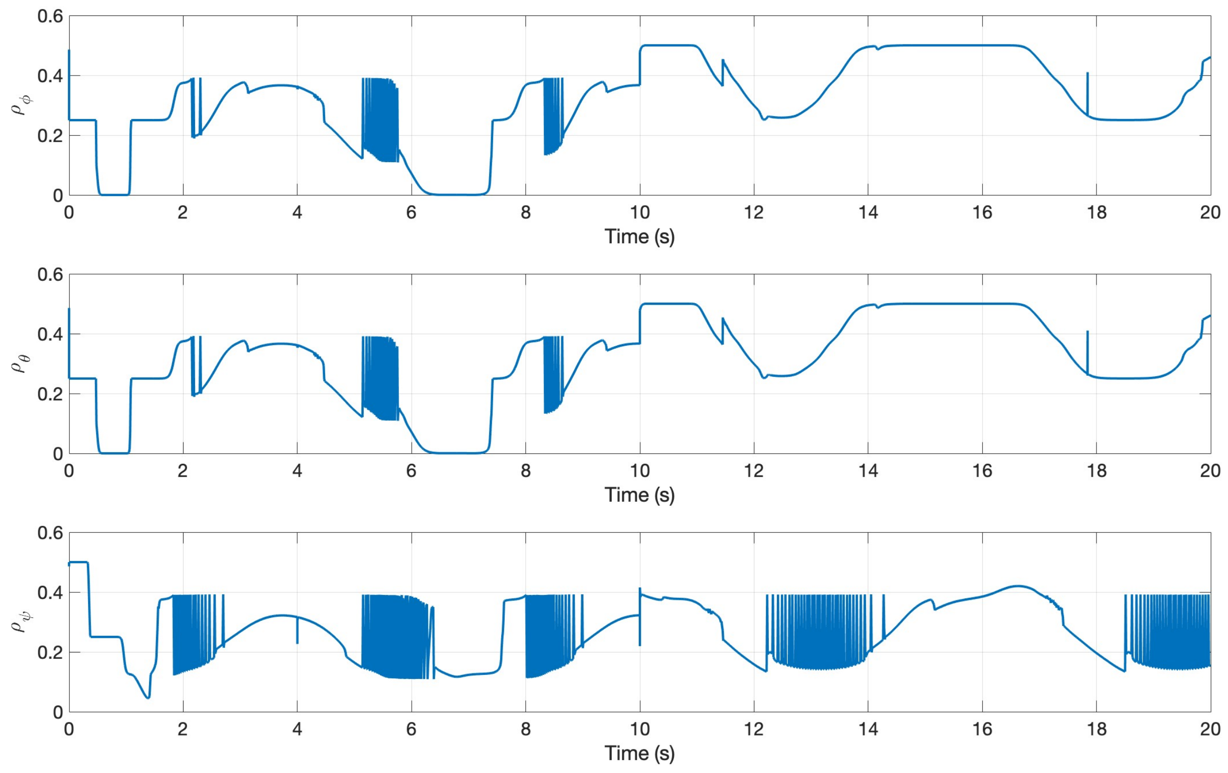


Figure 8.13: AFGS control gains with uncertainties and disturbances.

Table 8.3 shows the performance indices comparison between the three control methods with uncertainties and disturbances.

		ISMC	AFGS	Quasi
ϕ	IAE	0.3860	0.4993	0.5509
	ISE	0.0157	0.0198	0.0230
	ITAE	5.0706	6.0748	6.7852
	ITSE	0.2125	0.2530	0.3072
θ	IAE	0.3860	0.4993	0.5509
	ISE	0.0157	0.0198	0.0230
	ITAE	5.0706	6.0748	6.7852
	ITSE	0.2125	0.2530	0.3072
ψ	IAE	0.1399	0.2408	0.4525
	ISE	0.0111	0.0132	0.0213
	ITAE	0.3914	1.7487	4.3832
	ITSE	0.0080	0.0367	0.1510

Table 8.3: Performance indices with uncertainties and disturbances added.

The results show that ISMC with the use of the sign function shows better performance in terms of trajectory tracking, but the proposed AFGS and quasi controllers have an acceptable performance. The ISMC and AFGS does show better results in terms of robustness against model uncertainties and disturbance rejection than the quasi, but the quasi also shows acceptable robustness and disturbance rejection.
However, in terms of chattering reduction in the control inputs, the AFGS and quasi showed a significant reduction of chattering compared to the ISMC, where it is completely eliminated with the quasi controller.
By considering both trajectory tracking and chattering reduction, the AFGS and quasi controllers is considered to the better than the ISMC. In the subsequent sections of this chapter, the sign function in the ISMC control inputs was replaced with the approximated continuous function because of it's simplicity and effectiveness in reducing the chattering phenomena.

8.1.2 ISMC vs. Linear Control Methods

Several types of controllers has been introduced in this work, which includes PD, LQR and ISMC. A comparative study of the controllers will now be proposed where, as mentioned, the continuous approximation of the sign function was used in the ISMC control inputs. All the controllers were tuned by genetic algorithm as proposed in Chapter 6. Table 8.4, 8.5 and 8.6 shows the controller parameters obatained by genetic algorithm for PD, LQR and ISMC respectively.

i	k_{pi}	k_{di}
x	1.2404	0.4001
y	1.2404	0.4001
z	78.5735	12.1851
ϕ	98.8054	57.6252
θ	70.4077	43.8701
ψ	46.5495	14.5408

Table 8.4: PD controller parameters.

Controller	Q matrices	R matrices
x	$\begin{bmatrix} 11.7 & 0 \\ 0 & 0.6 \end{bmatrix}$	$\begin{bmatrix} 9.9 \end{bmatrix}$
y	$\begin{bmatrix} 11.7 & 0 \\ 0 & 0.6 \end{bmatrix}$	$\begin{bmatrix} 9.9 \end{bmatrix}$
z	$\begin{bmatrix} 1760.3 & 0 \\ 0 & 1393.5 \end{bmatrix}$	$\begin{bmatrix} 0.2 \end{bmatrix}$
ϕ	$\begin{bmatrix} 1062.7 & 0 \\ 0 & 305.2 \end{bmatrix}$	$\begin{bmatrix} 4.9 \end{bmatrix}$
θ	$\begin{bmatrix} 1791.8 & 0 \\ 0 & 462.5 \end{bmatrix}$	$\begin{bmatrix} 4.1 \end{bmatrix}$
ψ	$\begin{bmatrix} 785.9770 & 0 \\ 0 & 68.8622 \end{bmatrix}$	$\begin{bmatrix} 2.6613 \end{bmatrix}$

Table 8.5: LQR weighting matrices.

i	λ_i	ρ_i	ζ_i	k_i
x	2.9998	3.1285	0.6052	0.0034
y	2.9998	3.1285	0.6052	0.0034
z	5.0351	6.6602	2.7307	0.0011
ϕ	17.5038	38.5424	4.9774	0.0037
θ	16.2555	38.9482	4.2975	0.0032
ψ	39.0089	8.3646	2.3001	0.0016

Table 8.6: ISMC parameters.

To study the stabilization, attitude tracking, and position tracking ability of the controllers, a series of simulations were conducted. An altitude and attitude stabilization and tracking simulation was carried out to test the inner loop controllers' ability to stabilize and track the quadrotors altitude and attitude. To test the position tracking, a series of 3D trajectories were conducted.

Altitude and Attitude Tracking Simulations

In these simulations the quadrotor was first set to hover at a fixed altitude and attitude to test the inner loop controllers' ability to stabilize the quadrotor. Then the inner loop controllers' ability to track a sinusoidal reference was tested. The performance of the inner loop controllers was tested by considering two scenarios:
Scenario 1: In this scenario neither model uncertainty nor disturbance is considered.
Scenario 2: Uncertainty (25% subtracted) in rotary inertia and external disturbances is added. In this case, we consider model uncertainty 25% subtracted in all three moments of inertia. The external disturbances d_ϕ, d_θ, d_ψ are added to the quadrotor dynamic model as follows

$$\begin{aligned}\ddot{\phi} &= \frac{I_y - I_z}{I_x}\dot{\theta}\dot{\psi} - \frac{J_r}{I_x}\dot{\theta}\Omega + \frac{u_2}{I_x} + d_\phi \\ \ddot{\theta} &= \frac{I_z - I_x}{I_y}\dot{\phi}\dot{\psi} + \frac{J_r}{I_y}\dot{\phi}\Omega + \frac{u_3}{I_y} + d_\theta \\ \ddot{\psi} &= \frac{I_x - I_y}{I_z}\dot{\phi}\dot{\theta} + \frac{u_4}{I_z} + d_\psi\end{aligned} \tag{8.2}$$

and are defined as normal Gaussian noise

$$\begin{aligned}d_\phi &= \mathcal{N}_\phi(0,\, 0.5) \\ d_\theta &= \mathcal{N}_\theta(0,\, 0.5) \\ d_\psi &= \mathcal{N}_\psi(0,\, 0.5)\end{aligned} \tag{8.3}$$

with a sample time of 0.1. The disturbances represent the fast varying and unknown stochastic disturbances, such as stochastic wind and uncertain measurement noise.

For the first test the desired altitude and attitude was chosen as $z_d = -1$, $\phi_d = \theta_d = \psi_d = 0.5$. Figure 8.14 shows the desired altitude and attitude, and the actual altitude and attitude under Scenario 1. Table 8.7 shows the performance indices comparison between the three control methods under Scenario 1. Figure 8.15 and Table 8.8 shows the results under Scenario 2. All the controllers successfully stabilize the altitude and attitude of the quadrotors, although with varying performance levels. The PD controller, incorporating a feedforward term, effectively and rapidly stabilizes the altitude. The ISMC is also able to stabilize the altitude in a quick manner, while the LQR has the slowest response with considerably worse performance indices for the altitude. In terms of attitude stabilization, the ISMC controller outshines the others with its faster response time and significantly better performance indices. The response time of the yaw angle is noticeably affected by control input saturation, resulting in slower response times and overshoot for all controllers. The controllers are able to handle the uncertainties in the rotary inertias and the external disturbances under Scenario 2 with minor changes in the performance indices. Under Scenario 2, the performance indices for the attitude overall shows a slight reduction. This could be attributed to the fact that reducing the inertia makes the quadrotor less resistant to rotational changes, enabling it to respond more quickly to control inputs.

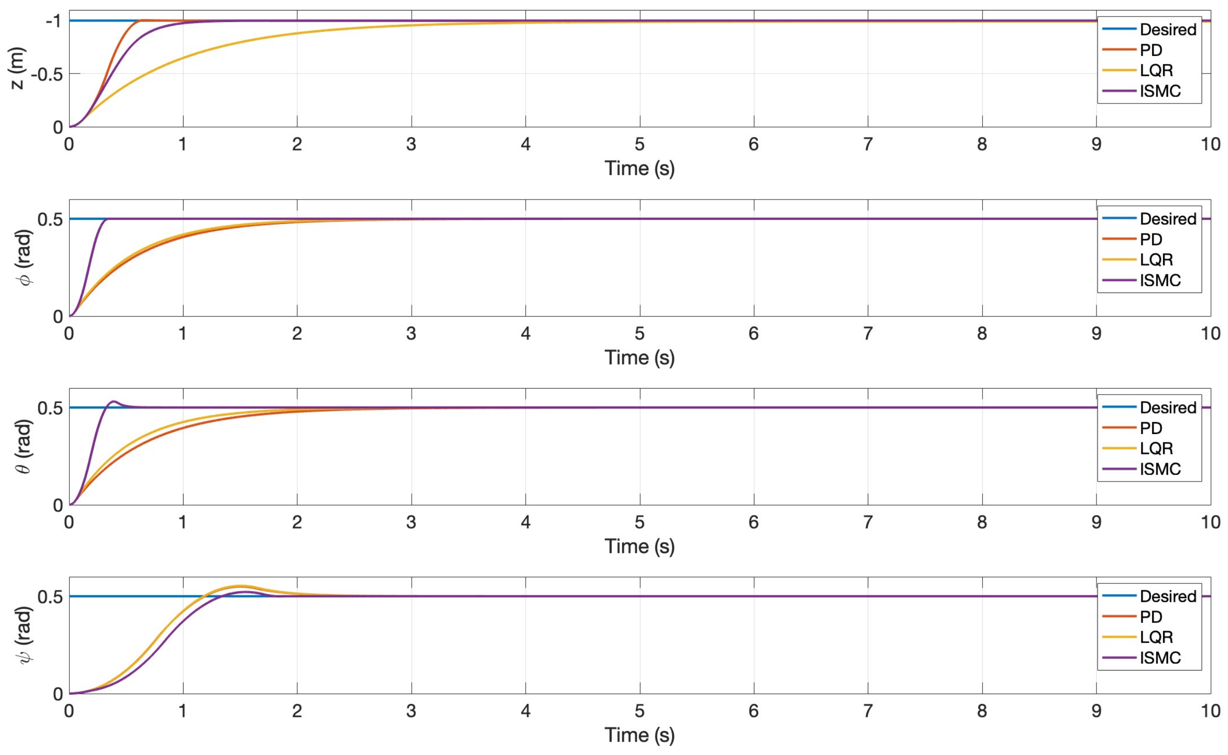


Figure 8.14: Quadrotor altitude and attitude under Scenario 1.

		PD	LQR	ISMC
z	IAE	0.3492	1.0234	0.4131
	ISE	0.2528	0.5095	0.2838
	ITAE	0.1779	1.2632	0.1177
	ITSE	0.0383	0.2389	0.0518
ϕ	IAE	0.3038	0.2811	0.0867
	ISE	0.0789	0.0735	0.0330
	ITAE	0.1774	0.1509	0.0097
	ITSE	0.0231	0.0198	0.0026
θ	IAE	0.3255	0.2709	0.0973
	ISE	0.0848	0.0719	0.0363
	ITAE	0.2032	0.1381	0.0125
	ITSE	0.0266	0.0184	0.0031
ψ	IAE	0.3812	0.3835	0.4010
	ISE	0.1386	0.1393	0.1562
	ITAE	0.1908	0.1935	0.1873
	ITSE	0.0462	0.0467	0.0570

Table 8.7: Performance indices under Scenario 1.

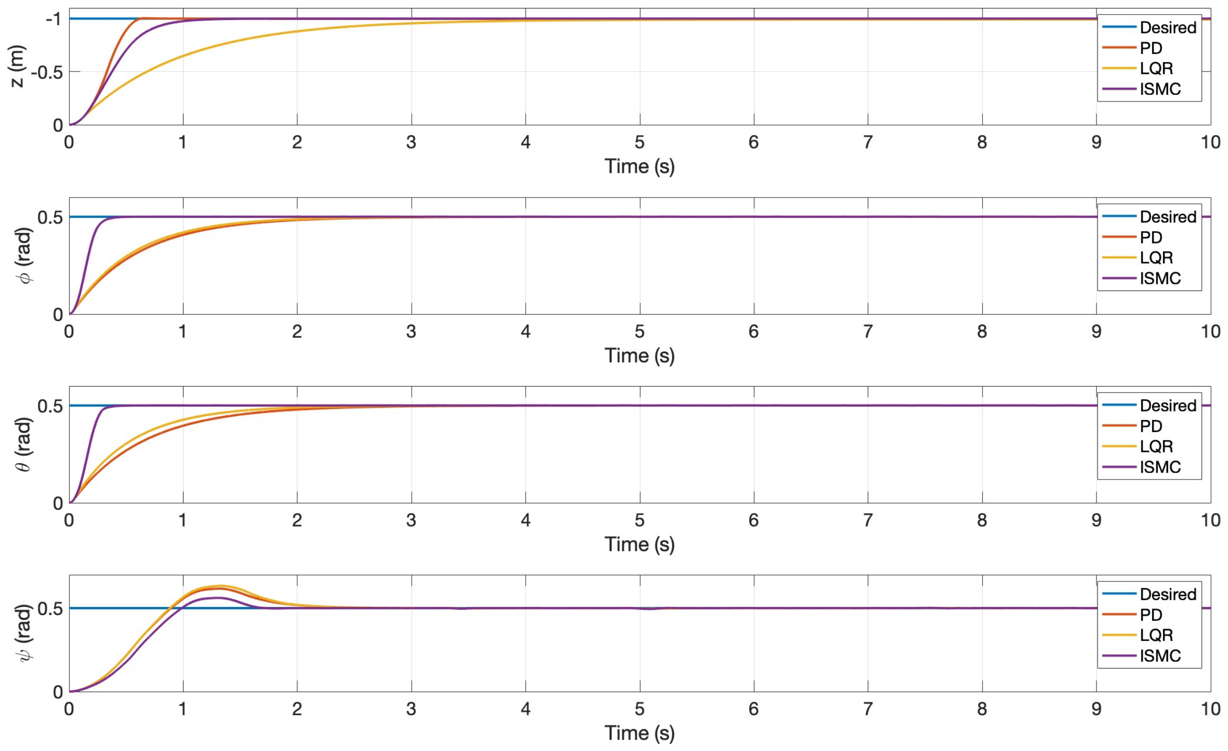


Figure 8.15: Quadrotor altitude and attitude tracking under Scenario 2.

		PD	LQR	ISMC
z	IAE	0.3494	1.0235	0.4137
	ISE	0.2530	0.5095	0.2843
	ITAE	0.1779	1.2634	0.1181
	ITSE	0.0383	0.2389	0.0520
ϕ	IAE	0.3005	0.2776	0.0794
	ISE	0.0773	0.0718	0.0287
	ITAE	0.1754	0.1489	0.0120
	ITSE	0.0226	0.0193	0.0020
θ	IAE	0.3216	0.2663	0.0844
	ISE	0.0829	0.0696	0.0312
	ITAE	0.2006	0.1356	0.0130
	ITSE	0.0259	0.0178	0.0023
ψ	IAE	0.3502	0.3636	0.3215
	ISE	0.1084	0.1111	0.1135
	ITAE	0.2179	0.2376	0.1518
	ITSE	0.0341	0.0375	0.0321

Table 8.8: Performance indices under Scenario 2.

For the second test the desired altitude and attitude was chosen as $z_d = -\sin(t) - 1$, $\phi_d = \theta_d = 0.5\sin(4t)$, $\psi_d = 0.5\sin(t)$. Figure 8.16 shows the desired altitude and attitude, and the actual altitude and attitude under Scenario 1. Table 8.9 shows the performance indices comparison between the three control methods under Scenario 1. Figure 8.17 and Table 8.10 shows the results under Scenario 2. Similar to the first test, all the controllers successfully track the desired trajectories, with varying performance levels. The PD controller still shows the best performance with the tracking of the altitude, where ISCM has slightly worse performance indices, and LQR performs the worst. In terms of attitude stabilization, the ISMC controller still outperforms the PD and LQR. All the controllers are able to handle the uncertainties in the rotary inertia's and the external disturbances under Scenario 2 with minor changes in the performance indices.

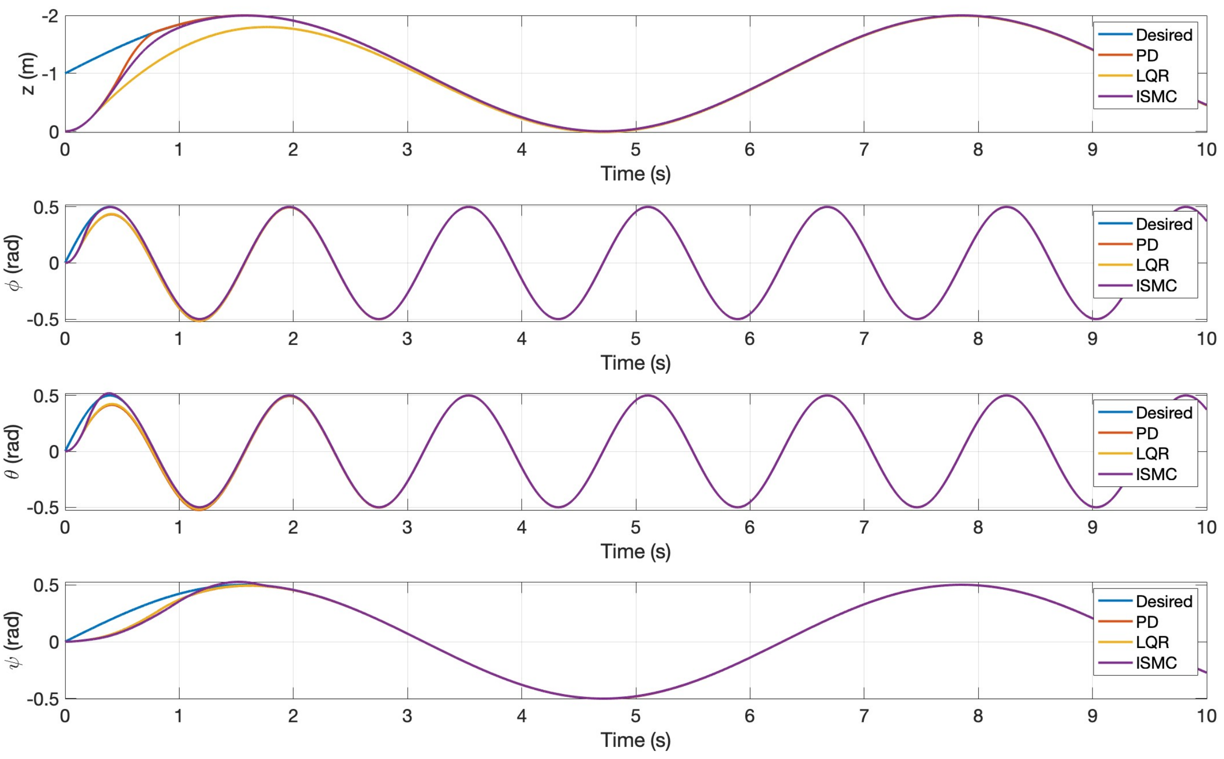


Figure 8.16: Quadrotor altitude and attitude tracking under Scenario 1.

		PD	LQR	ISMC
z	IAE	0.4995	1.1831	0.5715
	ISE	0.4212	0.6872	0.4522
	ITAE	0.1774	1.3766	0.1925
	ITSE	0.0897	0.3341	0.1077
ϕ	IAE	0.0740	0.0689	0.0186
	ISE	0.0042	0.0039	0.0014
	ITAE	0.0596	0.0532	0.0068
	ITSE	0.0015	0.0013	0.00015671
θ	IAE	0.0947	0.0800	0.0262
	ISE	0.0065	0.0056	0.0023
	ITAE	0.0766	0.0581	0.0089
	ITSE	0.0025	0.0018	0.00030173
ψ	IAE	0.1148	0.1131	0.1261
	ISE	0.0113	0.0113	0.0143
	ITAE	0.0786	0.0729	0.0818
	ITSE	0.0063	0.0063	0.0084

Table 8.9: Performance indices under Scenario 1.

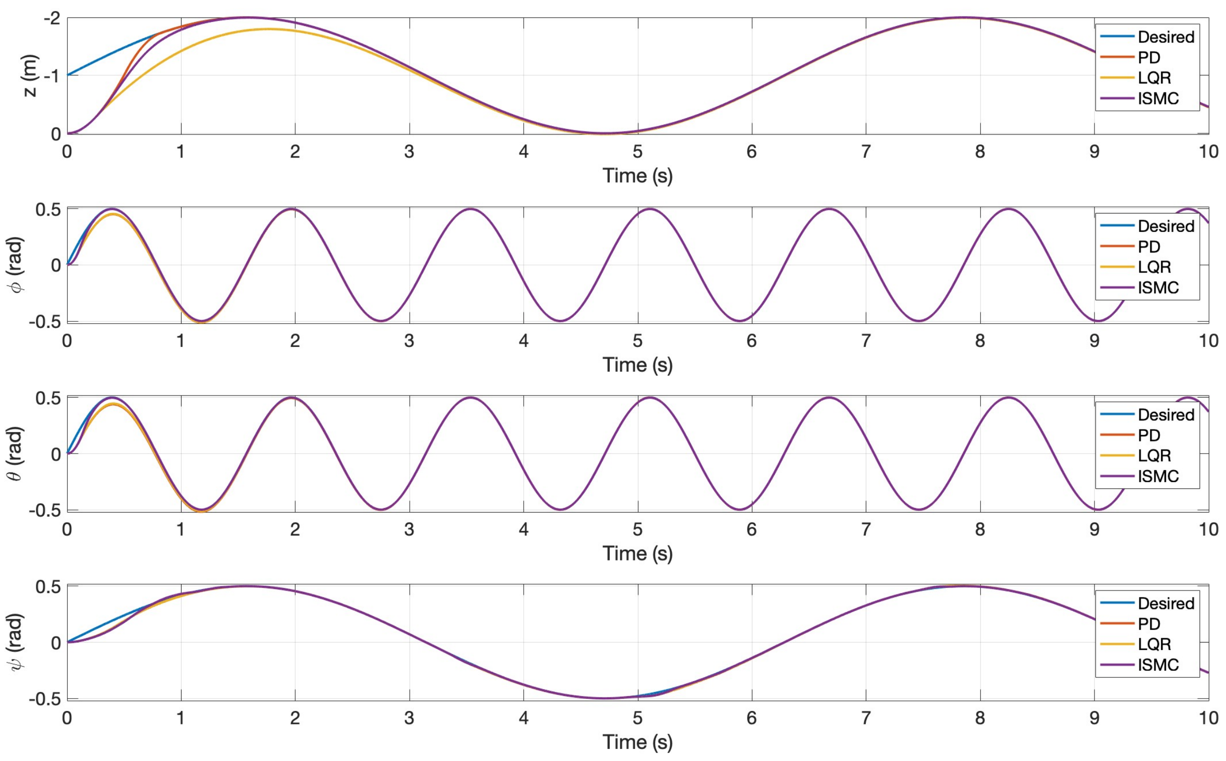


Figure 8.17: Quadrotor altitude and attitude tracking under Scenario 2.

		PD	LQR	ISMC
z	IAE	0.5065	1.1853	0.5778
	ISE	0.4272	0.6895	0.4581
	ITAE	0.1810	1.3791	0.1963
	ITSE	0.0924	0.3356	0.1103
ϕ	IAE	0.0542	0.0505	0.0156
	ISE	0.0022	0.0021	0.00062307
	ITAE	0.0469	0.0425	0.0241
	ITSE	0.00074884	0.00064999	0.000068148
θ	IAE	0.0693	0.0584	0.0199
	ISE	0.0035	0.0029	0.0010
	ITAE	0.0590	0.0457	0.0262
	ITSE	0.0013	0.00089846	0.00012054
ψ	IAE	0.0585	0.0561	0.0485
	ISE	0.0021	0.0021	0.0023
	ITAE	0.1207	0.1086	0.0689
	ITSE	0.0015	0.0013	0.0012

Table 8.10: Performance indices under Scenario 2.

Trajectory Tracking Simulations

In order to test the proposed controllers' ability to track a 3D trajectory, three different trajectories were simulated: a square trajectory, ascent helix and a more complex ascent helix. The ascent helix and the complex ascent helix are examples of relatively hard trajectories for quadrotors in the literature, and they are considered as a benchmark for demonstrating the complete control systems capability of executing complex trajectories. The trajectories are given by

$$\begin{aligned}
&\text{Square trajectory}\begin{cases} x_d = [0,\ 0,\ 2,\ 2,\ 0,\ 0,\ 0], & \text{for} \quad t = [0,\ 2,\ 4,\ 6,\ 8,\ 10,\ 12] \\ y_d = [0,\ 2,\ 2,\ 0,\ 0,\ 0,\ 0], & \text{for} \quad t = [0,\ 2,\ 4,\ 6,\ 8,\ 10,\ 12] \\ z_d = -2 \end{cases} \\
&\text{Ascent helix}\begin{cases} x_d = \cos(t) \\ y_d = \sin(t) \\ z_d = -t \end{cases} \\
&\text{Complex ascent helix}\begin{cases} x_d = \cos(t) + \cos^2(t) \\ y_d = \sin(t) - \cos^2(t) \\ z_d = -t \end{cases}
\end{aligned} \tag{8.4}$$

The desired yaw angle was set to $\psi_d = 0$ for all the trajectories.

The performance of the controllers was tested considering two scenarios:
Scenario 1: In this scenario neither model uncertainty nor disturbance is considered.
Scenario 2: Uncertainty (25% subtracted) in rotary inertia and external disturbances is added. In this case, we consider model uncertainty 25% subtracted in all three moments of inertia.
The external disturbances d_x, d_y, d_z, d_ϕ, d_θ, d_ψ were added to the quadrotor dynamic model as follows

$$\begin{aligned}
\ddot{x} &= -\frac{u_1}{m}\left(\sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta)\right) + d_x \\
\ddot{y} &= -\frac{u_1}{m}\left(\cos(\phi)\sin(\psi)\sin(\theta) - \cos(\psi)sin(\phi)\right) + d_y \\
\ddot{z} &= g - \frac{u_1}{m}\left(\cos(\phi)\cos(\theta)\right) + d_z \\
\ddot{\phi} &= \frac{I_y - I_z}{I_x}\dot{\theta}\dot{\psi} - \frac{J_r}{I_x}\dot{\theta}\Omega + \frac{u_2}{I_x} + d_\phi \\
\ddot{\theta} &= \frac{I_z - I_x}{I_y}\dot{\phi}\dot{\psi} + \frac{J_r}{I_y}\dot{\phi}\Omega + \frac{u_3}{I_y} + d_\theta \\
\ddot{\psi} &= \frac{I_x - I_y}{I_z}\dot{\phi}\dot{\theta} + \frac{u_4}{I_z} + d_\psi
\end{aligned} \tag{8.5}$$

and were defined as normal Gaussian noise

$$\begin{aligned}
d_x &= \mathcal{N}_x(0,\, 0.1) \\
d_y &= \mathcal{N}_y(0,\, 0.1) \\
d_z &= \mathcal{N}_z(0,\, 0.1) \\
d_\phi &= \mathcal{N}_\phi(0,\, 0.5) \\
d_\theta &= \mathcal{N}_\theta(0,\, 0.5) \\
d_\psi &= \mathcal{N}_\psi(0,\, 0.5)
\end{aligned} \tag{8.6}$$

with a sample time of 0.1. They represent the fast varying and unknown stochastic disturbances, such as stochastic wind and uncertain measurement noise.

Square Trajectory

The flight of the quadrotor in the 3D space for the case of the square trajectory under Scenario 1 is visually depicted in Figure 8.18, while the response for the position is shown in Figure 8.19, and the performance indices are listed in Table 8.11. All the proposed controllers enable the quadrotor to follow the square trajectory without major errors. While considering the performance indices, the PD controller demonstrates the best performance. However, despite its stronger performance indices, an overshoot in altitude is observed when utilizing the PD controller, which is not observed in the case of the LQR and ISMC controllers. Nevertheless, Figure 8.19 reveals that the ISMC controller achieves the desired values in closer proximity in a faster time than with the PD and LQR.

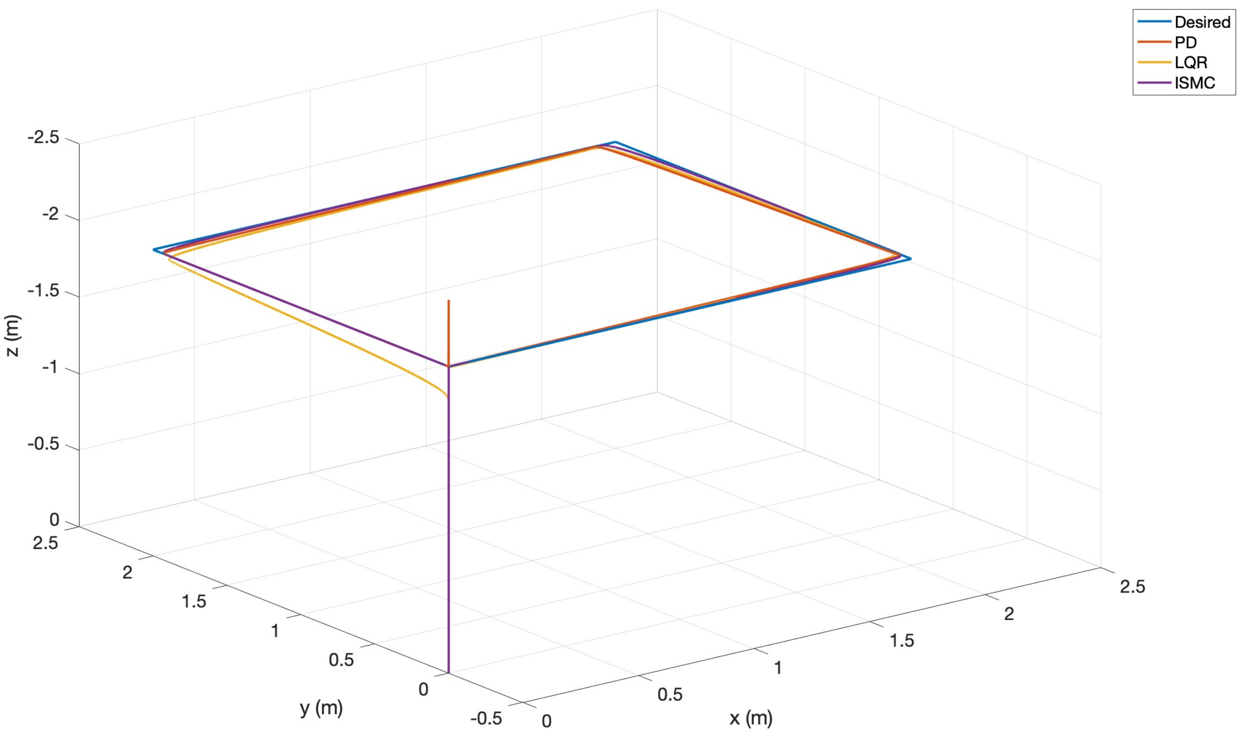


Figure 8.18: 3D view of the square trajectory responses under Scenario 1.

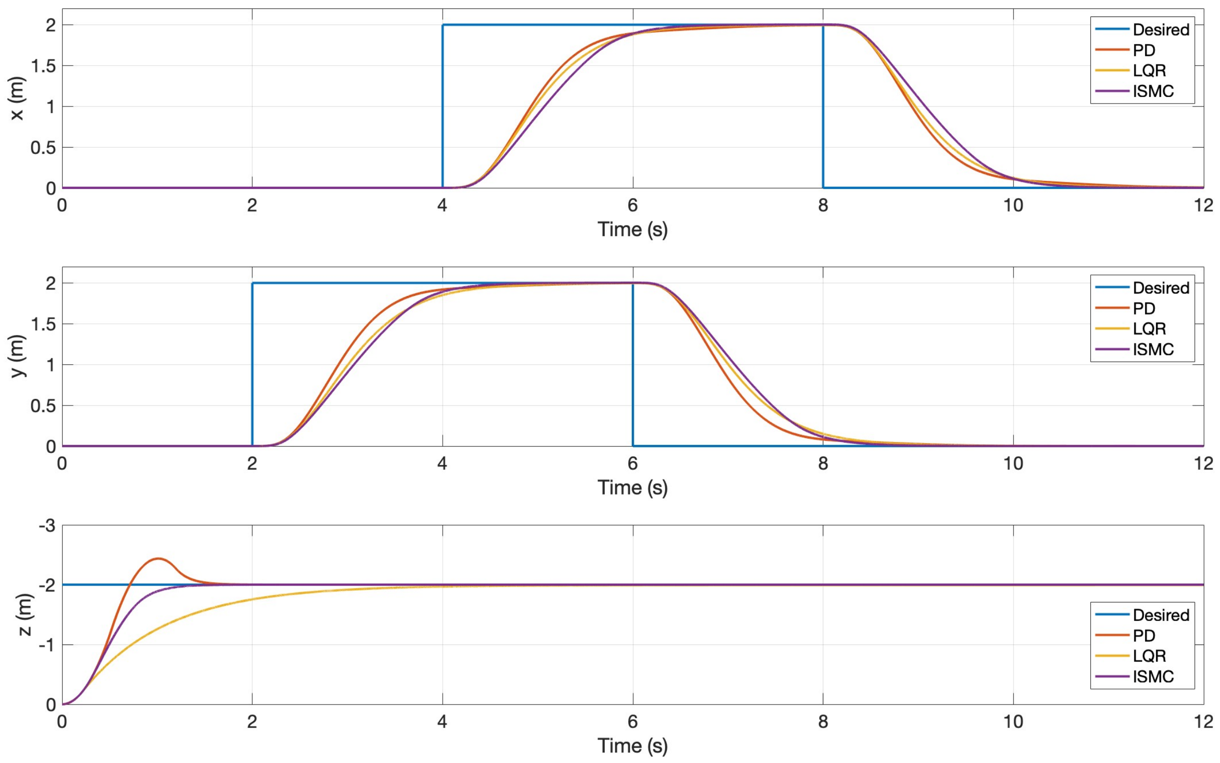


Figure 8.19: Position tracking of the square trajectory under Scenario 1.

		PD	LQR	ISMC
x	IAE	4.1393	4.2700	4.5047
	ISE	5.9847	6.2006	6.6999
	ITAE	27.6089	28.4878	30.1161
	ITSE	38.5303	40.0612	43.5622
y	IAE	3.9861	4.4363	4.4877
	ISE	5.8086	6.3713	6.6527
	ITAE	18.4964	20.8789	21.0221
	ITSE	25.7023	28.5161	29.9369
z	IAE	1.0888	2.0769	1.0617
	ISE	1.4712	2.2313	1.5147
	ITAE	0.4428	2.2740	0.3717
	ITSE	0.3558	1.0395	0.3574

Table 8.11: Performance indices for the square trajectory under Scenario 1.

Figure 8.20 illustrates the quadrotors flight in 3D space, for the square trajectory under Scenario 2, which incorporates disturbances and uncertainties. Figure 8.21 displays the response of the quadrotors position. Additionally, Table 8.12 provides a list of the performance indices pertaining to the quadrotors flight in this scenario. In this scenario, the PD controller is significantly impacted by uncertainties and disturbances, whereas the ISMC and LQR controllers demonstrate robustness by effectively handling them with minimal changes in performance from Scenario 1.

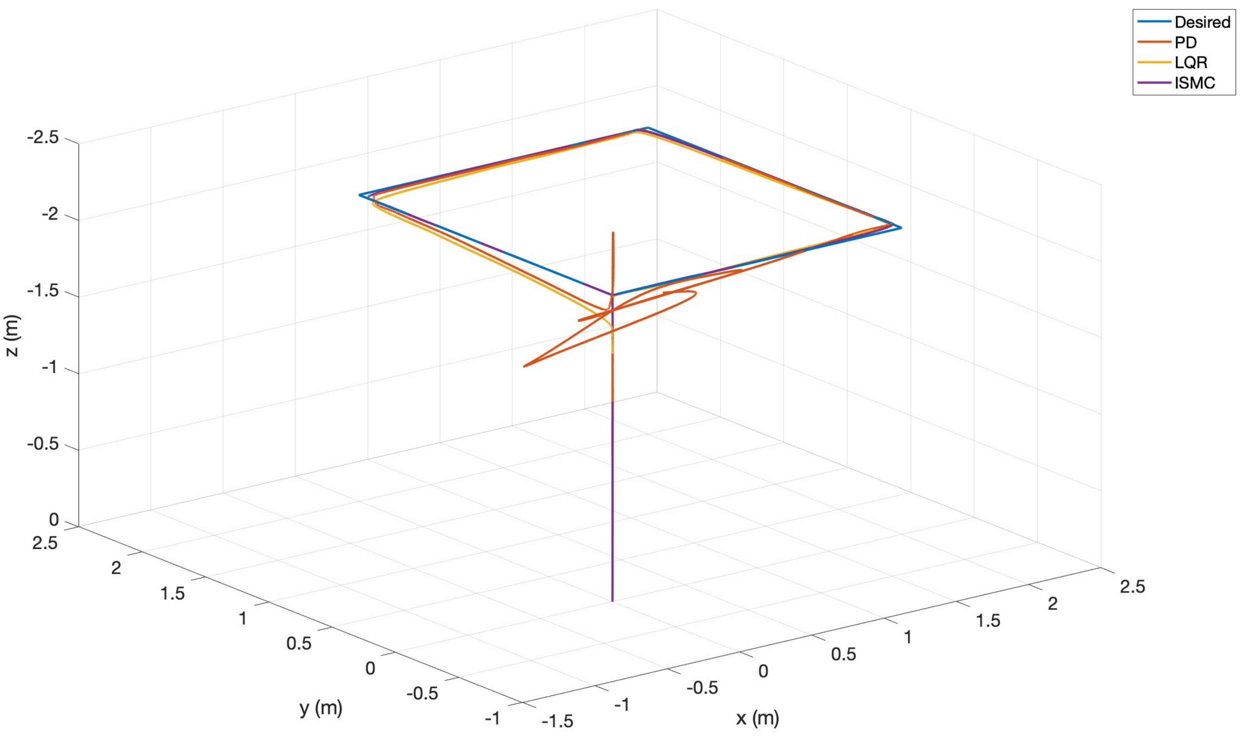


Figure 8.20: 3D view of the square trajectory responses under Scenario 2.

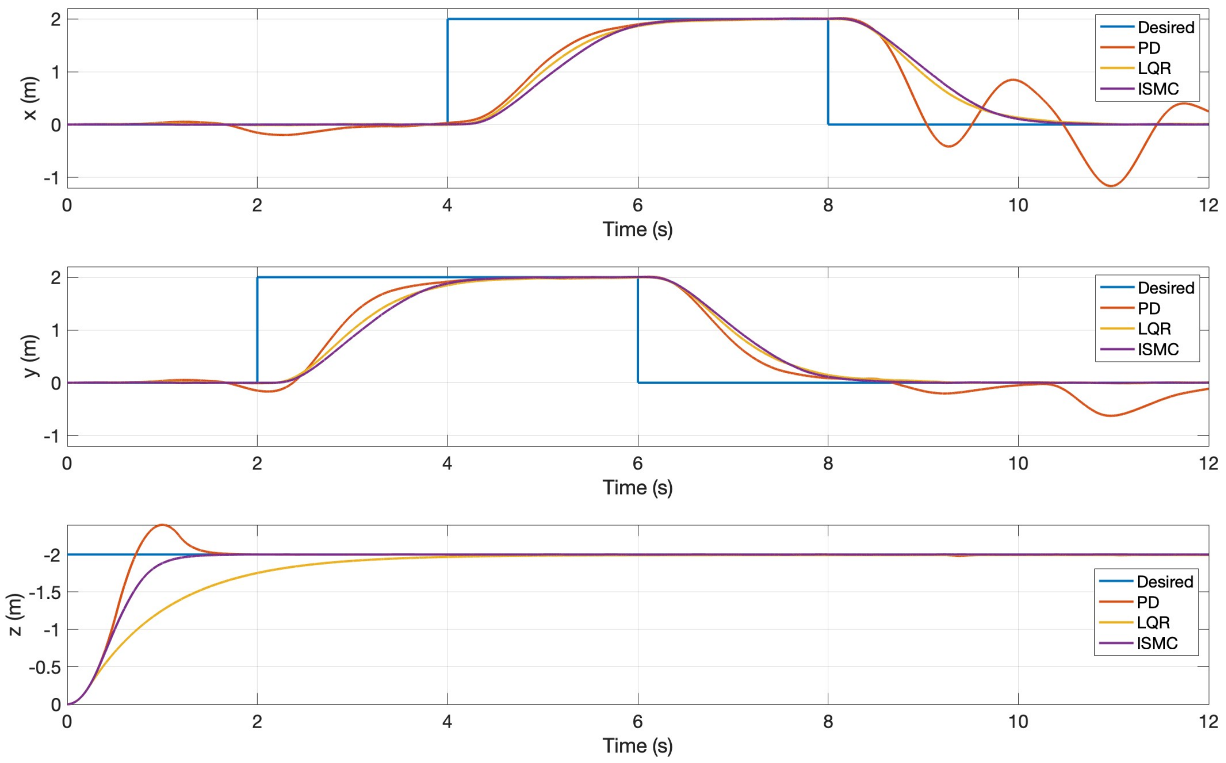


Figure 8.21: Position tracking of the square trajectory under Scenario 2.

		PD	LQR	ISMC
x	IAE	5.2153	4.3545	4.5431
	ISE	6.5024	6.2488	6.7403
	ITAE	38.0983	28.8542	30.1102
	ITSE	45.3382	40.2182	43.4095
y	IAE	4.7400	4.4368	4.5174
	ISE	6.1803	6.3035	6.6790
	ITAE	26.2653	20.8315	20.9960
	ITSE	28.6075	28.0272	29.6081
z	IAE	1.0881	2.0843	1.0761
	ISE	1.4762	2.2458	1.5334
	ITAE	0.5540	2.2820	0.4022
	ITSE	0.3502	1.0479	0.3656

Table 8.12: Performance indices for the square trajectory under Scenario 2.

Ascent Helix

The flight of the quadrotor in the 3D space for the case of the ascent helix trajectory under Scenario 1 is depicted in Figure 8.22, while the response for the position is shown in Figure 8.23, and the performance indices are listed in Table 8.13. The results shows that the ISMC controller exhibits significantly better tracking performance, closely following the desired trajectory with minimal errors. In contrast, both the PD and LQR control strategies reveal overshoots in the x- and y-positions for the sinusoidal-based reference signal. These findings demonstrate that the PD and LQR controllers are less effective in controlling the quadrotor during more complex maneuvers compared to the ISMC. The PD and LQR controllers demonstrate comparable performance in the x- and y-positions, with the PD controller showing slightly better performance indices. However, the PD controller excels in precisely following the desired altitude.

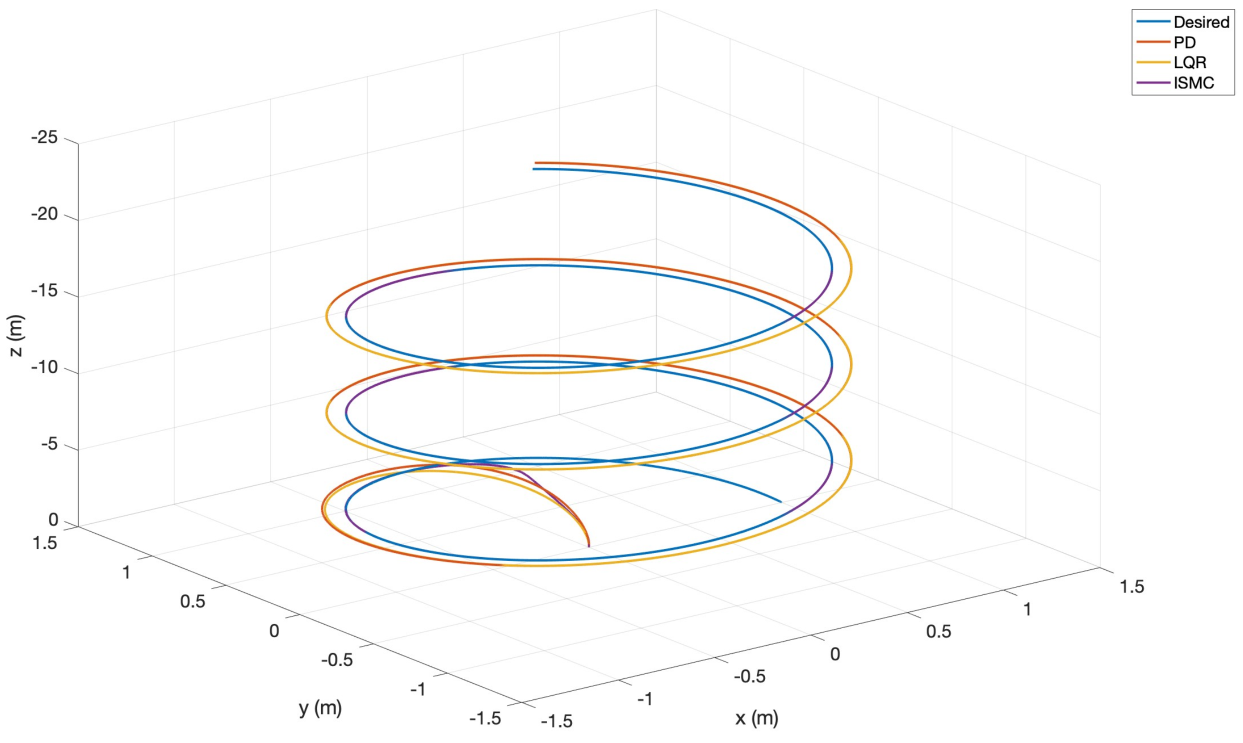


Figure 8.22: 3D view of the ascent helix trajectory responses under Scenario 1.

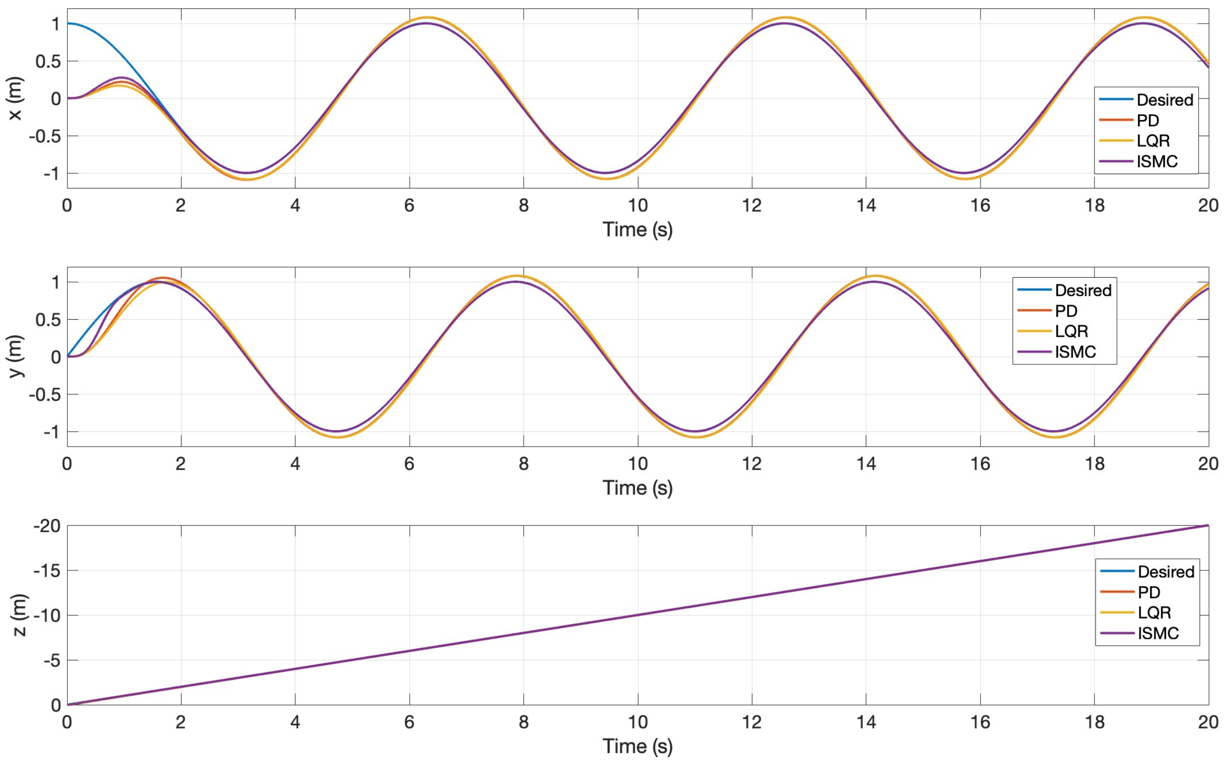


Figure 8.23: Position tracking of the ascent helix trajectory under Scenario 1.

		PD	LQR	ISMC
x	IAE	1.8845	1.9910	0.8118
	ISE	0.6848	0.7258	0.5720
	ITAE	11.5337	12.3651	0.5745
	ITSE	0.9822	1.1227	0.2009
y	IAE	1.2199	1.3205	0.1380
	ISE	0.1232	0.1461	0.0245
	ITAE	10.4090	11.2001	0.0907
	ITSE	0.6967	0.8134	0.0095
z	IAE	0.0141	0.1783	0.0186
	ISE	0.00044502	0.0026	0.00067075
	ITAE	0.0123	1.3683	0.0057
	ITSE	0.000064507	0.0099	0.00012379

Table 8.13: Performance indices for the ascent helix trajectory under Scenario 1.

Under Scenario 2, the flight of the quadrotor in 3D space for the case of the ascent helix trajectory is depicted in Figure 8.24. The response for the position is shown in Figure 8.25, and the performance indices are listed in Table 8.14. Notably, in the presence of uncertainties and disturbances, the PD controller's ineffectiveness becomes apparent. It struggles to adequately handle these external factors, leading to deviations from the desired trajectory. In contrast, both the ISMC and LQR controllers exhibit robustness by effectively managing the uncertainties and disturbances, ensuring minimal impact on their overall performance compared to Scenario 1.

The results reinforce the superiority of the ISMC controller, where it showcases its ability to mitigate the influence of uncertainties and disturbances, maintaining precise trajectory tracking with minimal errors.

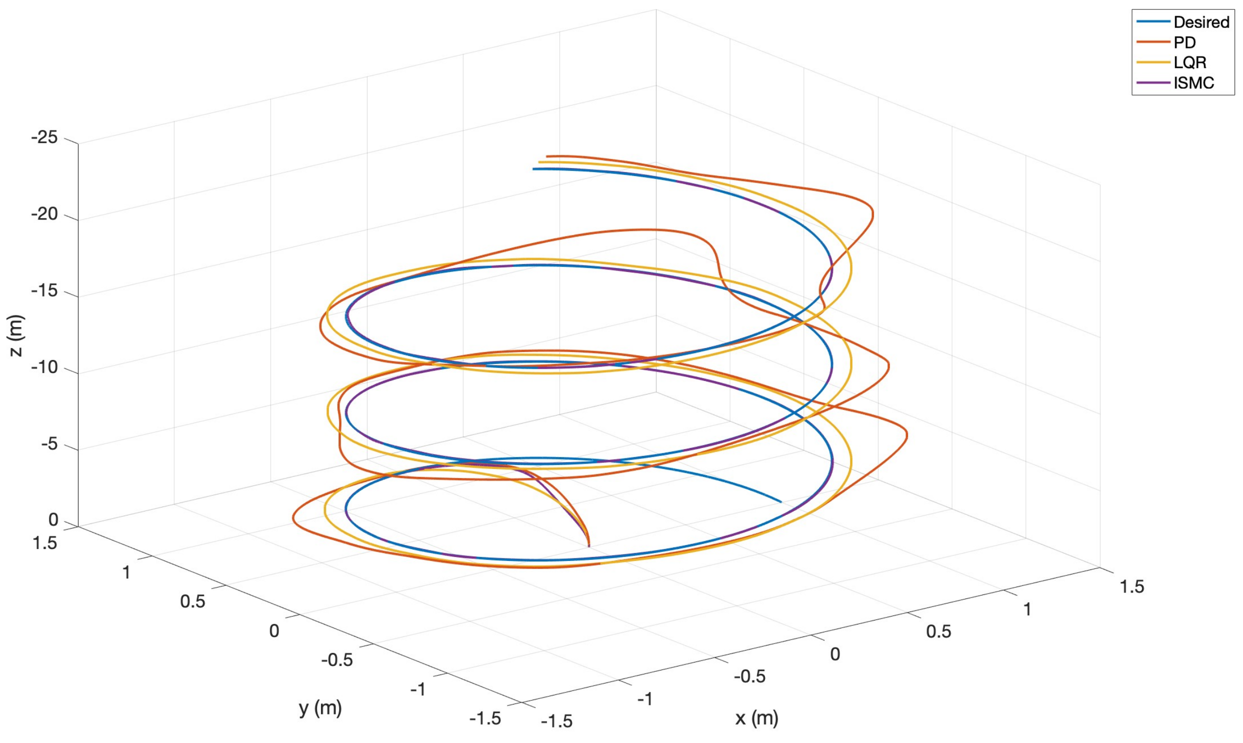


Figure 8.24: 3D view of the ascent helix trajectory responses under Scenario 2.

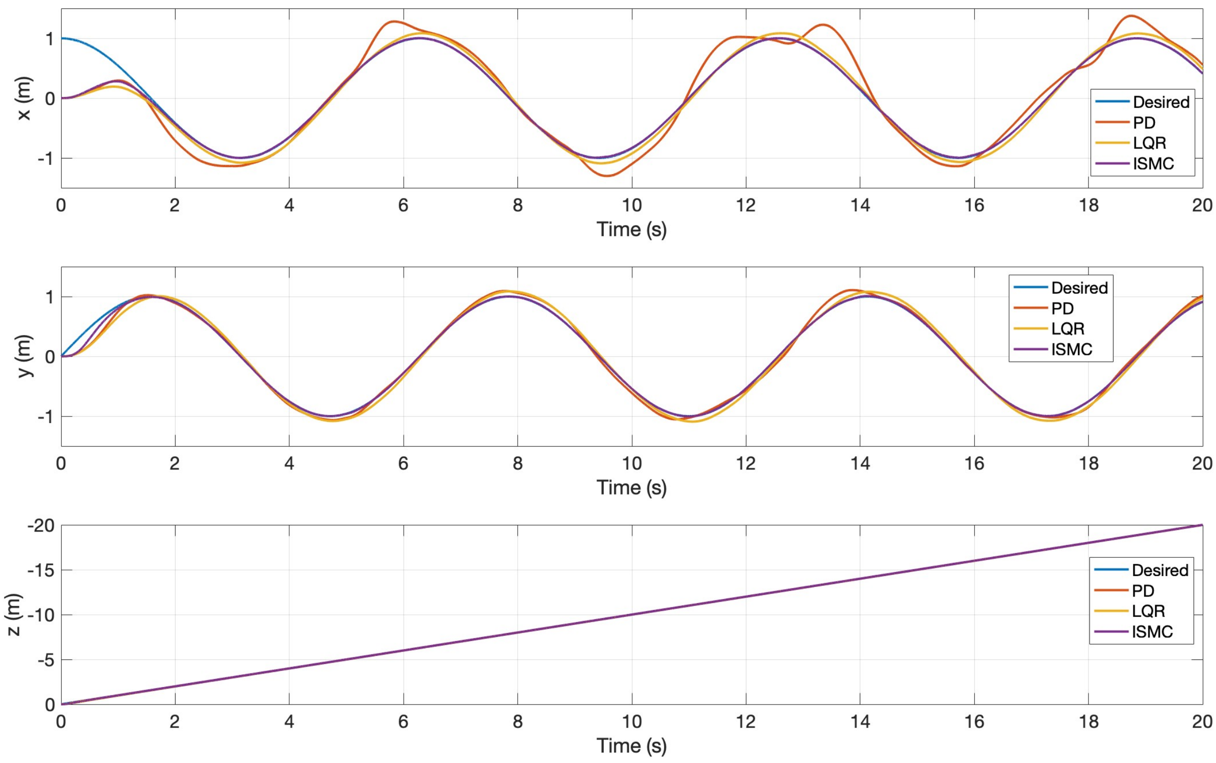


Figure 8.25: Position tracking of the ascent helix trajectory under Scenario 2.

		PD	LQR	ISMC
x	IAE	3.7837	1.9577	0.8627
	ISE	1.3563	0.7017	0.5650
	ITAE	32.7994	12.1865	1.2335
	ITSE	8.7994	1.0913	0.2061
y	IAE	1.0579	1.3044	0.1870
	ISE	0.1120	0.1362	0.0212
	ITAE	9.4875	11.3497	0.5699
	ITSE	0.7783	0.8383	0.0107
z	IAE	0.0181	0.1808	0.0306
	ISE	0.00049565	0.0028	0.00075739
	ITAE	0.0489	1.3728	0.1169
	ITSE	0.000093477	0.0100	0.00024285

Table 8.14: Performance indices for the ascent helix trajectory under Scenario 2.

Complex Ascent Helix

The complex ascent helix is showcased in Figure 8.26, which was set as the desired trajectory for the quadrotor to follow.

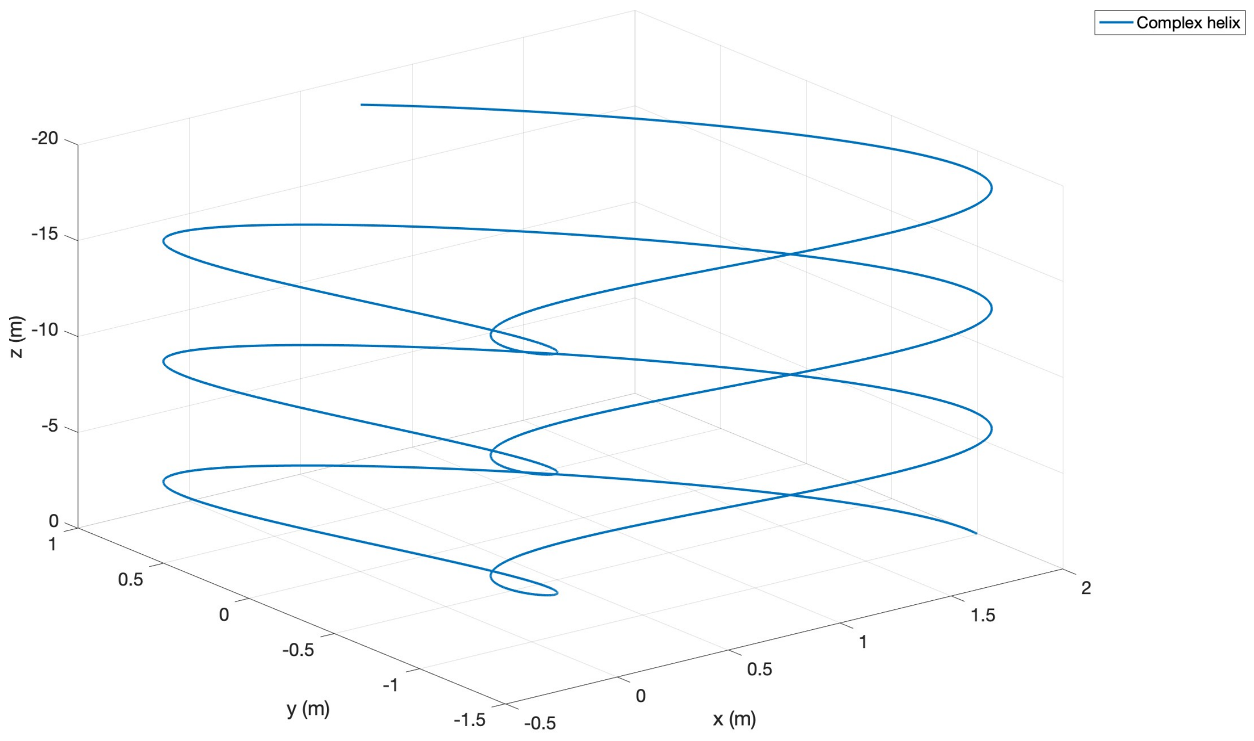


Figure 8.26: Complex ascent helix.

The flight of the quadrotor in 3D space for the case of the complex ascent helix trajectory under Scenario 1 is depicted in Figure 8.27, while the response for the position is shown in Figure 8.28, and the performance indices are listed in Table 8.15. From the results it can be seen that the ISMC is still able to follow the desired trajectory very closely with minimal error. The PD and LQR does struggle with more complex trajectories and has considerably worse performance than the ISMC. This shows that the proposed ISMC is able to effectively control the quadrotor during more complex maneuvers than the PD and LQR controllers.

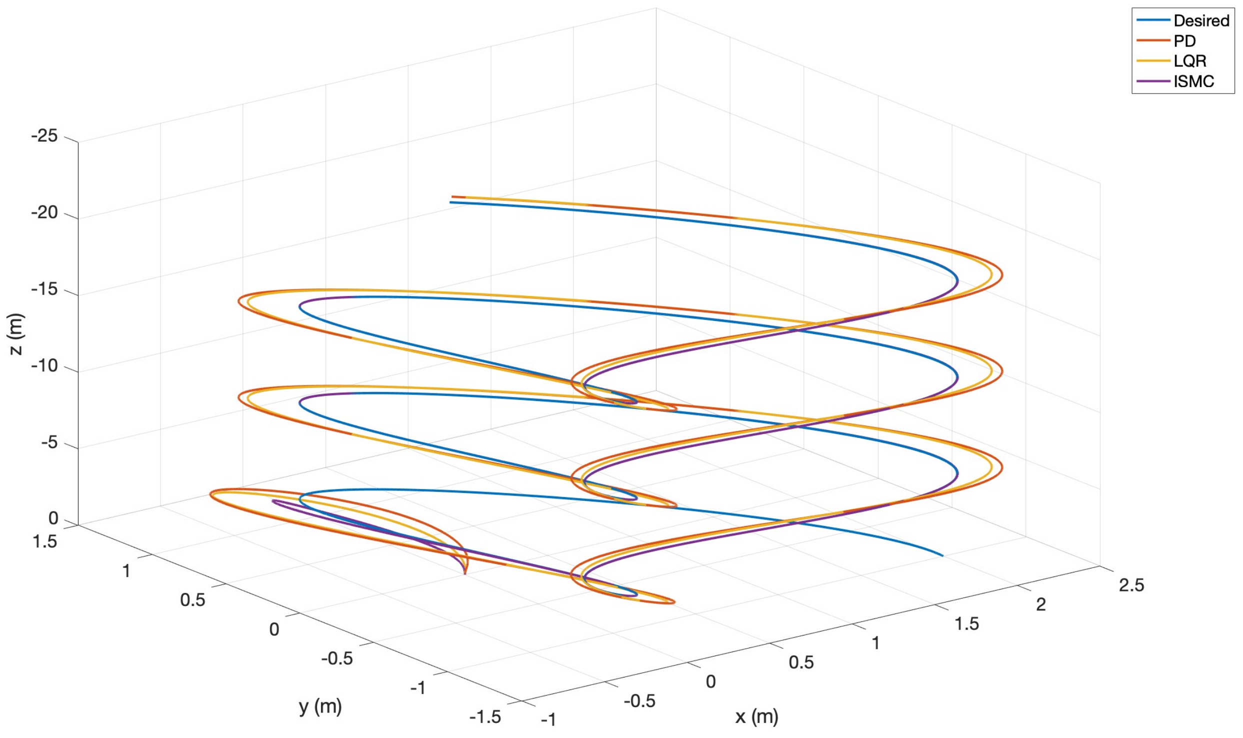


Figure 8.27: 3D view of the complex ascent helix trajectory responses under Scenario 1.

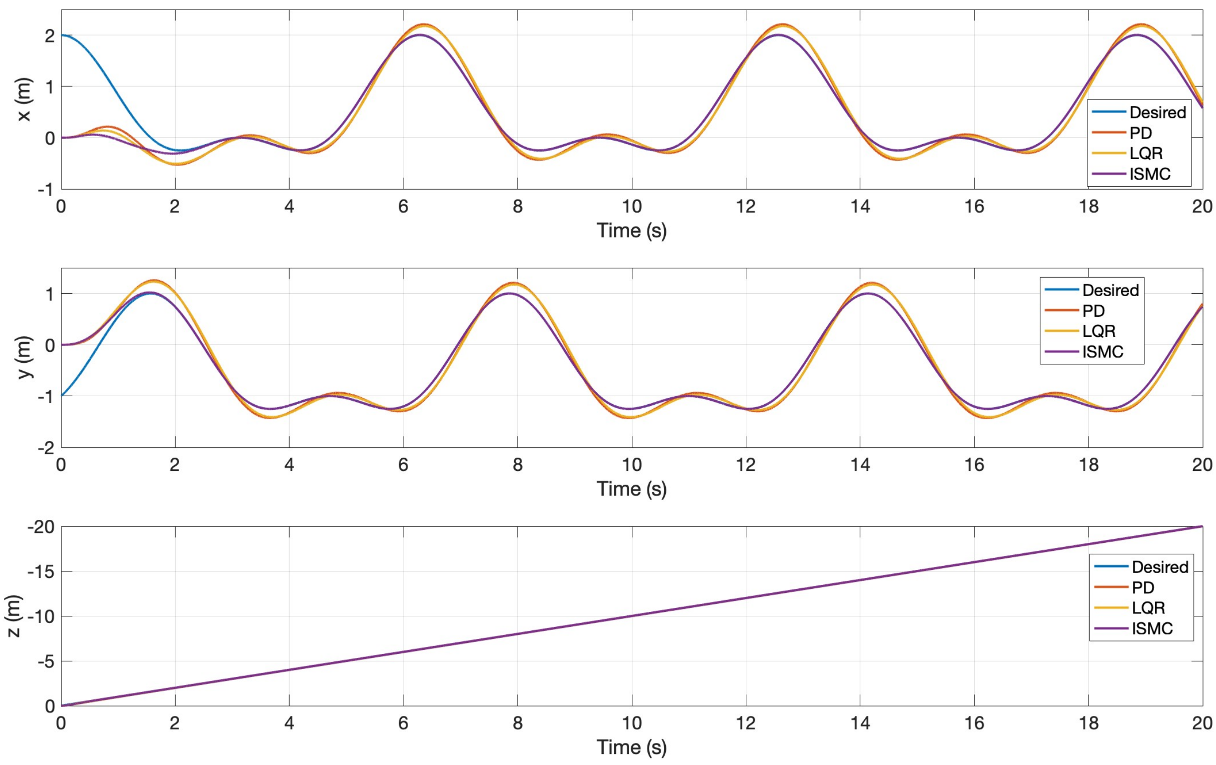


Figure 8.28: Position tracking of the complex ascent helix trajectory under Scenario 1.

		PD	LQR	ISMC
x	IAE	3.9736	3.9374	1.9631
	ISE	2.8083	2.9038	2.7028
	ITAE	25.0974	23.9386	1.6263
	ITSE	4.8703	4.6747	1.1795
y	IAE	2.6802	2.5912	0.5231
	ISE	0.6464	0.6302	0.3064
	ITAE	21.6706	20.5567	0.3121
	ITSE	3.2196	2.9659	0.0747
z	IAE	0.0188	0.1816	0.0186
	ISE	0.00044661	0.0027	0.00067024
	ITAE	0.0617	1.4048	0.0057
	ITSE	0.000090018	0.0104	0.00012374

Table 8.15: Performance indices for the complex ascent helix trajectory under Scenario 1.

For Scenario 2, the flight of the quadrotor in 3D space for the case of the complex ascent helix trajectory under Scenario 1 is depicted in Figure 8.29, the response for the position is shown in Figure 8.30, and the performance indices are listed in Table 8.16. Under the presence of model uncertainties and disturbances, the PD controller fails to accurately track the desired trajectory, resulting in divergence starting at approximately $t = 13$ s. The ISMC and LQR is still relatively unaffected by the added uncertainties and disturbances, and has a minimal overall changes in performance compared to Scenario 1.

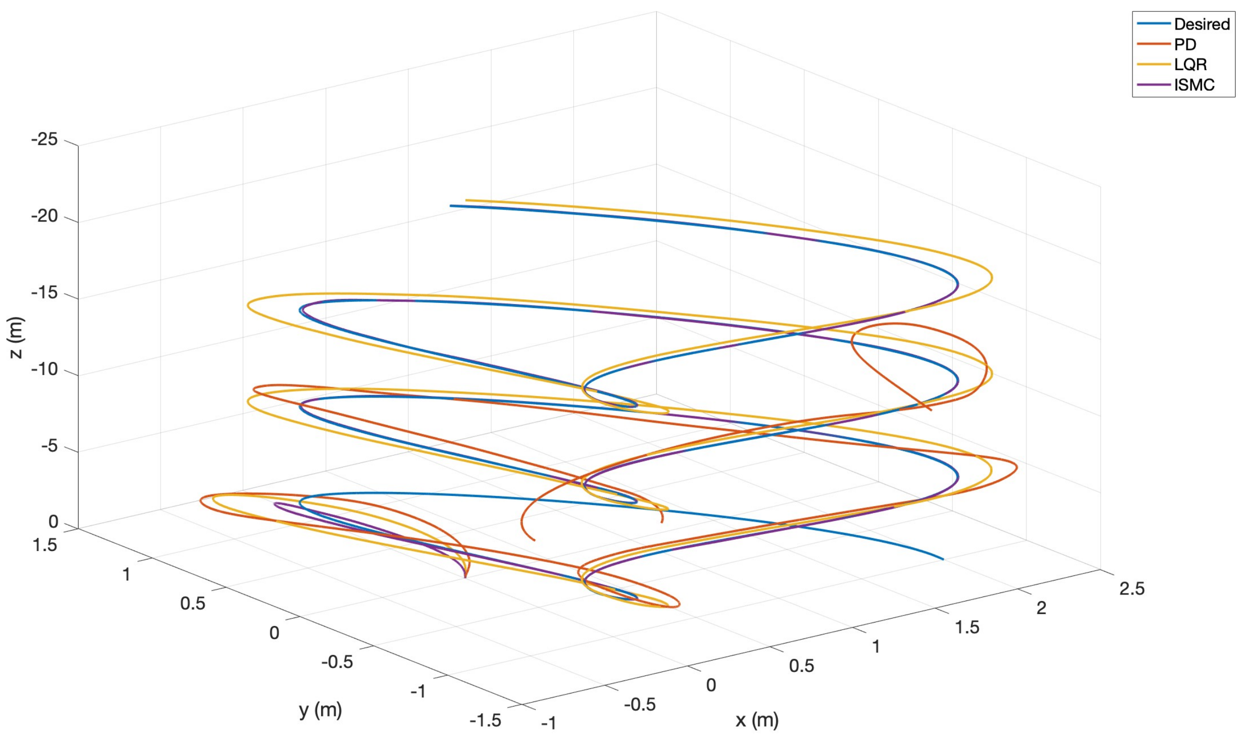


Figure 8.29: 3D view of the complex ascent helix trajectory responses under Scenario 2.

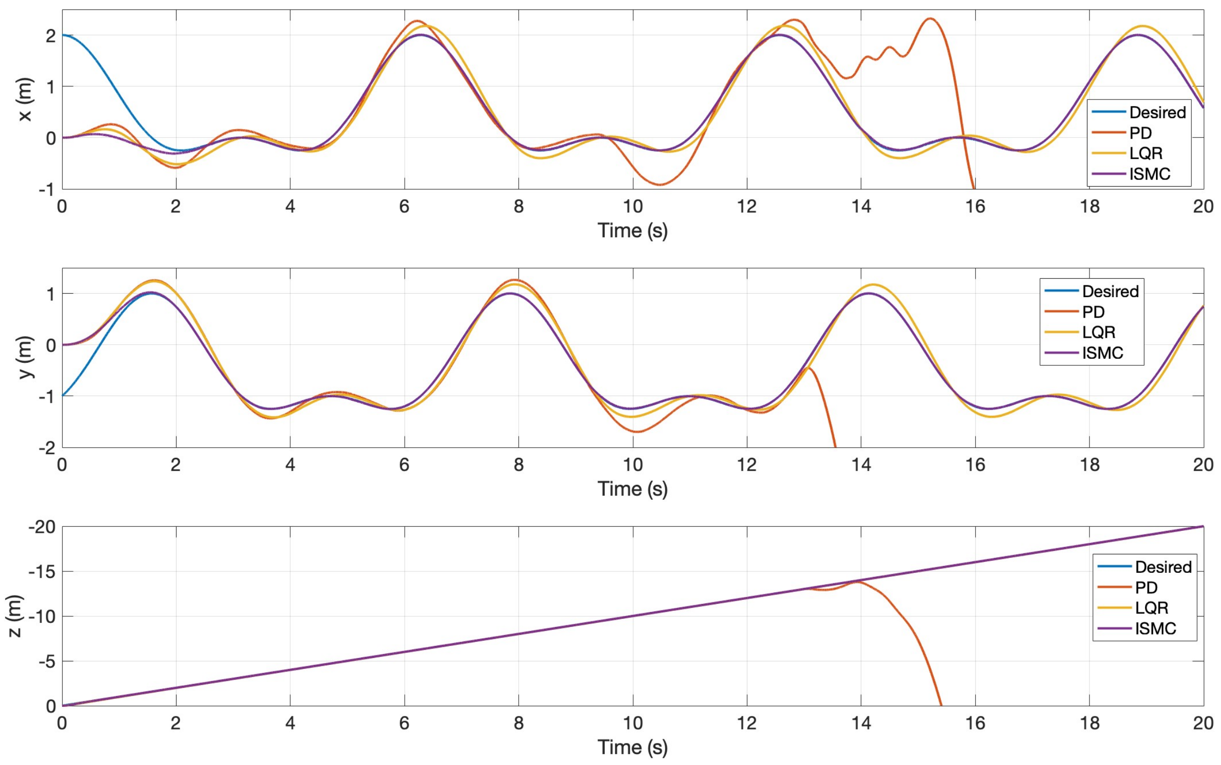


Figure 8.30: Position tracking of the complex ascent helix trajectory under Scenario 2.

		PD	LQR	ISMC
x	IAE	110.9979	3.8987	1.9716
	ISE	3893.3	2.8441	2.6563
	ITAE	2017.1	23.7450	2.0579
	ITSE	74210	4.5762	1.1482
y	IAE	85.1106	2.6316	0.5665
	ISE	1211.5	0.6507	0.3161
	ITAE	1455.8	20.7458	0.6489
	ITSE	21844	3.0328	0.0811
z	IAE	523.7911	0.1841	0.0306
	ISE	77844	0.0028	0.00075814
	ITAE	9649.1	1.4093	0.1170
	ITSE	1472100	0.0105	0.00024308

Table 8.16: Performance indices for the complex ascent helix trajectory under Scenario 2.

8.1.3 Analysis and Discussion

The simulations aimed to evaluate the effectiveness of the proposed control system architectures for altitude, attitude and 3D trajectory tracking of a quadrotor, both in the presence and absence of uncertainties and disturbances. The results clearly establish the ISMC controller as the superior choice, demonstrating it's capability to accurately track all the proposed 3D trajectories with minimal errors without exceeding the input saturation limits. Comparatively, the PD controller exhibits similar performance to the LQR controller when uncertainties and disturbances are absent. However, the PD controller's lack of robustness becomes evident when subjected to uncertainties and disturbances, resulting in a significantly impacted response.

8.2 3 DOF Hover

To evaluate the effectiveness of the proposed controllers in real-world scenarios, a laboratory experiment utilizing the 3 DOF Hover system was conducted. In this section, for the experimental testing and analysis of the physical 3 DOF Hover system, degrees are utilized to represent the results instead of radians. All the controllers were tuned by trial and error for a fast response with minimal overshoot and oscillations. Table 8.17, 8.18 and 8.19 show the controller parameters used for PD, LQR and ISMC respectively.

i	k_{pi}	k_{di}
ϕ	30	10
θ	30	10
ψ	25	5

Table 8.17: PD parameters.

Controller	Q matrices	R matrices
ϕ	$\begin{bmatrix} 950 & 0 \\ 0 & 150 \end{bmatrix}$	$\begin{bmatrix} 0.1 \end{bmatrix}$
θ	$\begin{bmatrix} 950 & 0 \\ 0 & 150 \end{bmatrix}$	$\begin{bmatrix} 0.1 \end{bmatrix}$
ψ	$\begin{bmatrix} 1000 & 0 \\ 0 & 100 \end{bmatrix}$	$\begin{bmatrix} 0.1 \end{bmatrix}$

Table 8.18: LQR weighting matrices.

i	λ_i	ρ_i	ζ_i	k_i
ϕ	3	40	35	0.001
θ	3	40	35	0.001
ψ	10	5	2	0.001

Table 8.19: ISMC parameters.

The small scalar constant in the sign approximation function in the ISMC control inputs was chosen as $\epsilon = 0.05$.

A total of four experiments were conducted to evaluate the performance of the PD, LQR, and ISMC controllers for the 3 DOF Hover system. These experiments consisted of a range of different reference signals to test the controllers under different conditions. In these experiments cases closer to the linearization region of the LQR controller are tested, as well as further away from the linearization region due to reference signals with larger amplitudes. Initially, for roll and pitch, a reference signal consisting of a step signal followed by a sinusoidal signal, $\sin(t)$, was tested. The step signal had a step time at $t = 2$ s, where it transitioned to a sinusoidal signal at $t = \frac{5}{2}\pi \approx 7.85$ s. Simultaneously, a ramp reference was set for the yaw angle from $t = 15$ s. The amplitude for the roll and pitch angle was first set as 16°. Then a second test was done with an amplitude of 28°. The slope of the ramp signal for the yaw angle was first set as 8°, followed by 14° for the second test.

To assess the response during more aggressive maneuvers with varying frequencies, a linear chirp signal with an amplitude of 16° was used as the reference for roll and pitch, followed by a test with an amplitude of 28°. The linear chirp signal is a sinusoidal signal with a frequency that varies linearly with time. The linear chirp signal is defined as shown under

$$\mathcal{C}(t) = \sin\left(\beta_0 + 2\pi\left(\frac{c}{2}t^2 + f_0 t\right)\right) \tag{8.7}$$

where β_0 is the the initial phase, f_0 is the starting frequency, and c is the chirp rate given by

$$c = \frac{f_1 - f_0}{T} \tag{8.8}$$

where f_1 is the final frequency and T is the time it takes to go from f_0 to f_1. For the experimental tests the chirp signal was set as

$$\mathcal{C}(t) = A\sin(0.05t^2 + t)$$

where A is the amplitude. A ramp reference was set for the yaw angle, with a slope of 16° when the amplitude of the chirp signal was set to 16°, and a slope of 28° was set when the amplitude of the chirp signal was set to 28°.

Each experiment was performed and repeated five times with each controller to account for potential variations, except for the chirp signal with an amplitude of 28°, which was carried out once to mitigate any potential harm to the 3 DOF Hover system. The performance indicators IAE, ISE, ITAE, and ITSE were selected to evaluate and compare the performance of the different controllers. The average values were calculated when multiple tests were conducted to ensure accurate assessment. All the experimental results, including the variations observed in each test, are presented in Appendix C, offering insights into the performance and variations encountered throughout the experiments. The figures in this section depict the results of the middle test (no. 3) out of the five total tests conducted for each controller.

Step-to-Sinusoidal Reference

Figure 8.31 shows the attitude response of the 3 DOF Hover with the proposed controllers, with a step-to-sinusoidal with an amplitude of 16° as a reference for the pitch and roll angles. Table 8.20 shows the corresponding average performance indices. All the controllers effectively stabilize the attitude of the system with minor errors. However, it should be noted that the PD and LQR controllers exhibit some more oscillations in the roll and pitch angles than ISMC. The performance indices show that ISMC performs the best with considering roll and pitch, while there are minor changes in performance for the yaw angle.

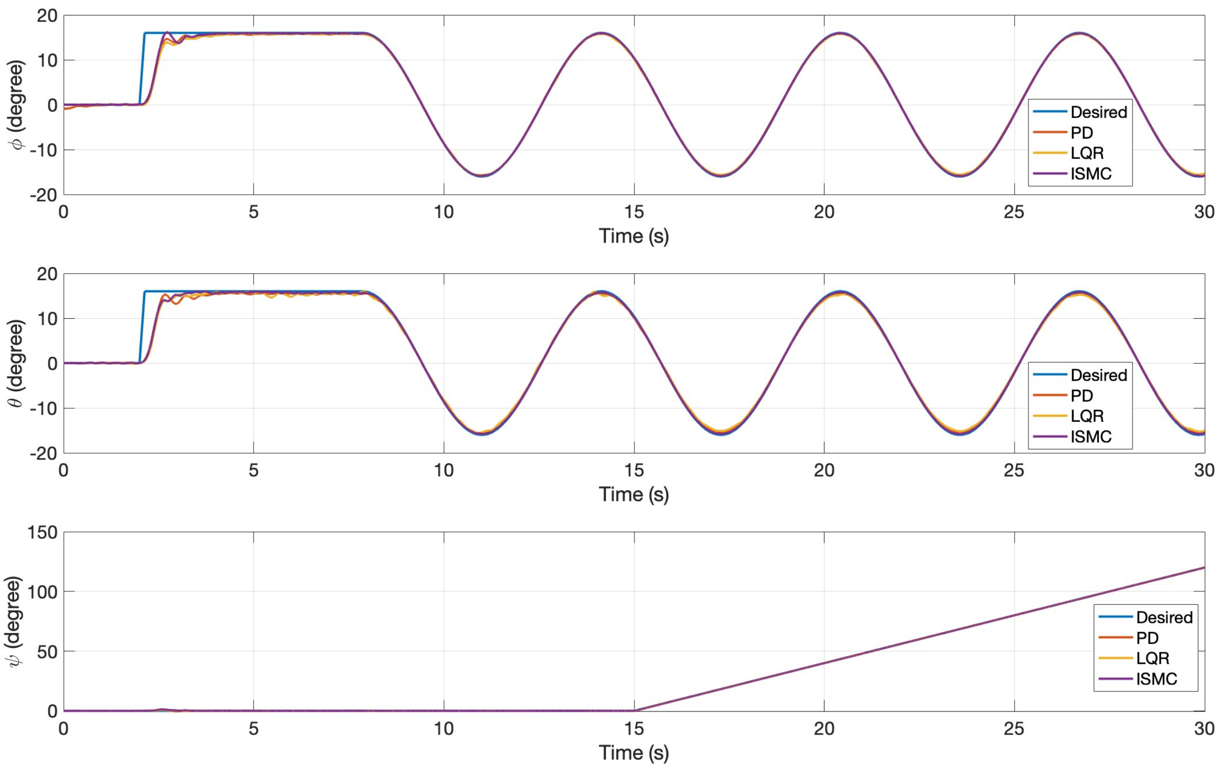


Figure 8.31: 3 DOF Hover attitude response for a step-to-sinusoidal reference with an amplitude of 16°.

		PD	LQR	ISMC
ϕ	IAE	10.8459	13.4355	8.9474
	ISE	67.7305	73.5433	66.4632
	ITAE	73.3021	109.4957	48.6243
	ITSE	161.0616	189.7740	152.8323
θ	IAE	16.0131	20.0373	10.7909
	ISE	69.2576	74.0389	61.2498
	ITAE	162.5766	225.4345	84.4049
	ITSE	208.4243	276.1837	149.8759
ψ	IAE	1.4070	1.3138	1.4467
	ISE	0.2645	0.1549	0.4537
	ITAE	16.2772	15.9770	14.2226
	ITSE	1.4244	1.1738	1.7563

Table 8.20: Average performance indices for a step-to-sinusoidal reference with an amplitude of 16°.

Figure 8.32 shows the attitude response of the 3 DOF Hover, with an amplitude of 28°. Table 8.21 shows the corresponding average performance indices. Notably, the ISMC controller continues to exhibit better performance in terms of the roll and pitch angles. However, it shows a slightly inferior performance in the yaw angle. This could be attributed to the observed coupling effect at approximately $t = 3$ s, which also was observed in the first experiment. This could explain the larger ISE for the two experiments, compared to the PD and LQR. The larger coupling affect could be attributed to the fact that the rotational velocities incorporated in the control inputs for the ISMC are not measured directly, but estimated with a derivative block and a second order filter.

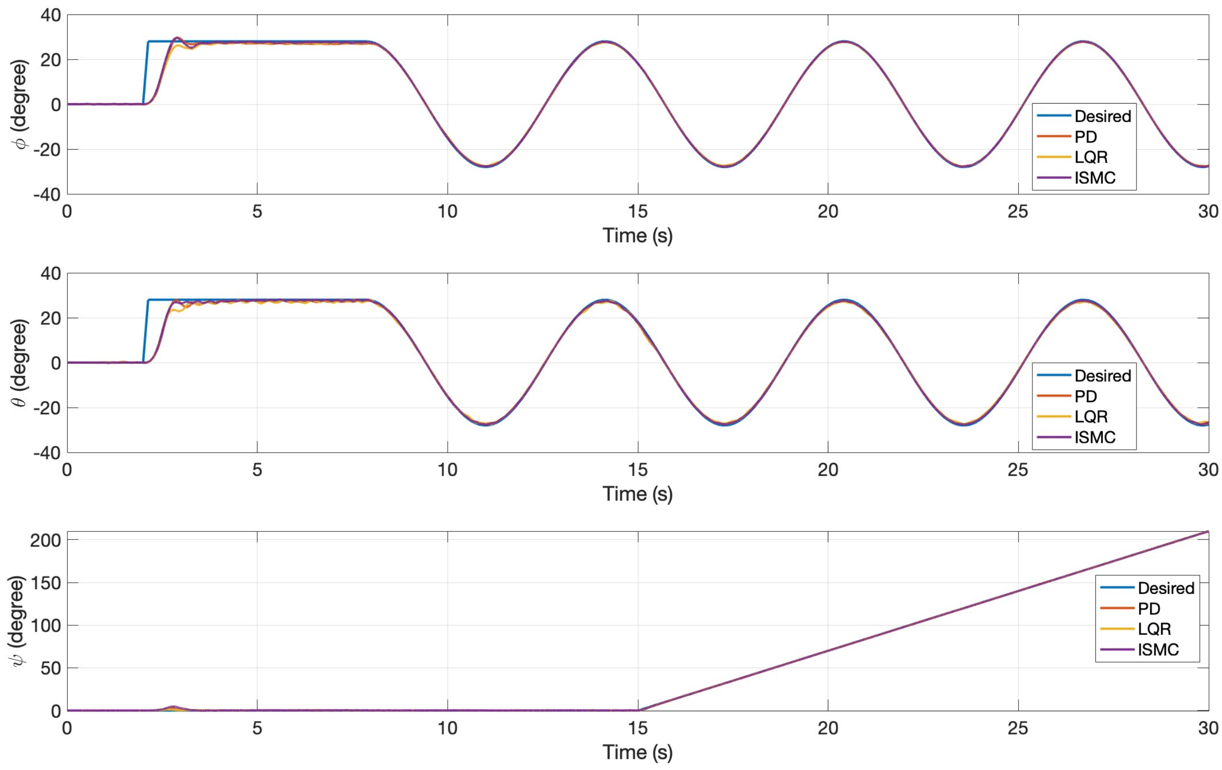


Figure 8.32: 3 DOF Hover attitude response for a step-to-sinusoidal reference with an amplitude of 28°.

		PD	LQR	ISMC
ϕ	IAE	23.0869	24.6633	18.4331
	ISE	262.9998	271.2178	256.0416
	ITAE	148.0226	169.5198	98.1211
	ITSE	641.3642	671.5906	597.4875
θ	IAE	26.8358	30.6957	22.3255
	ISE	266.6785	262.6389	254.1412
	ITAE	231.8864	298.9329	173.1427
	ITSE	709.0085	783.6478	638.7743
ψ	IAE	3.9302	3.2149	4.4978
	ISE	2.6912	0.9186	6.6754
	ITAE	37.4714	37.8273	36.2429
	ITSE	11.5212	6.9293	22.8500

Table 8.21: Average performance indices for a step-to-sinusoidal reference with an amplitude of 28°.

Linear Chirp Reference

Figure 8.33 depicts the attitude response of the 3 DOF Hover using the proposed controllers, with a linear chirp signal of 16° amplitude serving as the reference for the pitch and roll angles. Table 8.22 shows the corresponding average performance indices. The performance superiority of the ISMC controller becomes evident as it outperforms the PD and LQR controllers across all Euler angles. Notably, the PD and LQR controllers struggle to track the increasing frequency starting from approximately $t = 27$ s, resulting in big overshoots. The ISMC controller exhibits superior capability in following the higher frequencies.

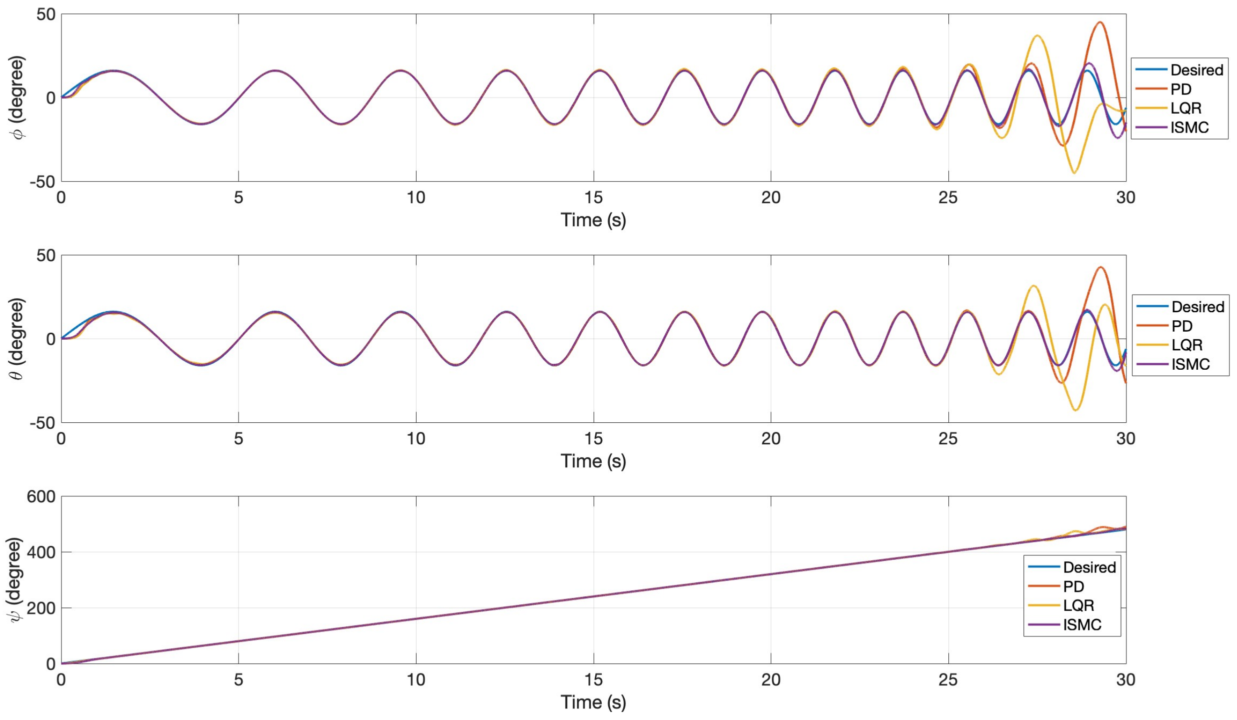


Figure 8.33: 3 DOF Hover attitude response for a linear chirp reference with an amplitude of 16°.

		PD	LQR	ISMC
ϕ	IAE	50.3605	75.5110	14.4941
	ISE	961.7991	1697.7	59.0408
	ITAE	1325.1	1976.9	333.5234
	ITSE	27847	48143	1619.8
θ	IAE	39.5178	63.0323	8.6760
	ISE	760.9422	1331	15.9845
	ITAE	977.8385	1585.9	105.7166
	ITSE	21895	37639	141.2120
ψ	IAE	16.5686	18.2073	7.4964
	ISE	143.8428	116.5188	15.2036
	ITAE	425.7416	462.6718	135.5470
	ITSE	4152.9	3296.7	265.4976

Table 8.22: Average performance indices for a linear chirp reference with an amplitude of 16°.

Figure 8.34 depicts the attitude response of the 3 DOF Hover using the proposed controllers, with a linear chirp signal of 28° amplitude serving as the reference for the pitch and roll angles. This experiment was conducted only once, and the performance indices are presented in two separate tables. Table 8.23 displays the corresponding performance indices for the first 20 seconds, while Table 8.24 provides the performance indices for the entire 30 second duration. The decision to split the performance indices into two separate tables was motivated by the observed behavior of the system. After approximately 20 seconds, the angles of the 3 DOF Hover started to exhibit overshooting and approached the physical boundaries of the system. Therefore, this allows evaluating and analyzing the performance of the controllers separately for the initial 20 second period and the entire 30 second duration.

Again, the ISMC shows superiority in handling the higher frequencies and larger angles showing considerably better performance indices across all the Euler angles for both the 30 and 20 second duration, except for the yaw angle for the 30 second duration where the LQR performed the best. The PD and LQR show similar performance in terms of the performance indices.

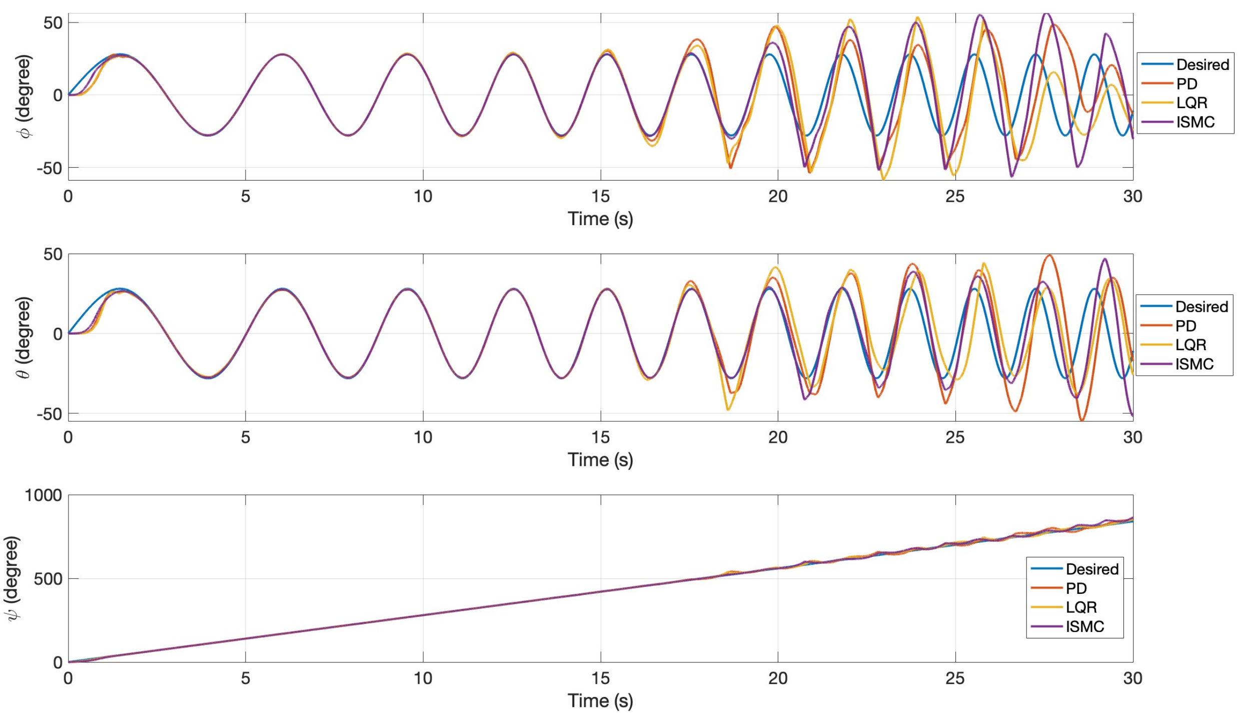


Figure 8.34: 3 DOF Hover attitude response for a linear chirp reference with an amplitude of 28°.

		PD	LQR	ISMC
ϕ	IAE	43.9914	43.0608	12.9492
	ISE	505.3881	409.3270	61.4372
	ITAE	636.1244	597.5513	120.2271
	ITSE	8107.7	6252.7	599.2716
θ	IAE	28.6646	32.8136	14.0225
	ISE	202.7386	305.2860	82.4638
	ITAE	295.8572	343.9586	53.8028
	ITSE	1821.2	3412	69.2422
ψ	IAE	17.7076	15.2074	10.2952
	ISE	112.8878	84.6662	40.7732
	ITAE	241.2167	195.1691	71.4903
	ITSE	1789.9	1328.5	89.2532

Table 8.23: Performance indices for a linear chirp reference with an amplitude of 28° for the first 20 seconds.

		PD	LQR	ISMC
ϕ	IAE	296.2097	309.6168	239.4952
	ISE	9072.9	9496.3	7330.8
	ITAE	7104.5	7296.4	6015.4
	ITSE	233540	237030	195260
θ	IAE	233.5727	229.5657	128.3496
	ISE	7034.8	5526.1	2482.9
	ITAE	5695.7	5389.7	3116.8
	ITSE	189210	140740	66989
ψ	IAE	99.3213	73.6205	84.0729
	ISE	1080	615.0246	926.4874
	ITAE	2334.6	1672.5	1996.1
	ITSE	27033	14745	23873

Table 8.24: Performance indices for a linear chirp reference with an amplitude of 28°.

8.2.1 Analysis and discussion

In these experiments, our focus was on assessing the controllers ability to accurately control the attitude of the 3 DOF Hover system. The results clearly demonstrate that the ISMC controller outperforms both the PD and LQR controllers, particularly when dealing with larger angles and more aggressive maneuvers. It is important to note that the tuning process for the ISMC controller was significantly more time-consuming compared to the PD and LQR controllers. The tuning of the ISMC controller proved to be a more complex process compared to the other controllers.

8.3 Line Tracking

8.3.1 Test Setup

The line tracking algorithm was tested with both the ISMC designed in this thesis and the PID-based controller developed by MathWorks that comes with the Parrot Minidrone Competition Project.

The parameters for the ISMC were obtained using GA, where the control parameters were optimized for tracking a fixed altitude and ramp references for the x- and y-position. The yaw angle is not actively utilized in the line tracking algorithm and has been tuned by trial and error to maintain and stabilize a fixed value of zero radians, which resulted in an acceptable performance. The obtained control parameters are shown in Table 8.25.

i	λ_i	ρ_i	ζ_i	k_i	ϵ_i
x	0.6766	0.2145	1.8376	0.01	0.3430
y	0.6766	0.2145	1.8376	0.01	0.3430
z	5.6479	5.3435	3.1581	0.01	1.4904
ϕ	0.8100	26.3578	23.7246	10.1140	1.5242
θ	1.1739	2.3434	22.2347	13.7415	1.3286
ψ	10	5	2	0.01	0.5

Table 8.25: ISMC parameters for the minidrone competition.

Here ϵ_x, ϵ_y, ϵ_z, ϵ_ϕ, ϵ_θ and ϵ_ψ denote the scalar constants in the continuous sign approximation function in the different controller inputs.

The algorithms were tested on three different tracks of increasing complexity. The tracks were built using the Track Builder that comes with the Parrot Minidrone Competition Project. The tracks are shown in Figure 8.35, 8.36 and 8.37.

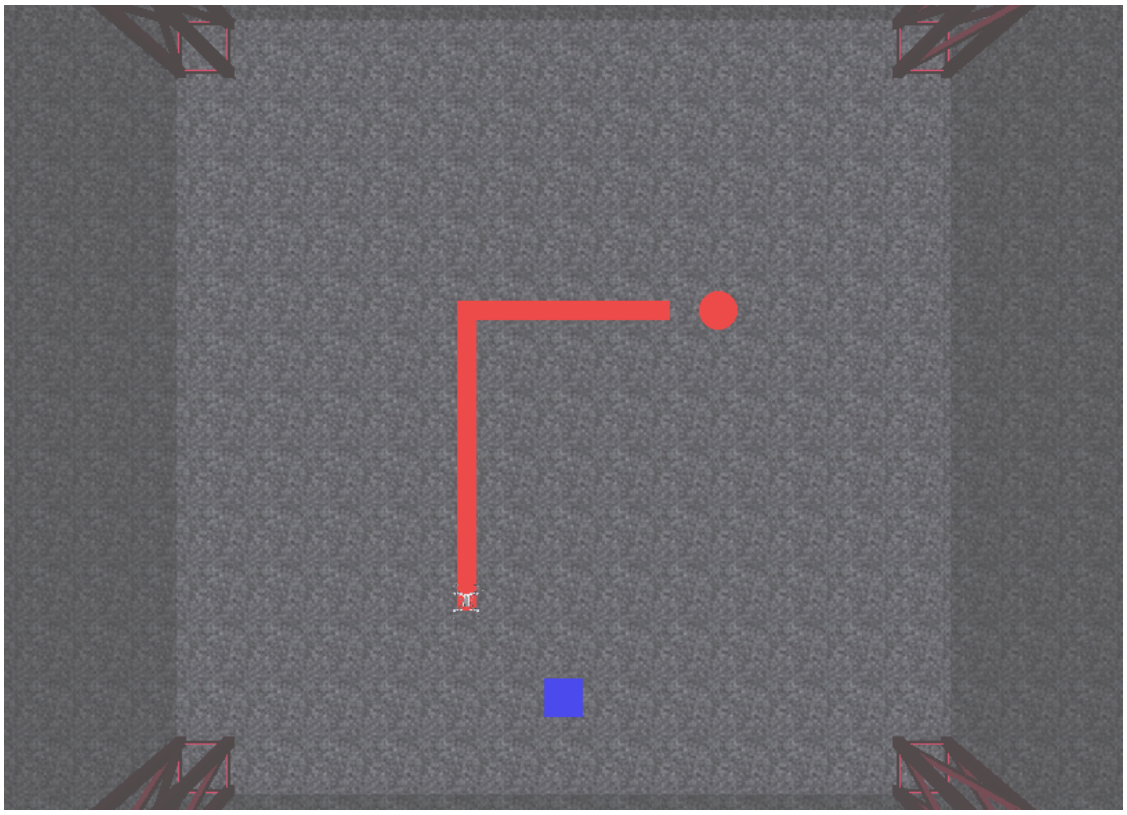


Figure 8.35: The first test track.

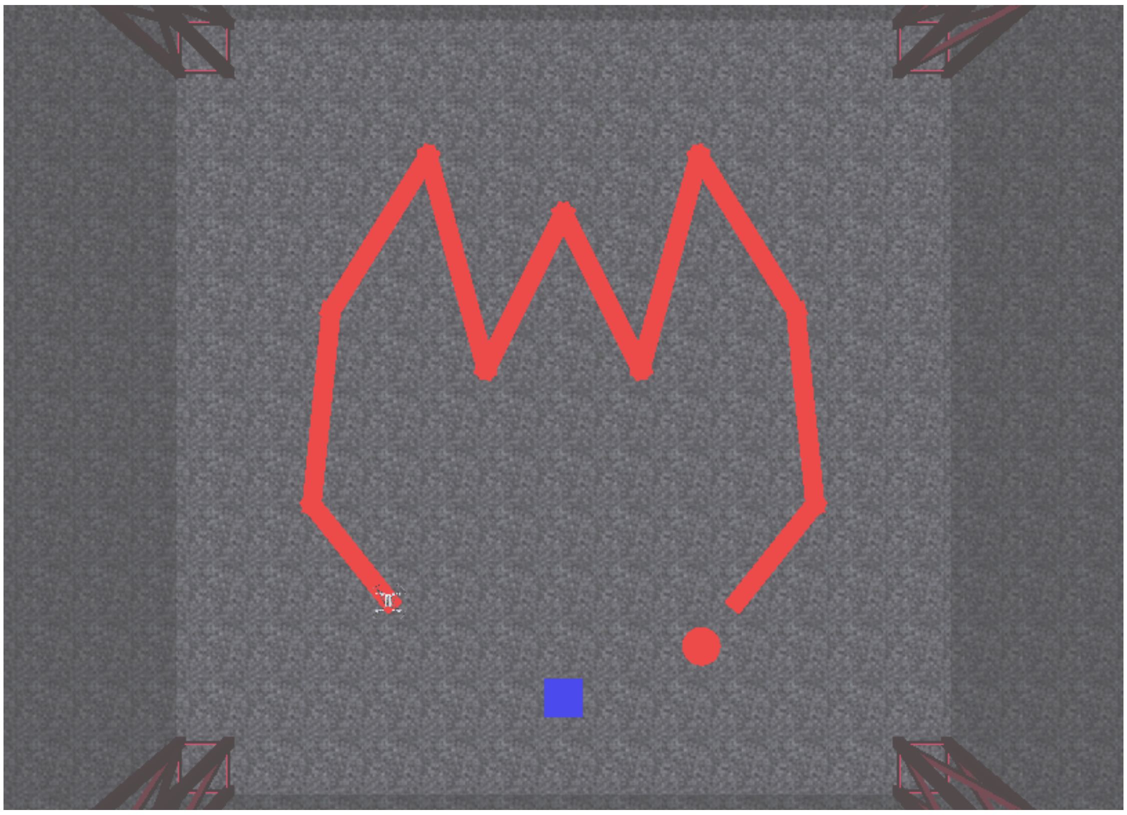


Figure 8.36: The second test track.

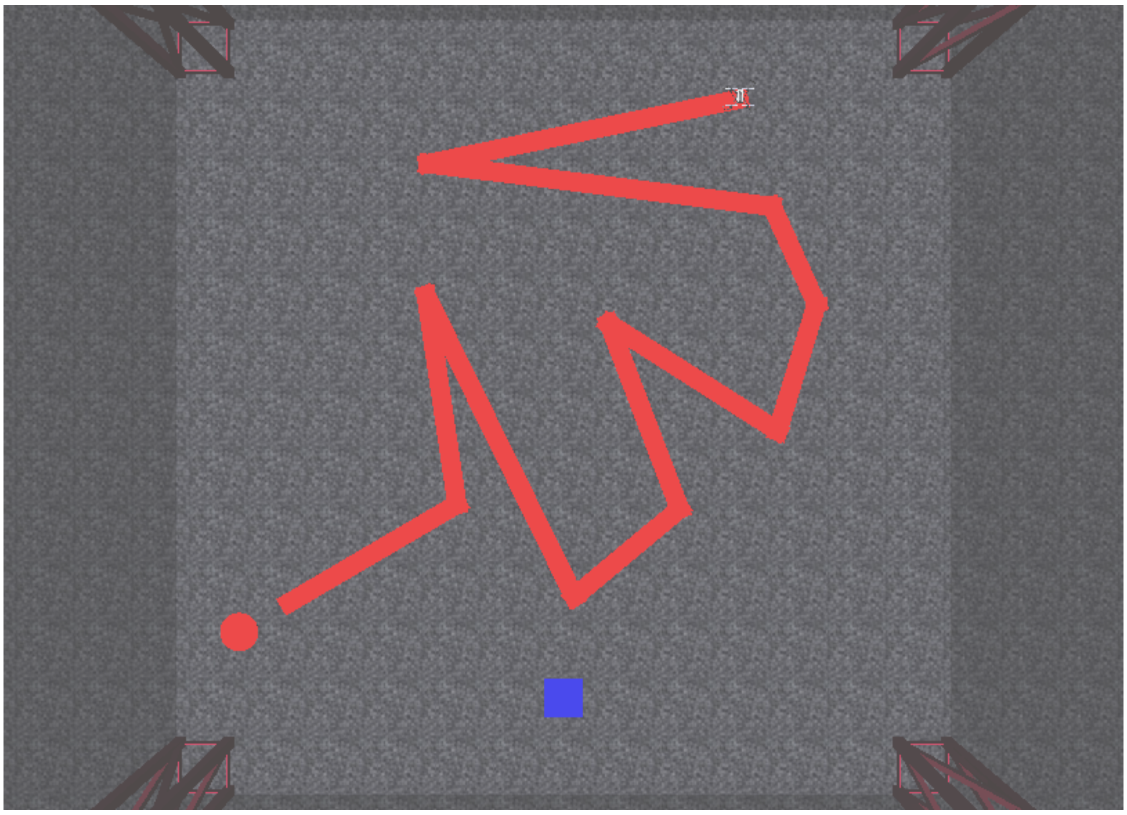


Figure 8.37: The third test track.

Preliminary tests showed that the PID-based controller did not work very well when the drone was allowed to run at high speeds. The decision was then made to test the line tracking algorithm and controllers with several different maximum speeds, shown in Table 8.26.

Max speed (m/sample)	Max speed (m/s)
0.0007	0.14
0.0008	0.16
0.0009	0.18
0.0010	0.20
0.0011	0.22

Table 8.26: Maximum speeds used for testing.

The two simplest metrics used to analyze the algorithms were the track completion time, or "run time" for short, and whether the track was completed or not. Following the track until the end and landing on the circle was considered a success. If the minidrone at any point crashed or started heading away from the track or in the direction it came from, or if other similar events occurred, the simulation was stopped and the attempt was considered a failure.

The performance metrics IAE and ISE were used to analyze the ability of the controllers to follow the reference signals. The reference signals were in this case compared to the values coming from the State Estimator block of the control system (see Figure 7.2). The simulations were carried out with added sensor noise which explains the noise that can be seen in the resulting figures. The IAE- and ISE-values have also been divided by the track completion times in order to give a more fair comparison. IAE and ISE have been computed for the linear positions x, y and z and the angular positions ϕ, θ and ψ.

The distance from the center of the image to the nearest white pixel has been used as a measure of the accuracy of the line tracking algorithm. As explained in Chapter 7.5, because of an even number of pixels in both the x- and y-direction, the distance must be calculated from an "artificially" inserted origin. The distance has been approximately converted from number of pixels to metres. IAE and ISE have also been computed for this distance. Naturally, these values are usually nonzero, either because of the gap between the last track segment and the circle or because of the case where the algorithm fails to complete the track.

As mentioned in Chapter 7.6 (see Figure 7.17), the back sensors regularly overlap with the track segment pixels when the path forward has not yet become stable. The amount of time these regions overlap with at least one pixel, dubbed "sensor time", has therefore been used as a measure of instability. This value has also been divided by the track completion times for better comparison. There will always be some overlap with the track because of the turns, so these values are usually nonzero.

8.3.2 Results

All of the test results can be found in Appendix D.1, but some of the results are repeated here for convenience.

Table 8.27, 8.28 and 8.29 show the first performance index comparisons of the simulations done for track 1, track 2 and track 3 respectively. Table 8.30 and 8.31 show the IAE- and ISE-comparisons for the simulation done on track 1 with max speed set to 0.0011 m/s and track 2 with max speed set to 0.0008 m/sample, respectively.

Using the ISMC-controller, the line tracking algorithm was able to complete the track in every case but one, namely track 3 with the max speed set to 0.0011 m/sample. The PID-controller fared much worse, completing track 1 for each max speed, but failing track 2 for the three highest max speeds tested, and failing track 3 in every case. In each case where they both completed the track, the track was completed quicker when using ISMC. In most of the cases where they both completed the track the minidrone flew more stably with the ISMC.

Because of the interconnectedness between the controller outputs, the minidrone movements, the line tracking and the camera vision, the two controllers are fed with different reference signals, which should be kept in mind when comparing the IAE- and ISE-values for the ISMC and PID. Regardless, ISMC is better than PID at tracking the linear position references it is given throughout the simulation. The yaw reference is held constant at 0 by the line tracker and ISMC is better than PID at keeping yaw stable at this value. When it comes to roll and pitch there is more variation in the results. This could be due to the ISMC-outer loop being more aggressive when it comes to translating the x- and y-references into the required roll and pitch references, resulting in larger reference angles fed by the outer loop control.

Max speed	Controller	Track completed?	Sensor time (s)	Run Time (s)	$\frac{\text{Sensor time}}{\text{Run time}}$ (%)
0.0007	PID	Yes	6.800	31.265	21.7
	ISMC	Yes	6.800	28.530	23.8
0.0008	PID	Yes	8.200	29.080	28.2
	ISMC	Yes	7.600	26.345	28.8
0.0009	PID	Yes	8.200	27.060	30.3
	ISMC	Yes	6.400	24.540	26.1
0.0010	PID	Yes	8.000	25.270	31.7
	ISMC	Yes	6.400	22.940	27.9
0.0011	PID	Yes	9.600	24.670	38.9
	ISMC	Yes	6.200	21.950	28.2

Table 8.27: Performance, track 1.

Max speed	Controller	Track completed?	Sensor time (s)	Run Time (s)	$\frac{\text{Sensor time}}{\text{Run time}}$ (%)
0.0007	PID	Yes	57.200	96.060	59.5
	ISMC	Yes	42.800	80.735	53.0
0.0008	PID	Yes	60.400	88.860	68.0
	ISMC	Yes	39.800	72.730	54.7
0.0009	PID	No	37.200	52.920	70.3
	ISMC	Yes	37.200	65.950	56.4
0.0010	PID	No	14.400	24.180	59.6
	ISMC	Yes	36.400	60.945	59.7
0.0011	PID	No	13.200	22.400	58.9
	ISMC	Yes	35.800	57.535	62.2

Table 8.28: Performance, track 2.

Max speed	Controller	Track completed?	Sensor time (s)	Run Time (s)	$\frac{\text{Sensor time}}{\text{Run time}}$ (%)
0.0007	PID	No	49.800	94.080	52.9
	ISMC	Yes	50.200	98.330	51.1
0.0008	PID	No	21.400	45.120	47.4
	ISMC	Yes	45.400	87.950	51.6
0.0009	PID	No	17.800	41.340	43.1
	ISMC	Yes	42.800	79.740	53.7
0.0010	PID	No	5.600	19.080	29.4
	ISMC	Yes	40.400	73.945	54.6
0.0011	PID	No	4.800	17.520	27.4
	ISMC	No	17.600	34.080	51.6

Table 8.29: Performance, track 3.

					Divided by Run Time	
Max speed	Param.	Index	PID	ISMC	PID	ISMC
0.0011	x	IAE	2.8156	0.41212	0.11413	0.018775
		ISE	0.70452	0.026214	0.028558	0.0011943
	y	IAE	2.3062	0.18653	0.09348	0.0084982
		ISE	0.4772	0.0048261	0.019343	0.00021987
	z	IAE	1.3891	0.9286	0.056308	0.042305
		ISE	0.68886	0.4595	0.027923	0.020934
	ϕ	IAE	0.024493	0.020677	0.00099281	0.000942
		ISE	5.7951e-05	0.00016156	2.3491e-06	7.3605e-06
	θ	IAE	0.027321	0.012163	0.0011075	0.00055413
		ISE	7.6381e-05	4.9065e-05	3.0961e-06	2.2353e-06
	ψ	IAE	0.00045028	0.00032897	1.8252e-05	1.4987e-05
		ISE	1.3039e-08	7.7258e-09	5.2855e-10	3.5197e-10
	d	IAE	0.27656	0.066338	0.01121	0.0030222
		ISE	0.033517	0.0031909	0.0013586	0.00014537

Table 8.30: IAE- and ISE-results for track 2 w/ maximum speed set to 0.0011 m/sample.

					Divided by Run Time	
Max speed	Param.	Index	PID	ISMC	PID	ISMC
0.0008	x	IAE	13.8897	1.8009	0.15631	0.024762
		ISE	2.4463	0.1268	0.027529	0.0017434
	y	IAE	6.0704	0.34267	0.068314	0.0047115
		ISE	0.56271	0.0038305	0.0063326	5.2667e-05
	z	IAE	1.6106	1.0433	0.018125	0.014344
		ISE	0.69387	0.45875	0.0078086	0.0063075
	ϕ	IAE	0.05806	0.057257	0.00065339	0.00078725
		ISE	5.9963e-05	0.0001862	6.748e-07	2.5601e-06
	θ	IAE	0.087471	0.041939	0.00098437	0.00057664
		ISE	0.00025015	0.00022777	2.8151e-06	3.1318e-06
	ψ	IAE	0.0017554	0.0011424	1.9755e-05	1.5707e-05
		ISE	5.4592e-08	2.8238e-08	6.1436e-10	3.8826e-10
	d	IAE	0.23048	0.10147	0.0025938	0.0013952
		ISE	0.020913	0.018695	0.00023535	0.00025705

Table 8.31: IAE- and ISE-results for track 2 w/ maximum speed set to 0.0008 m/sample.

8.3.3 Analysis and Discussion

The simulations that were carried out showed that the line tracking algorithm works much better with ISMC than PID. During turns the line tracker quickly changes the position references to a new direction. The ISMC handles these changes well and does so while the drone is flying at higher speeds. The PID controller generally reacts quite slowly to the turns. In the best case this results in worse track completion times than when ISMC is used. In the worst case it results in the minidrone failing to complete the track.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

In this thesis a nonlinear model of a quadrotor was derived. The main goal of the thesis was to design controllers for the quadrotor UAV, with a main focus on SMC (ISMC), but also designing PID (PD) and LQR for comparison purposes. Simulations were carried out to evaluate the effectiveness of the proposed control system architectures for altitude, attitude and 3D trajectory tracking of a quadrotor, both in the presence and absence of uncertainties and disturbances. ISMC outperformed the other two, accurately tracking all the proposed 3D trajectories with minimal errors without exceeding the input saturation limits. The PD and LQR controllers performed well in the absence of uncertainties and disturbances. However, the ISMC, and to a certain extent the LQR, proved to be more robust than the PD when these uncertainties and disturbances were introduced. Also, the controllers' ability to accurately control attitude were tested on the 3 DOF Hover system. The results clearly demonstrated that the ISMC controller outperforms both the PD and LQR controllers, particularly when dealing with larger angles and more aggressive maneuvers.

A vision-based line tracking algorithm was developed and tested with the Parrot Minidrone project. The algorithm fares reasonably well, especially when pared with ISMC, letting the minidrone handle many turns at a variety of tested speeds. The ISMC showed itself superior when matched against PID, completing more tracks, completing them quicker and tracking the references with less error.

9.2 Future Work

The work presented in this thesis could be further improved by looking into the following aspects:

- Further improving the dynamical model of the quadrotor, including effects such as aerodynamic friction torque and air drag force which resists the quadrotor motion.

- In this work some simplification were made under the assumptions of small angles of movement, in both the quadrotor model and in the control architecture. To further enhance the controllers ability to perform more aggressive maneuvering, additional refinements can be explored.
- Higher-order sliding mode control can be explored, which yields improved performance with respect to chattering effects as well as higher accuracy.
- Measurements uncertainties/perturbations is unavoidable in practice, and a disturbance observer for the estimation of disturbances can be implemented to the sliding mode control to compensate for these uncertainties/perturbations and improve robustness.
- Integrated fault detection and isolation mechanisms, as well as fault-tolerant control strategies can be implemented to enhance the quadrotor's resilience against actuator failures or sensor faults. Redundancy-based approaches or adaptive control techniques can be employed to maintain stability and control performance in the presence of faults.
- Improving the line tracking algorithm. One of its main flaws is the fact that track segments that are not a part of the current turn can come into the camera vision and affect the next direction. Flying closer to the ground could fix this, but the line tracker then reduces its ability to detect turns/corners early. Another suggestion is flying at the same or a higher height, but using a subregion of the image for direction calculations. The algorithm also struggles with sharp $\sim 10^\circ$ turns. If improvements are made to the algorithm, one could try flying at even higher speeds.

Bibliography

[1] MathWorks. Parrot drone support from matlab, 2023. URL https://se.mathworks.com/hardware-support/parrot-drone-matlab.html. Accessed: 2023-04-11.

[2] MathWorks. Introduction to minidrone competition - ifac 2023, 2023. URL https://se.mathworks.com/videos/introduction-to-minidrone-competition-ifac-2023-1679045143904.html. Accessed: 2023-04-11.

[3] 3dof quanser documentation, 2023. URL https://www.quanser.com/products/3-dof-hover/#overview. Accessed: 2023-02-24.

[4] MathWorks. Mamdani and sugeno fuzzy inference systems, 2023. URL https://se.mathworks.com/help/fuzzy/types-of-fuzzy-inference-systems.html. Accessed: 2023-03-29.

[5] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80:8091–8126, 2021.

[6] GeoGebra Team. Geogebra - 2d graphing, 2023. URL https://www.geogebra.org/m/Adc44ZZq. Accessed: 2023-05-01.

[7] Syed Agha Hassnain Mohsan, Muhammad Asghar Khan, Fazal Noor, Insaf Ullah, and Mohammed H. Alsharif. Applications of unmanned aerial vehicles, 2022. URL https://encyclopedia.pub/entry/25512. Accessed: 2023-05-08.

[8] Andrew Zulu and Samuel John. A review of control algorithms for autonomous quadrotors. *arXiv preprint arXiv:1602.02622*, 2016.

[9] MathWorks. Mathworks minidrone competition, 2023. URL https://se.mathworks.com/academia/student-competitions/minidrones/ifac.html. Accessed: 2023-02-24.

[10] MathWorks. Mathworks minidrone competition rules and guidelines, 2021. URL https://se.mathworks.com/content/dam/mathworks/mathworks-dot-com/academia/student-competitions/minidrone-competition/mathworks-minidrone-competition-guidelines.pdf. Accessed: 2023-05-07.

[11] International Federation of Automatic Control. About, 2023. URL https://www.ifac2023.org/about/#section-2. Accessed: 2023-02-24.

[12] Francesco Sabatino. Quadrotor control: modeling, nonlinearcontrol design, and simulation, 2015.

[13] Michael A Johnson and Mohammad H Moradi. *PID control*. Springer, 2005.

[14] Kunwu Zhang, Jicheng Chen, Yufang Chang, and Yang Shi. Ekf-based lqr tracking control of a quadrotor helicopter subject to uncertainties. In *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 5426–5431. IEEE, 2016.

[15] Yuri Shtessel, Christopher Edwards, Leonid Fridman, Arie Levant, et al. *Sliding mode control and observation*, volume 10. Springer, 2014.

[16] WJ Cunningham. An introduction to lyapunov's second method. *Transactions of the American Institute of Electrical Engineers, Part II: Applications and Industry*, 80(6):325–332, 1962.

[17] R DeCarlo and S Zak. A quick introduction to sliding mode control and its applications. *elettronica, Università degli Studi di Cagliari (Department of Electrical and Electronic Engineering-DIEE, University of Cagliari, Cagliari CA, Italy)*, 2008.

[18] Weibing Gao and James C Hung. Variable structure control of nonlinear systems: A new approach. *IEEE transactions on Industrial Electronics*, 40(1):45–55, 1993.

[19] Brahim Brahmi, Mohamed Hamza Laraki, Abdelkrim Brahmi, Maarouf Saad, and Mohammad H Rahman. Improvement of sliding mode controller by using a new adaptive reaching law: Theory and experiment. *ISA transactions*, 97:261–268, 2020.

[20] BiTao Zhang, YouGuo Pi, and Ying Luo. Fractional order sliding-mode control based on parameters auto-tuning for velocity control of permanent magnet synchronous motor. *ISA transactions*, 51(5):649–656, 2012.

[21] Ahmed Eltayeb, Mohd Fuaad Rahmat, MA Mohammed Eltoum, and Mohd Ariffanan Mohd Basri. Adaptive fuzzy gain scheduling sliding mode control for quadrotor uav systems. In *2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*, pages 1–5. IEEE, 2019.

[22] Seyedali Mirjalili. Genetic algorithm. *Evolutionary Algorithms and Neural Networks: Theory and Applications*, pages 43–55, 2019.

[23] Pol Majó Casero. A vision-based line following method for micro air vehicles. B.S. thesis, Universitat Politècnica de Catalunya, 2021.

Appendix A

Quadrotor Simulations

A.1 MATLAB Code for Quadrotor Simulations

```matlab
clear all
clc
%% Vehicle Nonlinear Variables (from competiotion files)
% This file was derived from the work by Peter Corke and Fabian
   Reither.
% Copyright (C) 1993-2015, by Peter I. Corke
%
% This file is part of The Robotics Toolbox for MATLAB (RTB).
%
% http://www.petercorke.com

% Copyright 2013-2017 The MathWorks, Inc.
model = 'Mambo';
rho = 1.1840;
% Physical properties
% Airframe
switch (model)
    case 'Mambo'
        Vehicle.Airframe.mass = 0.063;
        Vehicle.Airframe.inertia = diag([0.0000582857 0.0000716914
   0.0001]);
    case 'RollingSpider'
        Vehicle.Airframe.mass = 0.068;
        Vehicle.Airframe.inertia = diag([0.0686e-3 0.092e-3 0.1366e-3]);
end
Vehicle.Airframe.d = 0.0624;
Vehicle.Airframe.xy = Vehicle.Airframe.d*sqrt(2)/2; % For diamond rotor set-up
Vehicle.Airframe.h = -0.015876;
Vehicle.Airframe.Cdx = 0;
Vehicle.Airframe.Cdy = 0;
Vehicle.Airframe.diameter = .01; % For drag calculation purposes
% Rotor
Vehicle.Rotor.blades = 2;
Vehicle.Rotor.radius = 0.033;
Vehicle.Rotor.chord = .008;
Vehicle.Rotor.flappingOffset = 0;
Vehicle.Rotor.bladeMass = 3.75e-04;
Vehicle.Rotor.bladeInertia = Vehicle.Rotor.bladeMass*Vehicle.Rotor.radius^2/4;
Vehicle.Rotor.hubMass = 0;
Vehicle.Rotor.hubInertia = 0;
Vehicle.Rotor.inertia = Vehicle.Rotor.hubInertia + Vehicle.Rotor.bladeInertia;
Vehicle.Rotor.Ct = .0107;
Vehicle.Rotor.Cq = Vehicle.Rotor.Ct*sqrt(Vehicle.Rotor.Ct/2);
Vehicle.Rotor.solidity = Vehicle.Rotor.chord*Vehicle.Rotor.blades/(pi*Vehicle.Rotor.radius);
Vehicle.Rotor.theta0 = 14.6*(pi/180);
Vehicle.Rotor.thetaTip = 6.8*(pi/180);
Vehicle.Rotor.theta1 = Vehicle.Rotor.thetaTip-Vehicle.Rotor.theta0;
Vehicle.Rotor.theta34 = Vehicle.Rotor.theta0+0.75*Vehicle.Rotor.theta1;
Vehicle.Rotor.a = 5.5; % Lift slope
Vehicle.Rotor.area = pi*Vehicle.Rotor.radius^2;
Vehicle.Rotor.lock = rho*Vehicle.Rotor.a*Vehicle.Rotor.chord*Vehicle.Rotor.radius^4/...
    (Vehicle.Rotor.hubInertia+Vehicle.Rotor.bladeInertia);
Vehicle.Rotor.b = Vehicle.Rotor.Ct*rho*Vehicle.Rotor.area*Vehicle.Rotor.radius^2;
Vehicle.Rotor.k = Vehicle.Rotor.Cq*rho*Vehicle.Rotor.area*Vehicle.
    Rotor.radius^3;
Vehicle.Rotor.w2ToThrustGain = Vehicle.Rotor.Ct*rho*Vehicle.Rotor.
    area*Vehicle.Rotor.radius^2;
% Motors
Vehicle.Motor.maxLimit = 500;
Vehicle.Motor.minLimit = 10;
Vehicle.Motor.commandToW2Gain = 13840.8; %motor command for Rolling
     Spider (0-500) to motorspeed^2
Vehicle.Motor.thrustToMotorCommand = 1/(Vehicle.Rotor.
    w2ToThrustGain*Vehicle.Motor.commandToW2Gain);
% Flight Controller Vars

% This file is derived from the work by Fabian Riether.

% Copyright 2013-2018 The MathWorks, Inc.

% Control Mixer
%Ts2Q transforms thrust [Nm] for motors 1 trhough 4 to u_mechanical
     =[totalThrust;Torqueyaw;pitch;roll]
Controller.Ts2Q = ...
        [1 1 1 1;
        Vehicle.Rotor.Cq/Vehicle.Rotor.Ct*Vehicle.Rotor.radius ....
        -Vehicle.Rotor.Cq/Vehicle.Rotor.Ct*Vehicle.Rotor.radius ...
        Vehicle.Rotor.Cq/Vehicle.Rotor.Ct*Vehicle.Rotor.radius ...
        -Vehicle.Rotor.Cq/Vehicle.Rotor.Ct*Vehicle.Rotor.radius;
        -Vehicle.Airframe.d*sqrt(2)/2 ...
        -Vehicle.Airframe.d*sqrt(2)/2  ...
        Vehicle.Airframe.d*sqrt(2)/2 Vehicle.Airframe.d*sqrt(2)/2;
        -Vehicle.Airframe.d*sqrt(2)/2  ...
        Vehicle.Airframe.d*sqrt(2)/2 ...
        Vehicle.Airframe.d*sqrt(2)/2 -Vehicle.Airframe.d*sqrt(2)
    /2];

%Q2Ts transform requested Q to thrust per motor
Controller.Q2Ts = inv(Controller.Ts2Q);

% Controllers (generic helpers)
switch model
```

```
  case 'RollingSpider'
    Controller.takeoffGain = 0.2;   %drone takes off with constant thrust x% above hover thrust
  case 'Mambo'
    Controller.takeoffGain = 0.45;   %drone takes off with constant thrust x% above hover thrust
end
Controller.totalThrustMaxRelative = 0.92;   %relative maximum total thrust that can be used for gaining altitude; rest is buffer for orientation control
Controller.motorsThrustPerMotorMax = Vehicle.Motor.maxLimit*Vehicle.Motor.commandToW2Gain*...
    Vehicle.Rotor.Ct*rho*Vehicle.Rotor.area*Vehicle.Rotor.radius^2;

%% Quadrotor parameters
x_0 = [0 0 0 0 0 0 0 0 0 0 0 0];

m = 0.0630;         % [kg]
m_un =  m;          % [kg]
L = 0.0624;          % [m]
b = 0.0107;     % [Ns^2]
d = 0.7826400e-3;       % [Nms^2]
g = 9.81;          % [m/s^2]
I_x = 0.0582857e-3;   % |kgm^2]
I_y = 0.0716914e-3;  % |kgm^2]
I_z = 0.1000000e-3;    % |kgm^2]
J_r = 0.1021e-6; % [kgm^2]

% Uncertain quadrotor parameters and disturbances (remove comments for
% uncertainties and disturbances)
I_x_un = I_x%*0.75;   % |kgm^2]
I_y_un = I_y%*0.75;  % |kgm^2]
I_z_un = I_z%*0.75;    % |kgm^2]
disturbance_var = 0%0.1;
disturbance_var_attitude = 0%0.5;

% Sencond order filters
zeta = 1/sqrt(2);
```

```
w_n = 1e3%1.5e2;

%% PD control parameters
% x-position
kp_x = 1.2404;
kd_x = 0.4001;

% y-position
kp_y = 1.2404;
kd_y = 0.4001;

% Altitude
kp_z = 78.5735;
kd_z = 12.1851;

% Roll
kp_phi = 98.8054;
kd_phi = 57.6252;

% Pitch
kp_theta = 70.4077;
kd_theta = 43.8701;

% Yaw
kp_psi = 46.5495;
kd_psi = 14.5408;

%% LQR controller
% x-position
A_x = [0 1; 0 0];
B_x = [0; -g];
C_x = [1 0];
D_x = [0];

Q_x = [11.7000 0; 0 0.6000];
R_x = [9.9000];
K_x = lqr(A_x,B_x,Q_x,R_x);

% y-position
```

```
A_y = [0 1; 0 0];
B_y = [0; -g];
C_y = [1 0];
D_y = [0];

Q_y = [11.7000 0; 0 0.6000];
R_y = [9.9000];
K_y = lqr(A_y,B_y,Q_y,R_y);

% Altitude
A_z = [0 1; 0 0];
B_z = [0; -1/m];
C_z = [1 0];
D_z = [0];

Q_z = [1760.3 0;0  1393.5];
R_z = [0.2000];
K_z = lqr(A_z,B_z,Q_z,R_z);

% Roll
A_roll = [0 1; 0 0];
B_roll = [0; 1/I_x];
C_roll = [1 0];
D_roll = [0];

Q_roll = [1062.7 0;0 305.2];
R_roll = [4.9000];
K_phi= lqr(A_roll,B_roll,Q_roll,R_roll);

% Pitch
A_pitch = [0 1; 0 0];
B_pitch = [0; 1/I_y];
C_pitch = [1 0];
D_pitch = [0];

Q_pitch = [1791.8 0;0 462.5];
R_pitch = [4.1000];
K_theta= lqr(A_pitch,B_pitch,Q_pitch,R_pitch);

```

```
% Yaw
A_yaw = [0 1; 0 0];
B_yaw = [0; 1/I_z];
C_yaw = [1 0];
D_yaw = [0];

Q_yaw = [785.9770 0;0 68.8622];
R_yaw = [2.6613];
K_psi = lqr(A_yaw,B_yaw,Q_yaw,R_yaw);

%% ISMC parameters
% x-position
lambda_x = 2.9998;
rho_x = 3.1285;
epsilon_x = 0.2;
k_x = 0.0034;
zeta_x = 0.6052;

% y-position
lambda_y = 2.9998;
rho_y = 3.1285;
epsilon_y = 0.2;
k_y = 0.0034;
zeta_y = 0.6052;

% Altitude
lambda_z = 5.0351;
rho_z = 6.6602;
epsilon_z = 0.2;
k_z = 0.0011;
zeta_z = 2.7307;

% Roll
lambda_phi = 17.5038;
rho_phi = 38.5424;
epsilon_phi = 0.2;
k_phi = 0.0037;
zeta_phi = 4.9774;

```

```
% Pitch
lambda_theta = 16.2555;
rho_theta = 38.9482;
epsilon_theta = 0.2;
k_theta = 0.0032;
zeta_theta = 4.2975;

% Yaw
lambda_psi = 39.0089;
rho_psi = 8.3646;
epsilon_psi = 0.2;
k_psi = 0.0016;
zeta_psi = 2.3001;
```

Listing A.1: MATLAB code for initializing variables/constants used by the quadrotor control system.

```
[System]
Name='Fuzzy_gain_scheduling_attitude'
Type='sugeno'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=25
AndMethod='prod'
OrMethod='probor'
ImpMethod='prod'
AggMethod='sum'
DefuzzMethod='wtaver'

[Input1]
Name='s'
Range=[-0.2 0.2]
NumMFs=5
MF1='NL':'gaussmf',[0.0424661 -0.2]
MF2='NS':'gaussmf',[0.0424661 -0.0833333]
MF3='ZE':'gaussmf',[0.0424661 0]
MF4='PS':'gaussmf',[0.0424661 0.0833333]
MF5='PL':'gaussmf',[0.0424661 0.2]

```

```
[Input2]
Name='s_dot'
Range=[-0.2 0.2]
NumMFs=5
MF1='NL':'gaussmf',[0.0424661 -0.2]
MF2='NS':'gaussmf',[0.0424661 -0.0833333]
MF3='ZE':'gaussmf',[0.0424661 0]
MF4='PS':'gaussmf',[0.0424661 0.0833333]
MF5='PL':'gaussmf',[0.0424661 0.2]

[Output1]
Name='rho'
Range=[0 0.5]
NumMFs=5
MF1='VL':'constant',[0]
MF2='L':'constant',[0.125]
MF3='M':'constant',[0.25]
MF4='H':'constant',[0.375]
MF5='VH':'constant',[0.5]

[Rules]
5 1, 3 (1) : 1
5 2, 2 (1) : 1
5 3, 1 (1) : 1
5 4, 1 (1) : 1
5 5, 1 (1) : 1
4 1, 4 (1) : 1
4 2, 3 (1) : 1
4 3, 2 (1) : 1
4 4, 2 (1) : 1
4 5, 1 (1) : 1
3 1, 4 (1) : 1
3 2, 4 (1) : 1
3 3, 3 (1) : 1
3 4, 2 (1) : 1
3 5, 2 (1) : 1
2 1, 5 (1) : 1
2 2, 4 (1) : 1
2 3, 4 (1) : 1
```

```
2 4, 3 (1) : 1
2 5, 2 (1) : 1
1 1, 5 (1) : 1
1 2, 5 (1) : 1
1 3, 5 (1) : 1
1 4, 4 (1) : 1
1 5, 3 (1) : 1
```

Listing A.2: Adaptive Fuzzy Gain Scheduling.

A.2 Simulink Schemes for Quadrotor Simulations

A.2.1 PD Control System

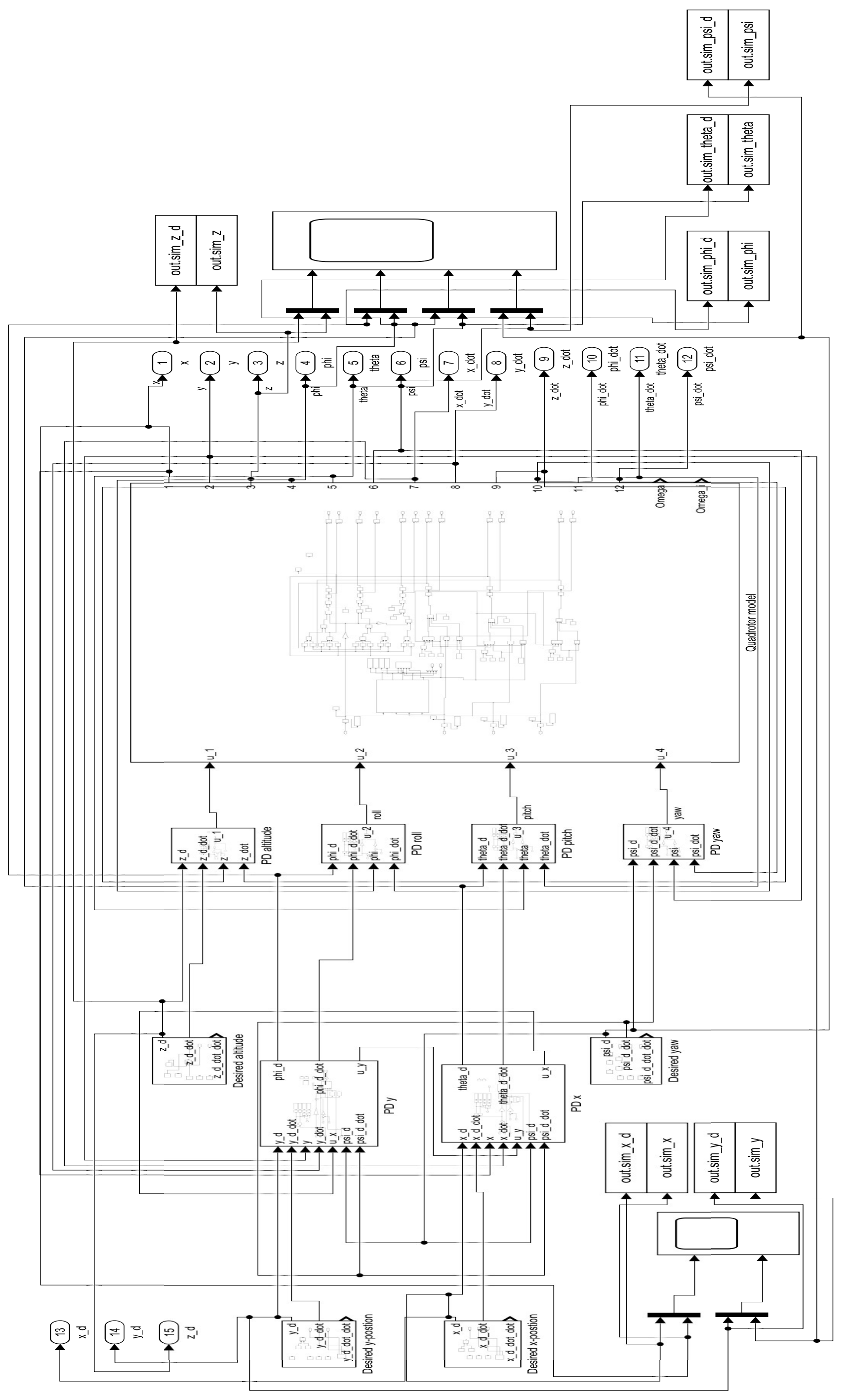


Figure A.1: PD Control System.

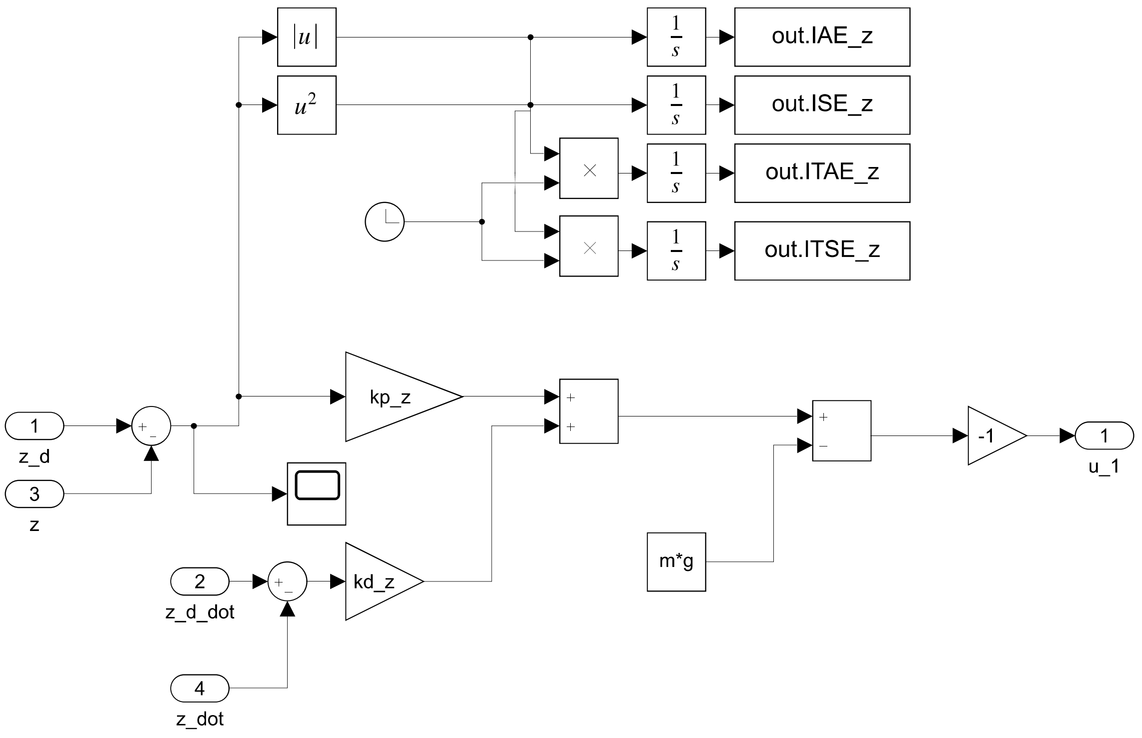


Figure A.2: PD control for altitude.

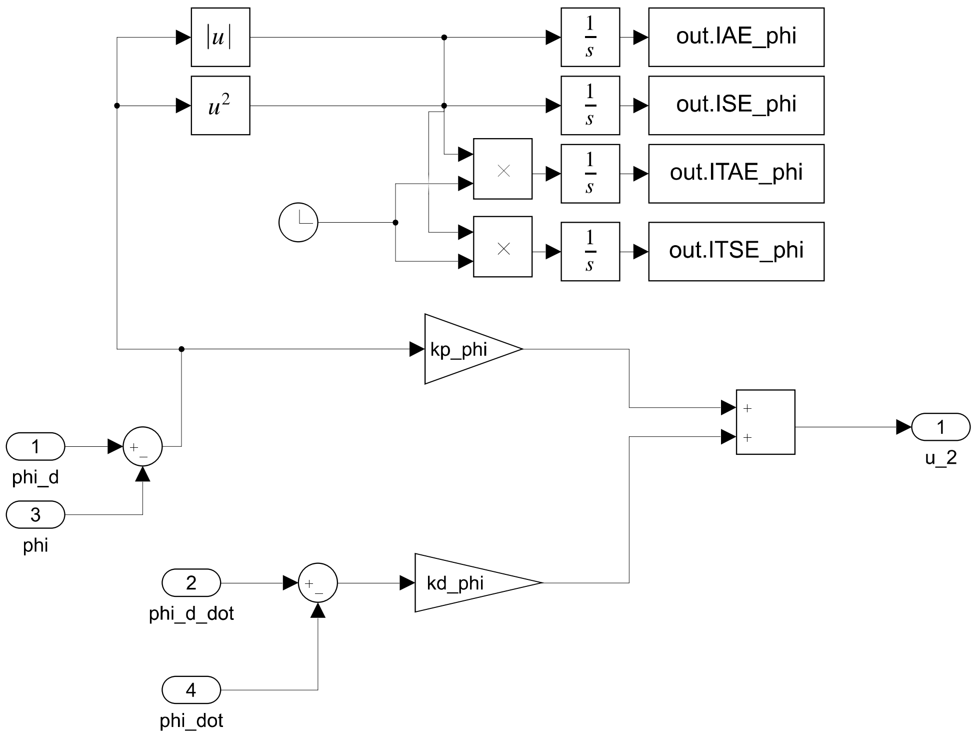


Figure A.3: PD control for roll.

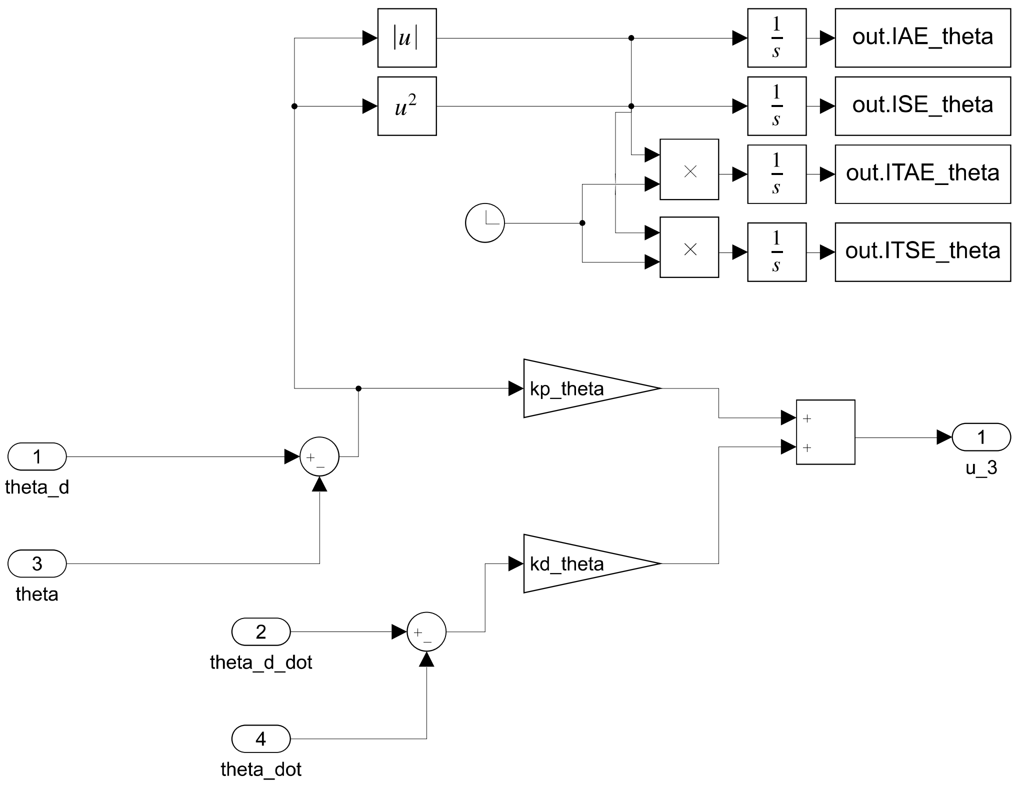


Figure A.4: PD control for pitch.

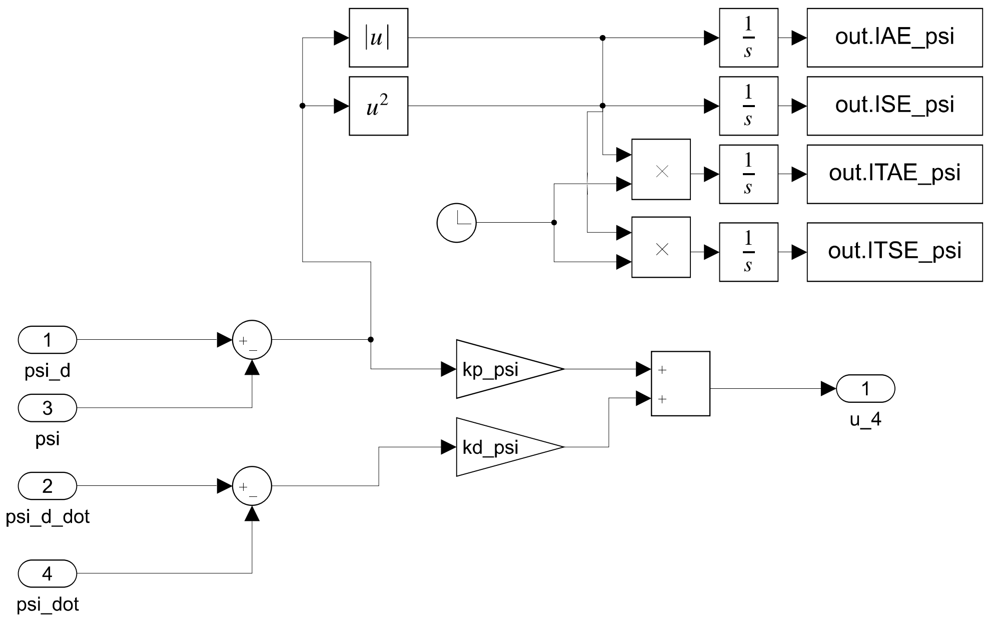


Figure A.5: PD control for yaw.

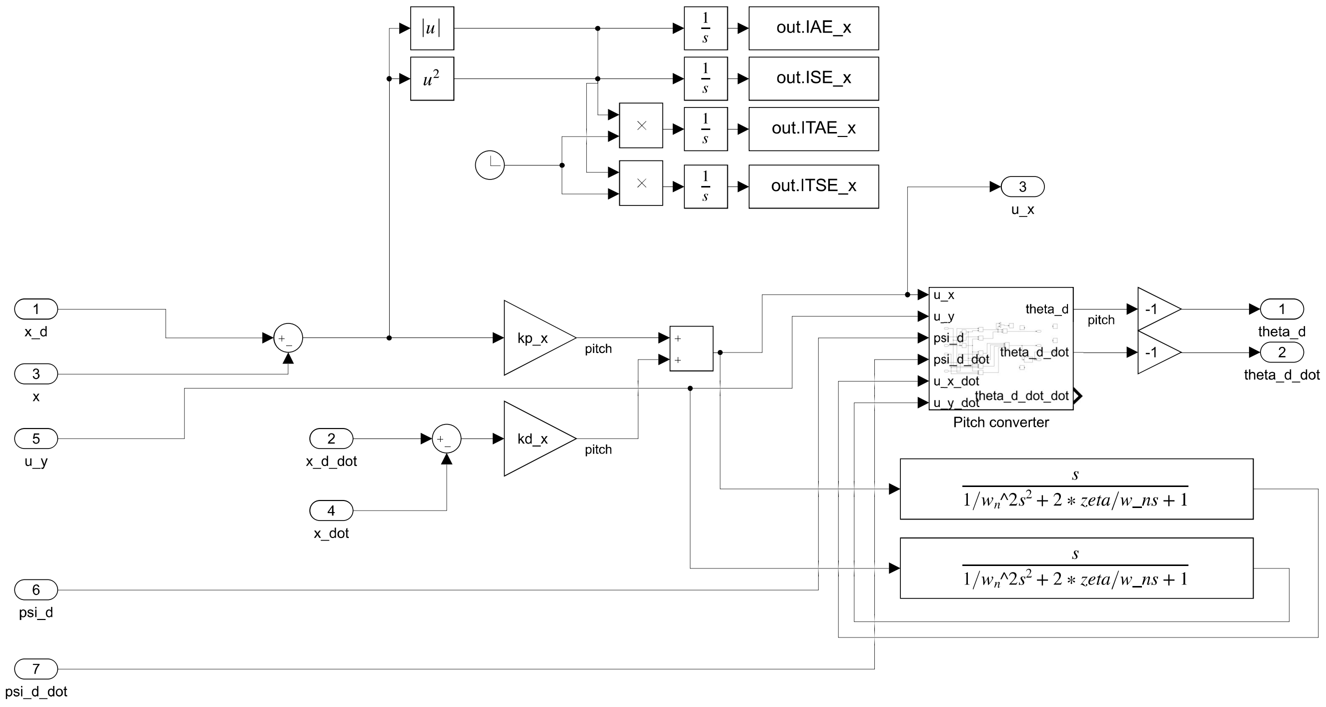


Figure A.6: PD for x-position.

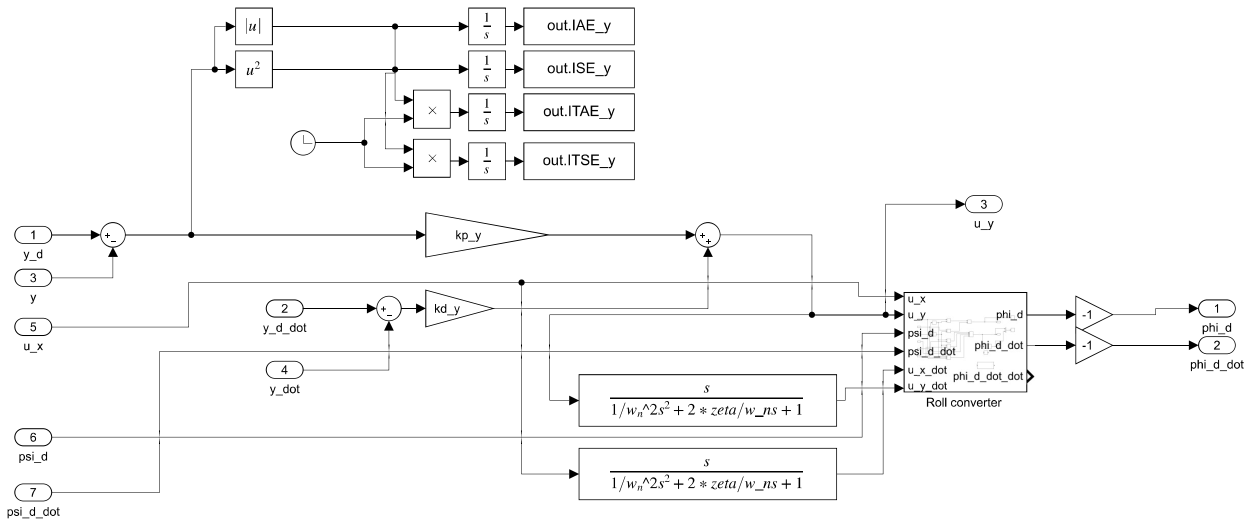


Figure A.7: PD for y-position.

A.2.2 LQR Control System

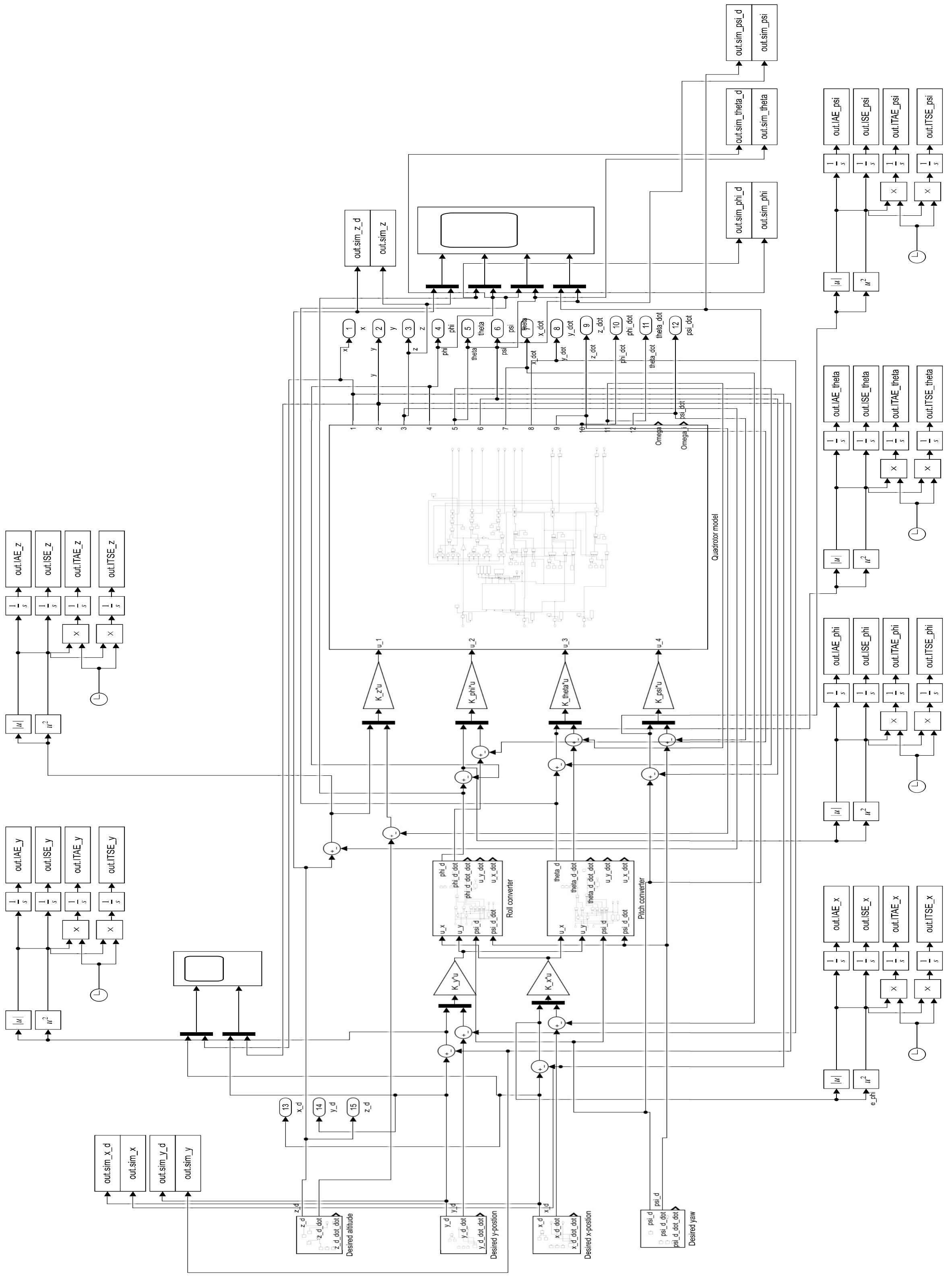


Figure A.8: LQR Control System.

A.2.3 ISMC Control System

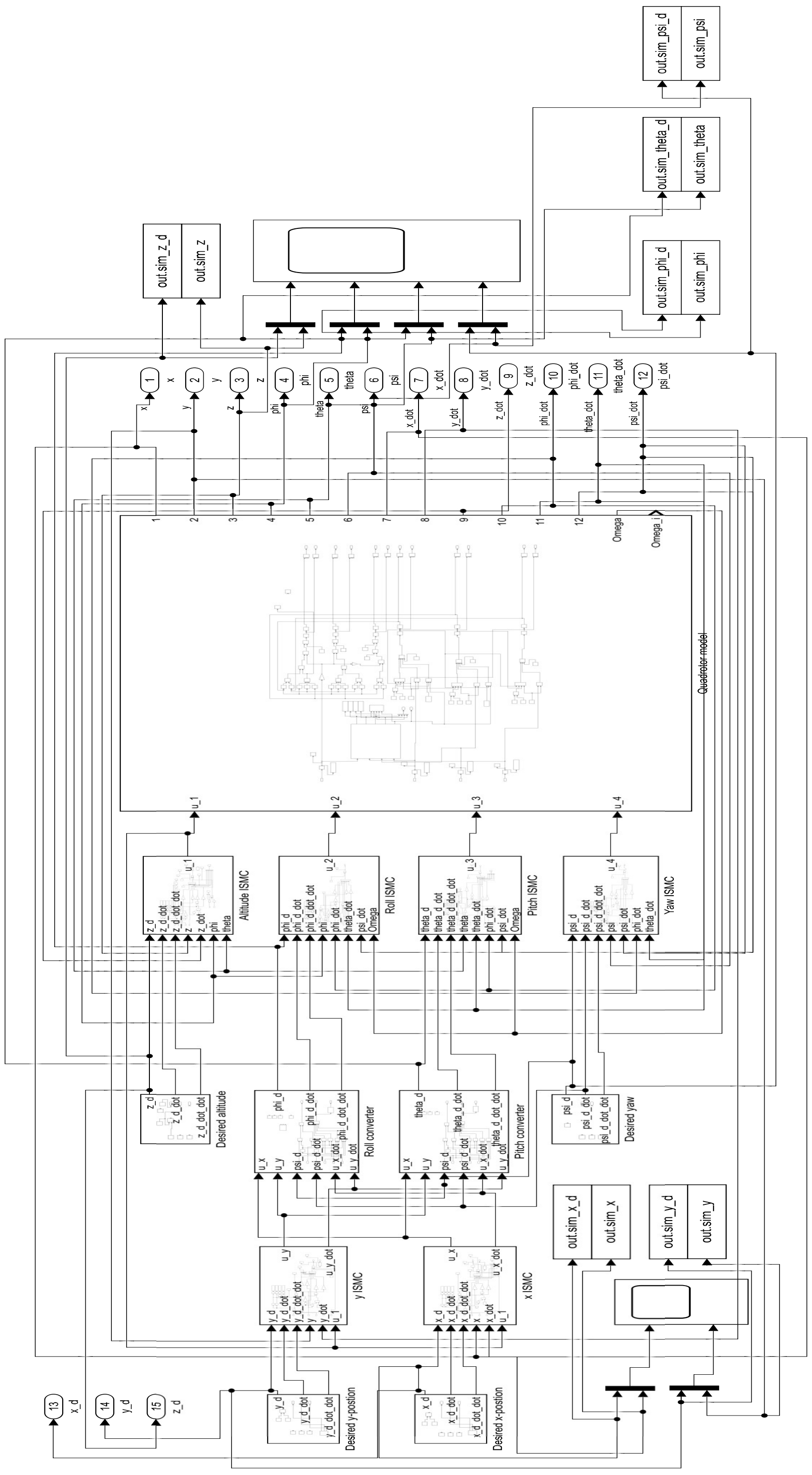


Figure A.9: ISMC Control System.

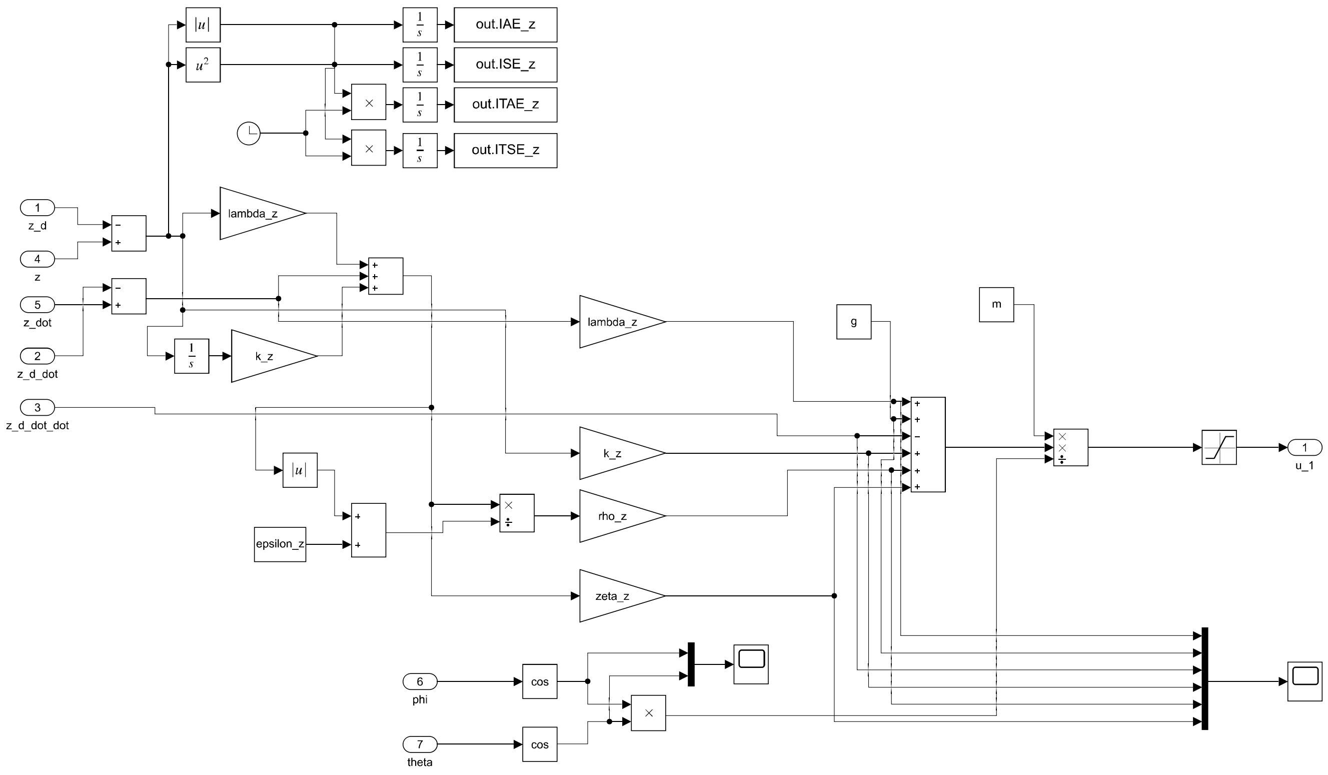


Figure A.10: ISMC for altitude.

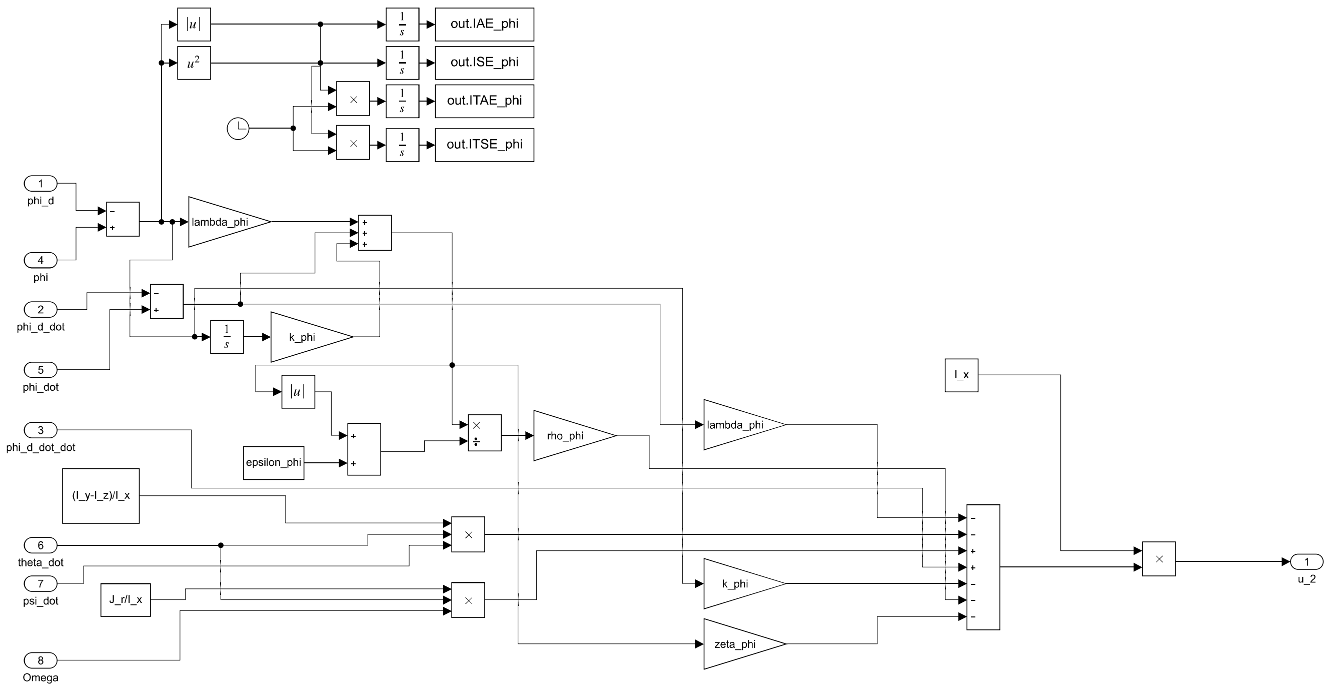


Figure A.11: ISMC for roll.

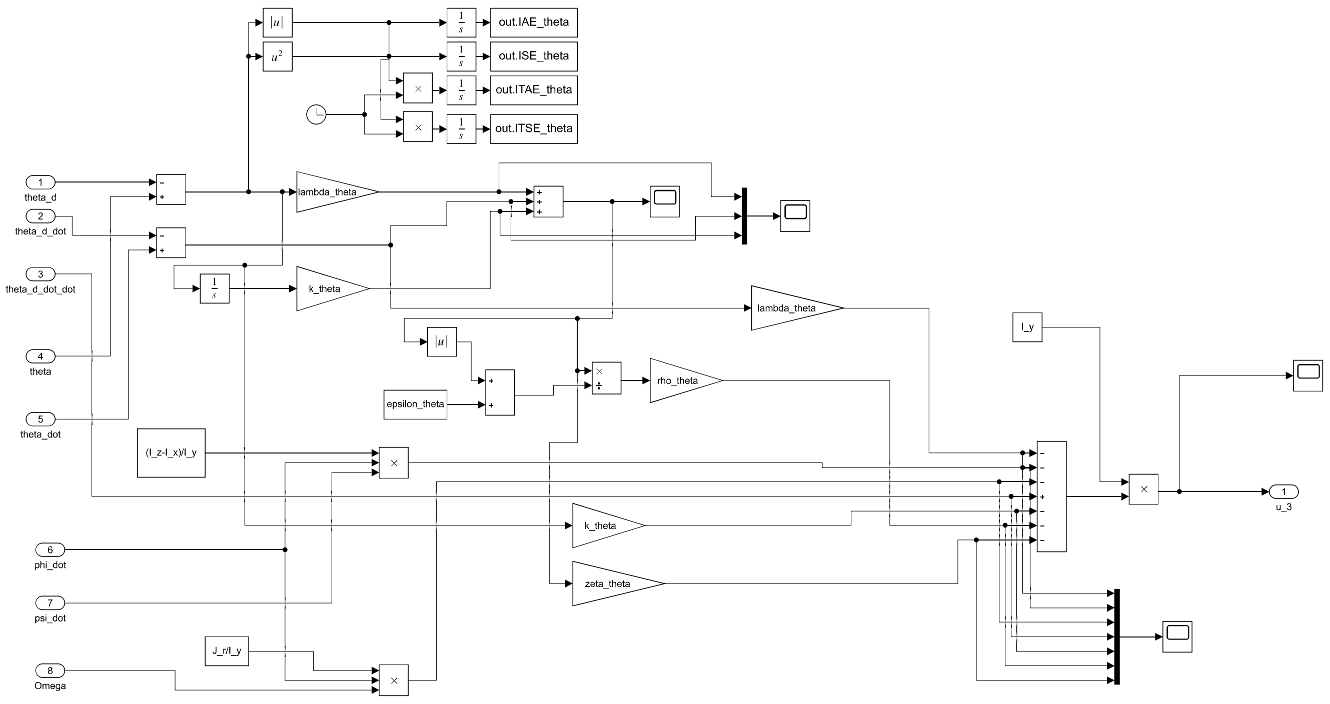


Figure A.12: ISMC for pitch.

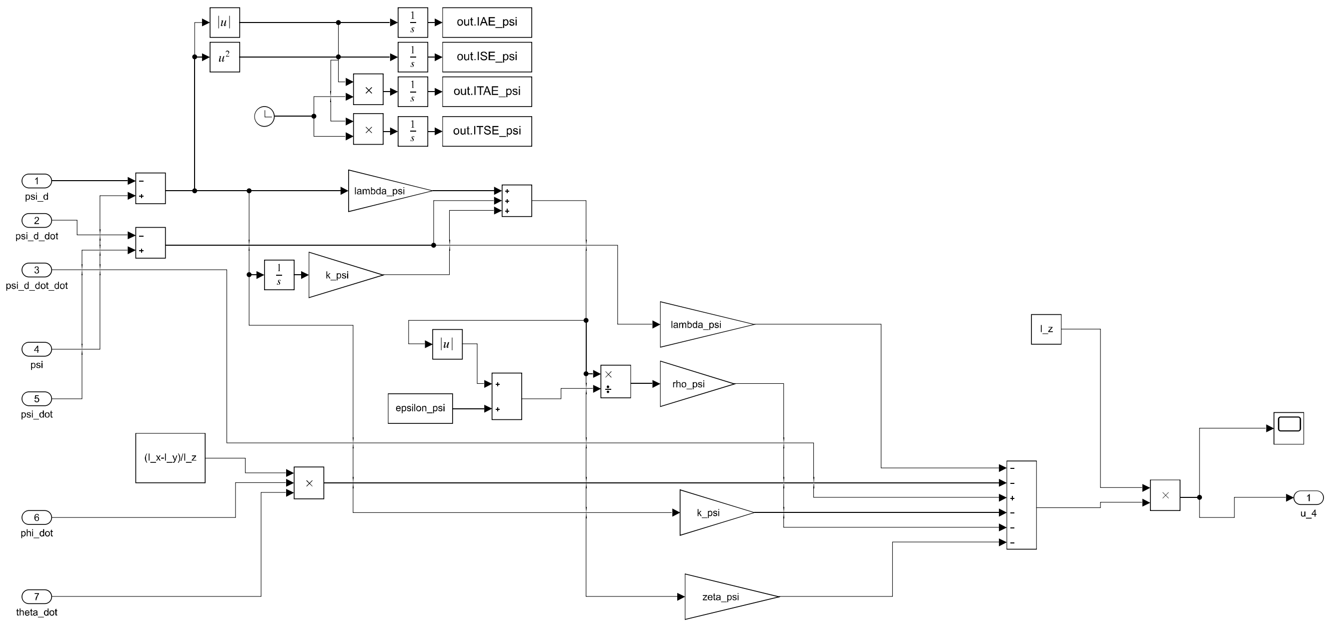


Figure A.13: ISMC for yaw.

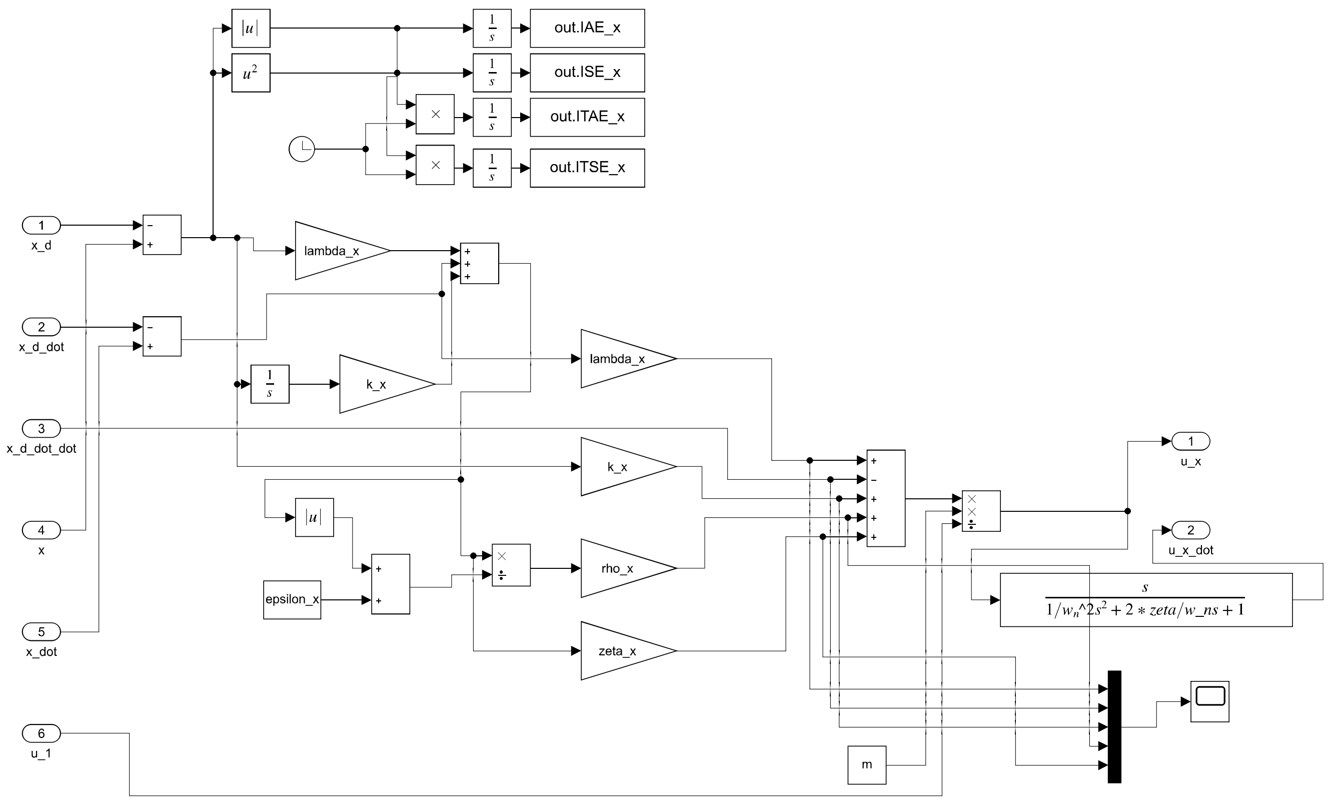


Figure A.14: ISMC for x-position.

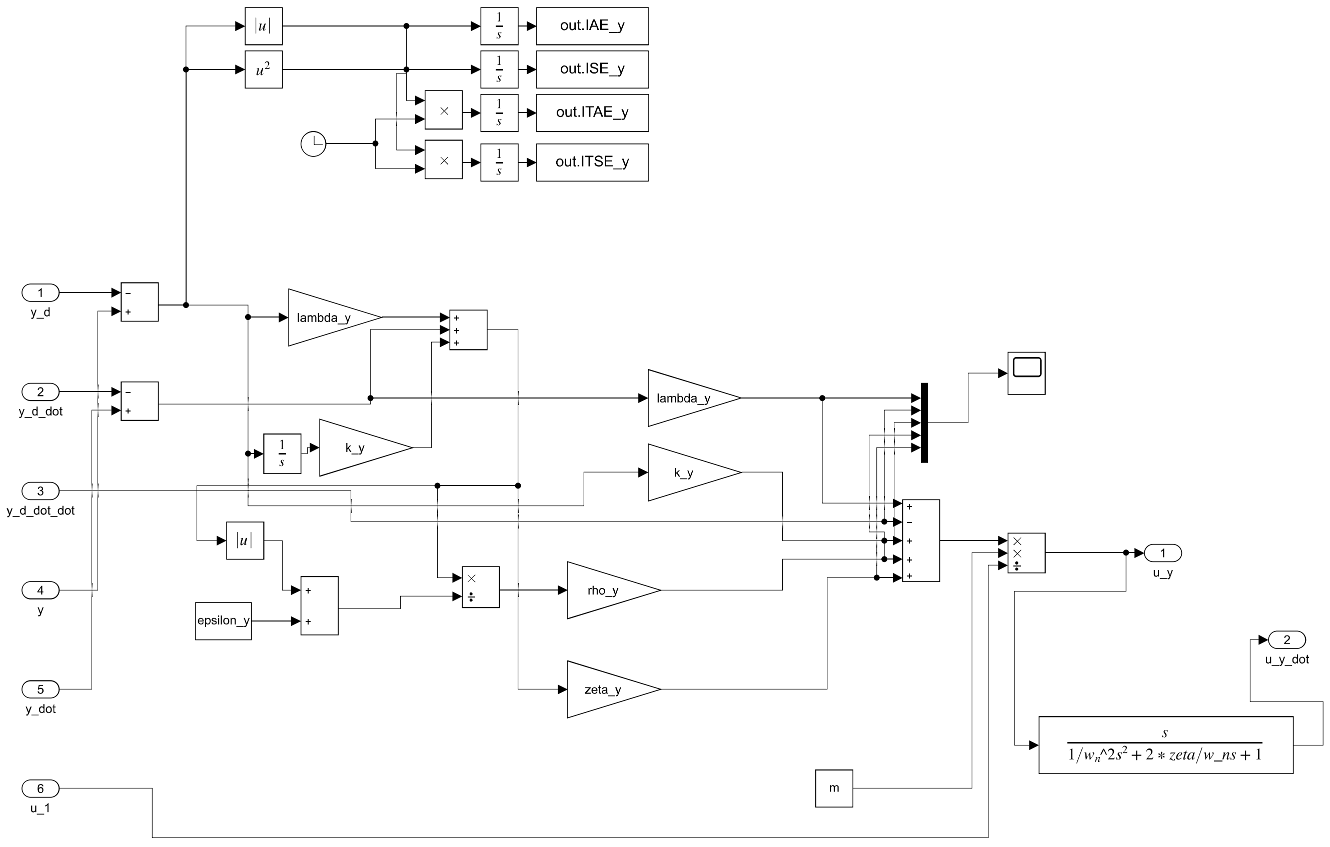


Figure A.15: ISMC for y-position.

A.2.4 Simulink Scheme of the Quadrotor Model

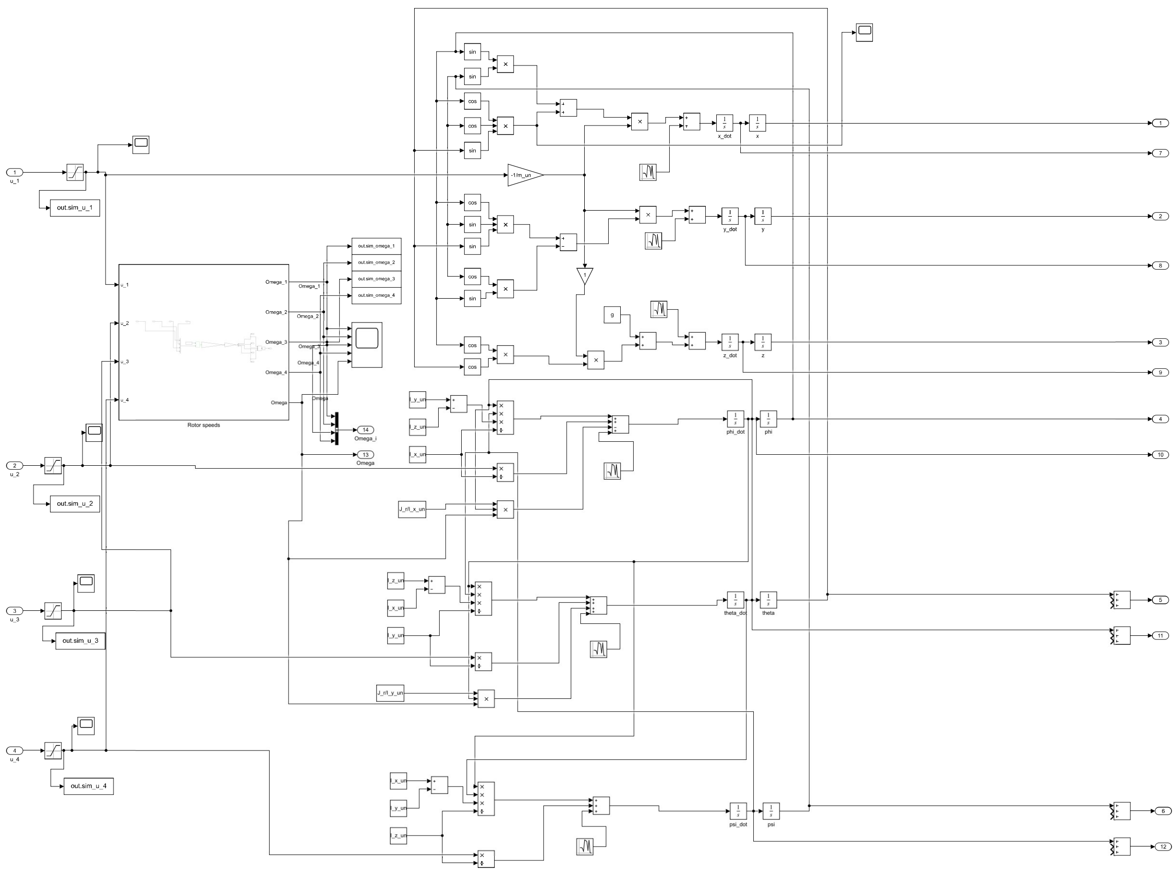


Figure A.16: Quadrotor model.

A.2.5 Simulink Schemes of Roll and Pitch Converter

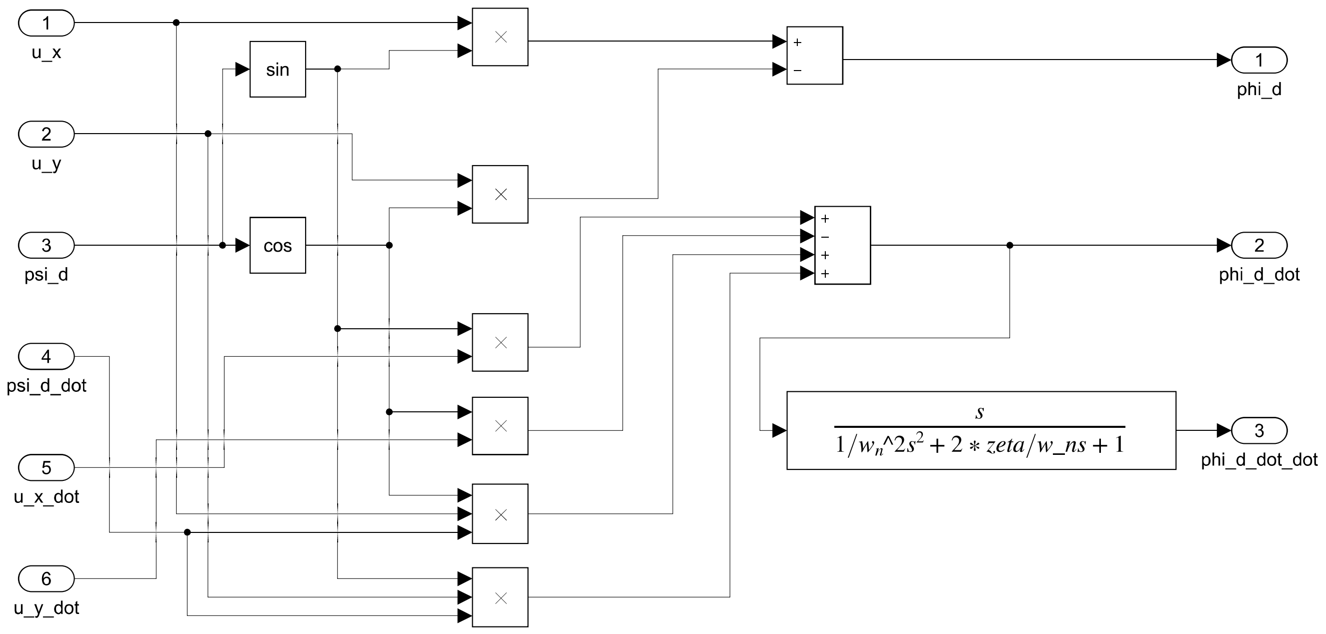


Figure A.17: Roll converter.

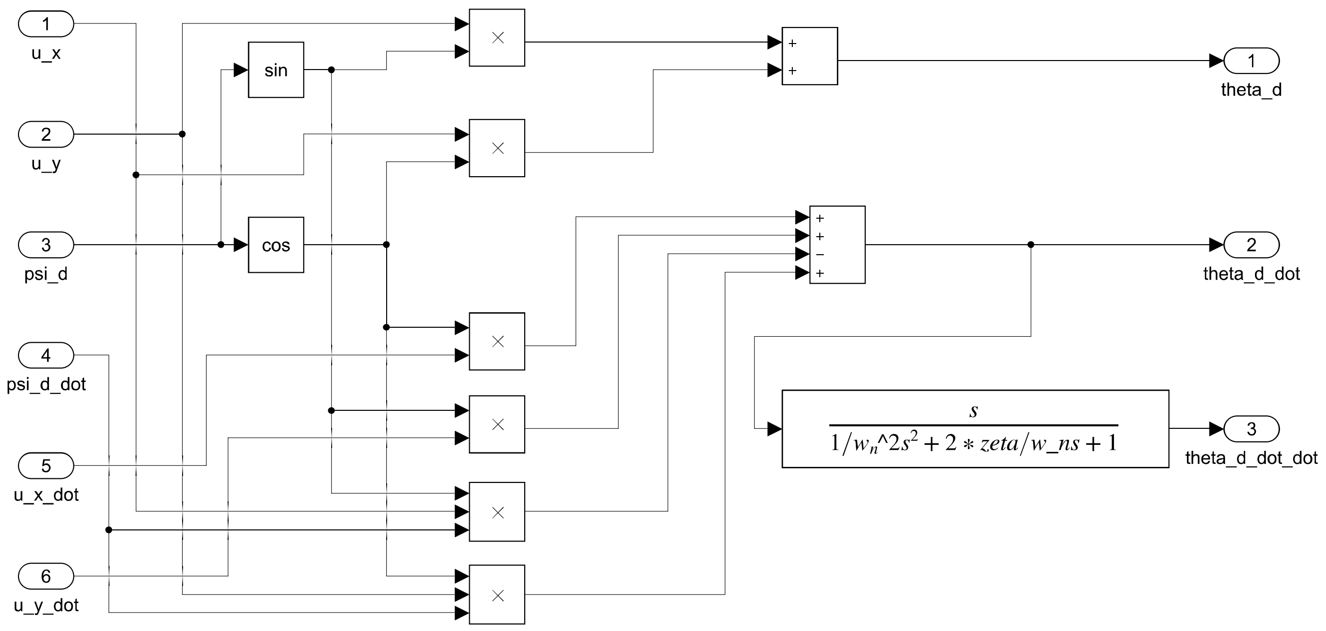


Figure A.18: Pitch converter.

Appendix B

Genetic Algorithm

B.1 Desired Trajectory for Genetic Algorithm

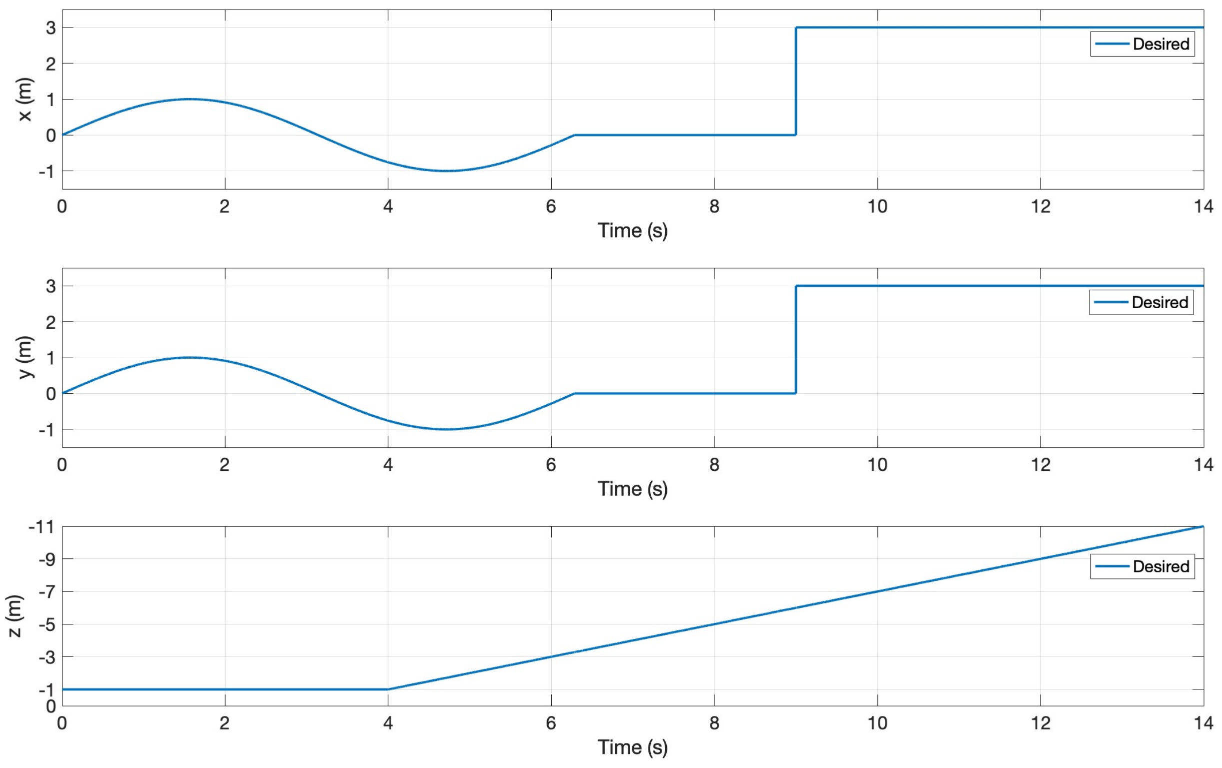


Figure B.1: The desired trajectory used for GA.

B.2 MATLAB Code and Functions for Genetic Algorithm

B.2.1 GA for PD

```
%% Genetic algorithm
no_var = 8;
% no_var = 2; for tuning yaw controller

lower_bound = [0 0 0 0 0 0 0 0] + 1e-2;
upper_bound = [5 5 100 100 100 100 100 100];
% lower_bound = [0 0] + 1e-2; for tuning yaw controller
% upper_bound = [100 100]; for tuning yaw controller

ga_opt = gaoptimset('Display','off','Generations',20,'
    populationsize',80,'PlotFcns',@gaplotbestf);
obj_fn = @(K) cost_function_PD(K);
% ga_opt = gaoptimset('Display','off','Generations',15,'
    populationsize',20,'PlotFcns',@gaplotbestf);
% obj_fn = @(K) cost_function_PD(K); for tuning yaw controller


[K,best] = ga((obj_fn),no_var,[],[],[],[],lower_bound,upper_bound
    ,[],ga_opt)
```

Listing B.1: MATLAB code for the genetic algorithm for PD.

```
function cost = cost_function_PD(K)
    % x-position
    assignin('base','kp_x',K(1));
    assignin('base','kd_x',K(2));

    % y-position
    assignin('base','kp_y',K(1));
    assignin('base','kd_y',K(2));

    % Altitude
    assignin('base','kp_z',K(3));
    assignin('base','kd_z',K(4));

    % Roll
    assignin('base','kp_phi',K(5));
    assignin('base','kd_phi',K(6));

    % Pitch
    assignin('base','kp_theta',K(7));
    assignin('base','kd_theta',K(8));

    % Yaw (for tuning yaw controller)
%     assignin('base','kp_psi',K(1));
%     assignin('base','kd_psi',K(2));

    try
        simulering = sim('Quadrotor_PID_GA.slx');
        if simulering.flag(end,1) > 0 || simulering.flag(end,2) >0
    % flag for divergence
            cost = 100;
        else
            cost = simulering.IAE_x.data(end) + simulering.IAE_y.
    data(end) + simulering.IAE_z.data(end) + simulering.IAE_phi.data
    (end) + simulering.IAE_theta.data(end) + simulering.IAE_psi.data
    (end);
        end
    catch
        cost = 100;
    end
end
```

Listing B.2: MATLAB code for the cost function for PD.

B.2.2 GA for LQR

```
%% Genetic algorithm
no_var = 12;
% no_var = 3; % for tuning yaw controller
lower_bound = [0 0 0 0 0 0 0 0 0 0 0 0] + 1e-2;
upper_bound = [100 100 10 2000 2000 10 2000 2000 10 2000 2000 10];
% lower_bound = [0 0 0] + 1e-2; for tuning yaw controller
% upper_bound = [1000 1000 10]; for tuning yaw controller

```

```
ga_opt = gaoptimset('Display','off','Generations',20,'
   populationsize',120,'PlotFcns',@gaplotbestf);
obj_fn = @(K) cost_function_LQR(K);
% ga_opt = gaoptimset('Display','off','Generations',15,'
   populationsize',30,'PlotFcns',@gaplotbestf);
% obj_fn = @(K) cost_function_LQR(K); for tuning yaw controller


[K,best] = ga((obj_fn),no_var,[],[],[],[],lower_bound,upper_bound
   ,[],ga_opt)
```

Listing B.3: MATLAB code for the genetic algorithm for LQR.

```
function cost = cost_function_LQR(K)
    %% LQR controllers
    % Quadrotor parameters
    m = 0.0630;          % [kg]
    L = 0.0624;            % [m]
    b = 0.0107;      % [Ns^2]
    d = 0.7826400e-3;      % [Nms^2]
    g = 9.81;          % [m/s^2]
    I_x = 0.0582857e-3;    % |kgm^2]
    I_y = 0.0716914e-3;   % |kgm^2]
    I_z = 0.1000000e-3;     % |kgm^2]
    J_r = 0.1021e-6; % [kgm^2]

    % x-position
    A_x = [0 1; 0 0];
    B_x = [0; -g];

    Q_x = diag([K(1) K(2)]);
    R_x = [K(3)];
    K_x = lqr(A_x,B_x,Q_x,R_x);
    assignin('base','K_x',K_x);

    % y-position
    A_y = [0 1; 0 0];
    B_y = [0; -g];

    Q_y = diag([K(1) K(2)]);
    R_y = [K(3)];
    K_y = lqr(A_y,B_y,Q_y,R_y);
    assignin('base','K_y',K_y);

    % Altitude
    A_z = [0 1; 0 0];
    B_z = [0; -1/m];

    Q_z = diag([K(4) K(5)]);
    R_z = [K(6)];
    K_z = lqr(A_z,B_z,Q_z,R_z);
    assignin('base','K_z',K_z);

    % Roll
    A_phi = [0 1; 0 0];
    B_phi = [0; 1/I_x];

    Q_phi = diag([K(7) K(8)]);
    R_phi = [K(9)];
    K_phi = lqr(A_phi,B_phi,Q_phi,R_phi);
    assignin('base','K_phi',K_phi);

    % Pitch
    A_theta = [0 1; 0 0];
    B_theta = [0; 1/I_y];

    Q_theta = diag([K(10) K(11)]);
    R_theta = [K(12)];
    K_theta = lqr(A_theta,B_theta,Q_theta,R_theta);
    assignin('base','K_theta',K_theta);

    % Yaw (for tuning yaw controller)
%     A_psi = [0 1; 0 0];
%     B_psi = [0; 1/I_z];
%
%     Q_psi = diag([K(1) K(2)]);
%     R_psi = [K(3)];
%     K_psi = lqr(A_psi,B_psi,Q_psi,R_psi);
%     assignin('base','K_psi',K_psi);
```

```

    try
        simulering = sim('Quadrotor_LQR_GA.slx');
        if simulering.flag(end,1) > 0 || simulering.flag(end,2) >0 % flag for divergence
            cost = 100;
        else
            cost = simulering.IAE_x(end) + simulering.IAE_y(end) + simulering.IAE_z(end) + simulering.IAE_phi(end) + simulering.IAE_theta(end) + simulering.IAE_psi(end);
        end
    catch
        cost = 100;
    end
end
```

Listing B.4: MATLAB code for the cost function for LQR.

B.2.3 GA for ISMC

```
%% Genetic algorithm
no_var = 16;
%no_var = 4; % for tuning yaw controller

lower_bound = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] + 1e-3;
upper_bound = [5 5 0.005 5 10 10 0.005 3 40 40 0.005 5 40 40 0.005 5];
% lower_bound = [0 0 0 0] + 1e-3; for tuning yaw controller
% upper_bound = [40 40 0.005 5];  for tuning yaw controller

ga_opt = gaoptimset('Display','off','Generations',20,'populationsize',160,'PlotFcns',@gaplotbestf);
obj_fn = @(K) cost_function_ISMC(K);
% ga_opt = gaoptimset('Display','off','Generations',15,'populationsize',40,'PlotFcns',@gaplotbestf);
% obj_fn = @(K) cost_function_ISMC(K); % for tuning yaw controller

[K,best] = ga((obj_fn),no_var,[],[],[],[],lower_bound,upper_bound,[],ga_opt)
```

Listing B.5: MATLAB code for the genetic algorithm for ISMC.

```matlab
function cost = cost_function_ISMC(K)
    % x-position
    assignin('base','lambda_x',K(1));
    assignin('base','rho_x',K(2));
    assignin('base','k_x',K(3));
    assignin('base','zeta_x',K(4));
    % y-position
    assignin('base','lambda_y',K(1));
    assignin('base','rho_y',K(2));
    assignin('base','k_y',K(3));
    assignin('base','zeta_y',K(4));
    % z-position
    assignin('base','lambda_z',K(5));
    assignin('base','rho_z',K(6));
    assignin('base','k_z',K(7));
    assignin('base','zeta_z',K(8));
    % phi
    assignin('base','lambda_phi',K(9));
    assignin('base','rho_phi',K(10));
    assignin('base','k_phi',K(11));
    assignin('base','zeta_phi',K(12));
    % theta
    assignin('base','lambda_theta',K(13));
    assignin('base','rho_theta',K(14));
    assignin('base','k_theta',K(15));
    assignin('base','zeta_theta',K(16));
    % psi (for tuning yaw controller)
%     assignin('base','lambda_psi',K(1));
%     assignin('base','rho_psi',K(2));
%     assignin('base','k_psi',K(3));
%     assignin('base','zeta_psi',K(4));
    try
        simulering = sim('Quadrotor_ISMC_vs.slx');
        cost = simulering.IAE_x.data(end) + simulering.IAE_y.data(
   end) + simulering.IAE_z.data(end) + simulering.IAE_phi.data(end)
    + simulering.IAE_theta.data(end) + simulering.IAE_psi.data(end)
   ;
    catch
        cost = 100;
    end
end
```

Listing B.6: MATLAB code for the cost function for ISMC.

B.3 Genetic Algorithm Simulink Schemes

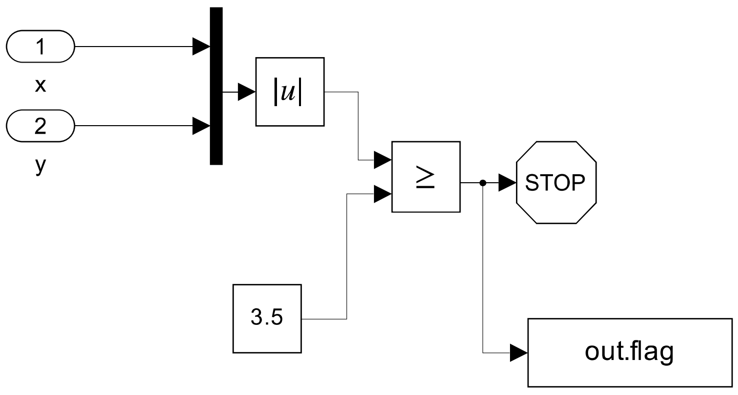


Figure B.2: Divergence flag for GA.

Appendix C

3 DOF Hover

C.1 Experimental Results for The 3 DOF Hover

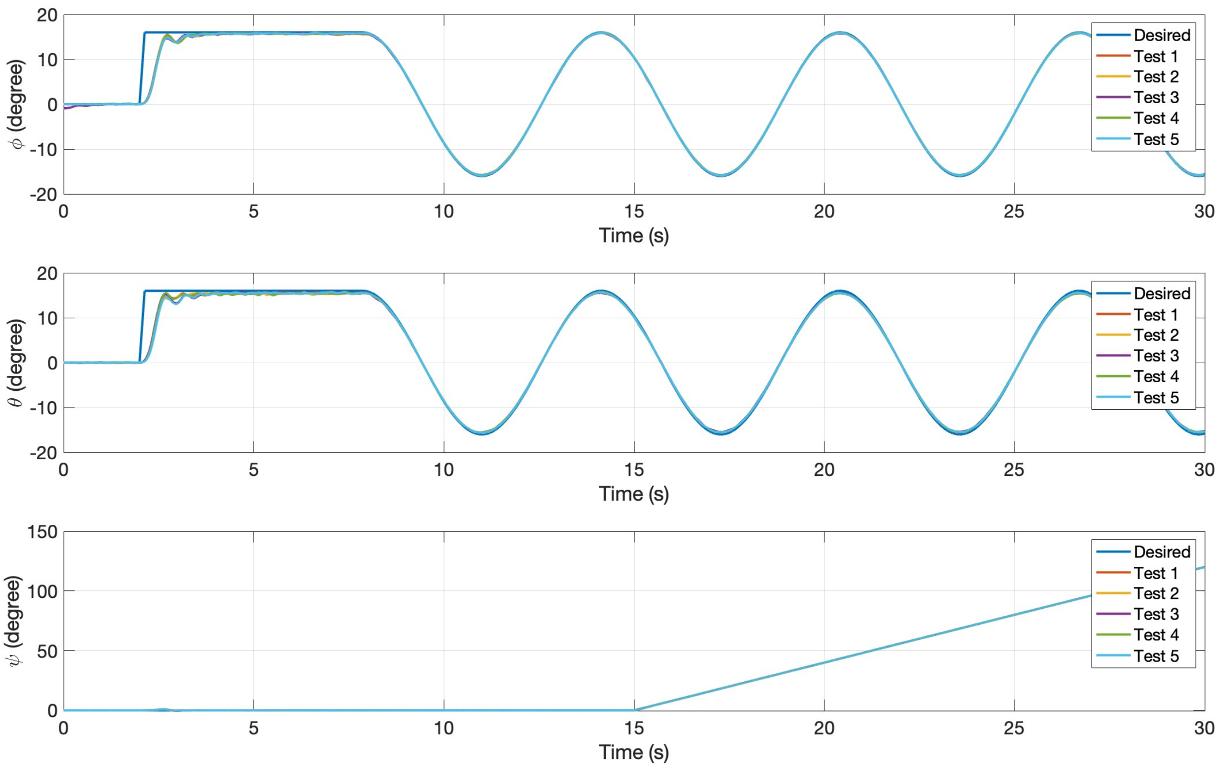


Figure C.1: 3 DOF Hover attitude response for a step-to-sinusoidal reference with an amplitude of 16° with the PD controller.

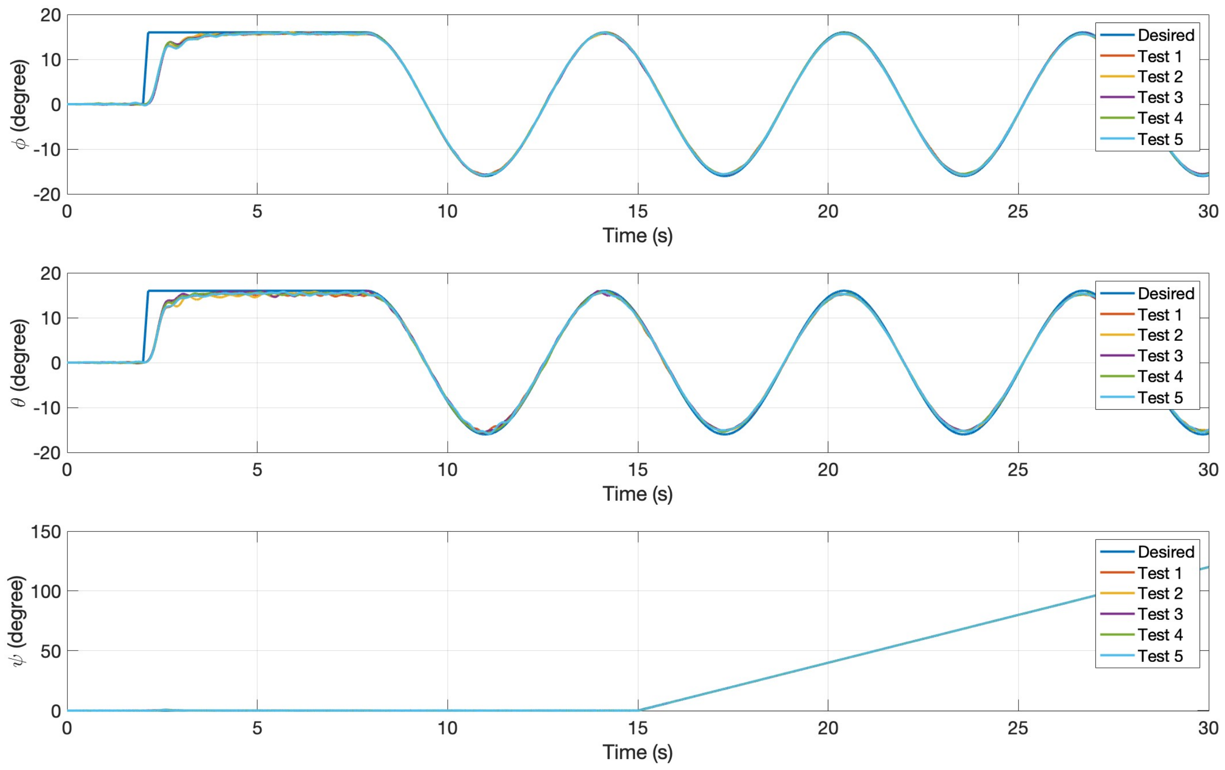


Figure C.2: 3 DOF Hover attitude response for a step-to-sinusoidal reference with an amplitude of 16° with the LQR controller.

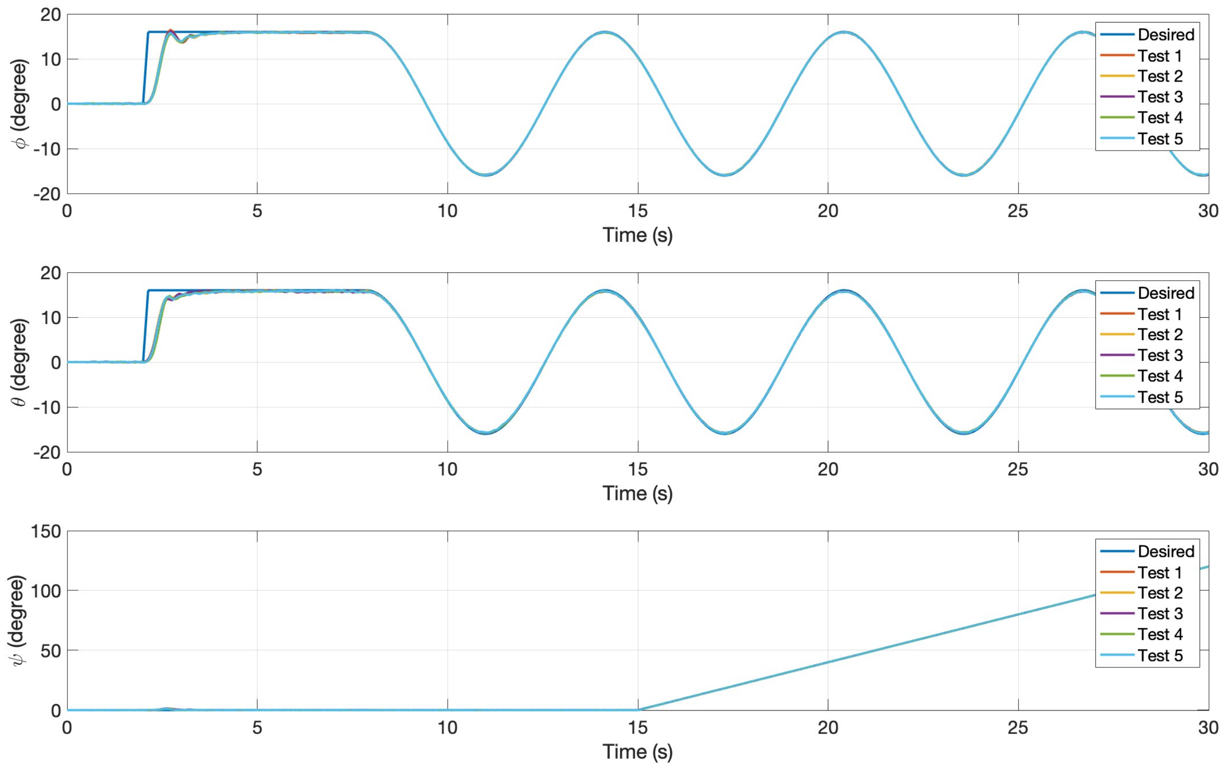


Figure C.3: 3 DOF Hover attitude response for a step-to-sinusoidal reference with an amplitude of 16° with the ISMC.

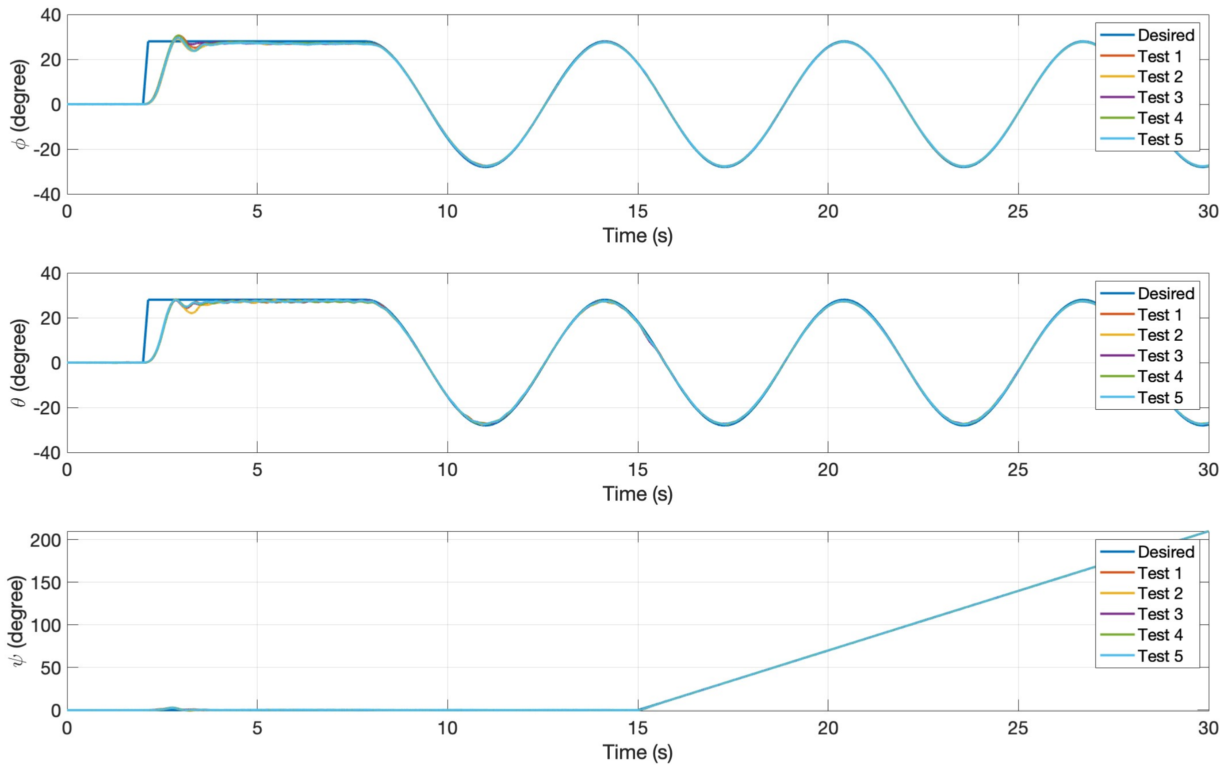


Figure C.4: 3 DOF Hover attitude response for a step-to-sinusoidal reference with an amplitude of 28° with the PD controller.

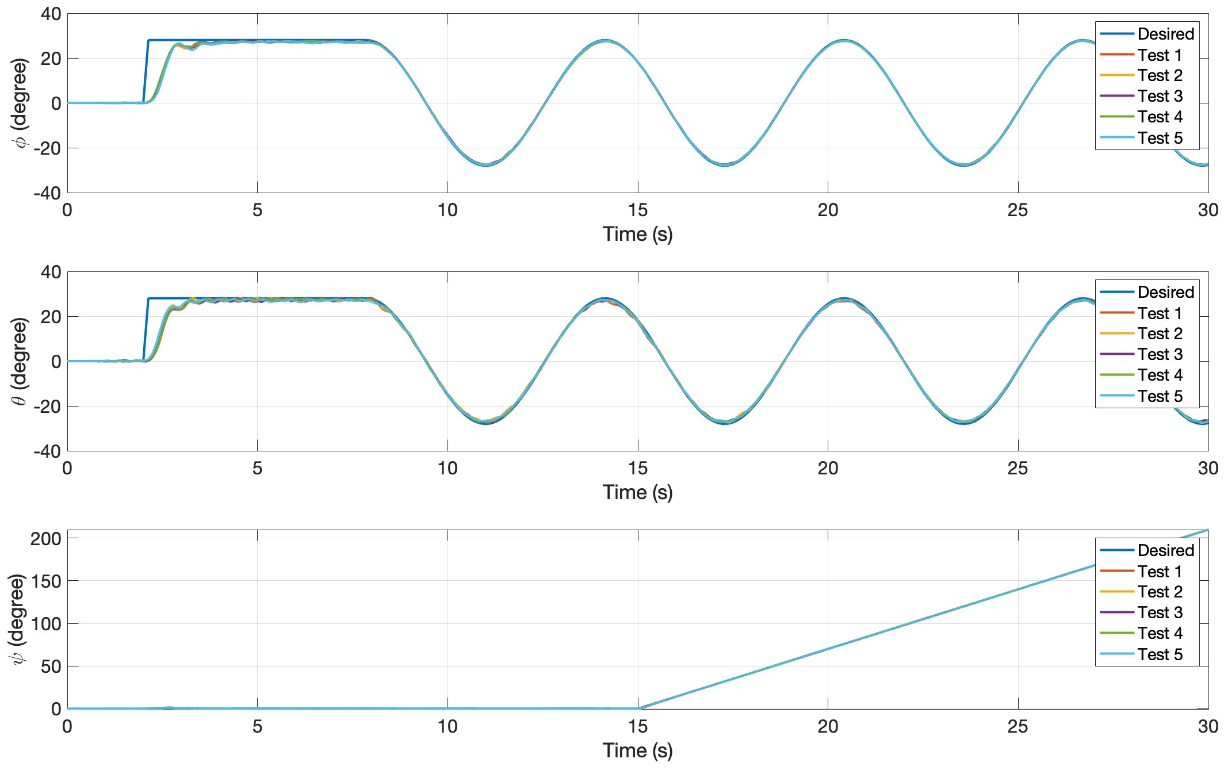


Figure C.5: 3 DOF Hover attitude response for a step-to-sinusoidal reference with an amplitude of 28° with the LQR controller.

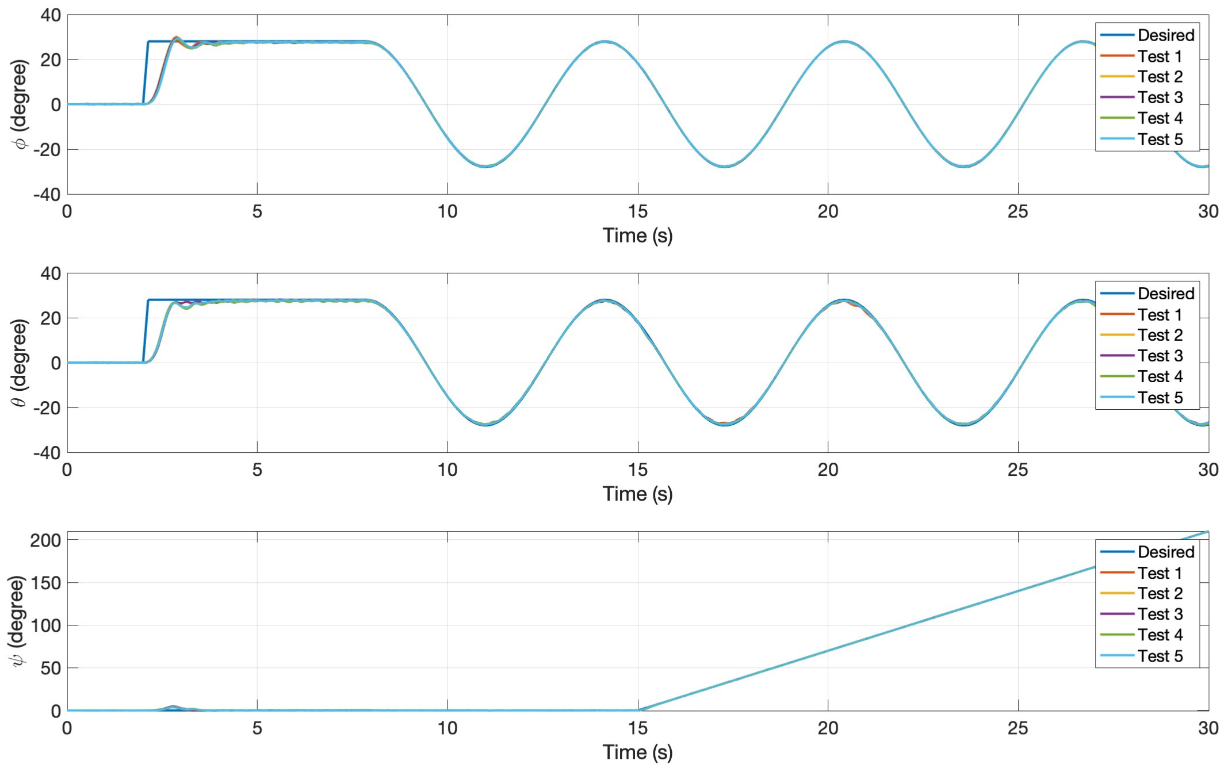


Figure C.6: 3 DOF Hover attitude response for a step-to-sinusoidal reference with an amplitude of 28° with the ISMC.

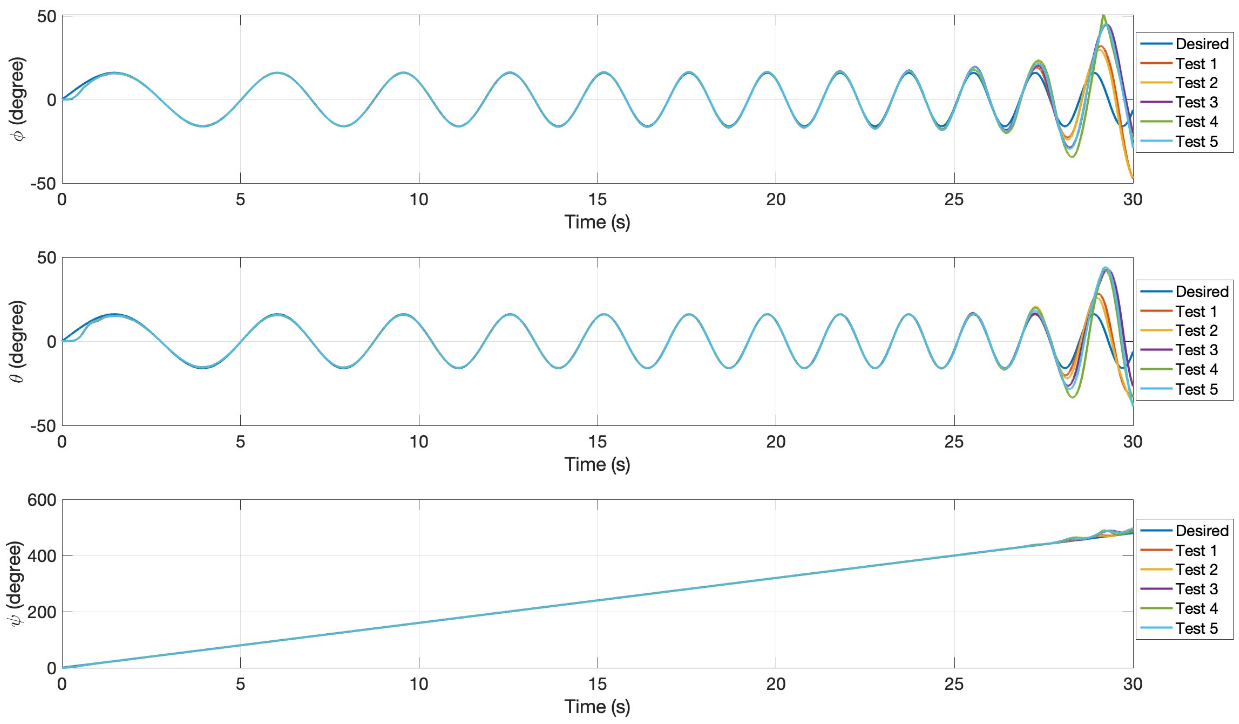


Figure C.7: 3 DOF Hover attitude response for a linear chirp reference with an amplitude of 16° with the PD controller.

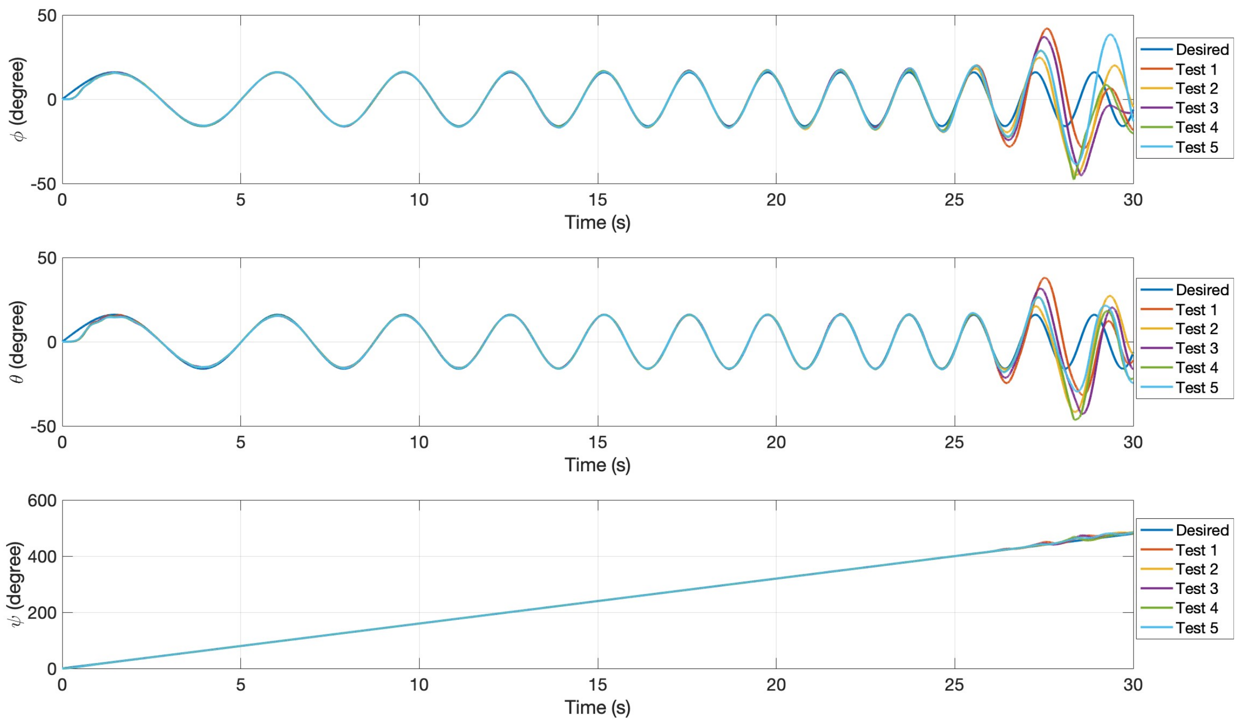


Figure C.8: 3 DOF Hover attitude response for a linear chirp reference with an amplitude of 16° with the LQR controller.

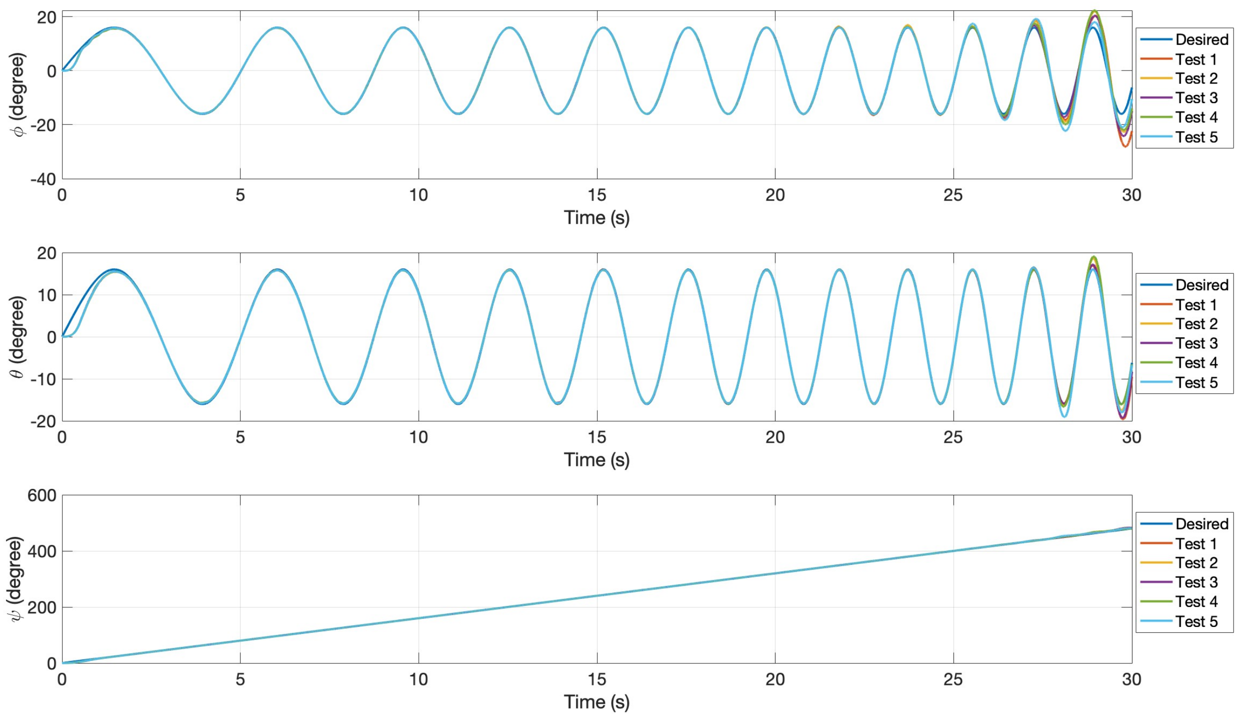


Figure C.9: 3 DOF Hover attitude response for a linear chirp reference with an amplitude of 16° with the ISMC.

C.2 MATLAB Code for the 3 DOF Hover

```
% 3 DOF HOVER Control Lab:
%
% SETUP_LAB_HOVER sets the model parameters.
%
% Copyright (C) 2010 Quanser Consulting Inc.
% Quanser Consulting Inc.
%

%% Amplifier Configuration
% Amplifier gain used for yaw and pitch axes.
K_AMP = 3;
% Amplifier Maximum Output Voltage (V)
VMAX_AMP = 24;
% Digital-to-Analog Maximum Voltage (V): set to 10 for Q4/Q8 cards
VMAX_DAC = 10;
%
% Filter and Rate Limiter Settings
% Specifications of a second-order low-pass filter
wcf = 2 * pi * 20; % filter cutting frequency
zetaf = 0.6;         % filter damping ratio
%
% Maximum Rate of Desired Position (rad/s)
CMD_RATE_LIMIT = 60 * pi/180; % 60 deg/s converted to rad/s

% Set the model parameters of the 3DOF HOVER.
% These parameters are used for model representation and controller
    design.
% Gravitational Constant (m/s^2)
g = 9.81;
% Motor Armature Resistance (Ohm)
Rm = 0.83;
% Motor Current-Torque Constant (N.m/A)
Kt_m = 0.0182;
% Motor Rotor Moment of Inertia (kg.m^2)
Jm = 1.91e-6;
J_r = Jm;
% Moving Mass of the Hover system (kg)
```

```
m_hover = 2.85;
% Mass of each Propeller Section = motor + shield + propeller +
   body (kg)
m_prop = m_hover / 4;
% Distance between Pivot to each Motor (m)
L = 7.75*0.0254;
% Propeller Force-Thrust Constant found Experimentally (N/V)
Kf = 0.1188;
cf = 7.32e-5;
% Propeller Torque-Thrust Constant found Experimentally (N-m/V)
Kt = 0.0036;
ct = 3.46e-6;
% note: front/back motor are counter-clockwise (negative torque)
   and
% left/right motor are clockwise (positive torque).
%
% Equivalent Moment of Inertia of each Propeller Section (kg.m^2)
Jeq_prop = Jm + m_prop*L^2;
% Equivalent Moment of Inertia about each Axis (kg.m^2)
Jp = 2*Jeq_prop;
Jy = 4*Jeq_prop;
Jr = 2*Jeq_prop;
I_x = Jr;
I_y = Jp;
I_z = Jy;
%
Kv = 54.945;
% Pitch and Yaw Axis Encoder Resolution (rad/count)
K_EC_Y = -2 * pi / ( 8 * 1024 );
K_EC_P = 2 * pi / ( 8 * 1024 );
K_EC_R = 2 * pi / ( 8 * 1024 );
% Bias voltage applied to motors (V)
V_bias = 4.0;


%% ISMC parameters
% Roll
lambda_phi = 3;
rho_phi = 40;
```

```
epsilon_phi = 0.05;
k_phi = 0.001;
zeta_phi = 35;

% Pitch
lambda_theta = 3;
rho_theta = 40;
epsilon_theta = 0.05;
k_theta = 0.001;
zeta_theta = 35;

% Yaw
lambda_psi = 10;
rho_psi = 5;
epsilon_psi = 0.05;
k_psi = 0.001;
zeta_psi = 2;

%% PID parameters
% Roll
kp_phi = 30;
kd_phi = 10;

% Pitch
kp_theta = 30;
kd_theta = 10;

% Yaw
kp_psi = 25;
kd_psi = 5;

%% LQR controllers
% Roll
A_roll = [0 1; 0 0];
B_roll = [0; 1/I_x];
C_roll = [1 0];
D_roll = [0];

Q_roll = [950 0;0 150];
```

```
R_roll = [0.1];
k_phi= lqr(A_roll,B_roll,Q_roll,R_roll);

% Pitch
A_pitch = [0 1; 0 0];
B_pitch = [0; 1/I_y];
C_pitch = [1 0];
D_pitch = [0];

Q_pitch = [950 0;0 150];
R_pitch = [0.1];
k_theta= lqr(A_pitch,B_pitch,Q_pitch,R_pitch);

% Yaw
A_yaw = [0 1; 0 0];
B_yaw = [0; 1/I_z];
C_yaw = [1 0];
D_yaw = [0];

Q_yaw = [1000 0;0 100];
R_yaw = [0.1];
k_psi = lqr(A_yaw,B_yaw,Q_yaw,R_yaw);
```

Listing C.1: MATLAB code for initializing variables/constants used by the 3 DOF Hover.

C.3 3 DOF Hover Simulink Schemes

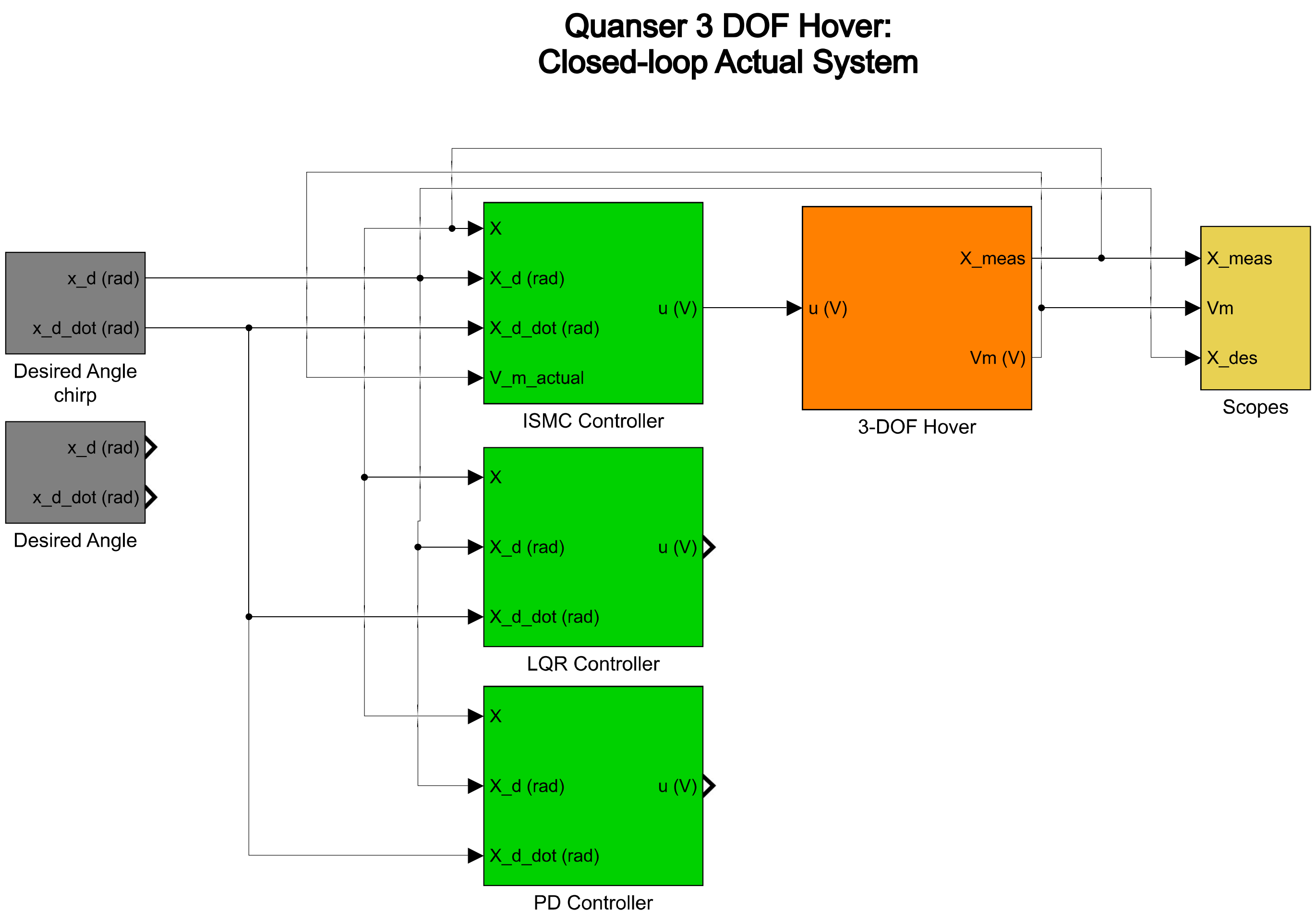


Figure C.10: Quanser 3 DOF Hover system simulink scheme.

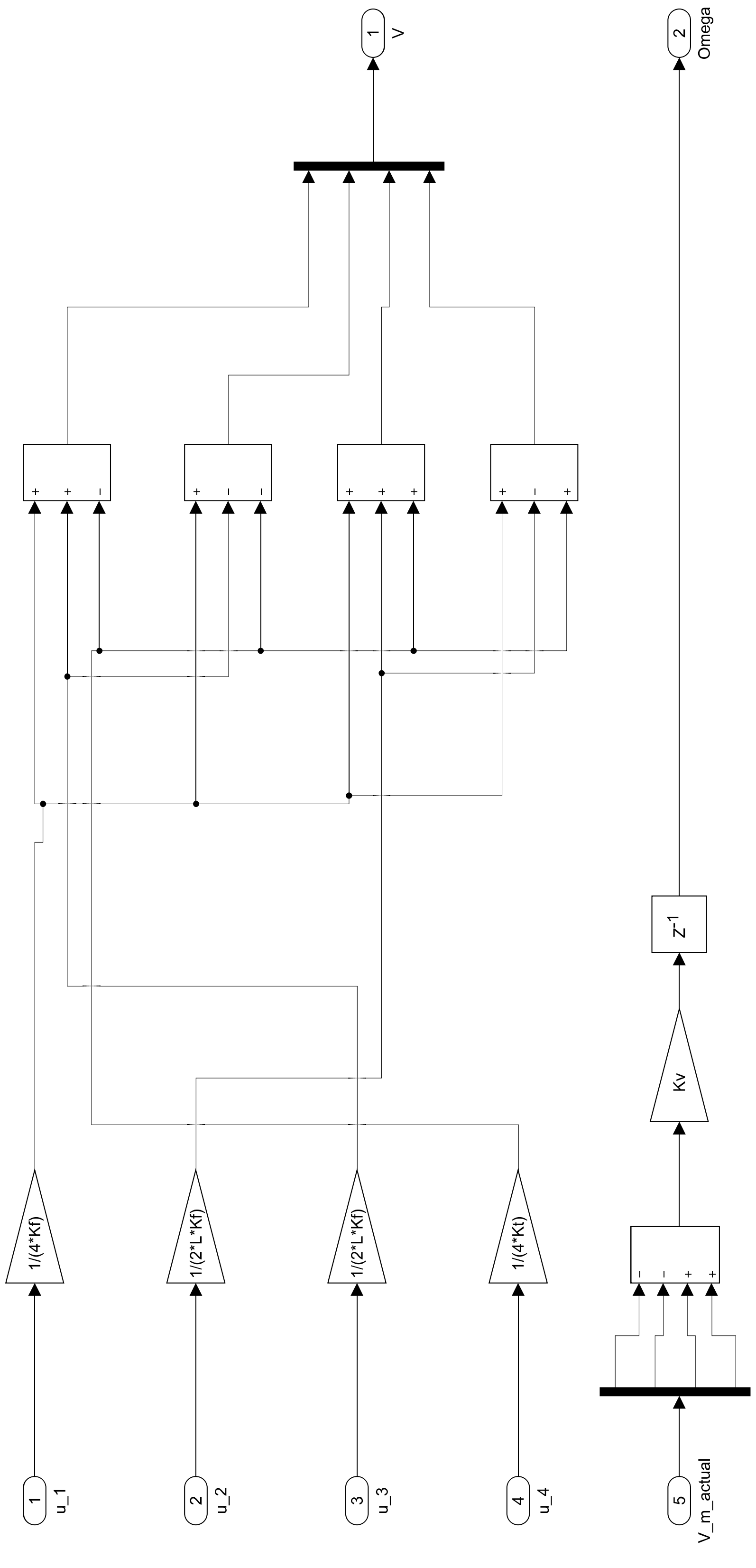


Figure C.11: Converter from control inputs to voltages for the 3 DOF Hover.

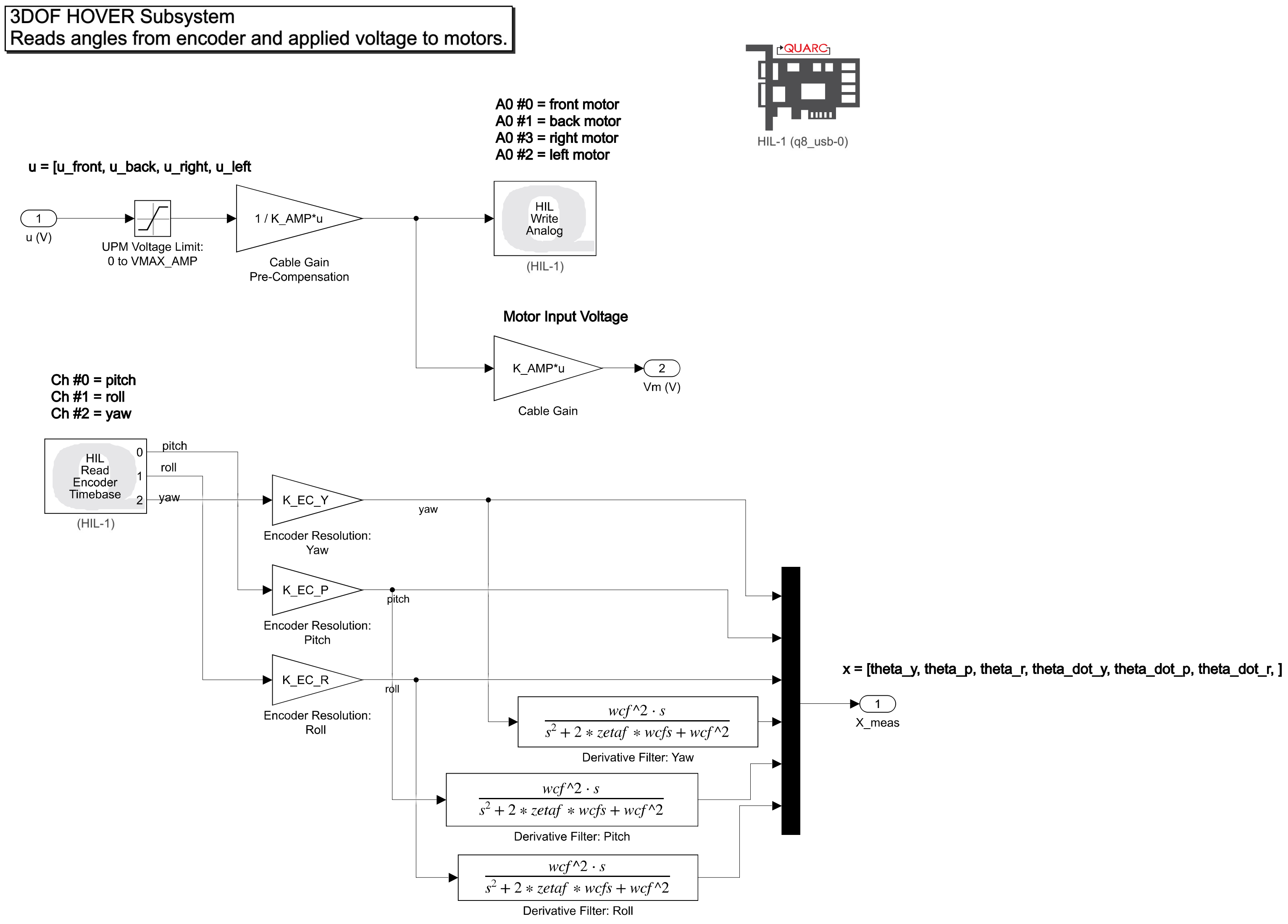


Figure C.12: 3 DOF Hover subsystem.

Scopes

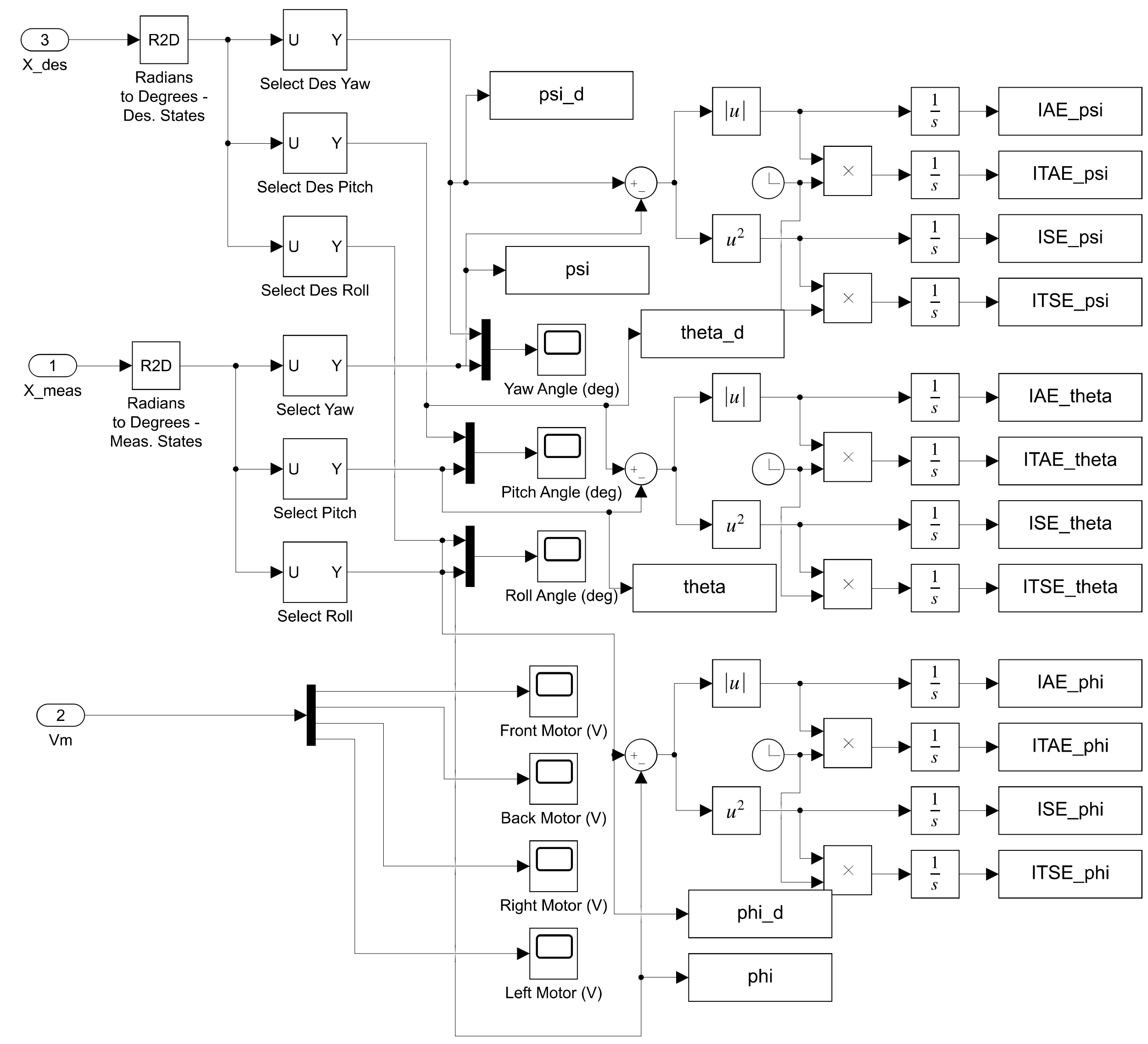


Figure C.13: 3 DOF Hover scopes.

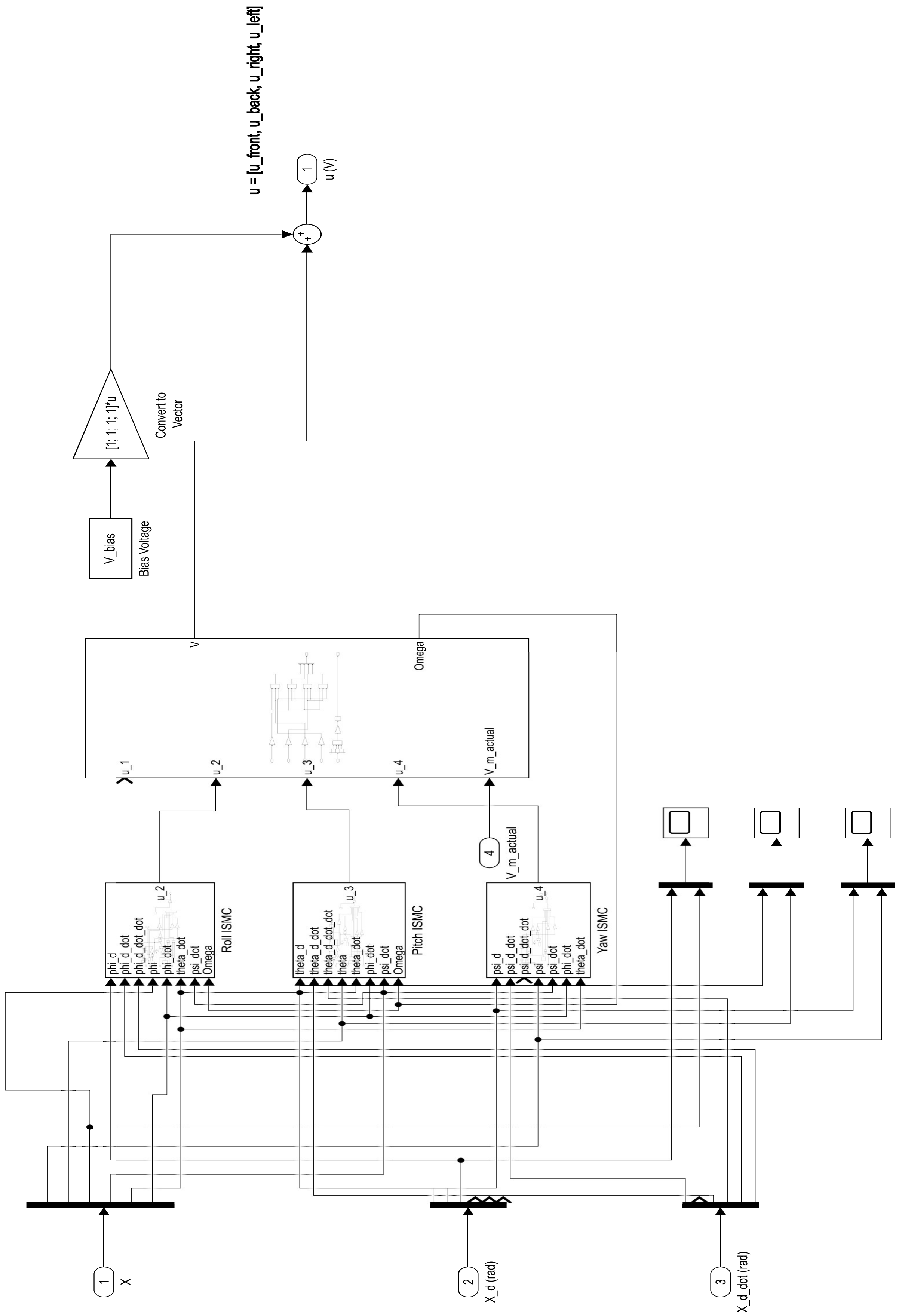


Figure C.14: 3 DOF ISMC controller.

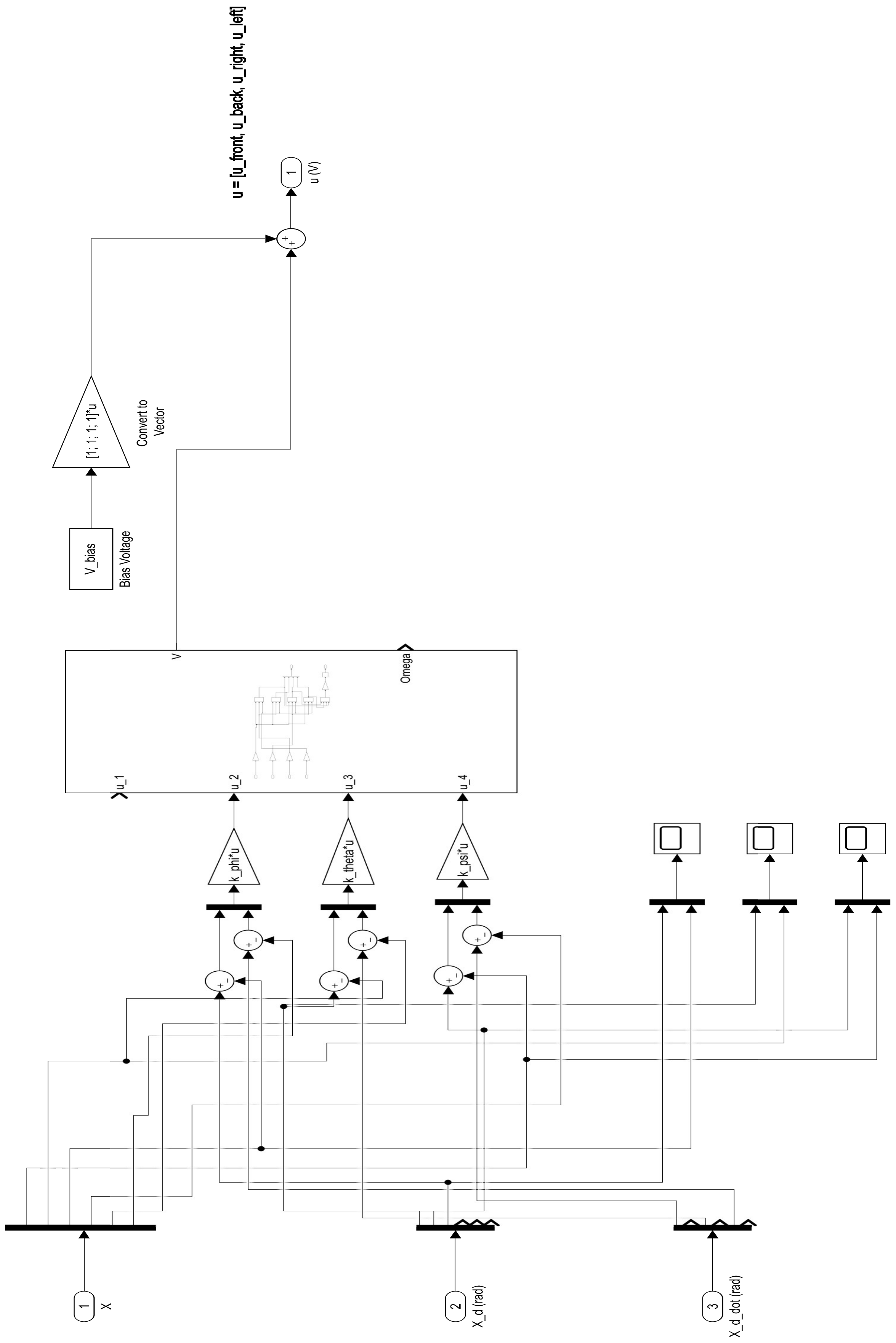


Figure C.15: 3 DOF LQR controller.

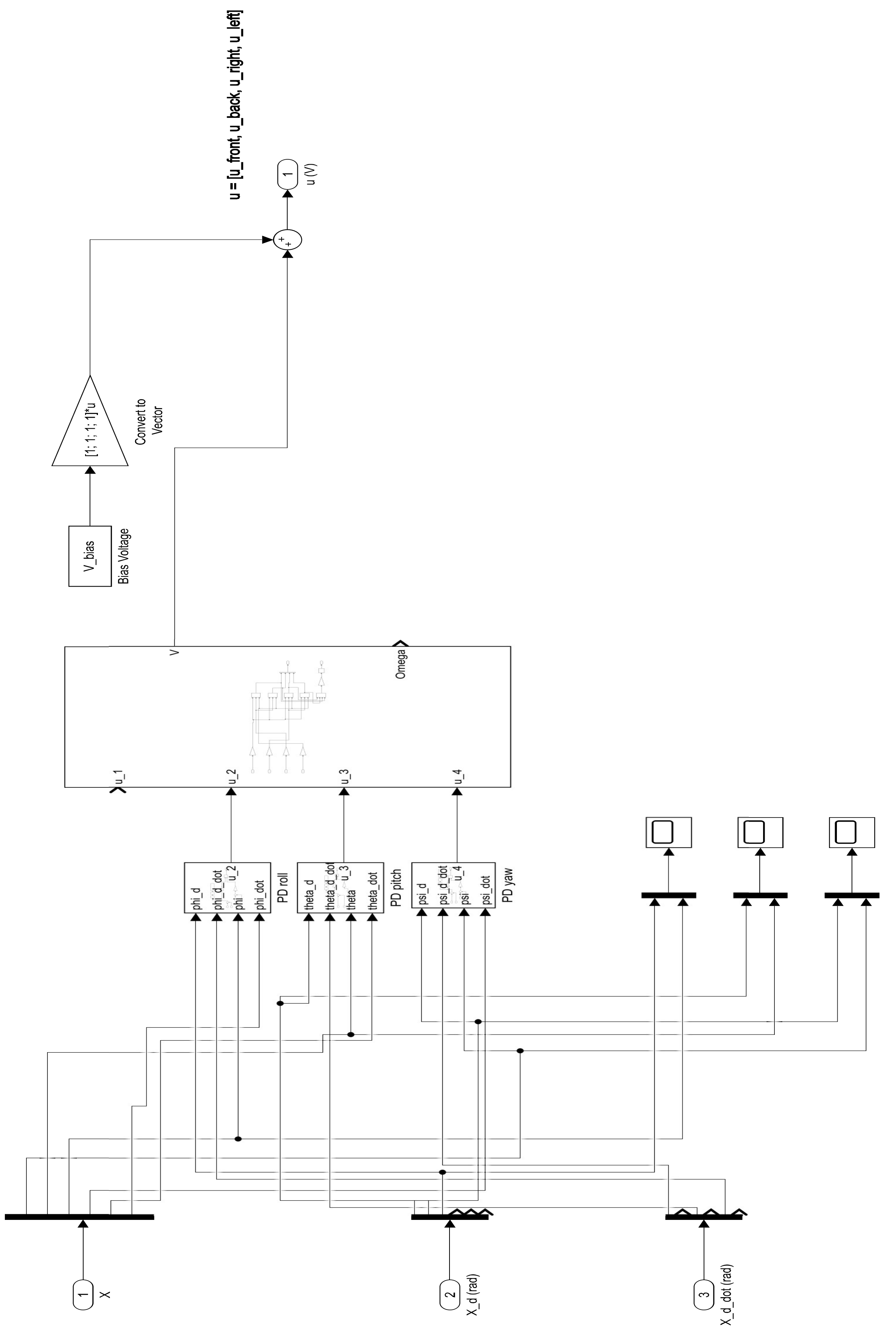


Figure C.16: 3 DOF PD controller.

Appendix D

Parrot Minidrone

D.1 Experimental Results of Parrot Minidrone Simulations

D.1.1 First Track Results

Figures

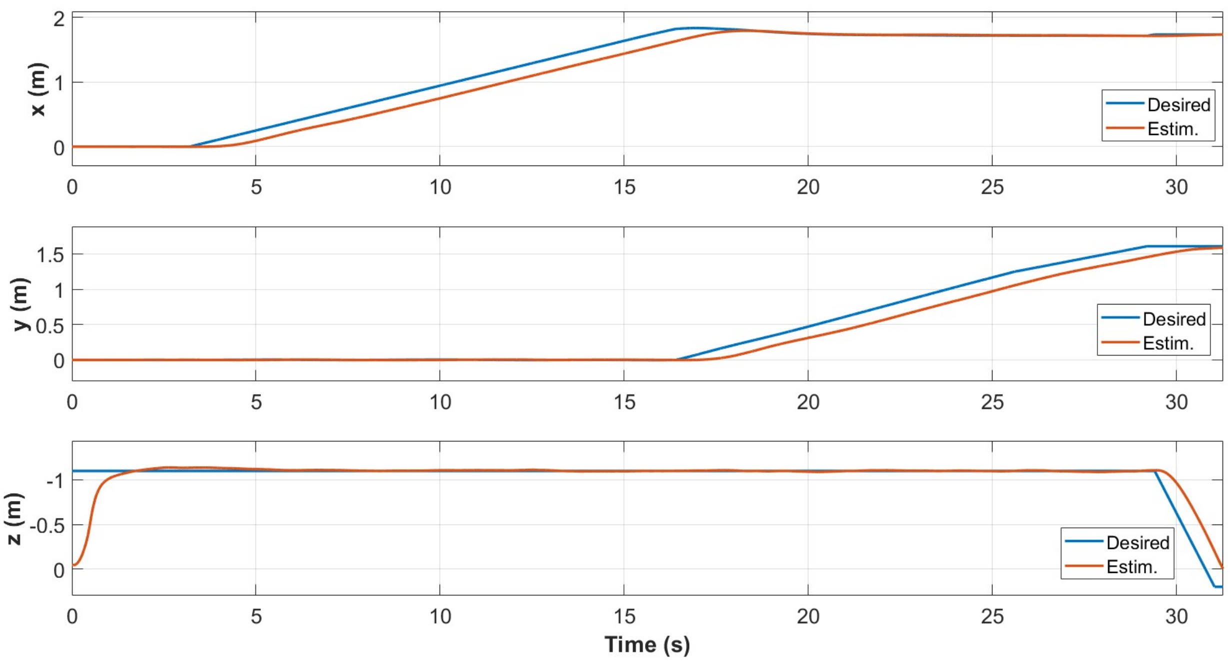


Figure D.1: Position w/ PID control during the first test with max speed set to 0.0007 m/sample.

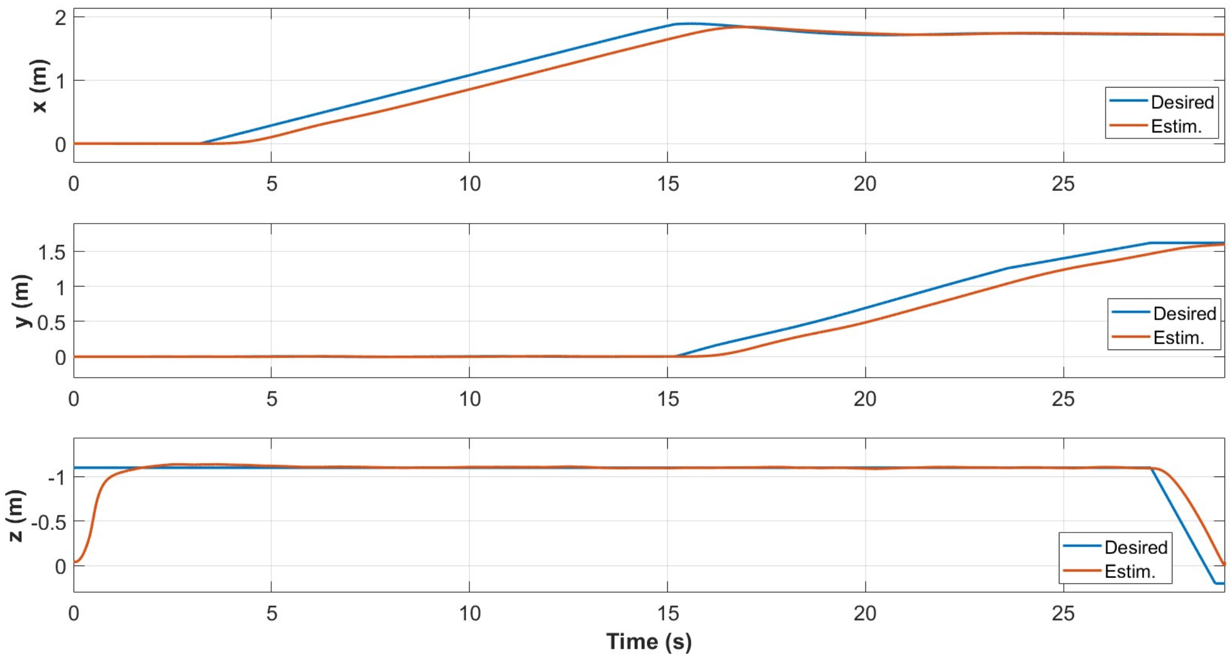


Figure D.2: Position w/ PID control during the first test with max speed set to 0.0008 m/sample.

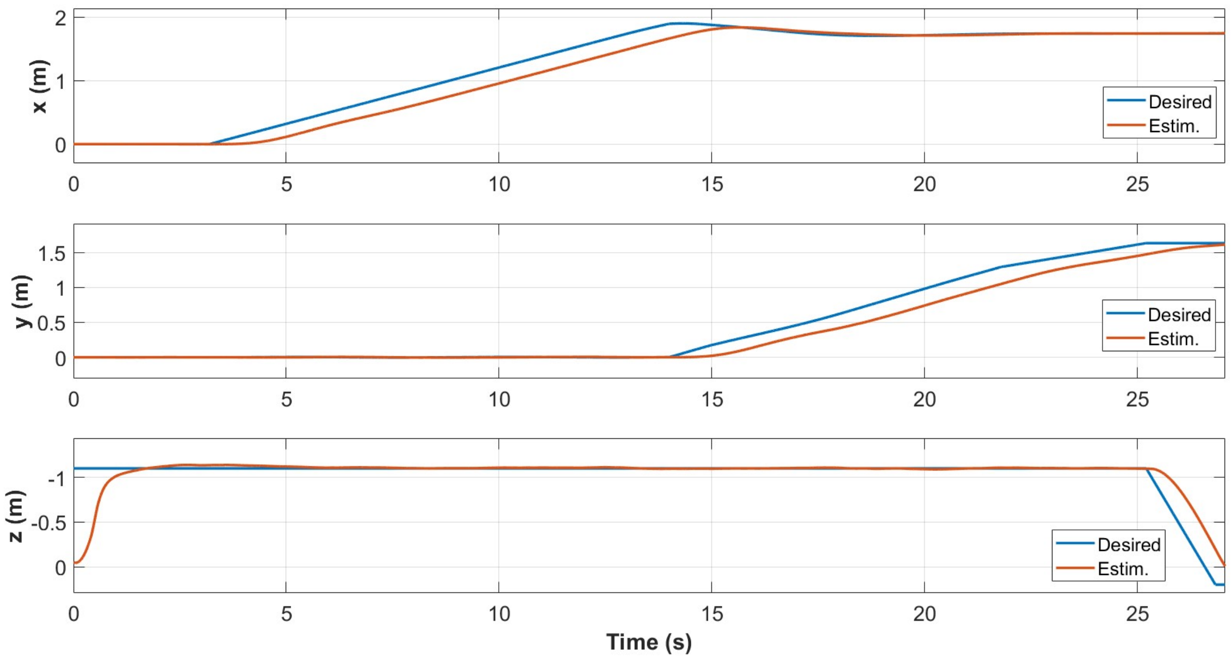


Figure D.3: Position w/ PID control during the first test with max speed set to 0.0009 m/sample.

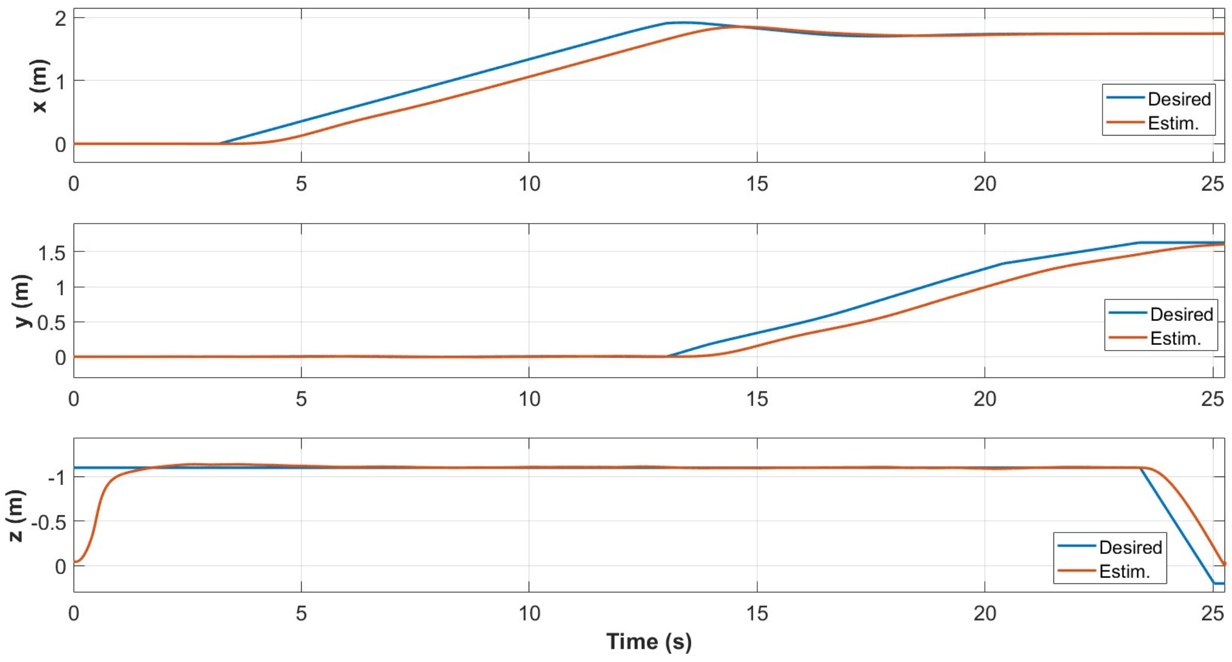


Figure D.4: Position w/ PID control during the first test with max speed set to 0.0010 m/sample.

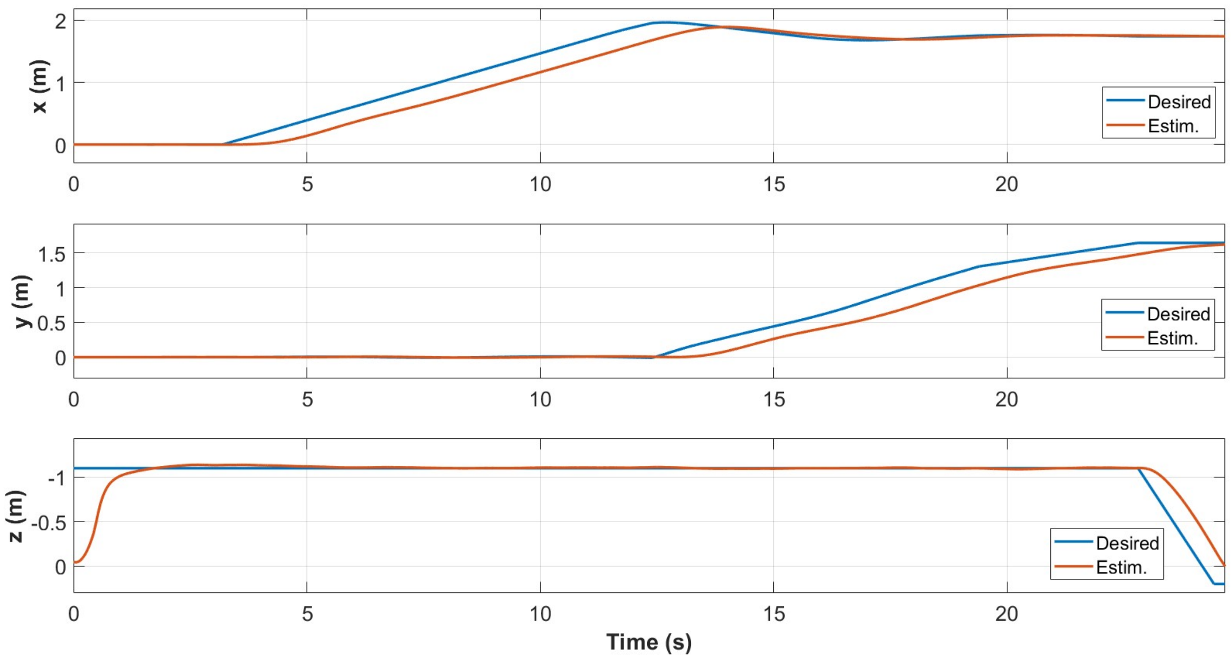


Figure D.5: Position w/ PID control during the first test with max speed set to 0.0011 m/sample.

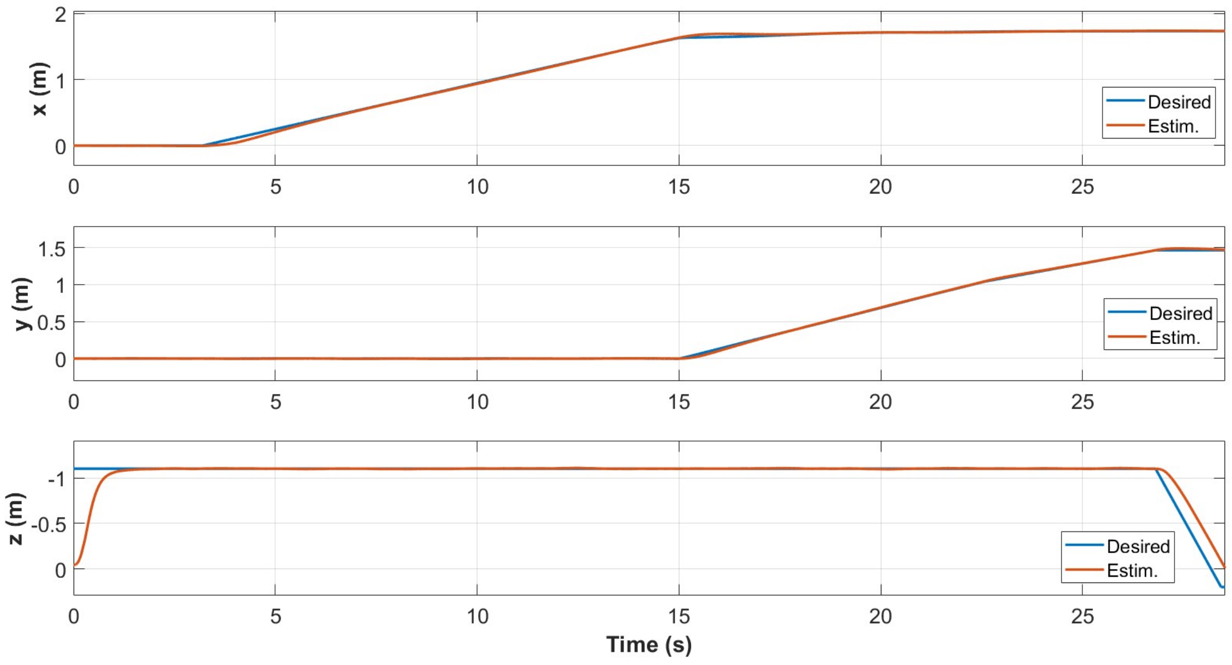


Figure D.6: Position w/ ISMC control during the first test with max speed set to 0.0007 m/sample.

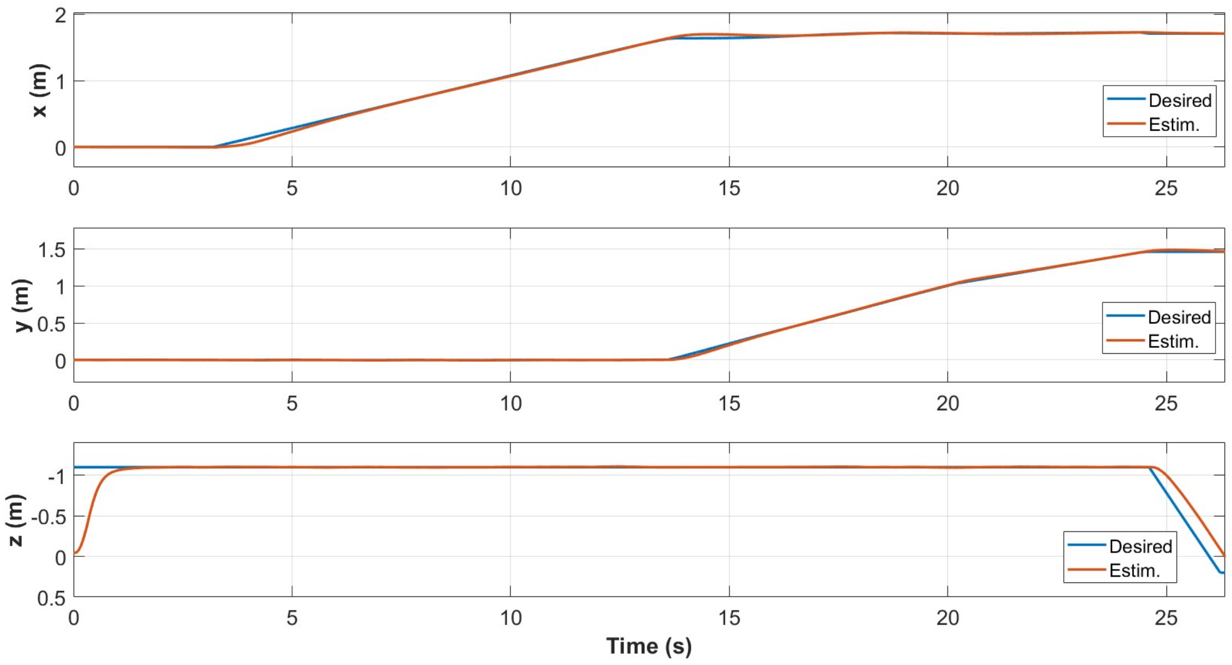


Figure D.7: Position w/ ISMC control during the first test with max speed set to 0.0008 m/sample.

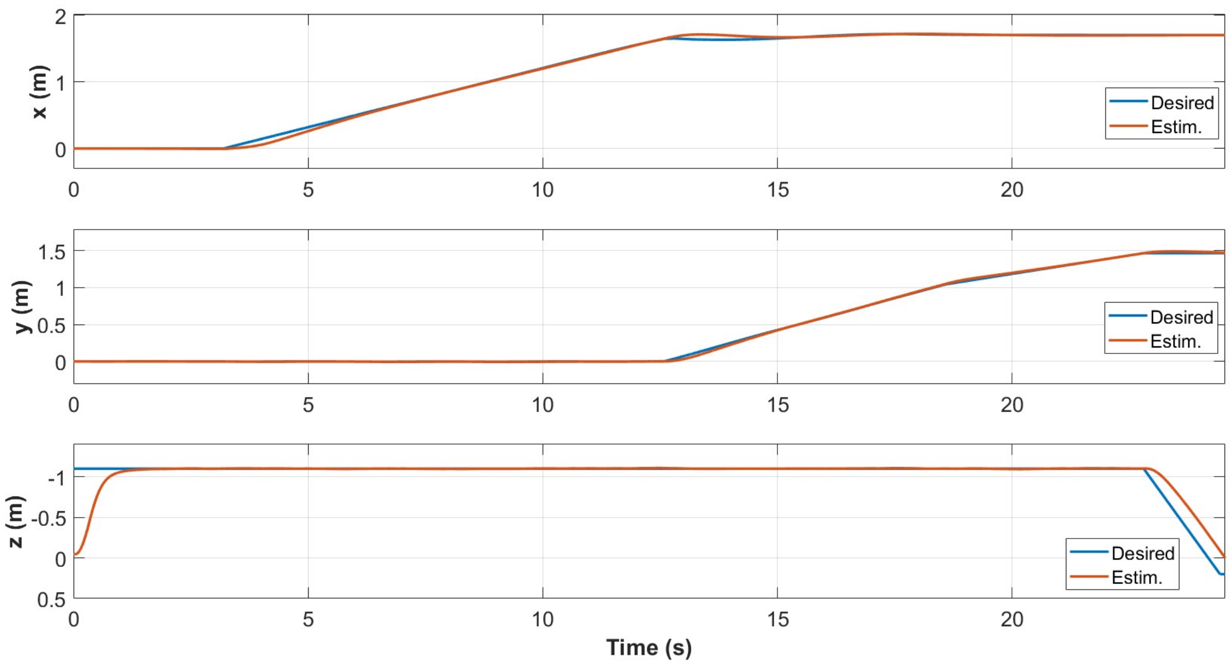


Figure D.8: Position w/ ISMC control during the first test with max speed set to 0.0009 m/sample.

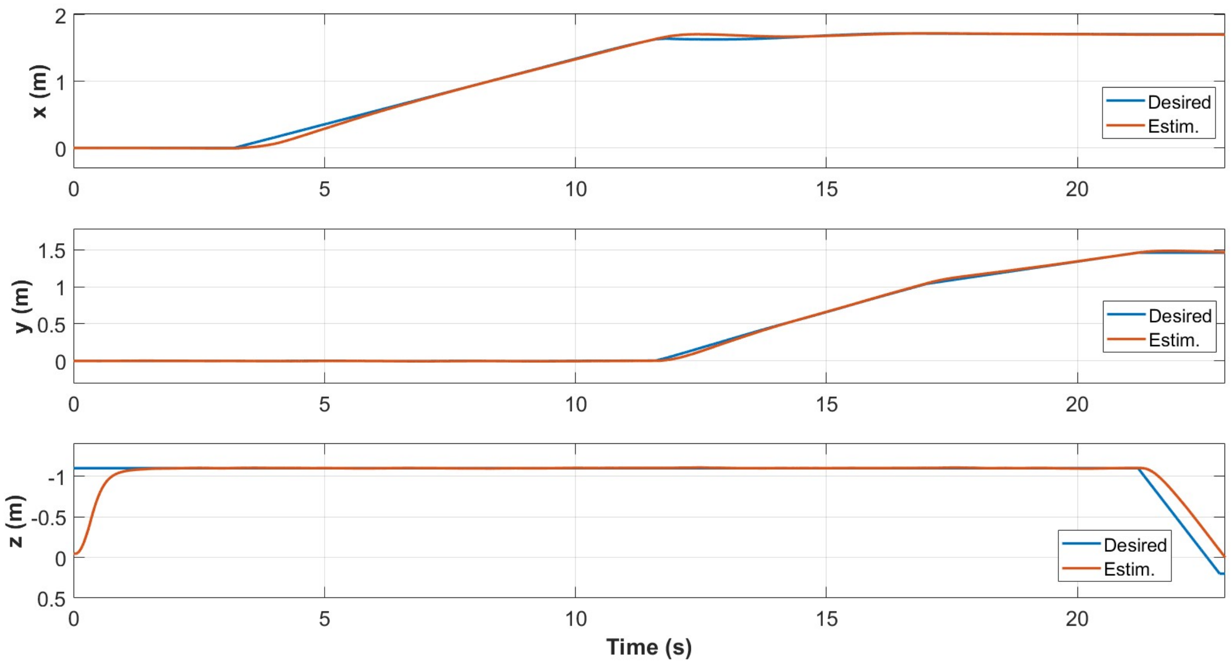


Figure D.9: Position w/ ISMC control during the first test with max speed set to 0.0010 m/sample.

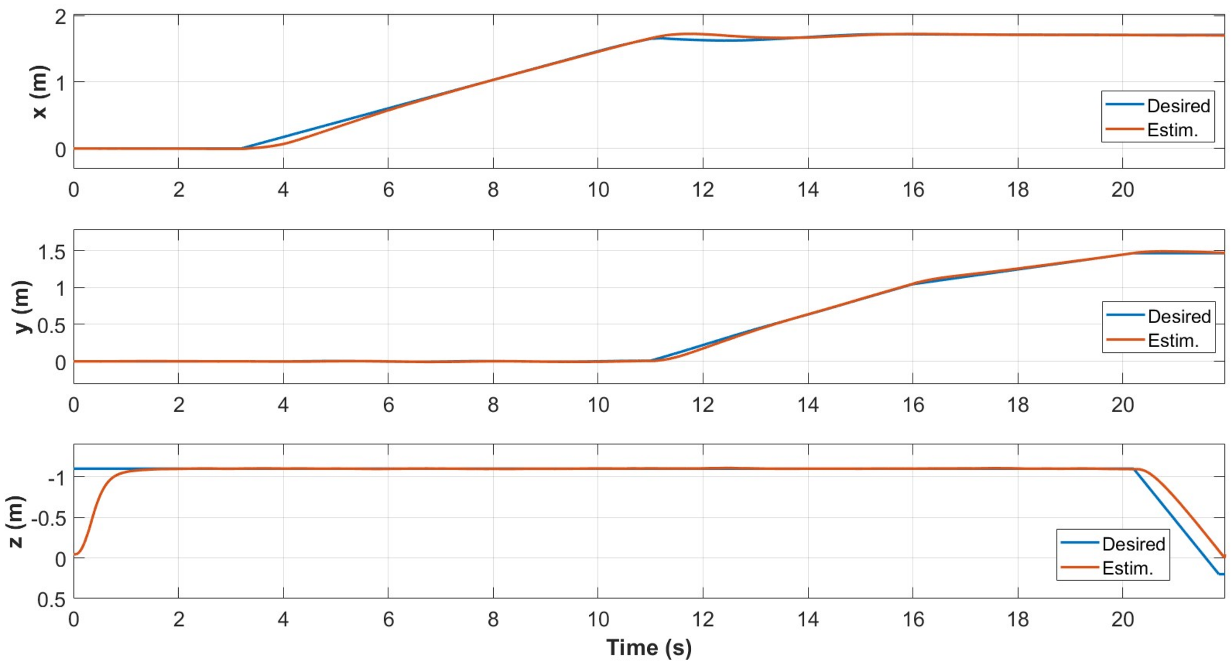


Figure D.10: Position w/ ISMC control during the first test with max speed set to 0.0011 m/sample.

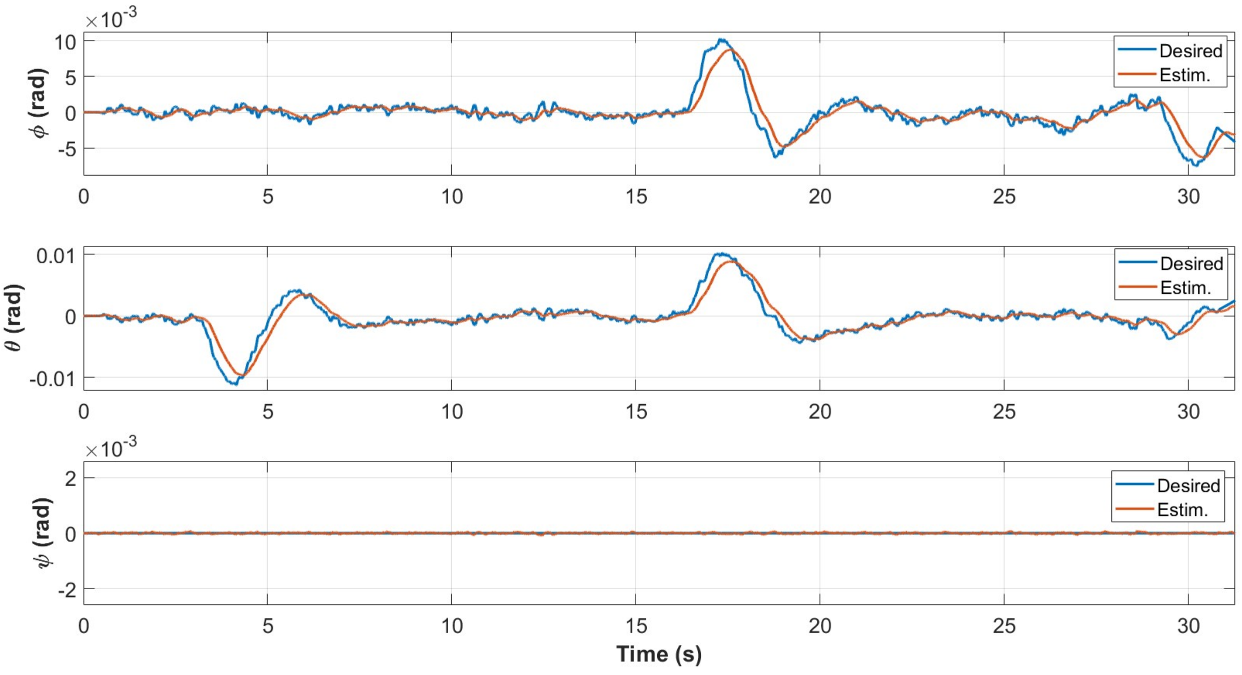


Figure D.11: Attitude w/ PID control during the first test with max speed set to 0.0007 m/sample.

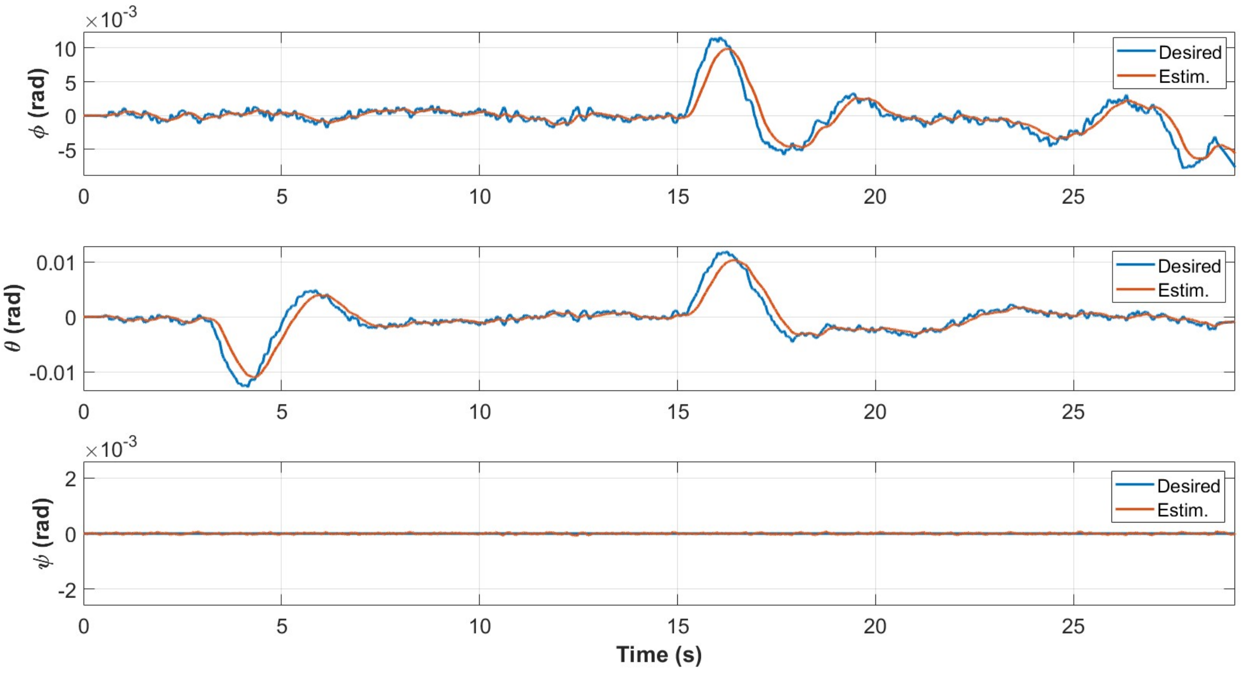


Figure D.12: Attitude w/ PID control during the first test with max speed set to 0.0008 m/sample.

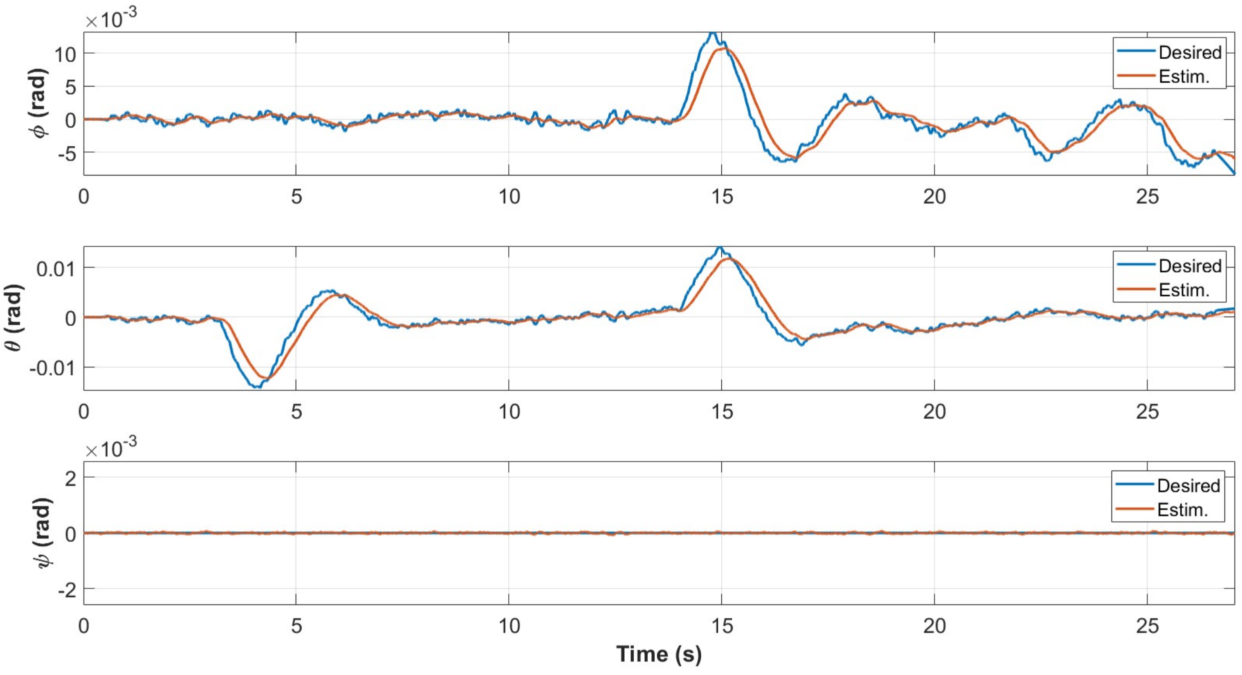


Figure D.13: Attitude w/ PID control during the first test with max speed set to 0.0009 m/sample.

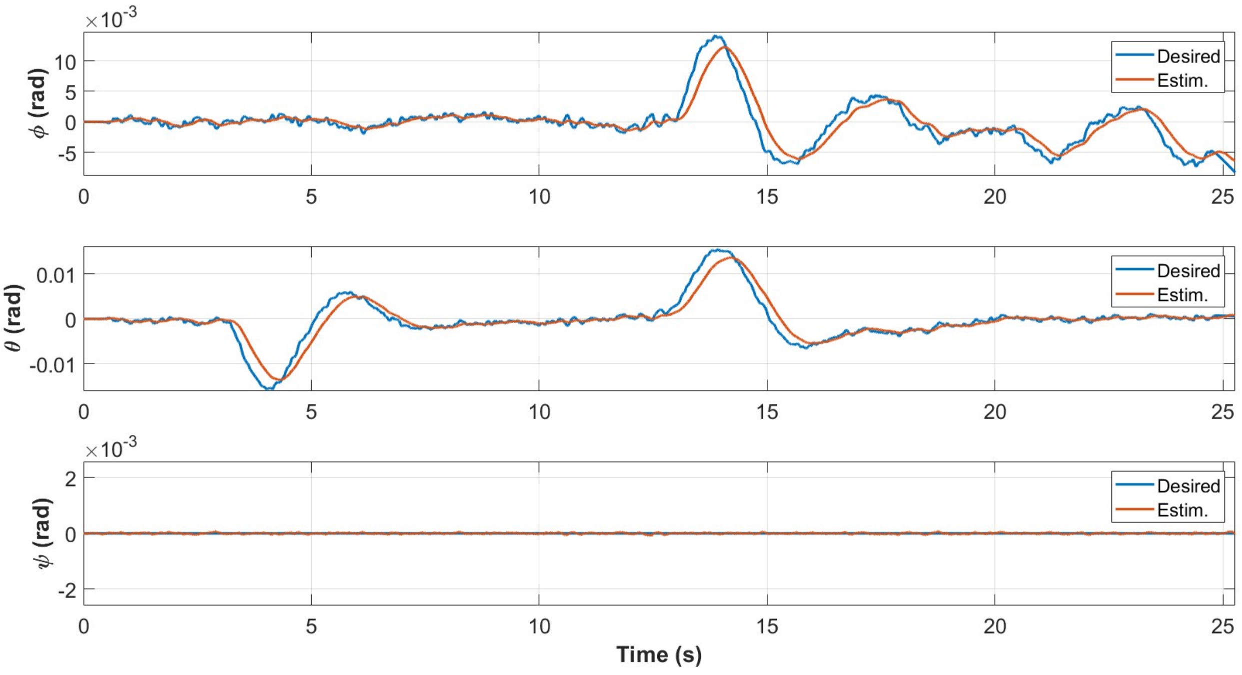


Figure D.14: Attitude w/ PID control during the first test with max speed set to 0.0010 m/sample.

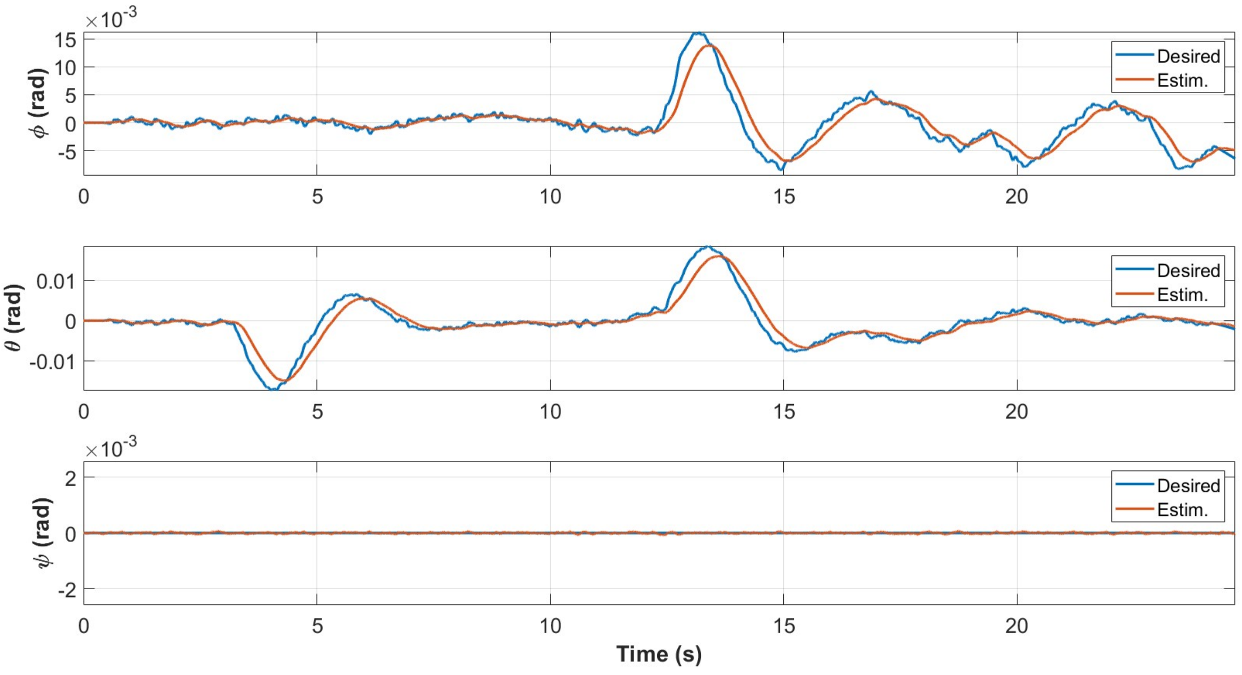


Figure D.15: Attitude w/ PID control during the first test with max speed set to 0.0011 m/sample.

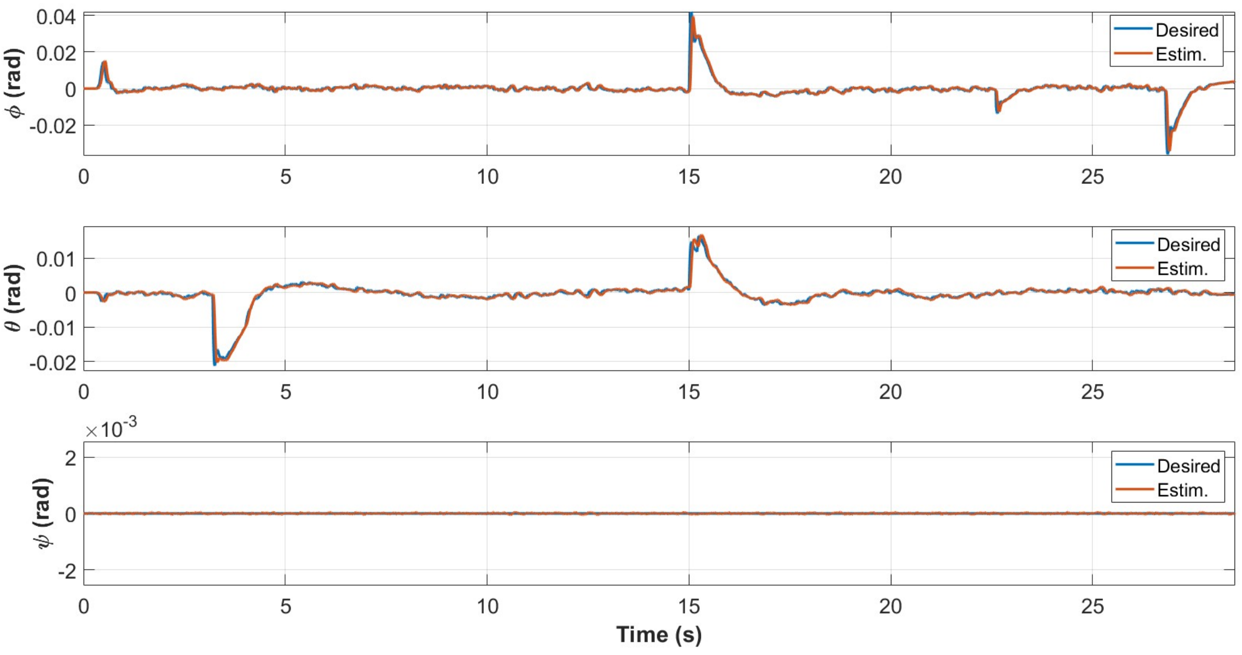


Figure D.16: Attitude w/ ISMC control during the first test with max speed set to 0.0007 m/sample.

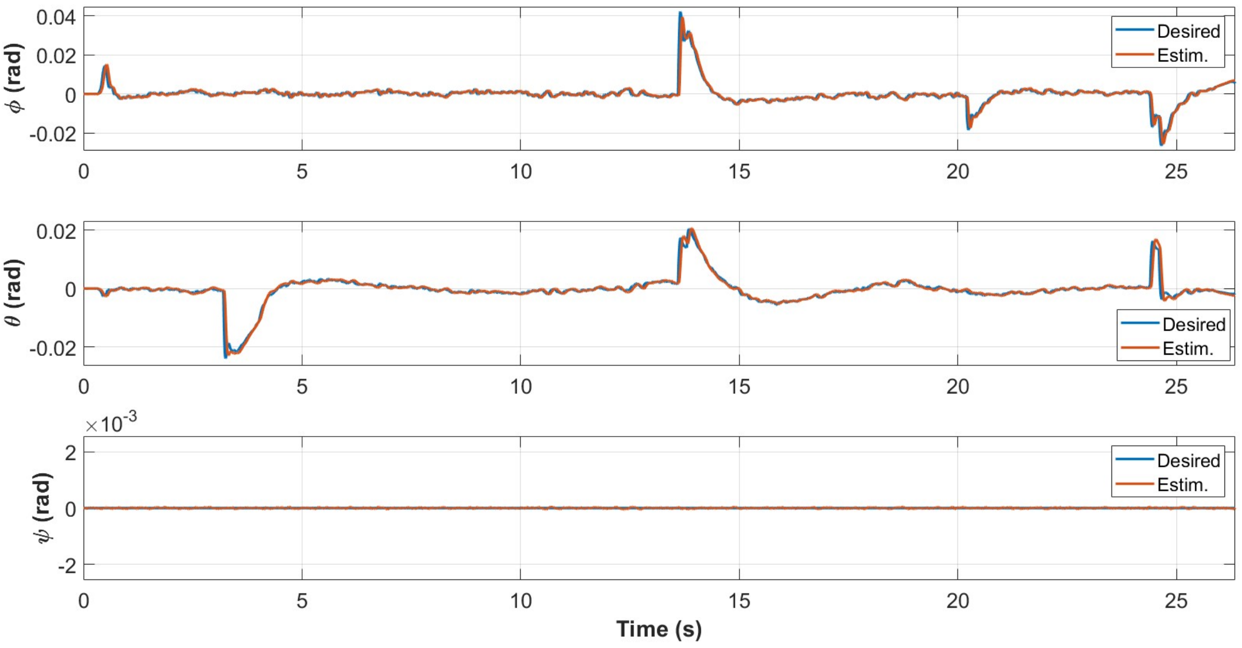


Figure D.17: Attitude w/ ISMC control during the first test with max speed set to 0.0008 m/sample.

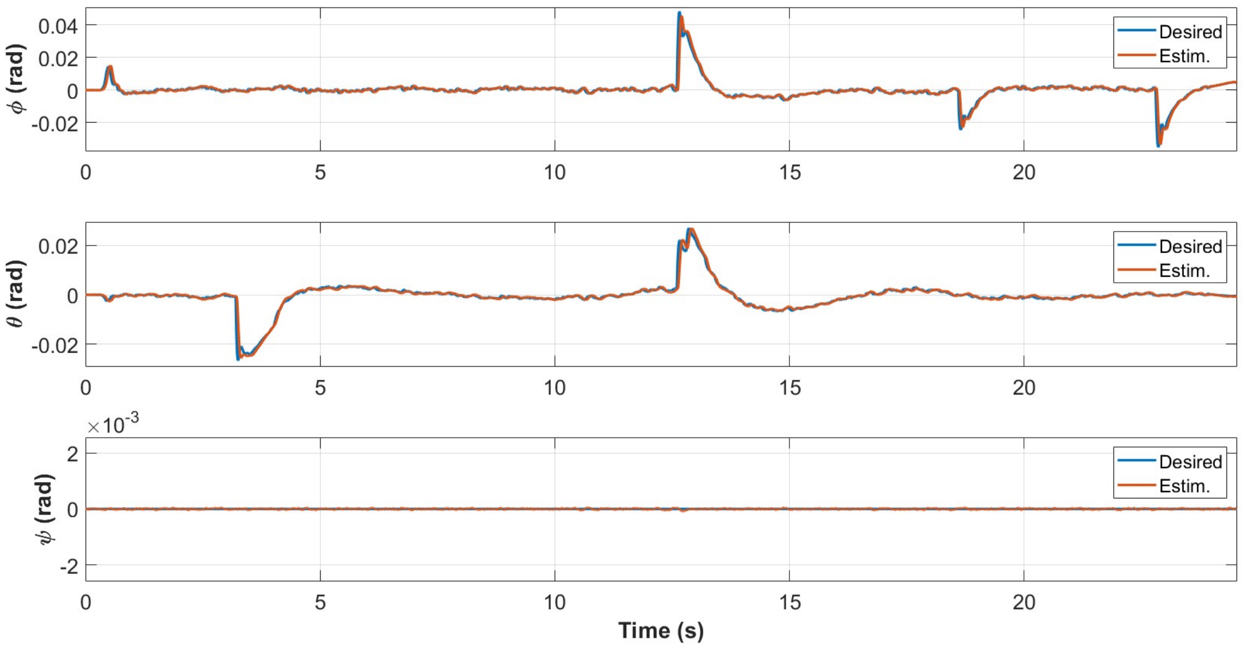


Figure D.18: Attitude w/ ISMC control during the first test with max speed set to 0.0009 m/sample.

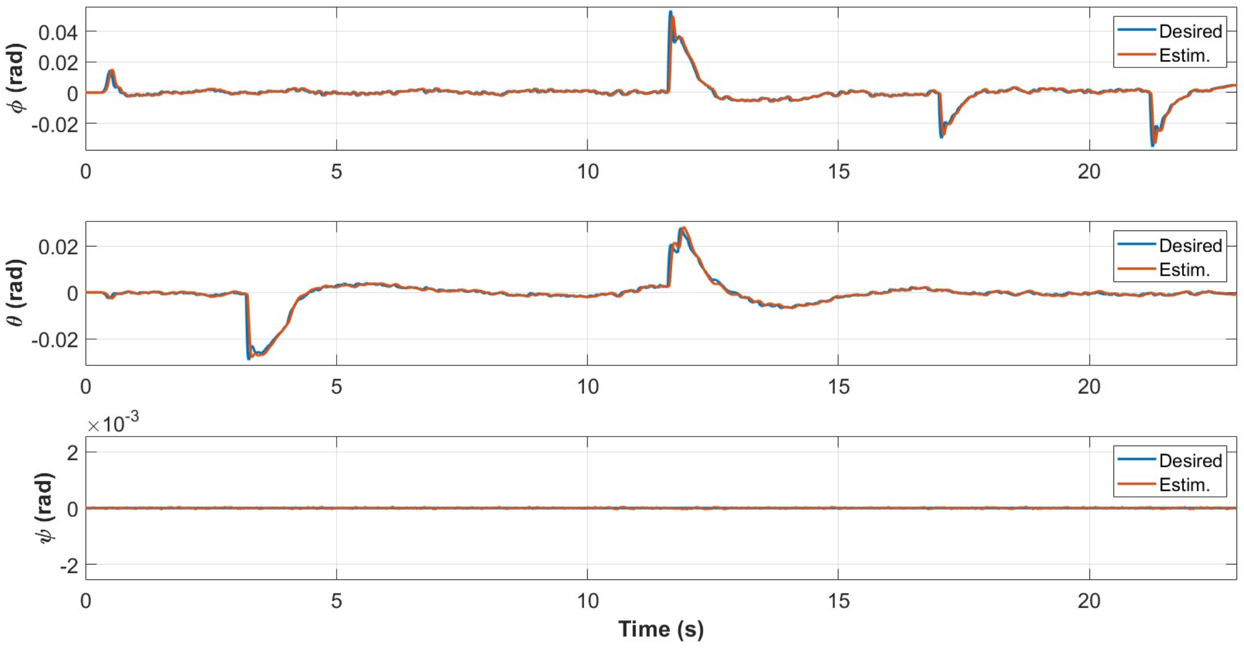


Figure D.19: Attitude w/ ISMC control during the first test with max speed set to 0.0010 m/sample.

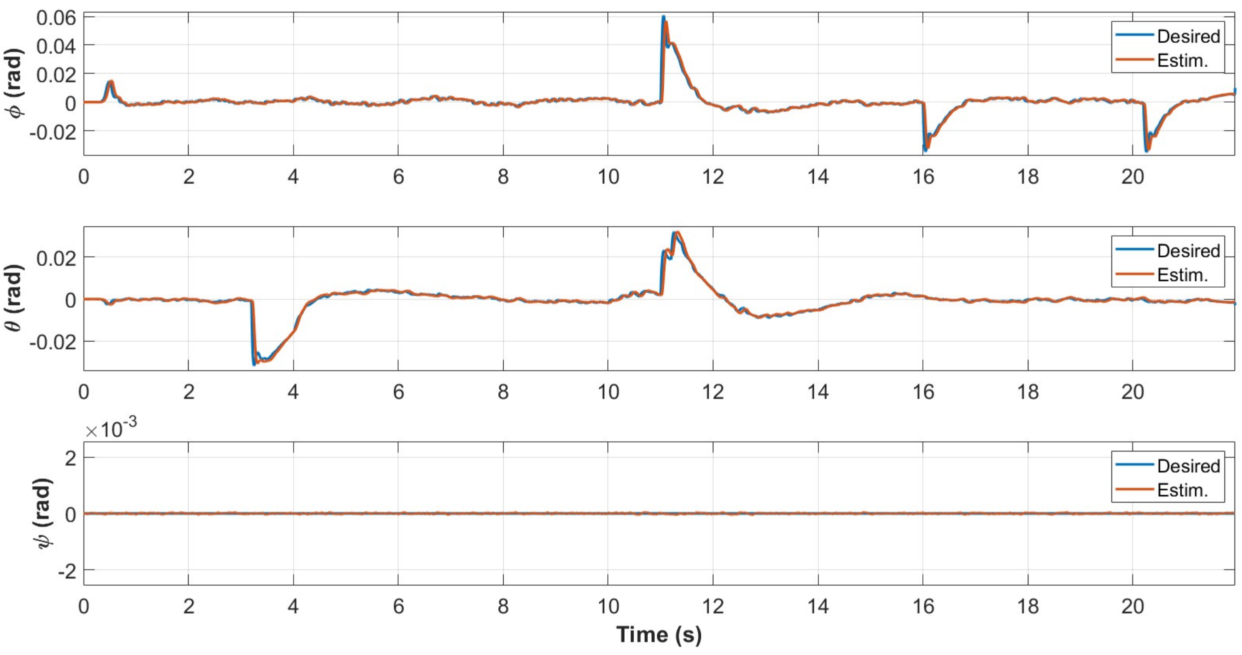


Figure D.20: Attitude w/ ISMC control during the first test with max speed set to 0.0011 m/sample.

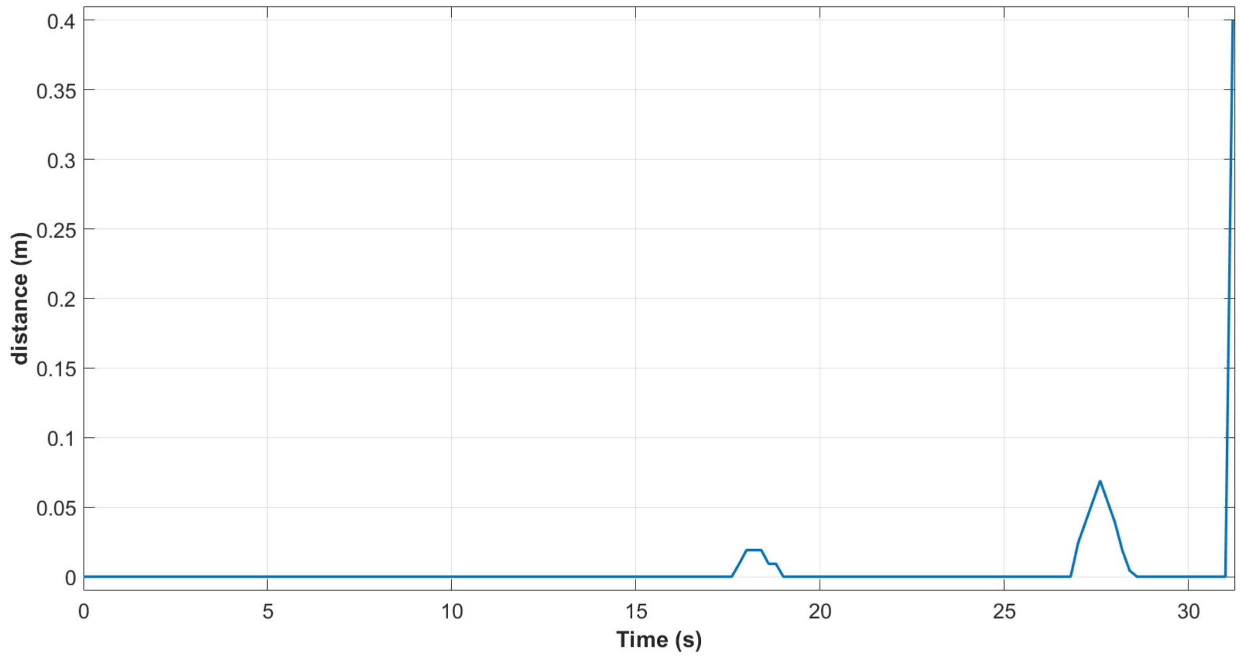


Figure D.21: Center displacement w/ PID control during the first test with max speed set to 0.0007 m/sample.

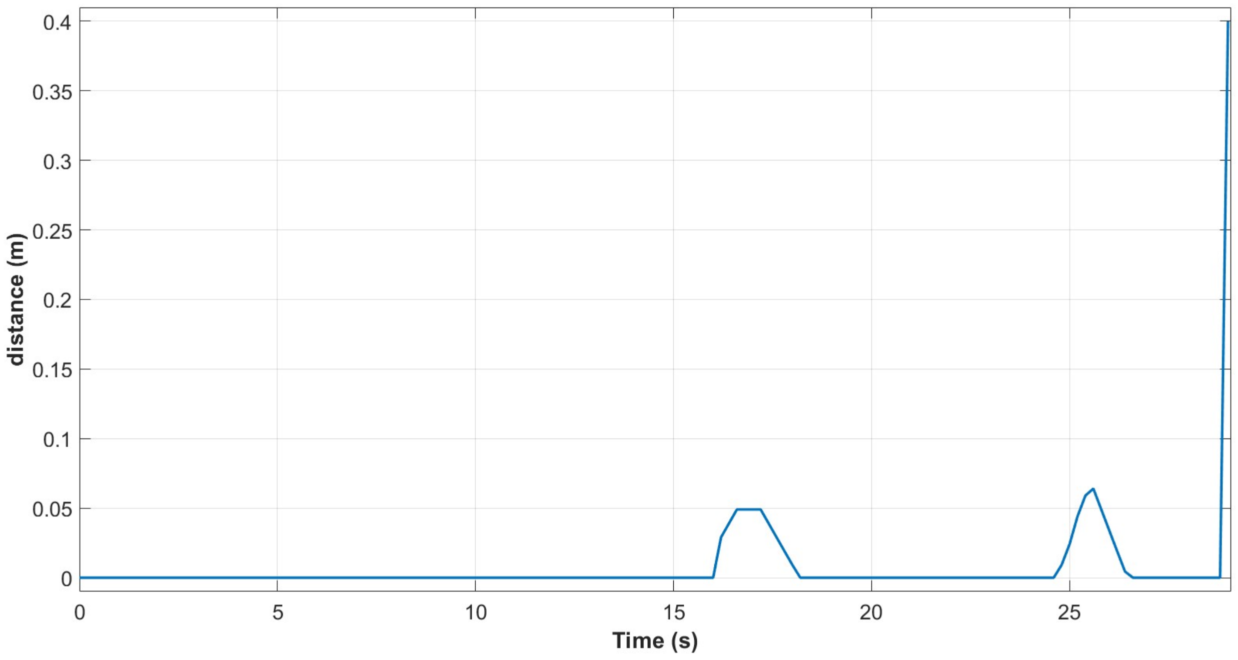


Figure D.22: Center displacement w/ PID control during the first test with max speed set to 0.0008 m/sample.

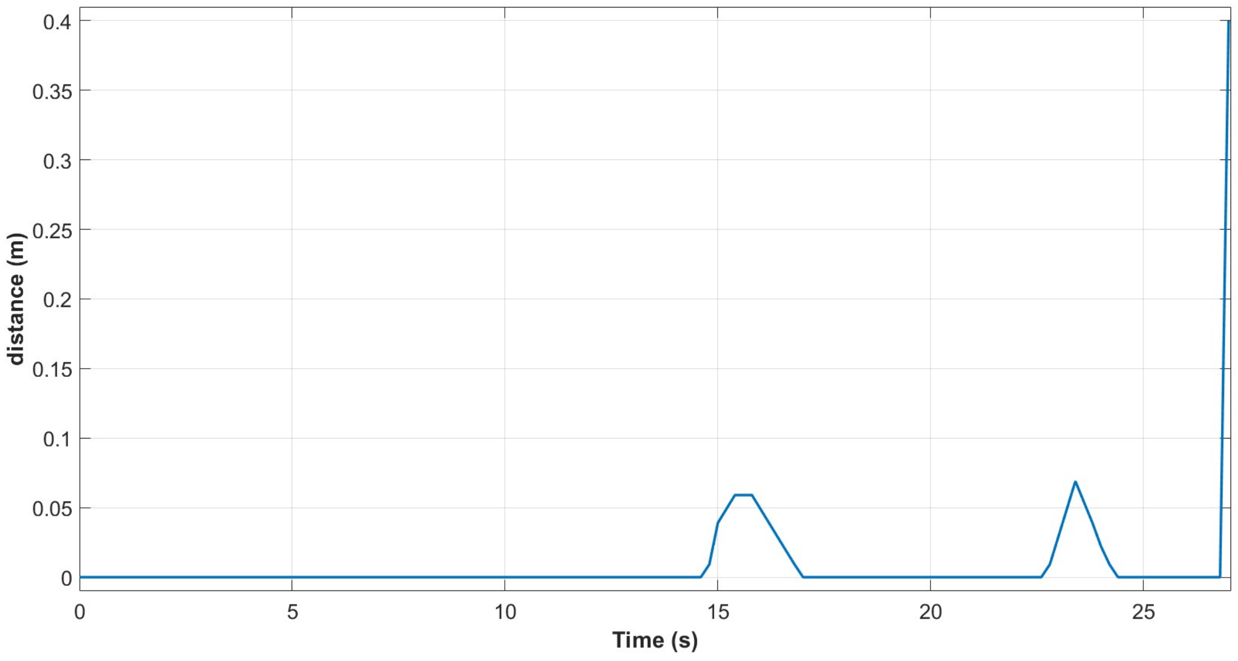


Figure D.23: Center displacement w/ PID control during the first test with max speed set to 0.0009 m/sample.

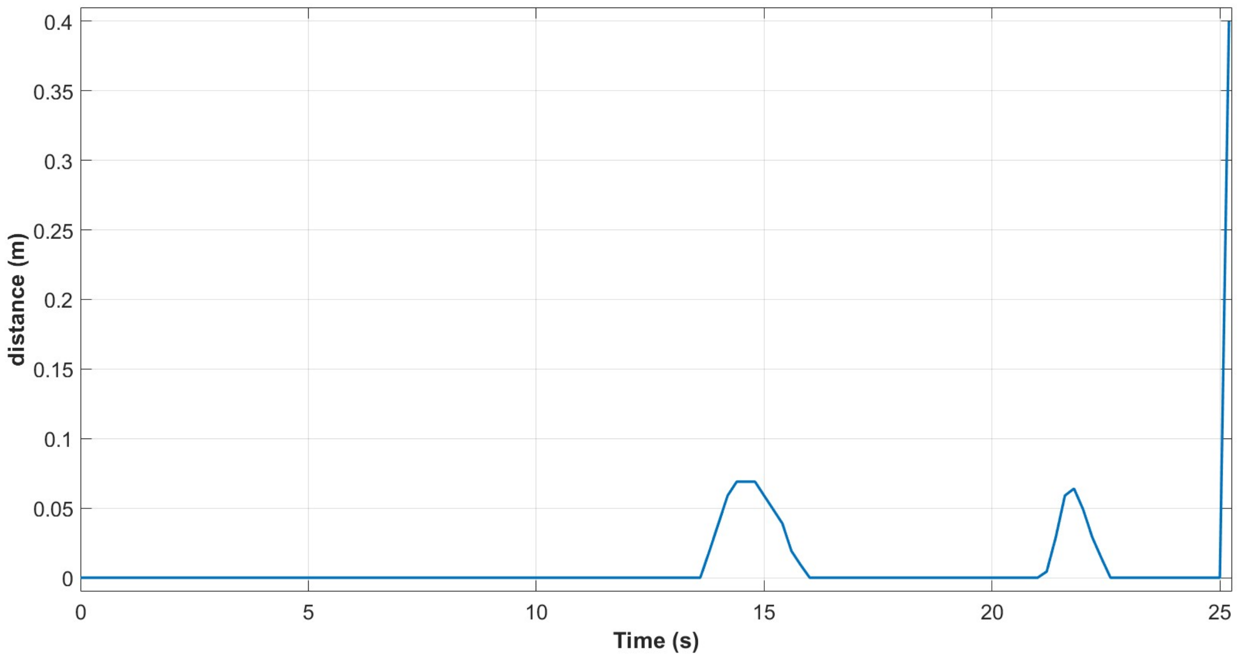


Figure D.24: Center displacement w/ PID control during the first test with max speed set to 0.0010 m/sample.

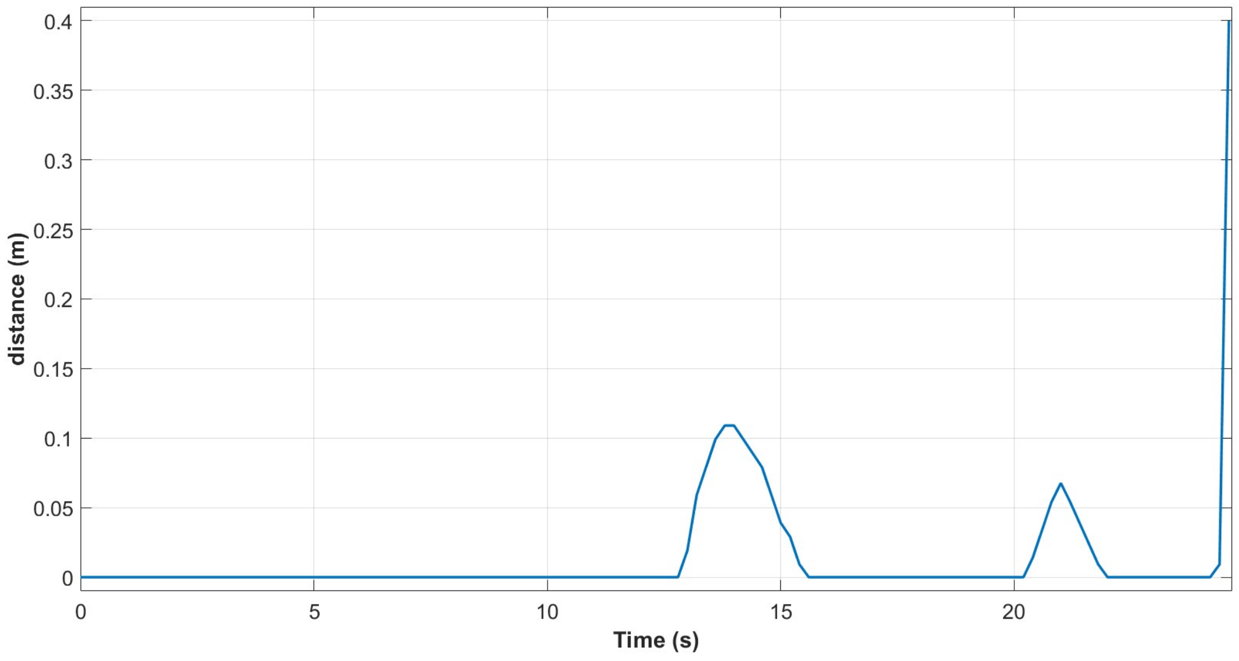


Figure D.25: Center displacement w/ PID control during the first test with max speed set to 0.0011 m/sample.

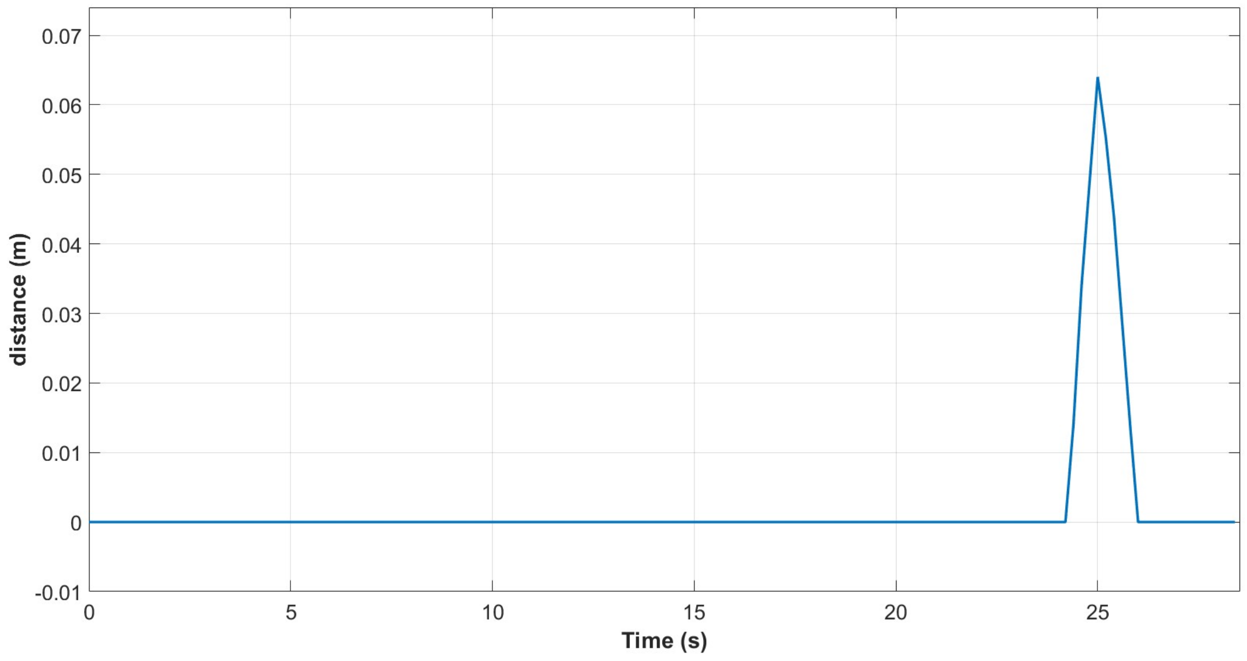


Figure D.26: Center displacement w/ ISMC control during the first test with max speed set to 0.0007 m/sample.

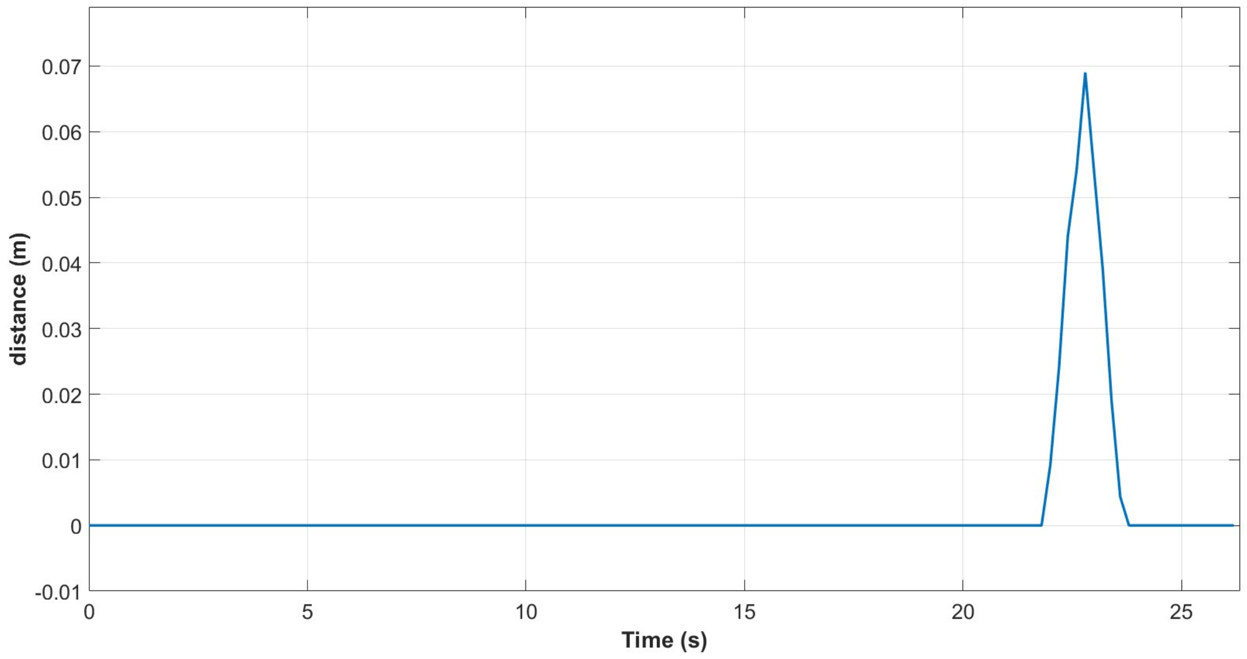


Figure D.27: Center displacement w/ ISMC control during the first test with max speed set to 0.0008 m/sample.

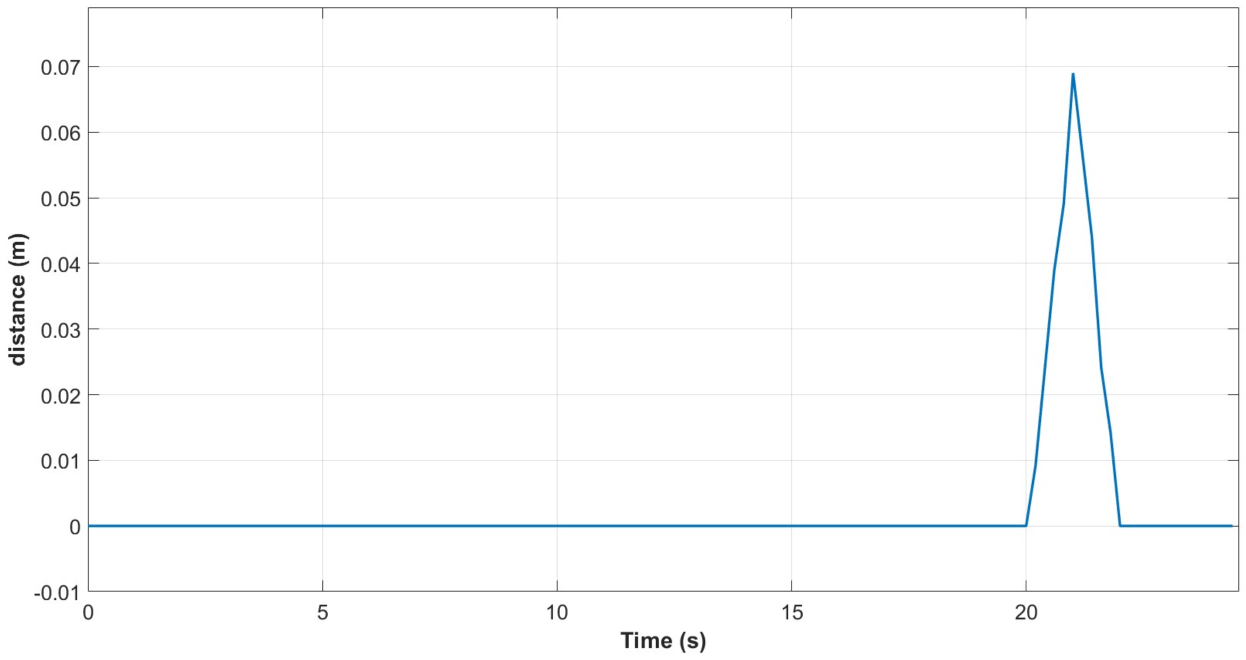


Figure D.28: Center displacement w/ ISMC control during the first test with max speed set to 0.0009 m/sample.

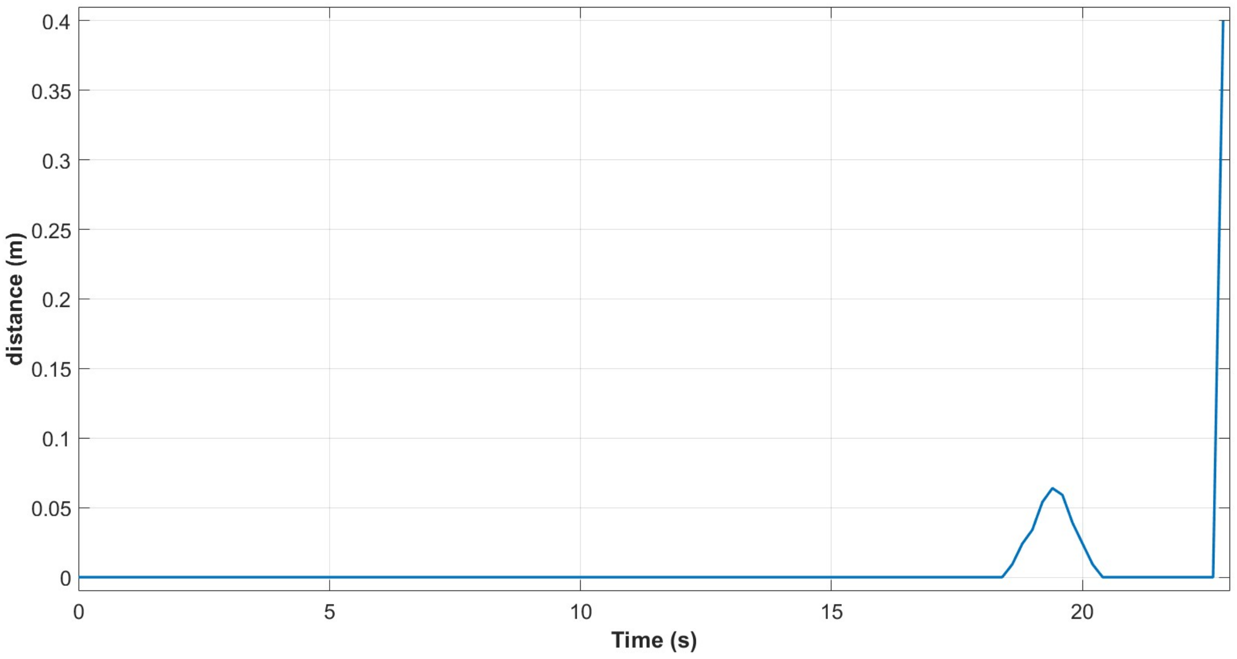


Figure D.29: Center displacement w/ ISMC control during the first test with max speed set to 0.0010 m/sample.

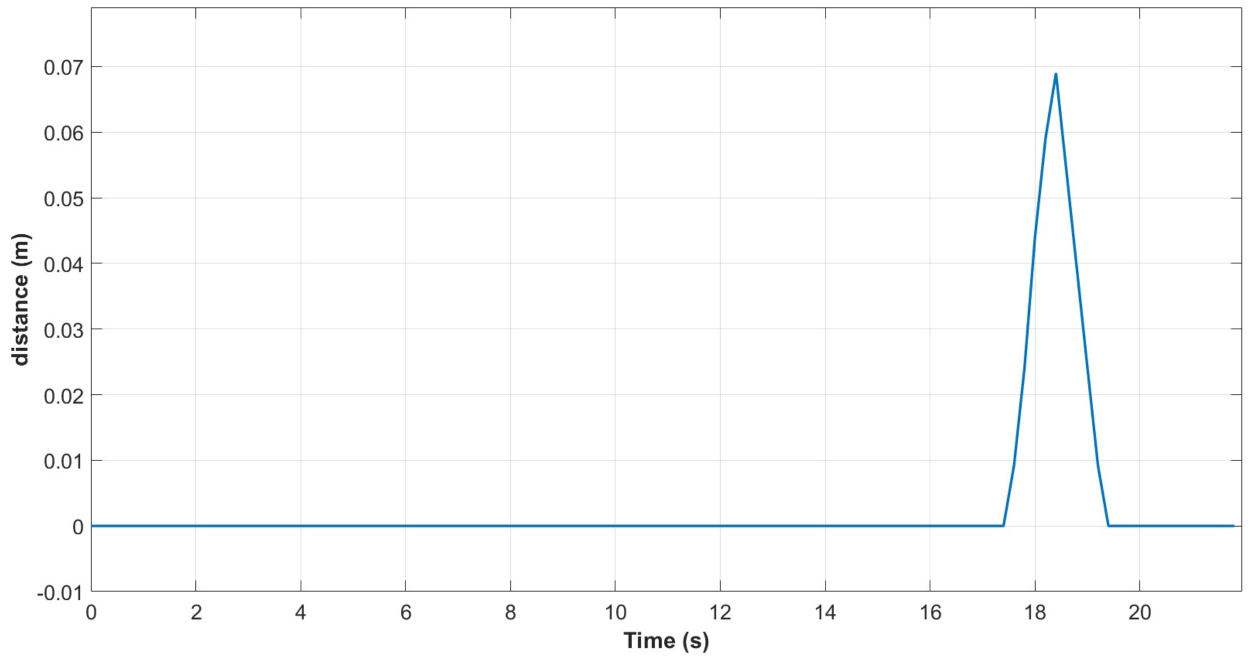


Figure D.30: Center displacement w/ ISMC control during the first test with max speed set to 0.0011 m/sample.

Tables

Max speed	Controller	Track completed?	Sensor time (s)	Run Time (s)	$\frac{\text{Sensor time}}{\text{Run time}}$ (%)
0.0007	PID	Yes	6.800	31.265	21.7
	ISMC	Yes	6.800	28.530	23.8
0.0008	PID	Yes	8.200	29.080	28.2
	ISMC	Yes	7.600	26.345	28.8
0.0009	PID	Yes	8.200	27.060	30.3
	ISMC	Yes	6.400	24.540	26.1
0.0010	PID	Yes	8.000	25.270	31.7
	ISMC	Yes	6.400	22.940	27.9
0.0011	PID	Yes	9.600	24.670	38.9
	ISMC	Yes	6.200	21.950	28.2

Table D.1: Performance, track 1.

					Divided by Run Time	
Max speed	Param.	Index	PID	ISMC	PID	ISMC
0.0007	x	IAE	2.5726	0.30322	0.082285	0.010628
		ISE	0.45041	0.010212	0.014406	0.00035792
	y	IAE	2.2578	0.1328	0.072214	0.0046548
		ISE	0.37001	0.0019878	0.011834	6.9673e-05
	z	IAE	1.4141	0.93432	0.045231	0.032749
		ISE	0.68717	0.45484	0.021979	0.015942
	ϕ	IAE	0.018836	0.021263	0.00060248	0.0007453
		ISE	2.7546e-05	9.3838e-05	8.8106e-07	3.2891e-06
	θ	IAE	0.019669	0.011233	0.00062912	0.00039372
		ISE	2.988e-05	2.1936e-05	9.5572e-07	7.6887e-07
	ψ	IAE	0.00058471	0.00043015	1.8702e-05	1.5077e-05
		ISE	1.7353e-08	1.0072e-08	5.5502e-10	3.5303e-10
	d	IAE	0.11752	0.060753	0.0037589	0.0021294
		ISE	0.019192	0.0027804	0.00061384	9.7456e-05

Continued on next page

Table D.2 – continued from previous page

					Divided by Run Time	
Max speed	Param.	Index	PID	ISMC	PID	ISMC
0.0008	x	IAE	2.6691	0.35527	0.091786	0.013485
		ISE	0.51896	0.014068	0.017846	0.00053399
	y	IAE	2.258	0.14227	0.077649	0.0054002
		ISE	0.39716	0.0024301	0.013657	9.2243e-05
	z	IAE	1.3878	0.94131	0.047725	0.03573
		ISE	0.67989	0.46029	0.02338	0.017472
	ϕ	IAE	0.019856	0.01996	0.00068281	0.00075763
		ISE	3.2722e-05	8.0033e-05	1.1252e-06	3.0379e-06
	θ	IAE	0.020213	0.013059	0.00069508	0.00049569
		ISE	3.7821e-05	4.2246e-05	1.3006e-06	1.6035e-06
	ψ	IAE	0.00054564	0.00039382	1.8763e-05	1.4948e-05
		ISE	1.6285e-08	9.1929e-09	5.6e-10	3.4894e-10
	d	IAE	0.1735	0.063375	0.0059663	0.0024056
		ISE	0.021786	0.0030219	0.00074918	0.0001147
0.0009	x	IAE	2.6531	0.37697	0.098045	0.015361
		ISE	0.57536	0.018446	0.021263	0.00075166
	y	IAE	2.2772	0.15593	0.084154	0.0063543
		ISE	0.43399	0.0031468	0.016038	0.00012823
	z	IAE	1.394	0.93724	0.051516	0.038192
		ISE	0.69029	0.45988	0.02551	0.01874
	ϕ	IAE	0.021244	0.020279	0.00078506	0.00082635
		ISE	3.9159e-05	0.00011663	1.4471e-06	4.7526e-06
	θ	IAE	0.021467	0.011558	0.00079333	0.000471
		ISE	4.6285e-05	3.6252e-05	1.7105e-06	1.4772e-06
	ψ	IAE	0.00049949	0.00036505	1.8459e-05	1.4876e-05
		ISE	1.4656e-08	8.7241e-09	5.416e-10	3.5551e-10
	d	IAE	0.18011	0.065836	0.0066558	0.0026828
		ISE	0.022559	0.0030545	0.00083368	0.00012447

Continued on next page

Table D.2 – continued from previous page

					Divided by Run Time	
Max speed	Param.	Index	PID	ISMC	PID	ISMC
0.0010	x	IAE	2.6626	0.38407	0.10537	0.016743
		ISE	0.63293	0.021201	0.025047	0.00092419
	y	IAE	2.2713	0.17322	0.089881	0.007551
		ISE	0.4641	0.0040432	0.018366	0.00017625
	z	IAE	1.3897	0.93397	0.054995	0.040713
		ISE	0.68803	0.46112	0.027227	0.020101
	ϕ	IAE	0.022066	0.019905	0.00087322	0.0008677
		ISE	4.4356e-05	0.00013327	1.7553e-06	5.8094e-06
	θ	IAE	0.022946	0.011773	0.00090805	0.00051323
		ISE	5.818e-05	3.9844e-05	2.3023e-06	1.7369e-06
	ψ	IAE	0.00046444	0.00034316	1.8379e-05	1.4959e-05
		ISE	1.357e-08	8.0368e-09	5.3701e-10	3.5034e-10
	d	IAE	0.18965	0.10334	0.007505	0.0045049
		ISE	0.023882	0.018902	0.00094507	0.00082398
0.0011	x	IAE	2.8156	0.41212	0.11413	0.018775
		ISE	0.70452	0.026214	0.028558	0.0011943
	y	IAE	2.3062	0.18653	0.09348	0.0084982
		ISE	0.4772	0.0048261	0.019343	0.00021987
	z	IAE	1.3891	0.9286	0.056308	0.042305
		ISE	0.68886	0.4595	0.027923	0.020934
	ϕ	IAE	0.024493	0.020677	0.00099281	0.000942
		ISE	5.7951e-05	0.00016156	2.3491e-06	7.3605e-06
	θ	IAE	0.027321	0.012163	0.0011075	0.00055413
		ISE	7.6381e-05	4.9065e-05	3.0961e-06	2.2353e-06
	ψ	IAE	0.00045028	0.00032897	1.8252e-05	1.4987e-05
		ISE	1.3039e-08	7.7258e-09	5.2855e-10	3.5197e-10
	d	IAE	0.27656	0.066338	0.01121	0.0030222
		ISE	0.033517	0.0031909	0.0013586	0.00014537

Table D.2: IAE and ISE results, track 1.

D.1.2 Second Track Results

Figures

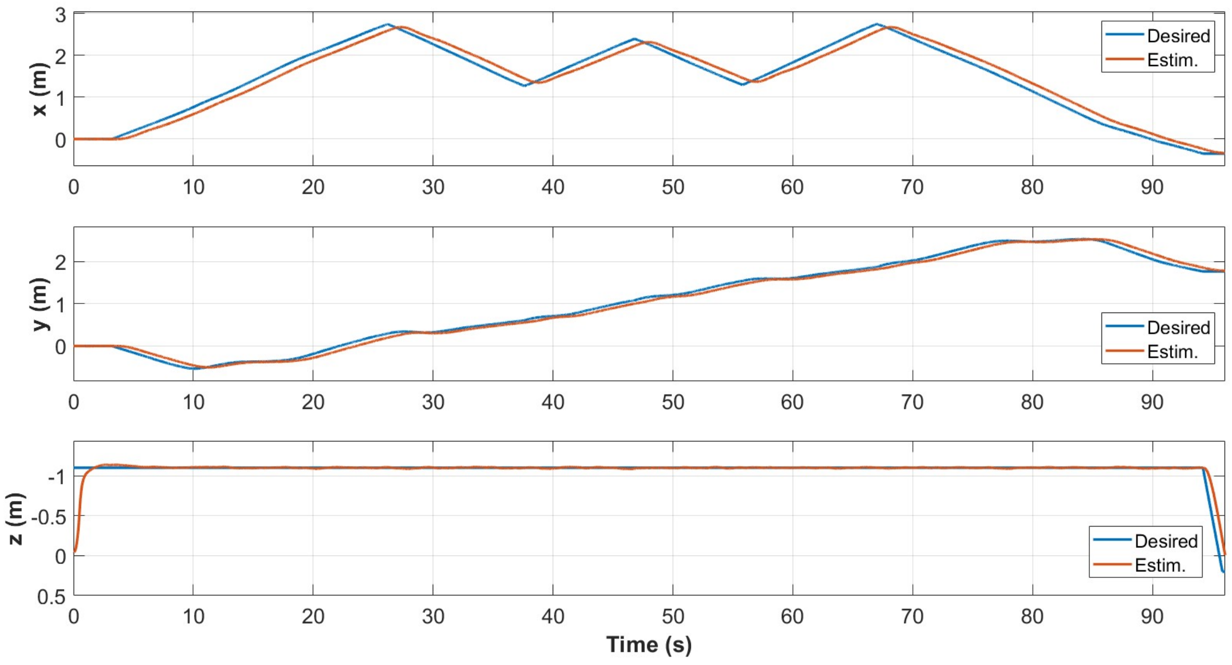


Figure D.31: Position w/ PID control during the second test with max speed set to 0.0007 m/sample.

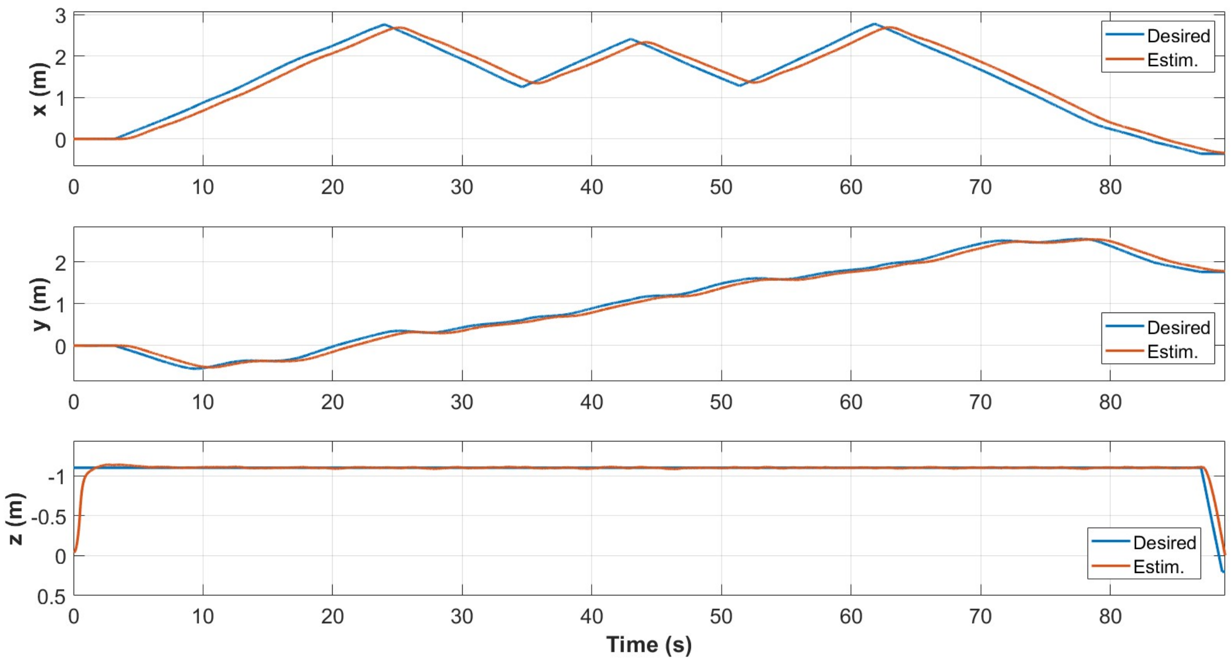


Figure D.32: Position w/ PID control during the second test with max speed set to 0.0008 m/sample.

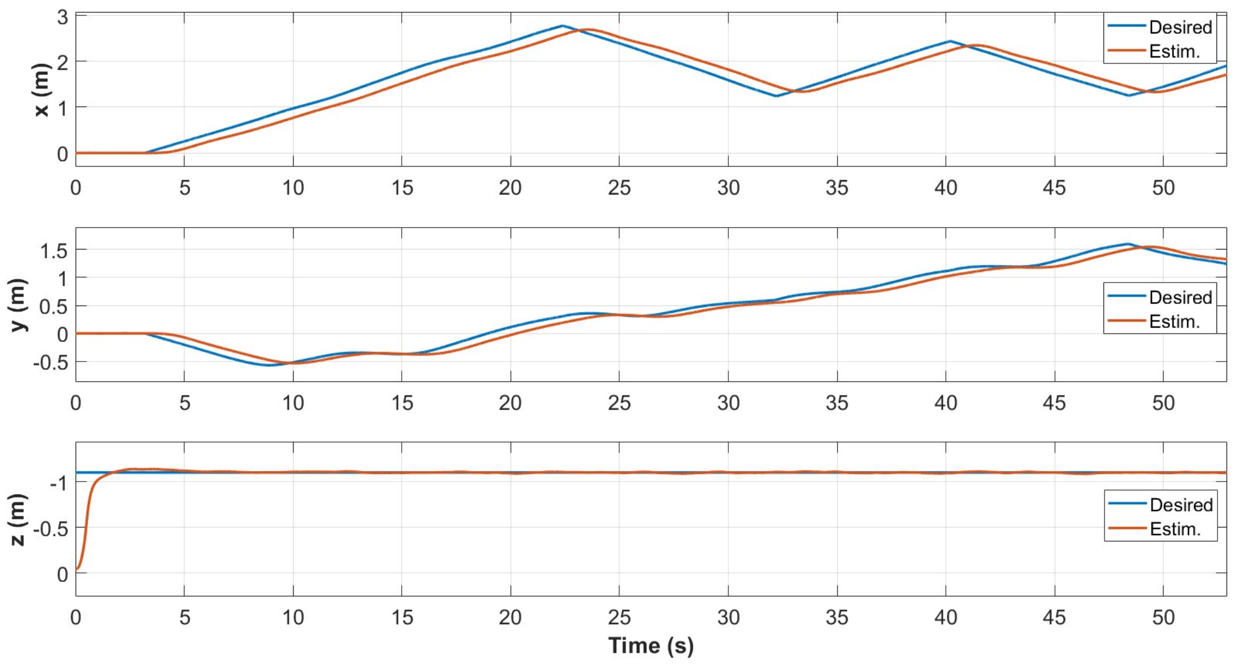


Figure D.33: Position w/ PID control during the second test with max speed set to 0.0009 m/sample.

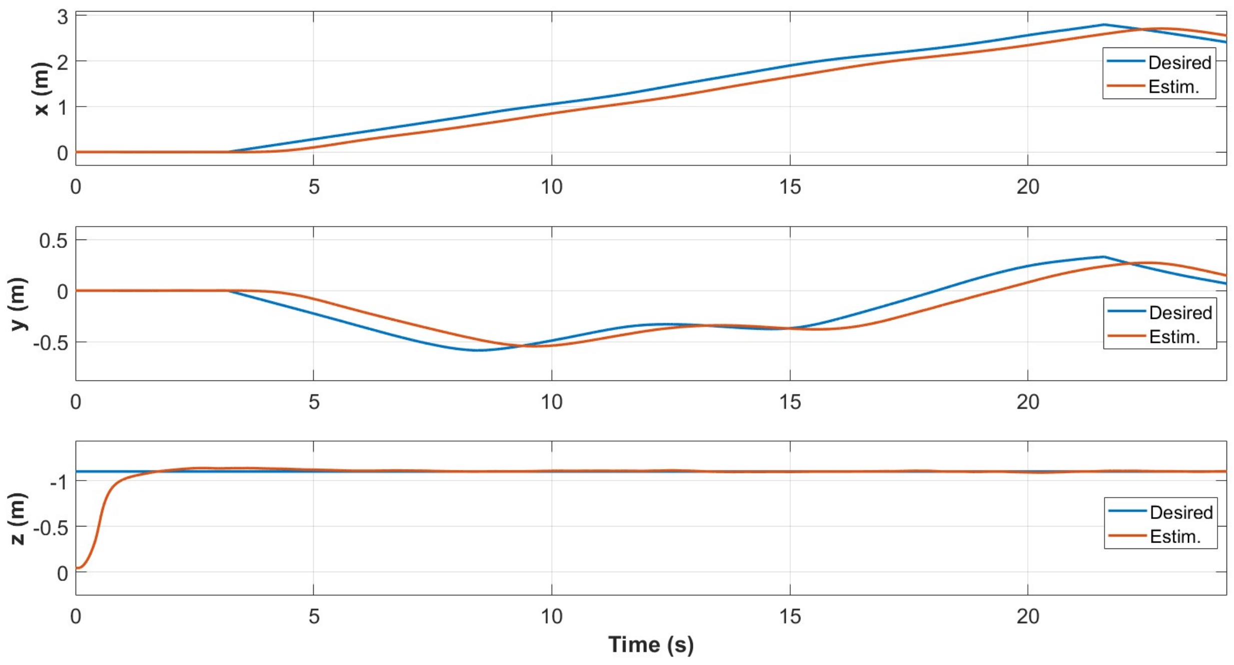


Figure D.34: Position w/ PID control during the second test with max speed set to 0.0010 m/sample.

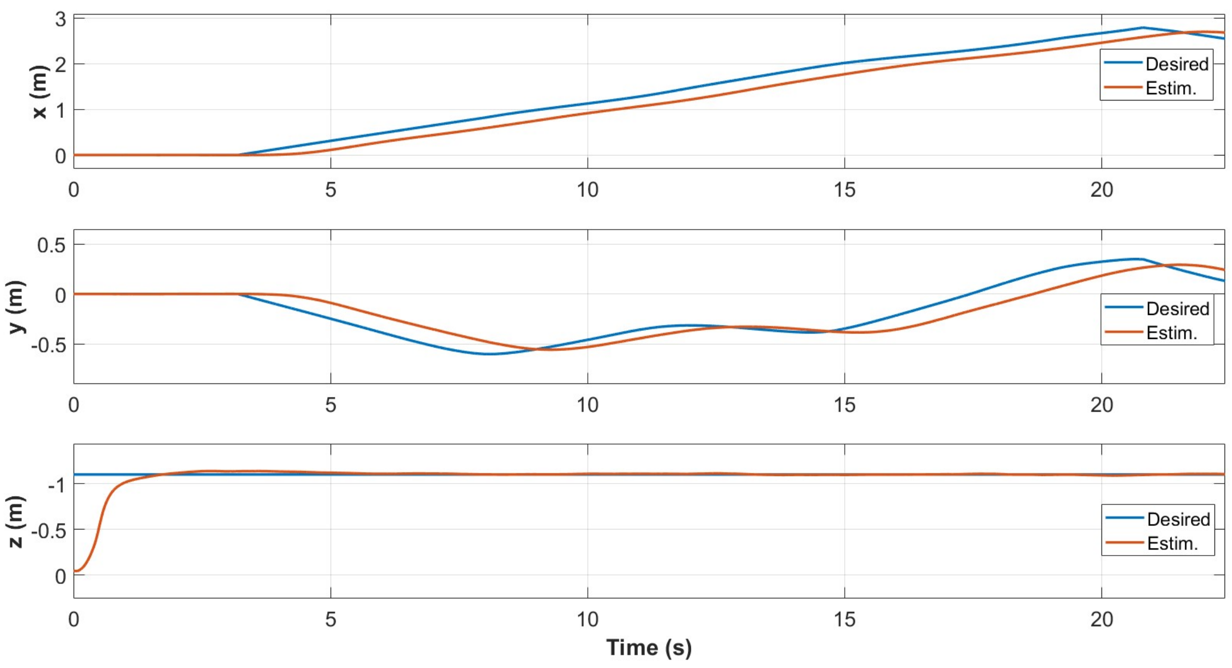


Figure D.35: Position w/ PID control during the second test with max speed set to 0.0011 m/sample.

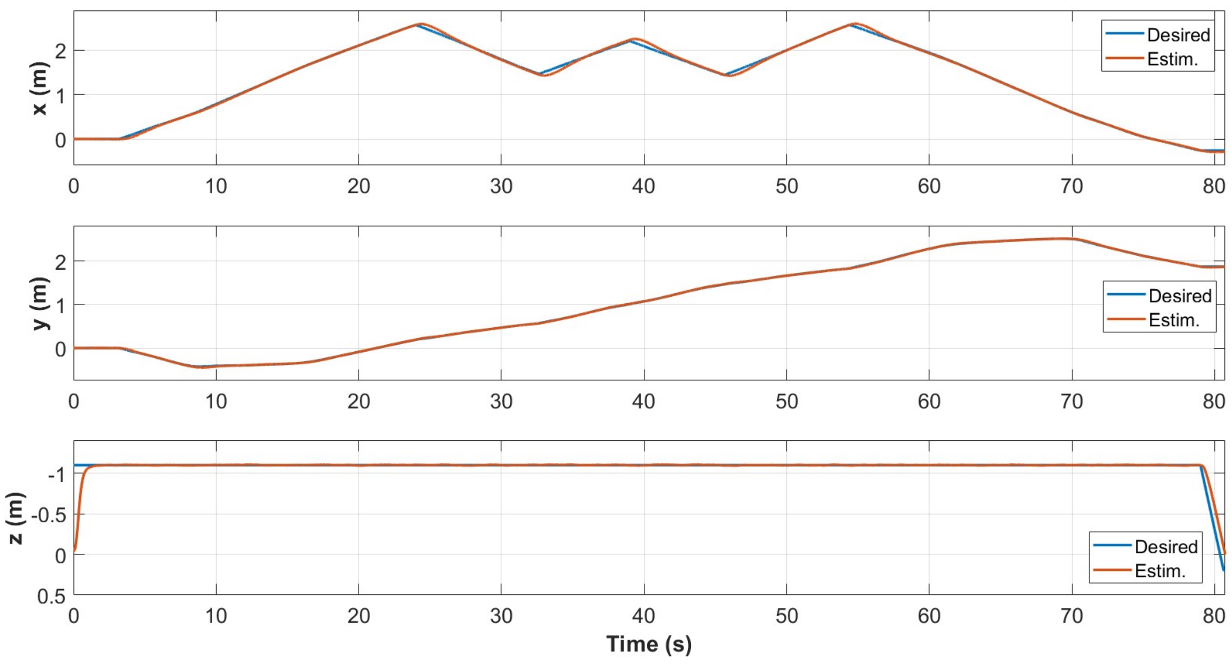


Figure D.36: Position w/ ISMC control during the second test with max speed set to 0.0007 m/sample.

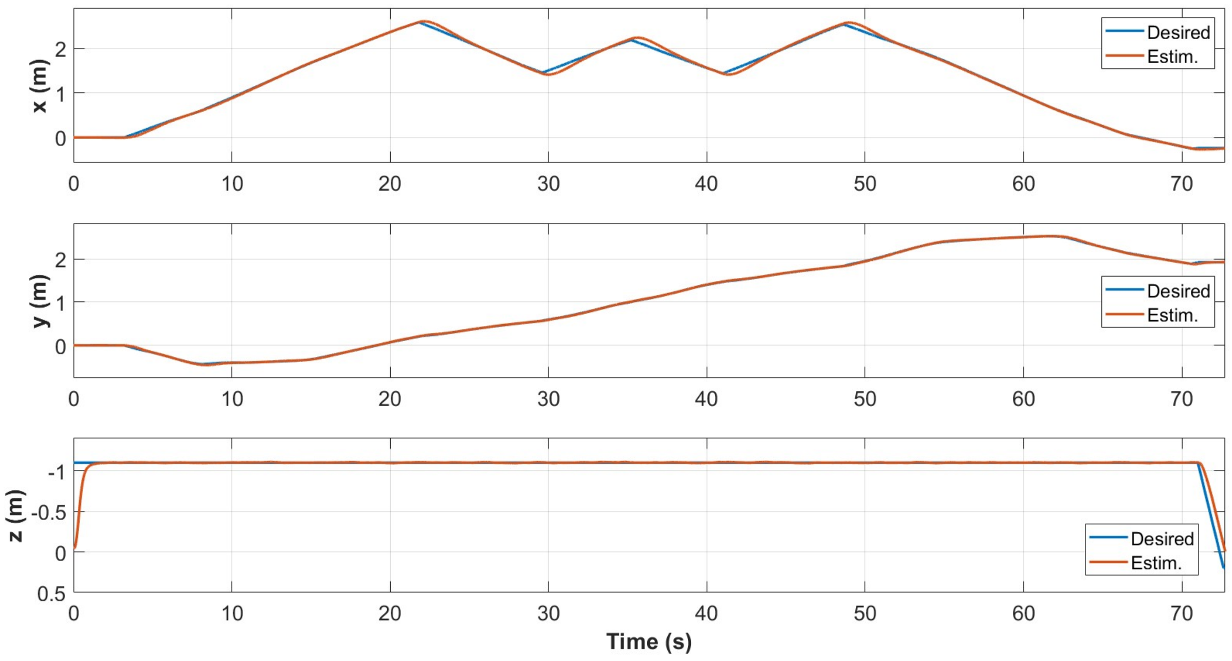


Figure D.37: Position w/ ISMC control during the second test with max speed set to 0.0008 m/sample.

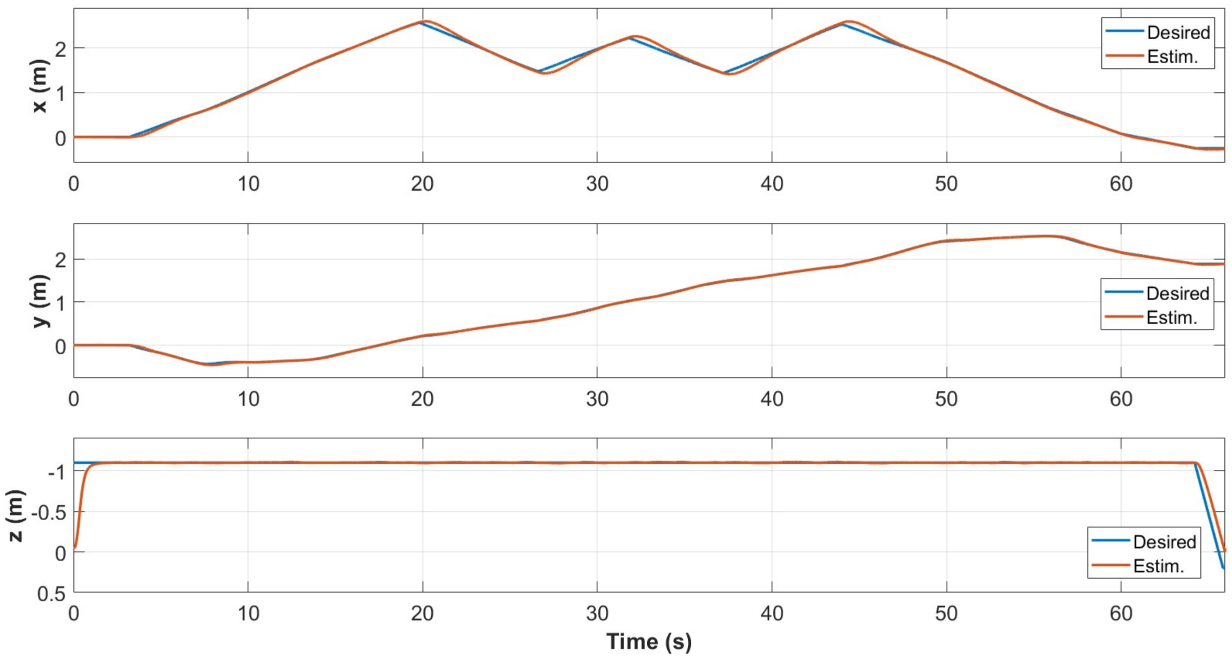


Figure D.38: Position w/ ISMC control during the second test with max speed set to 0.0009 m/sample.

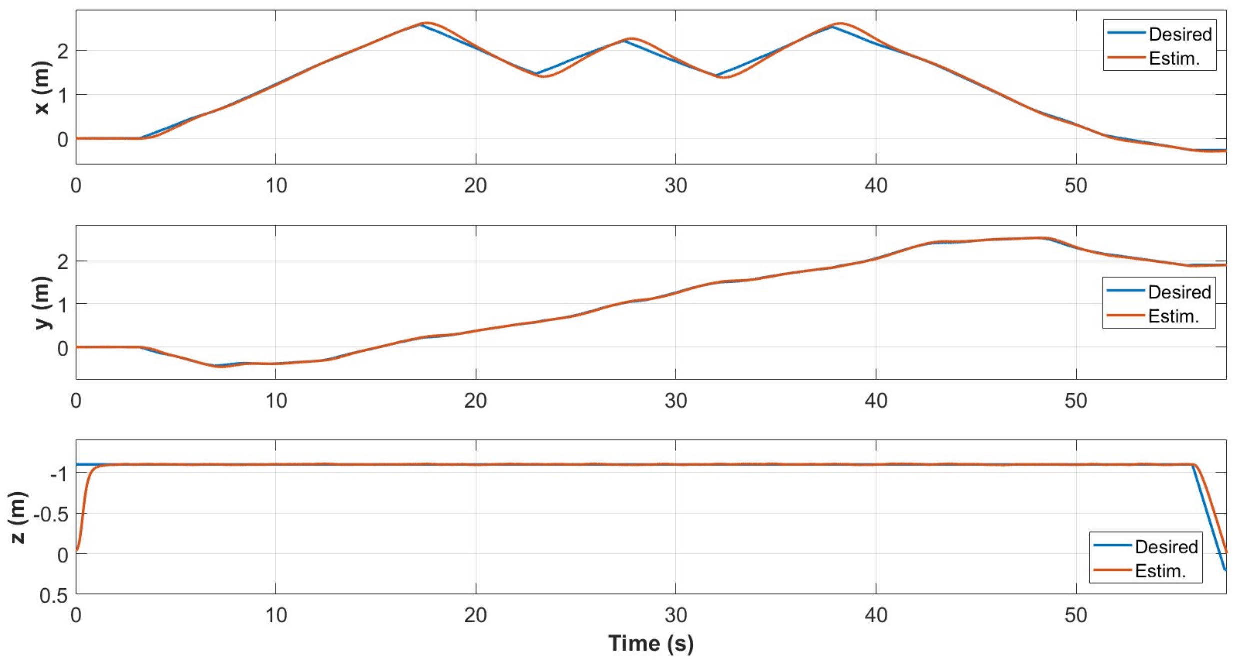


Figure D.39: Position w/ ISMC control during the second test with max speed set to 0.0010 m/sample.

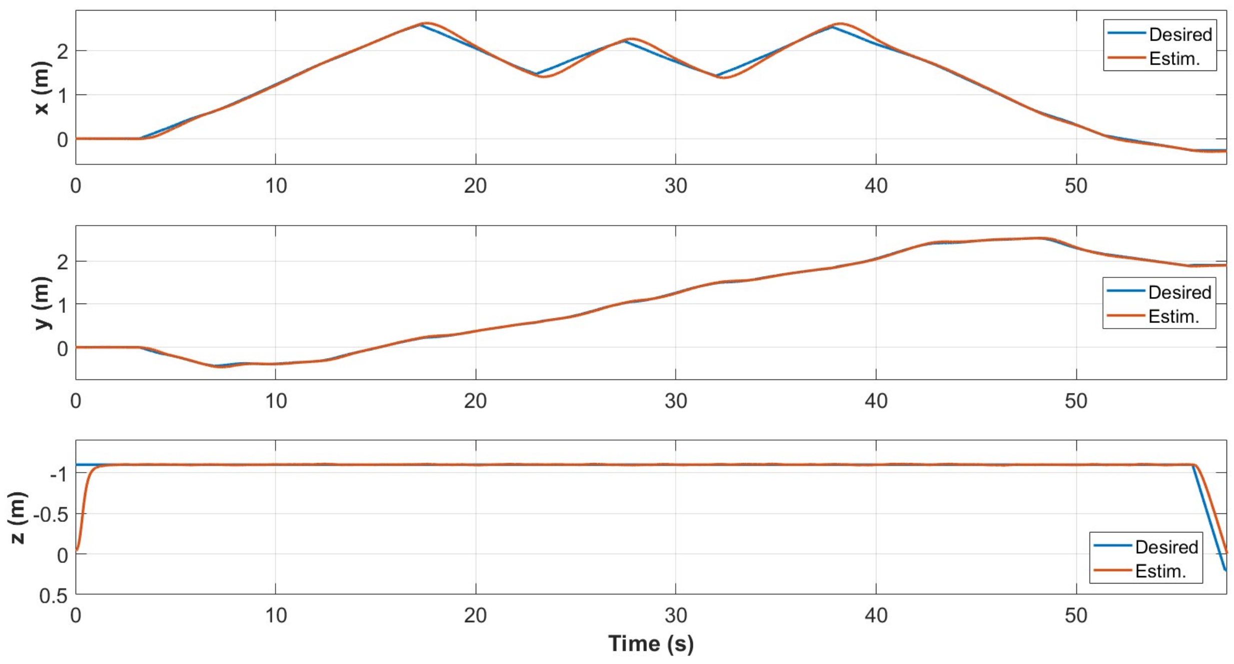


Figure D.40: Position w/ ISMC control during the second test with max speed set to 0.0011 m/sample.

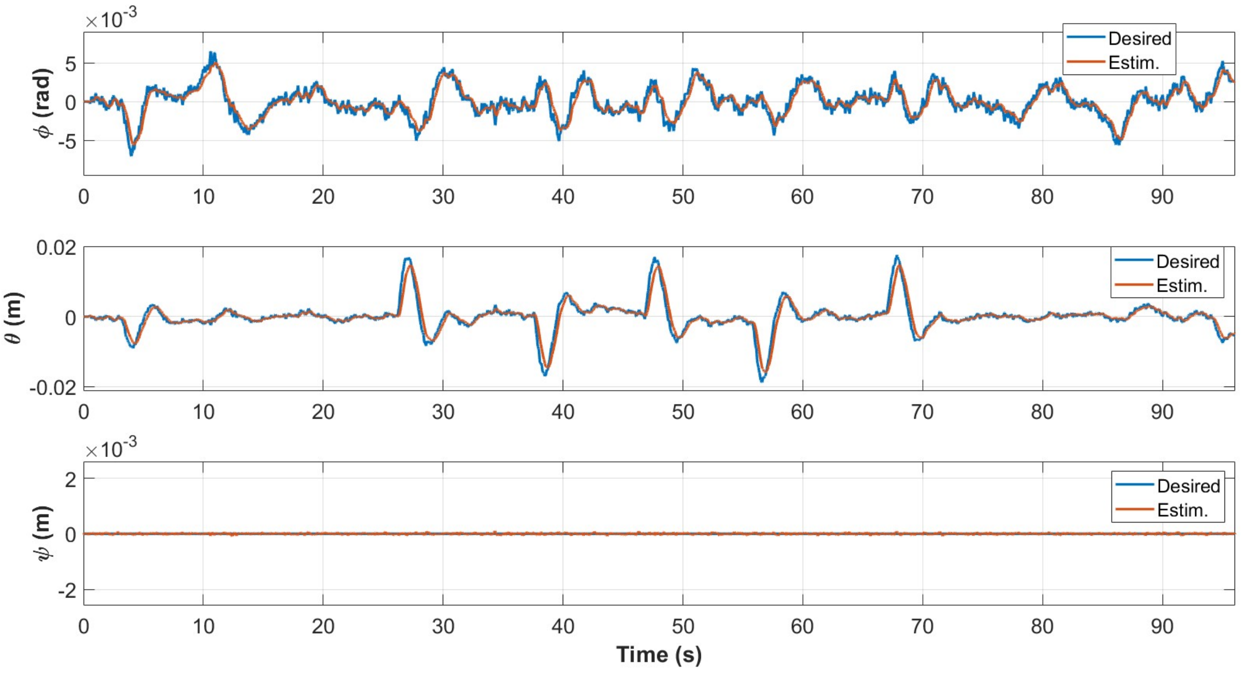


Figure D.41: Attitude w/ PID control during the second test with max speed set to 0.0007 m/sample.

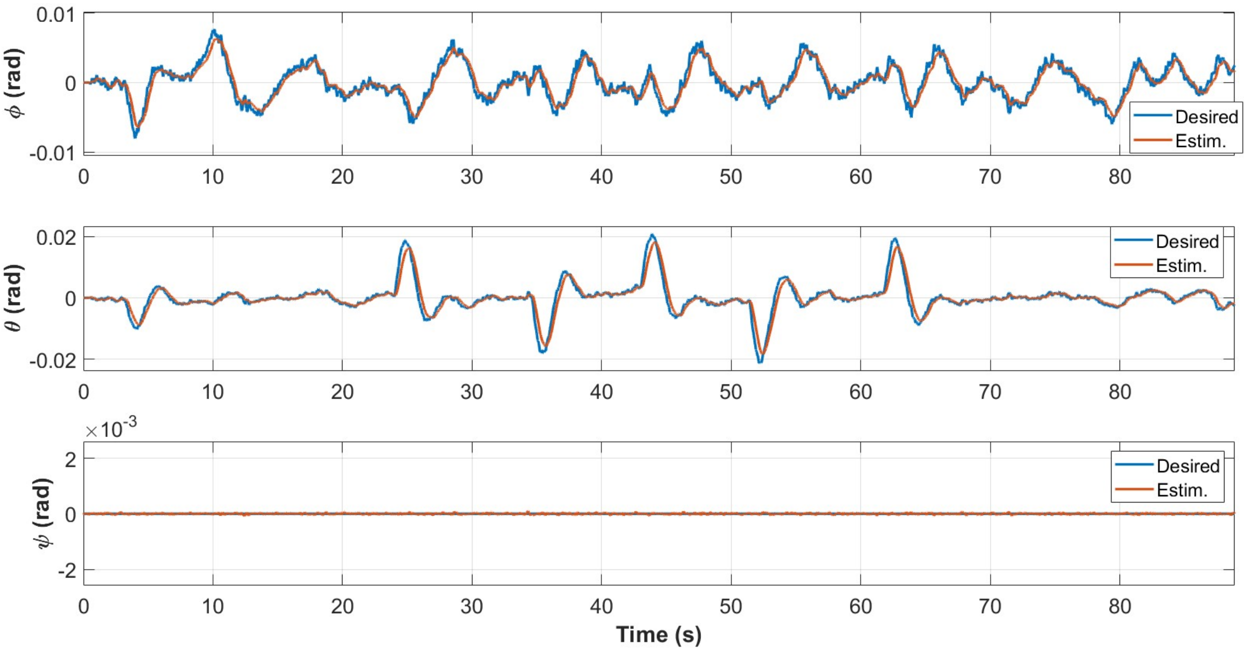


Figure D.42: Attitude w/ PID control during the second test with max speed set to 0.0008 m/sample.

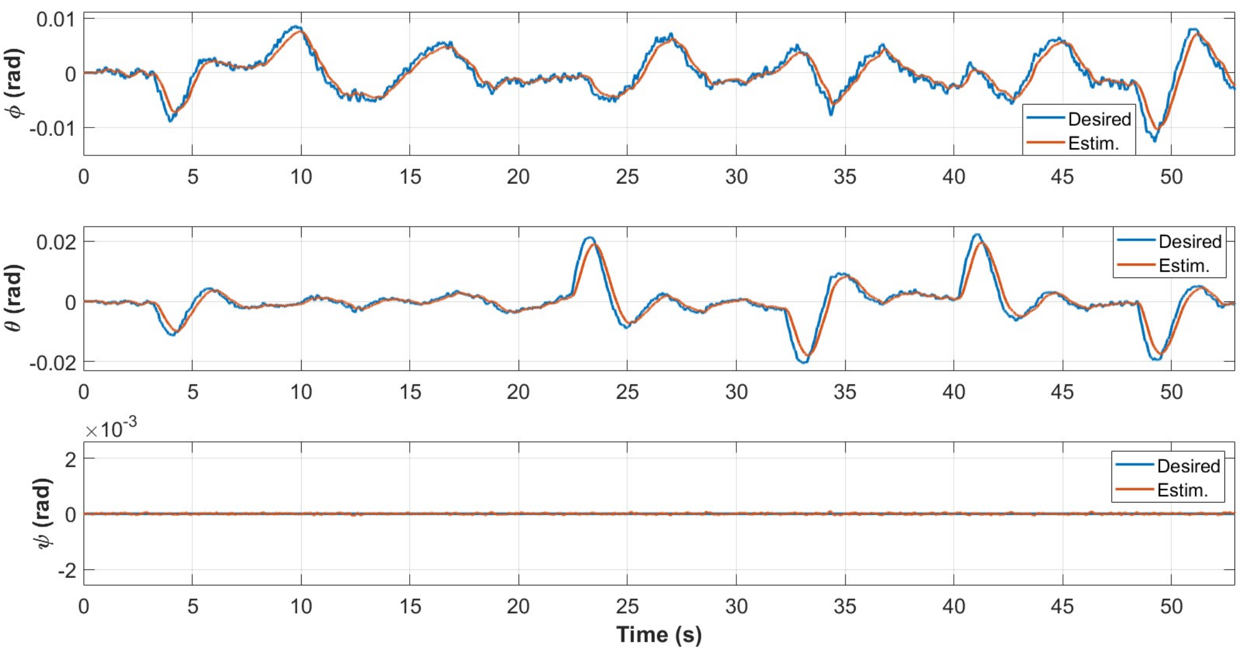


Figure D.43: Attitude w/ PID control during the second test with max speed set to 0.0009 m/sample.

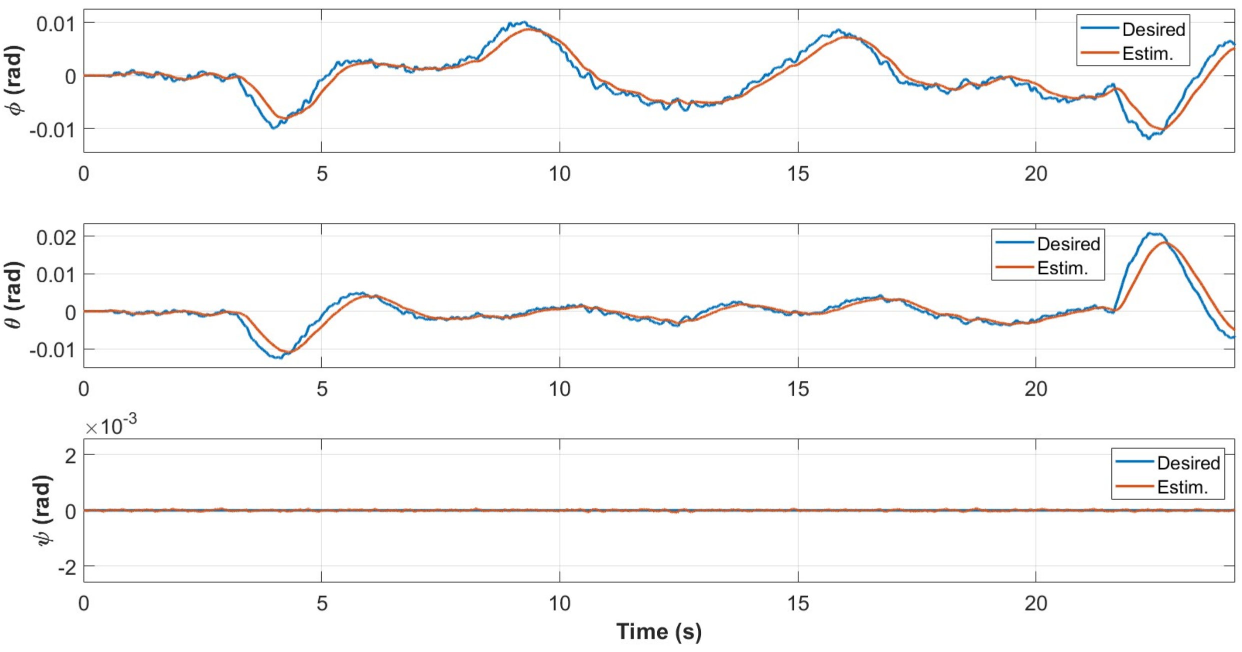


Figure D.44: Attitude w/ PID control during the second test with max speed set to 0.0010 m/sample.

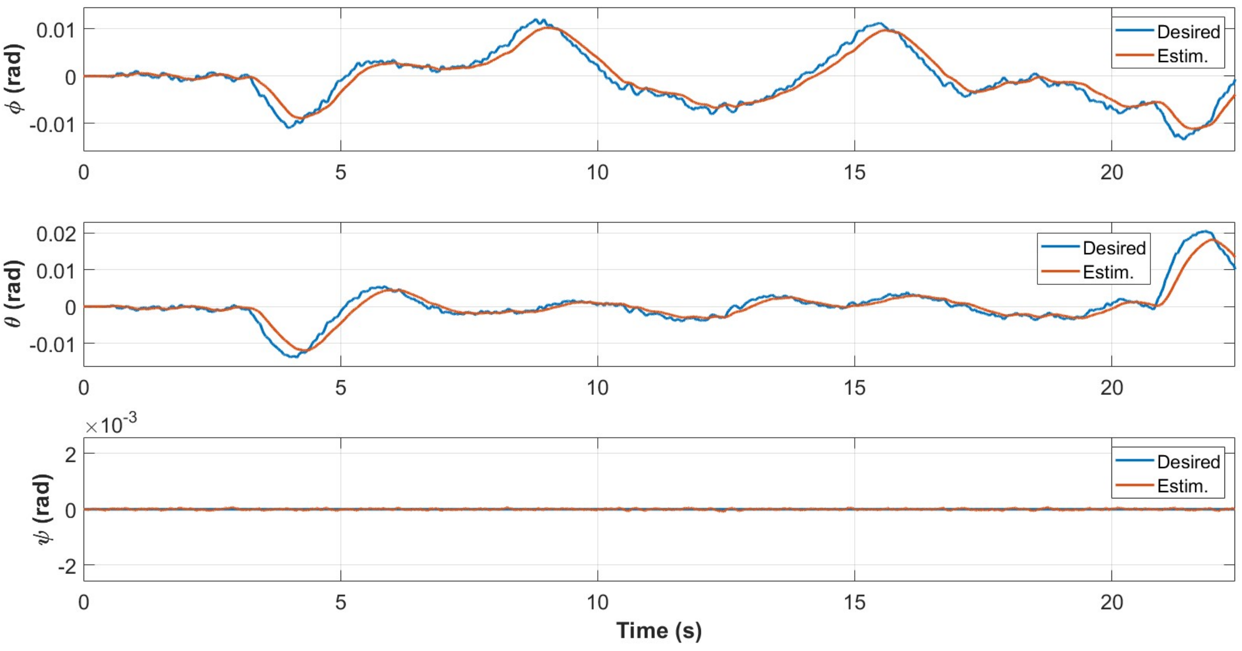


Figure D.45: Attitude w/ PID control during the second test with max speed set to 0.0011 m/sample.

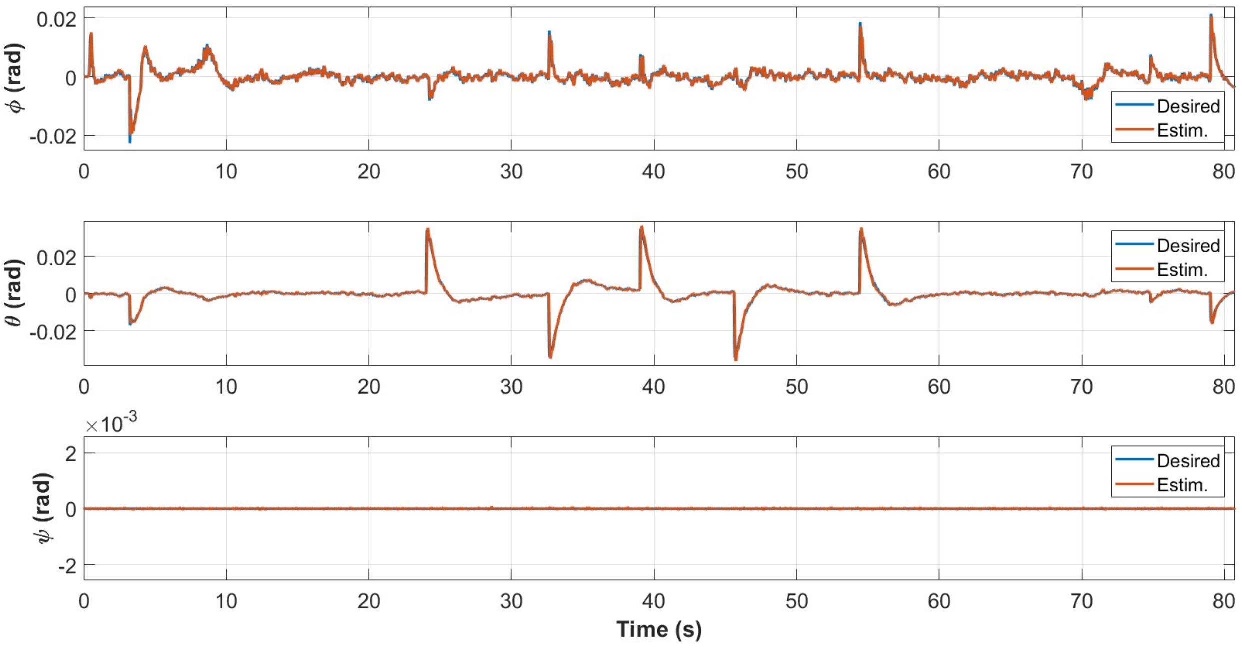


Figure D.46: Attitude w/ ISMC control during the second test with max speed set to 0.0007 m/sample.

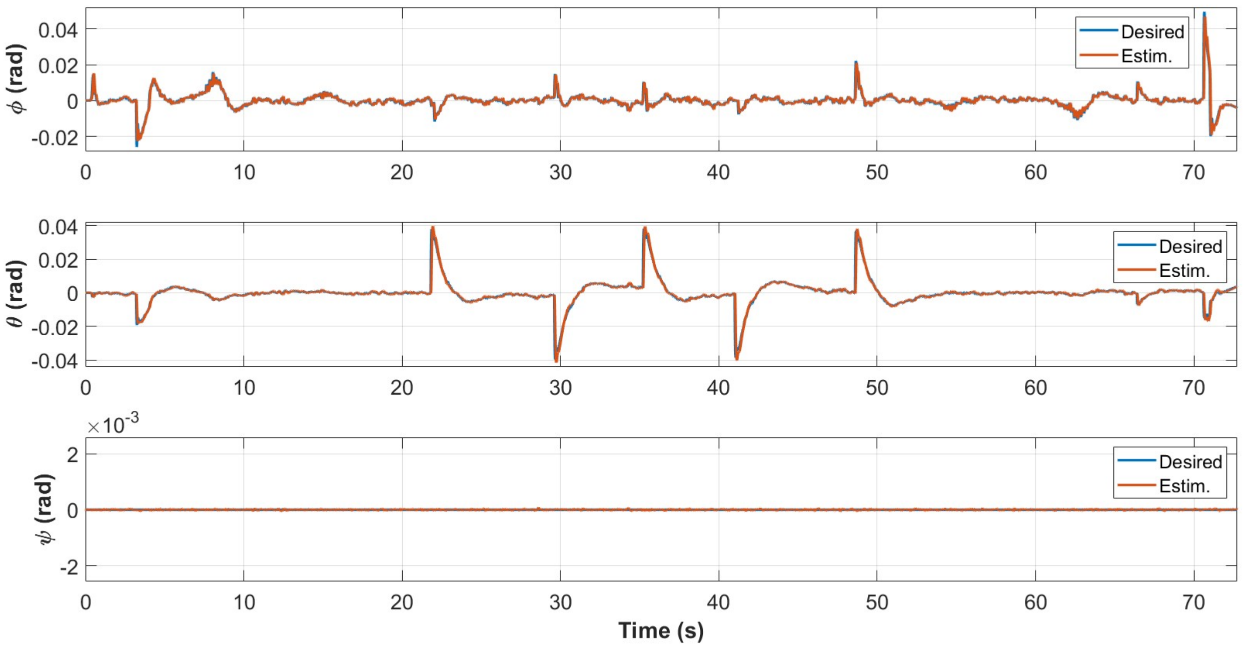


Figure D.47: Attitude w/ ISMC control during the second test with max speed set to 0.0008 m/sample.

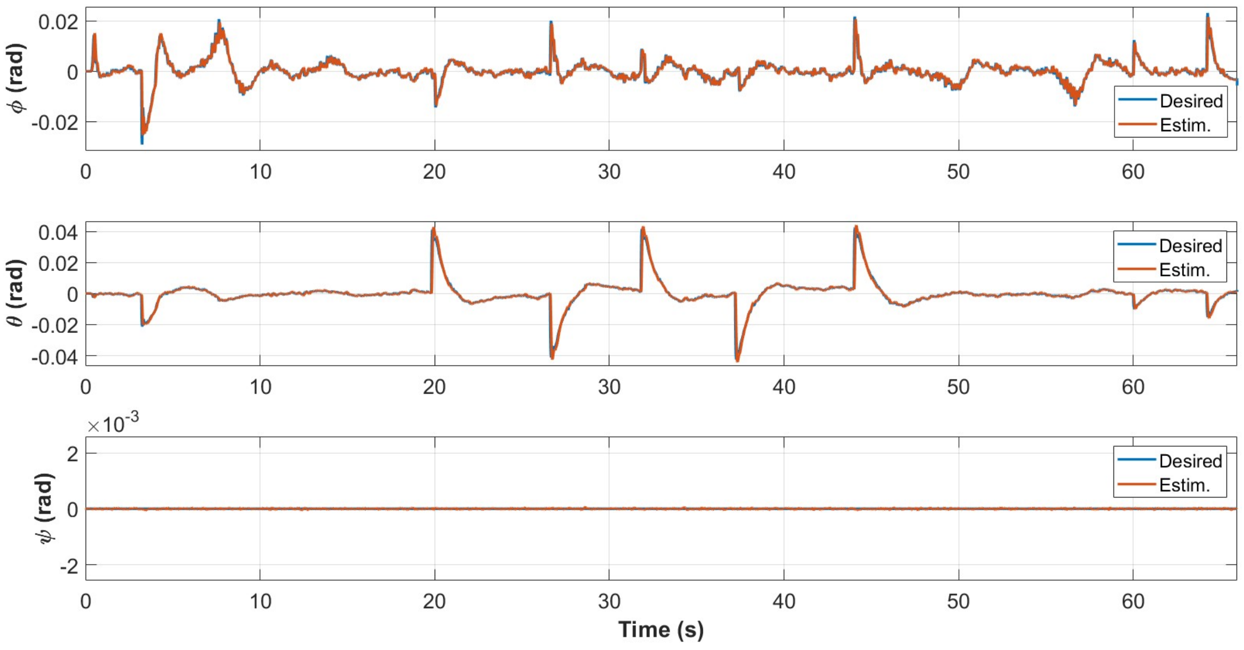


Figure D.48: Attitude w/ ISMC control during the second test with max speed set to 0.0009 m/sample.

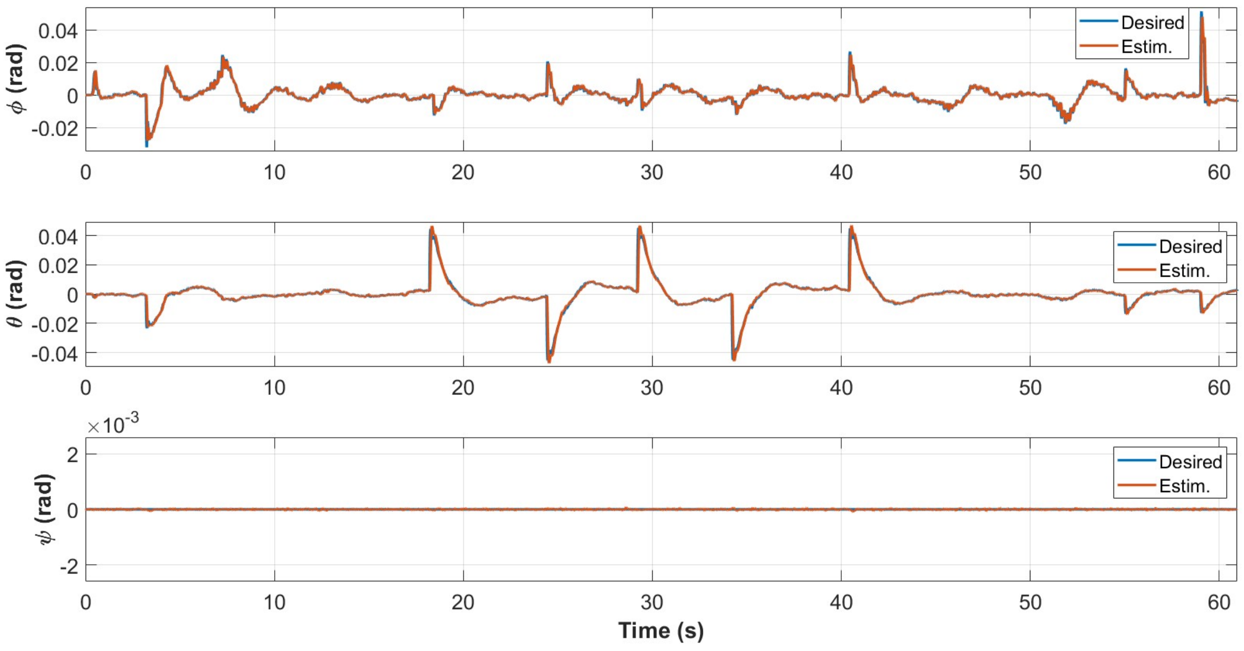


Figure D.49: Attitude w/ ISMC control during the second test with max speed set to 0.0010 m/sample.

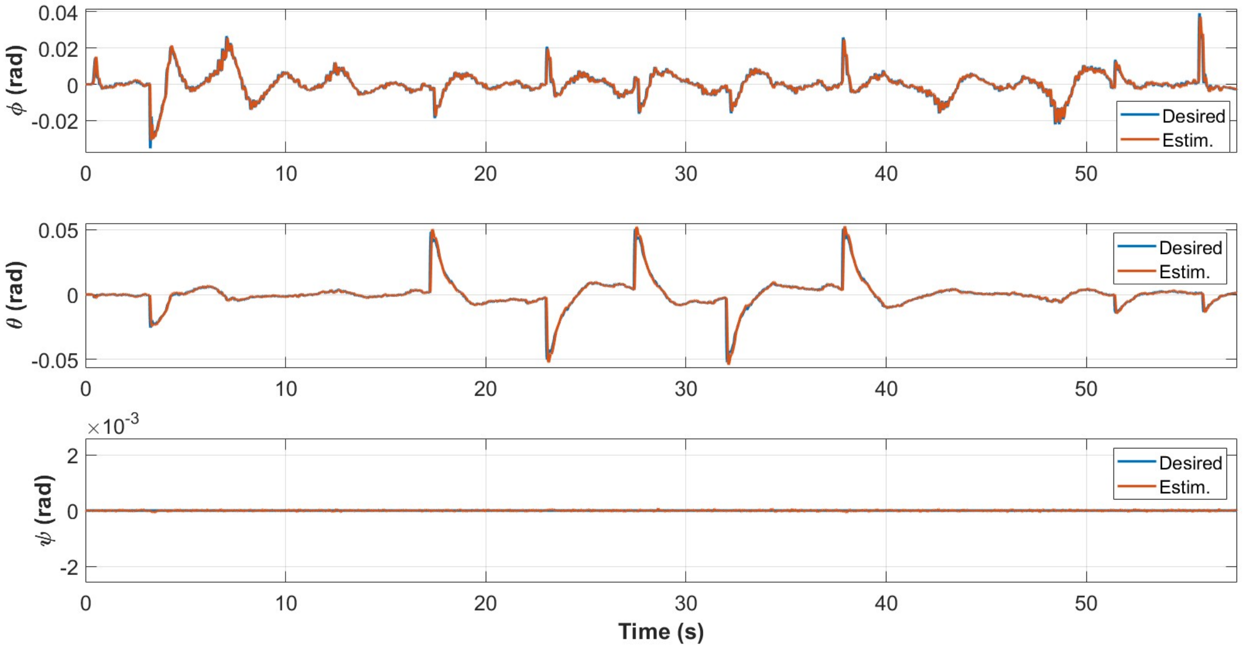


Figure D.50: Attitude w/ ISMC control during the second test with max speed set to 0.0011 m/sample.

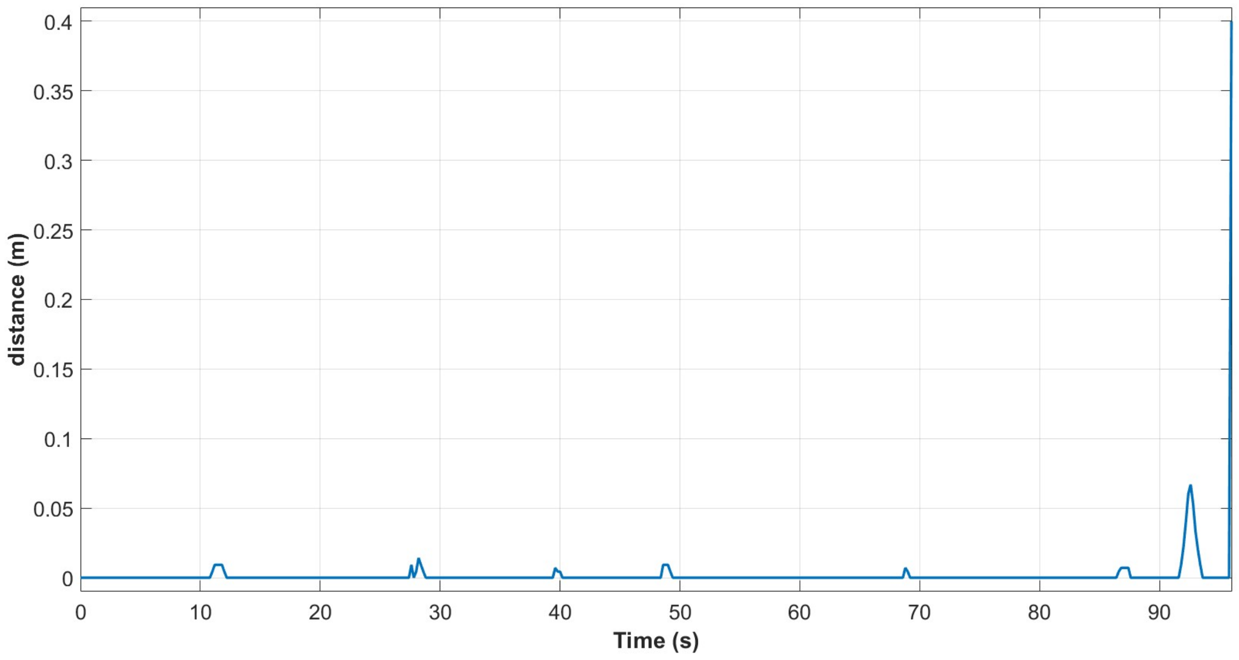


Figure D.51: Center displacement w/ PID control during the second test with max speed set to 0.0007 m/sample.

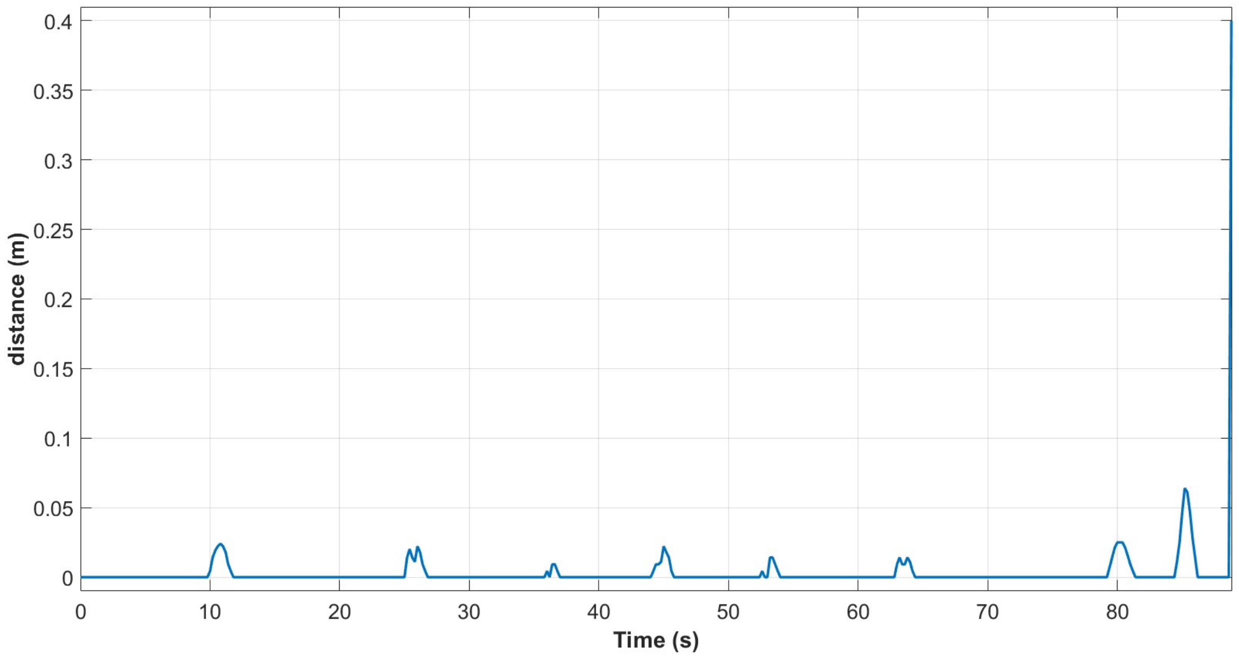


Figure D.52: Center displacement w/ PID control during the second test with max speed set to 0.0008 m/sample.

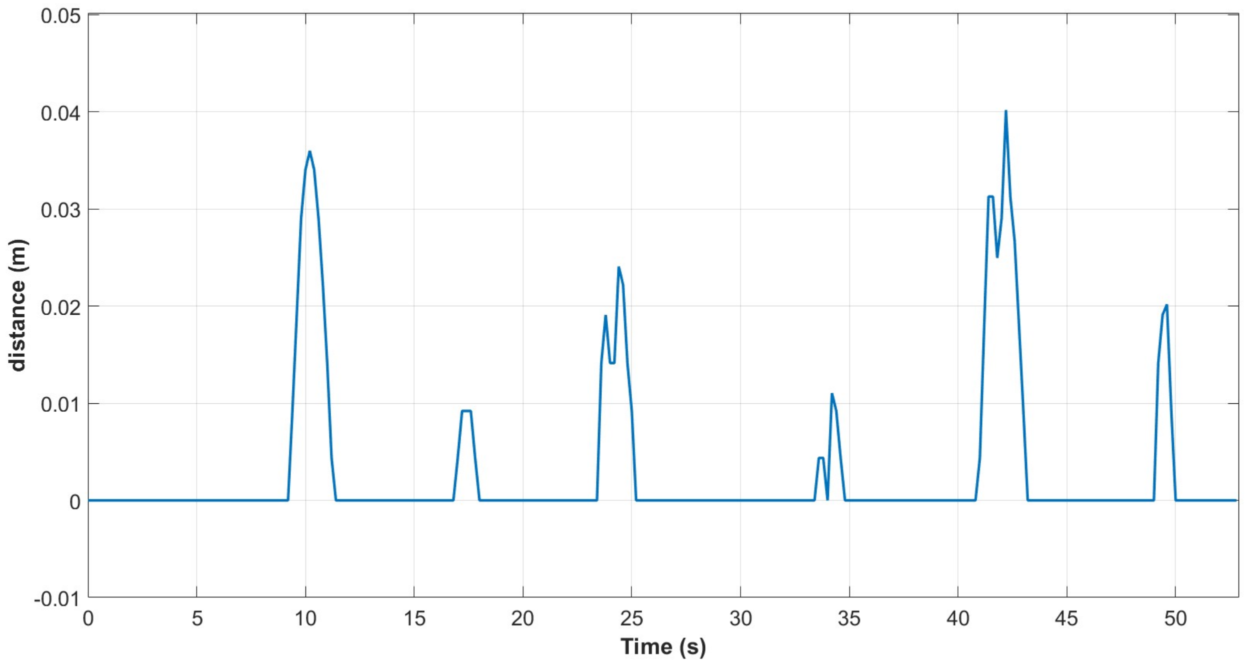


Figure D.53: Center displacement w/ PID control during the second test with max speed set to 0.0009 m/sample.

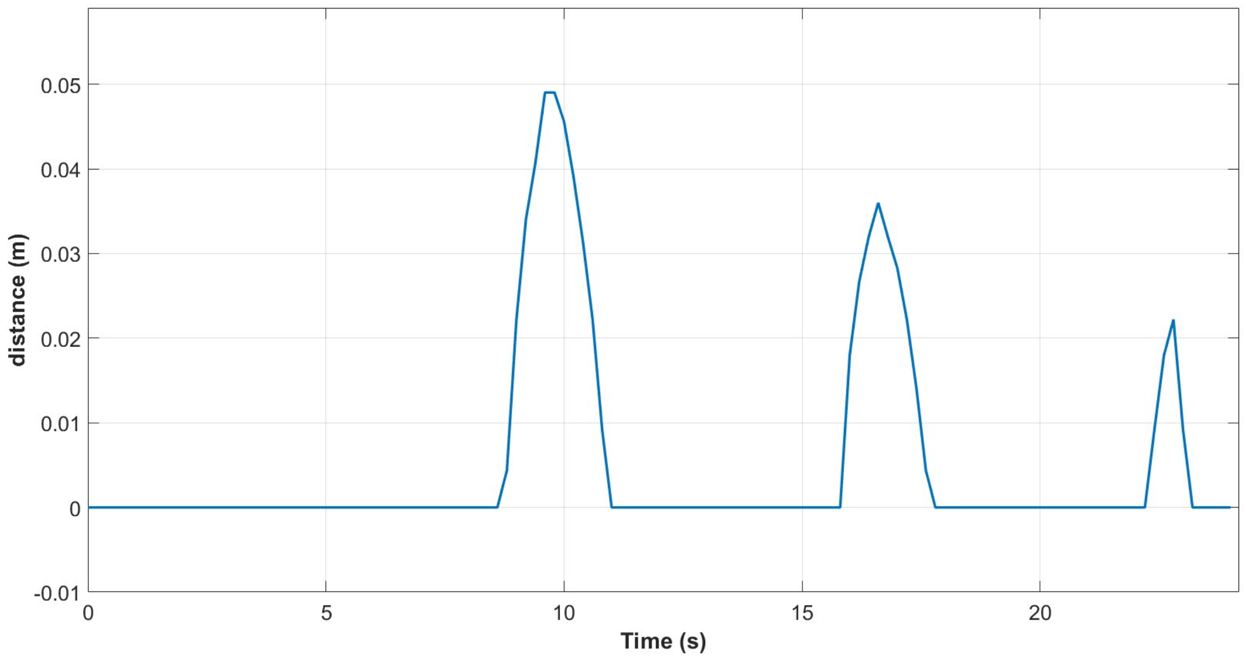


Figure D.54: Center displacement w/ PID control during the second test with max speed set to 0.0010 m/sample.

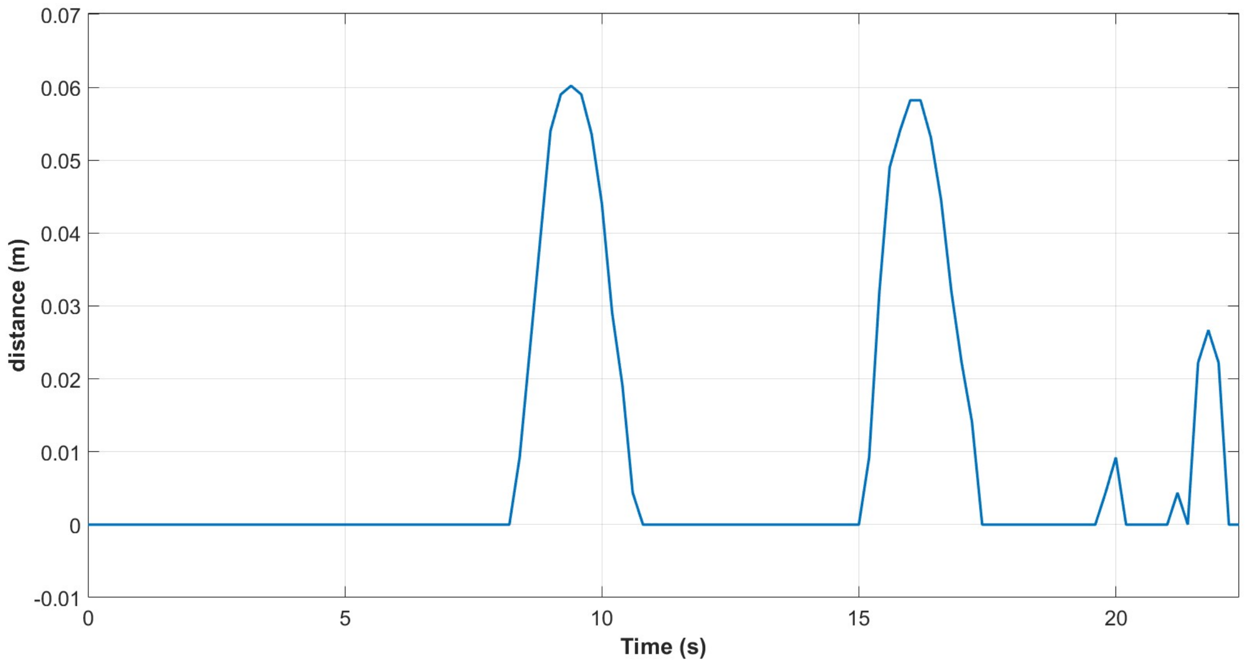


Figure D.55: Center displacement w/ PID control during the second test with max speed set to 0.0011 m/sample.

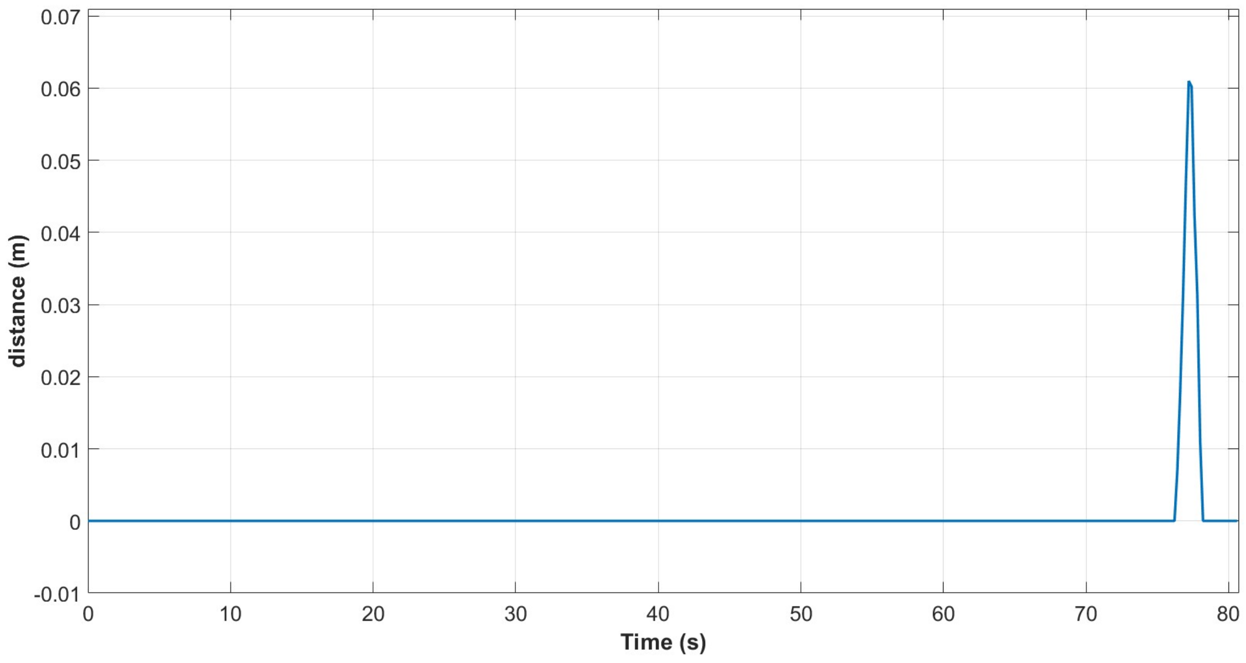


Figure D.56: Center displacement w/ ISMC control during the second test with max speed set to 0.0007 m/sample.

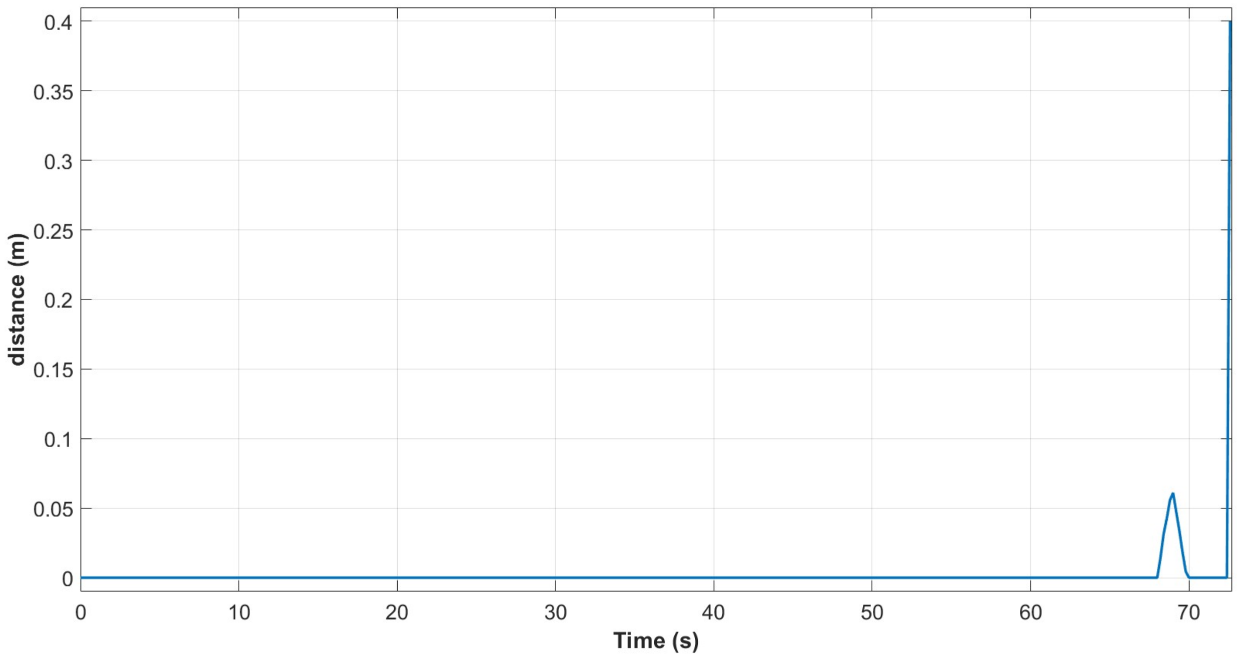


Figure D.57: Center displacement w/ ISMC control during the second test with max speed set to 0.0008 m/sample.

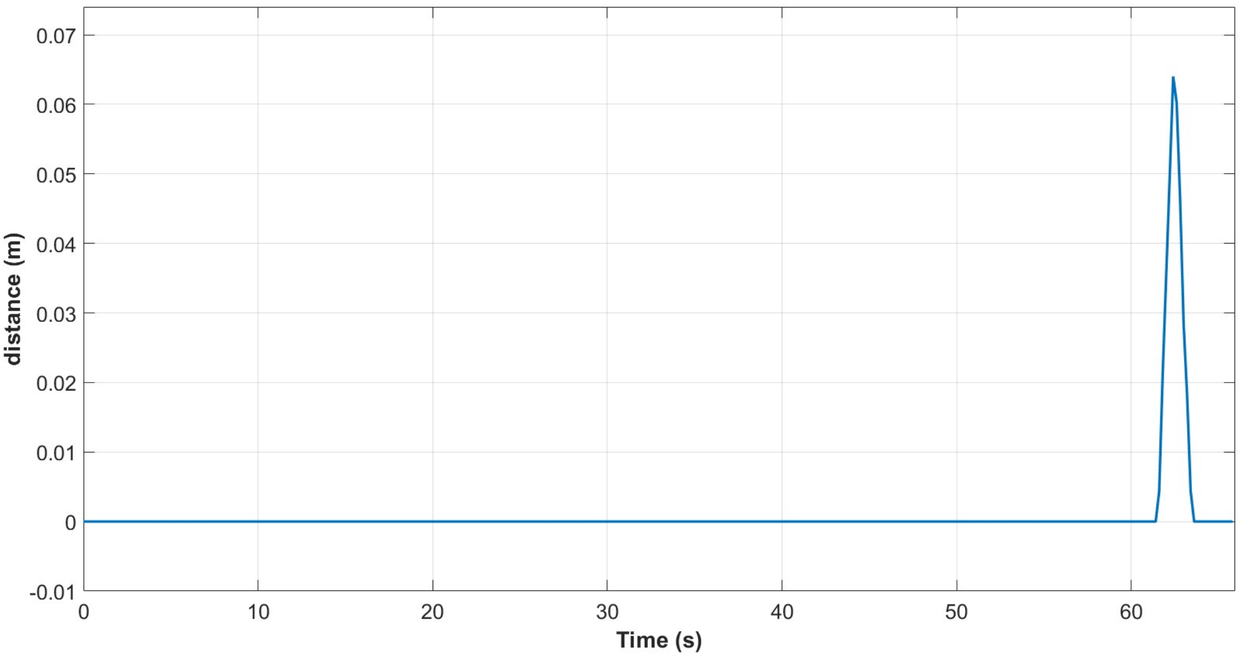


Figure D.58: Center displacement w/ ISMC control during the second test with max speed set to 0.0009 m/sample.

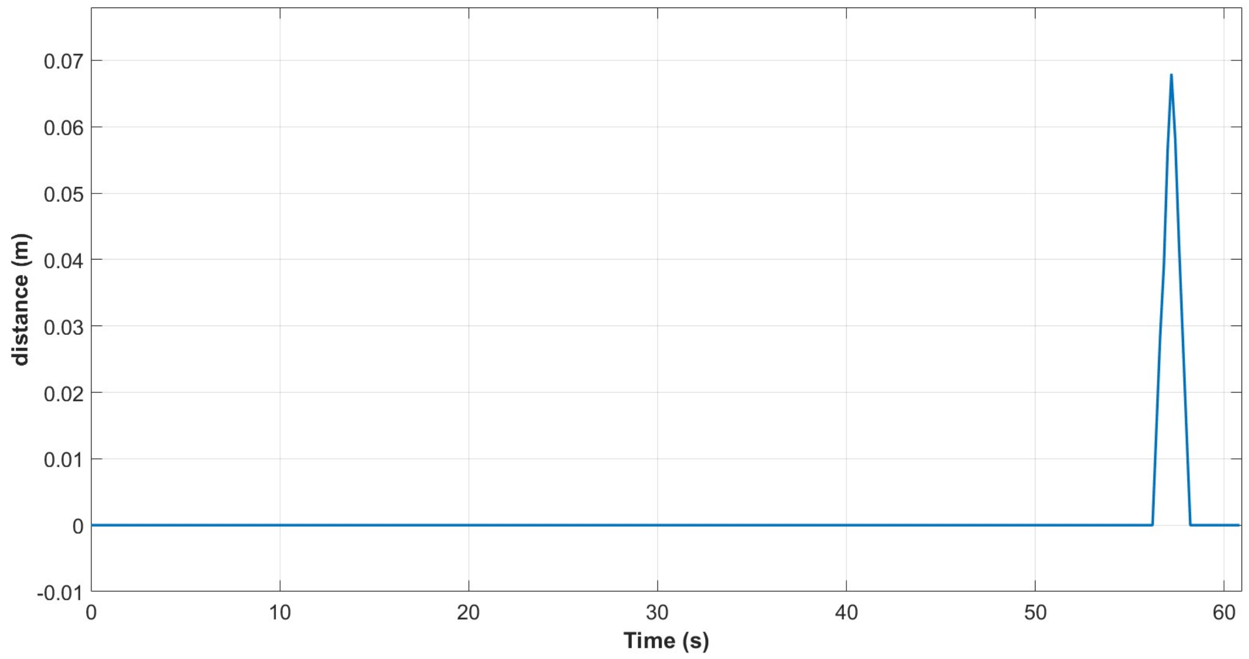


Figure D.59: Center displacement w/ ISMC control during the second test with max speed set to 0.0010 m/sample.

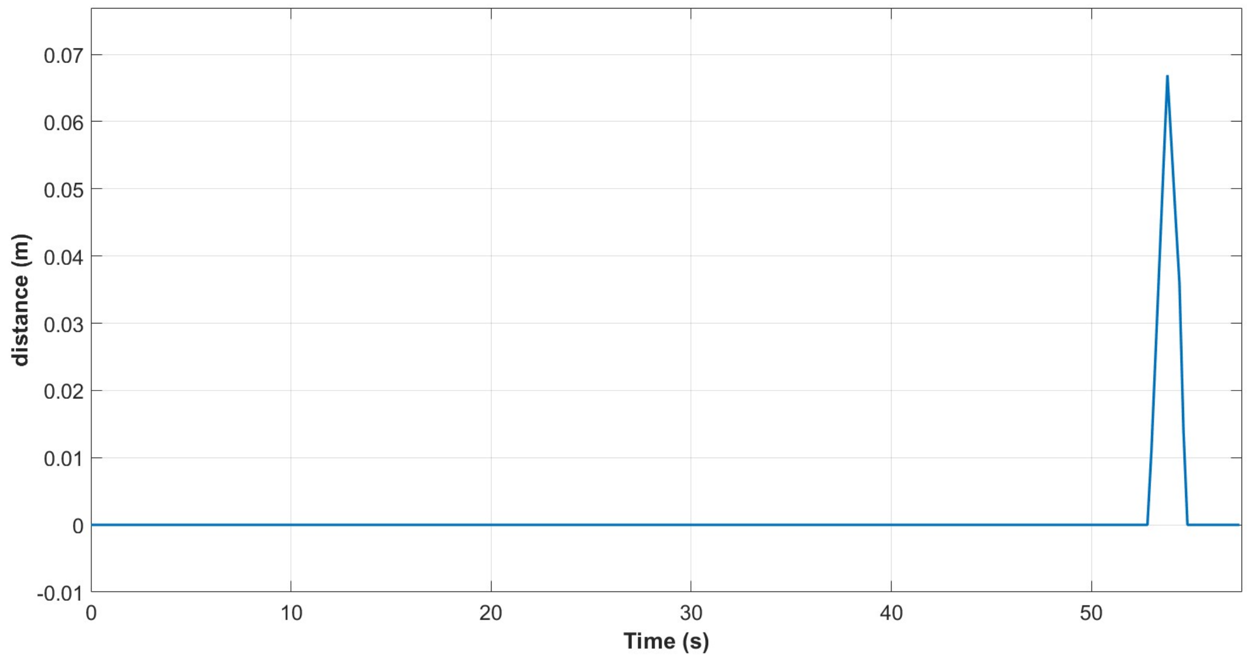


Figure D.60: Center displacement w/ ISMC control during the second test with max speed set to 0.0011 m/sample.

Tables

Max speed	Controller	Track completed?	Sensor time (s)	Run Time (s)	$\frac{\text{Sensor time}}{\text{Run time}}$ (%)
0.0007	PID	Yes	57.200	96.060	59.5
	ISMC	Yes	42.800	80.735	53.0
0.0008	PID	Yes	60.400	88.860	68.0
	ISMC	Yes	39.800	72.730	54.7
0.0009	PID	No	37.200	52.920	70.3
	ISMC	Yes	37.200	65.950	56.4
0.0010	PID	No	14.400	24.180	59.6
	ISMC	Yes	36.400	60.945	59.7
0.0011	PID	No	13.200	22.400	58.9
	ISMC	Yes	35.800	57.535	62.2

Table D.3: Performance, track 2.

					Divided by Run Time	
Max speed	Param.	Index	PID	ISMC	PID	ISMC
0.0007	x	IAE	13.8003	1.5647	0.14366	0.01938
		ISE	2.219	0.090617	0.0231	0.0011224
	y	IAE	5.9976	0.30488	0.062436	0.0037763
		ISE	0.49141	0.0025771	0.0051156	3.192e-05
	z	IAE	1.6272	1.0585	0.01694	0.013111
		ISE	0.68329	0.459	0.0071132	0.0056852
	ϕ	IAE	0.052752	0.054411	0.00054915	0.00067395
		ISE	4.6845e-05	9.8385e-05	4.8766e-07	1.2186e-06
	θ	IAE	0.081642	0.041	0.00084991	0.00050783
		ISE	0.00020867	0.00018864	2.1723e-06	2.3366e-06
	ψ	IAE	0.001902	0.0012615	1.98e-05	1.5626e-05
		ISE	5.9327e-08	3.0969e-08	6.1761e-10	3.8358e-10
	d	IAE	0.13869	0.06187	0.0014438	0.00076633
		ISE	0.019232	0.0027625	0.00020021	3.4217e-05

Continued on next page

Table D.4 – continued from previous page

					Divided by Run Time	
Max speed	Param.	Index	PID	ISMC	PID	ISMC
0.0008	x	IAE	13.8897	1.8009	0.15631	0.024762
		ISE	2.4463	0.1268	0.027529	0.0017434
	y	IAE	6.0704	0.34267	0.068314	0.0047115
		ISE	0.56271	0.0038305	0.0063326	5.2667e-05
	z	IAE	1.6106	1.0433	0.018125	0.014344
		ISE	0.69387	0.45875	0.0078086	0.0063075
	ϕ	IAE	0.05806	0.057257	0.00065339	0.00078725
		ISE	5.9963e-05	0.0001862	6.748e-07	2.5601e-06
	θ	IAE	0.087471	0.041939	0.00098437	0.00057664
		ISE	0.00025015	0.00022777	2.8151e-06	3.1318e-06
	ψ	IAE	0.0017554	0.0011424	1.9755e-05	1.5707e-05
		ISE	5.4592e-08	2.8238e-08	6.1436e-10	3.8826e-10
	d	IAE	0.23048	0.10147	0.0025938	0.0013952
		ISE	0.020913	0.018695	0.00023535	0.00025705
0.0009	x	IAE	8.4943	1.9768	0.16051	0.029975
		ISE	1.582	0.15879	0.029895	0.0024078
	y	IAE	3.8582	0.35367	0.072907	0.0053627
		ISE	0.38644	0.0043284	0.0073023	6.5631e-05
	z	IAE	0.91388	1.0357	0.017269	0.015705
		ISE	0.47439	0.45939	0.0089644	0.0069657
	ϕ	IAE	0.046653	0.053498	0.00088158	0.00081119
		ISE	6.8801e-05	0.00013365	1.3001e-06	2.0266e-06
	θ	IAE	0.069016	0.041104	0.0013042	0.00062326
		ISE	0.00022838	0.00026985	4.3155e-06	4.0917e-06
	ψ	IAE	0.0010554	0.0010431	1.9944e-05	1.5816e-05
		ISE	3.329e-08	2.5837e-08	6.2906e-10	3.9177e-10
	d	IAE	0.15181	0.066272	0.0028687	0.0010049
		ISE	0.0035765	0.0030314	6.7584e-05	4.5965e-05

Continued on next page

Table D.4 – continued from previous page

					Divided by Run Time	
Max speed	Param.	Index	PID	ISMC	PID	ISMC
0.0010	x	IAE	3.9847	2.1285	0.16479	0.034925
		ISE	0.81117	0.19156	0.033547	0.0031431
	y	IAE	1.9565	0.41357	0.080916	0.006786
		ISE	0.24312	0.0061505	0.010054	0.00010092
	z	IAE	0.79317	1.0292	0.032803	0.016887
		ISE	0.47357	0.45933	0.019585	0.0075368
	ϕ	IAE	0.024848	0.057665	0.0010276	0.00094618
		ISE	4.3788e-05	0.00024141	1.8109e-06	3.961e-06
	θ	IAE	0.026834	0.041924	0.0011098	0.0006879
		ISE	7.5766e-05	0.0003034	3.1334e-06	4.9783e-06
	ψ	IAE	0.00044117	0.0009696	1.8245e-05	1.5909e-05
		ISE	1.2794e-08	2.4501e-08	5.2909e-10	4.0202e-10
	d	IAE	0.1238	0.069812	0.0051198	0.0011455
		ISE	0.0040346	0.0033064	0.00016686	5.4251e-05
0.0011	x	IAE	3.8364	2.3751	0.17127	0.041281
		ISE	0.82075	0.24649	0.036641	0.0042842
	y	IAE	1.9741	0.47024	0.088128	0.0081732
		ISE	0.2712	0.0080135	0.012107	0.00013928
	z	IAE	0.78906	1.0225	0.035226	0.017772
		ISE	0.47355	0.45957	0.021141	0.0079877
	ϕ	IAE	0.025443	0.059811	0.0011358	0.0010396
		ISE	4.9068e-05	0.00022264	2.1905e-06	3.8696e-06
	θ	IAE	0.023446	0.043972	0.0010467	0.00076426
		ISE	6.1675e-05	0.00036487	2.7534e-06	6.3417e-06
	ψ	IAE	0.00041372	0.00091521	1.847e-05	1.5907e-05
		ISE	1.2114e-08	2.3036e-08	5.4082e-10	4.0038e-10
	d	IAE	0.19409	0.069579	0.0086649	0.0012093
		ISE	0.0086576	0.003278	0.0003865	5.6974e-05

Table D.4: IAE and ISE results, track 2.

D.1.3 Third Track Results

Figures

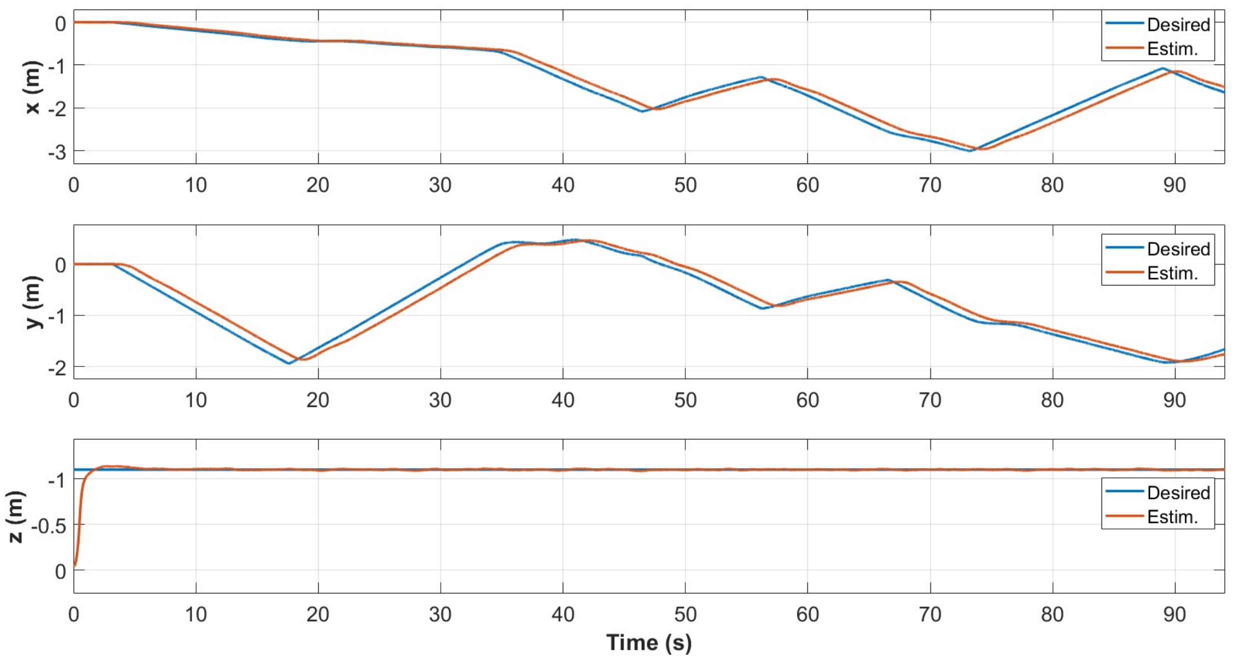


Figure D.61: Position w/ PID control during the third test with max speed set to 0.0007 m/sample.

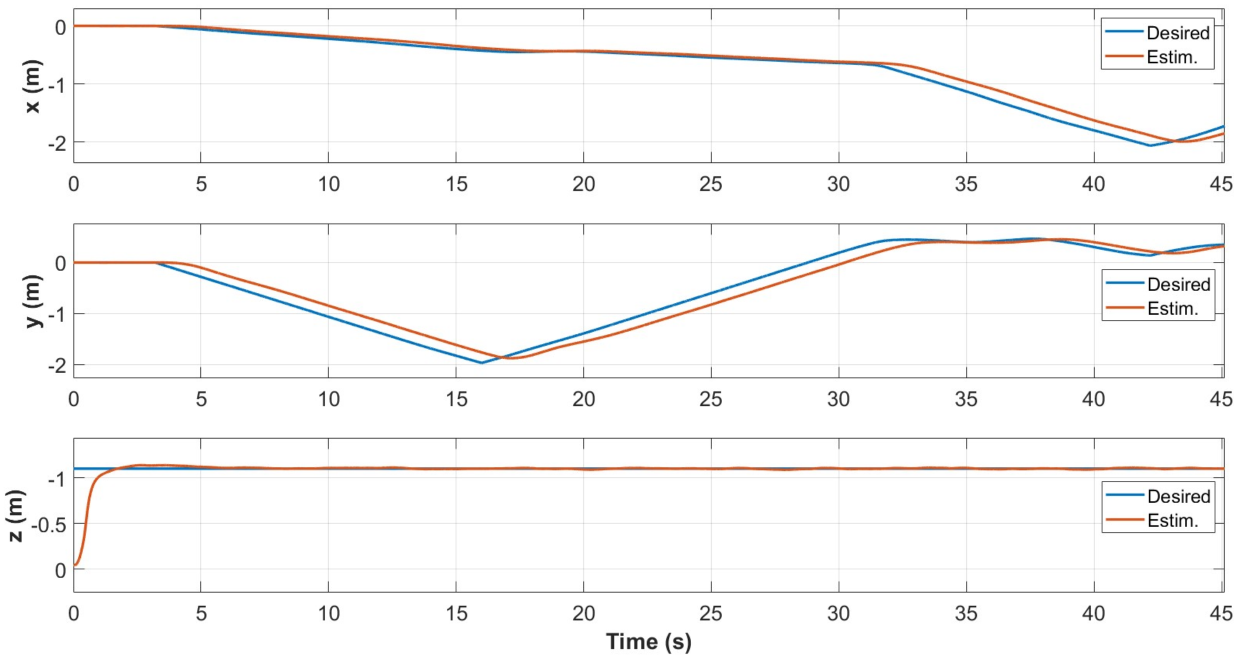


Figure D.62: Position w/ PID control during the third test with max speed set to 0.0008 m/sample.

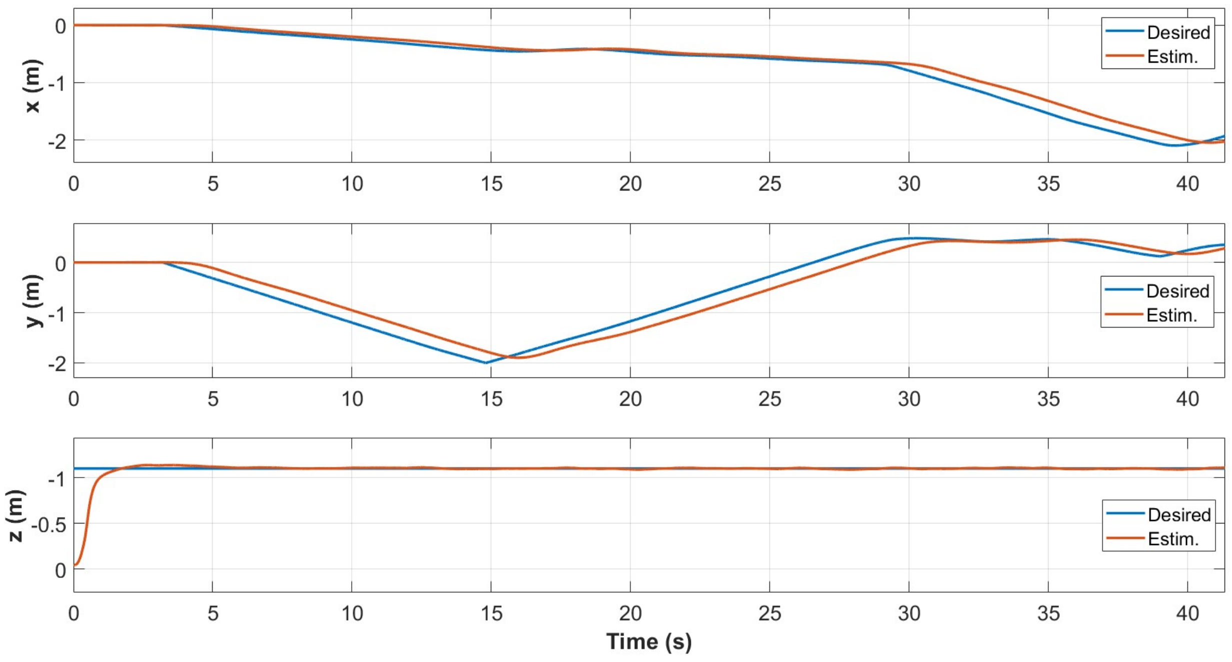


Figure D.63: Position w/ PID control during the third test with max speed set to 0.0009 m/sample.

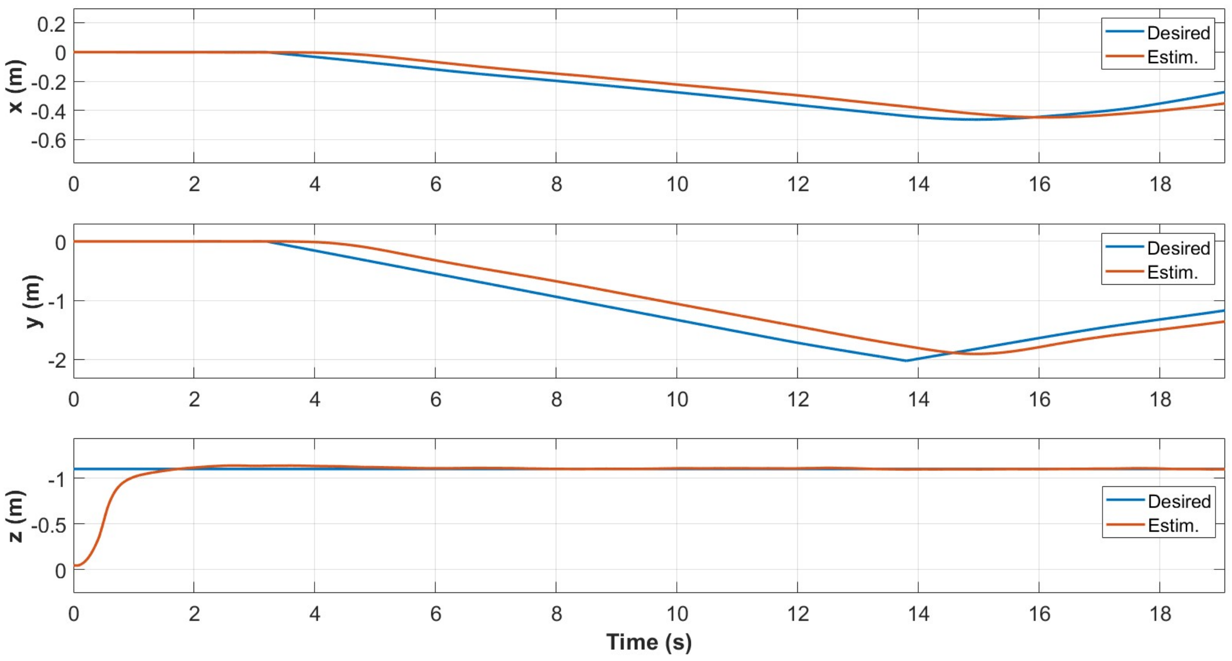


Figure D.64: Position w/ PID control during the third test with max speed set to 0.0010 m/sample.

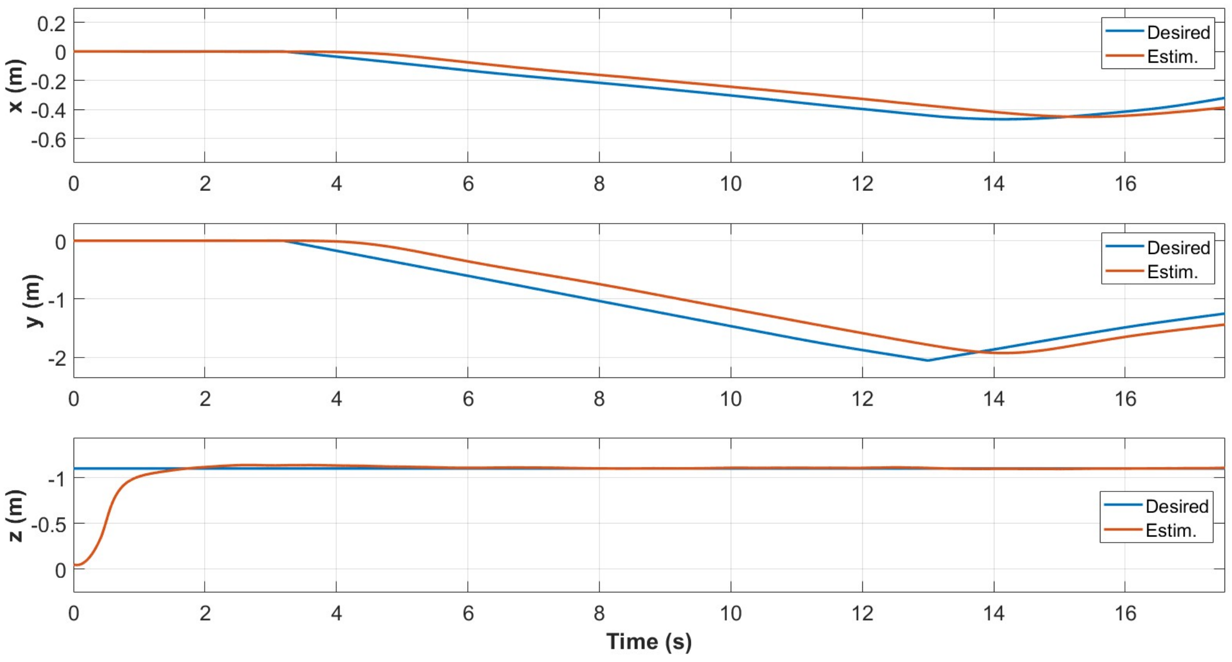


Figure D.65: Position w/ PID control during the third test with max speed set to 0.0011 m/sample.

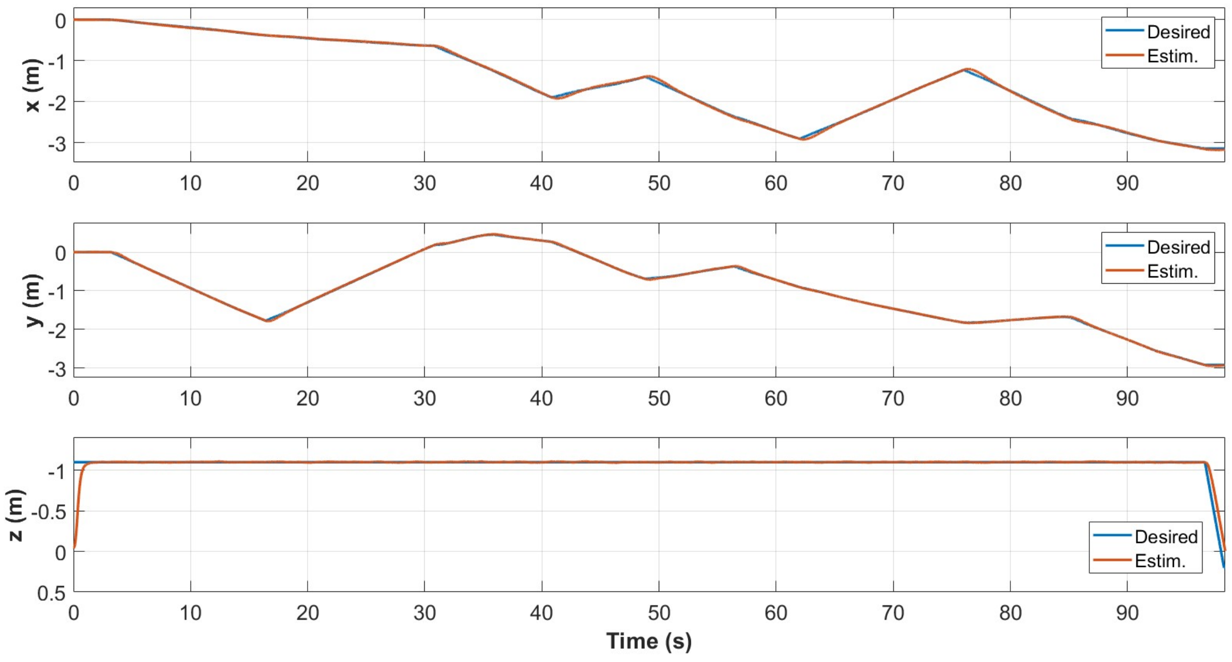


Figure D.66: Position w/ ISMC control during the third test with max speed set to 0.0007 m/sample.

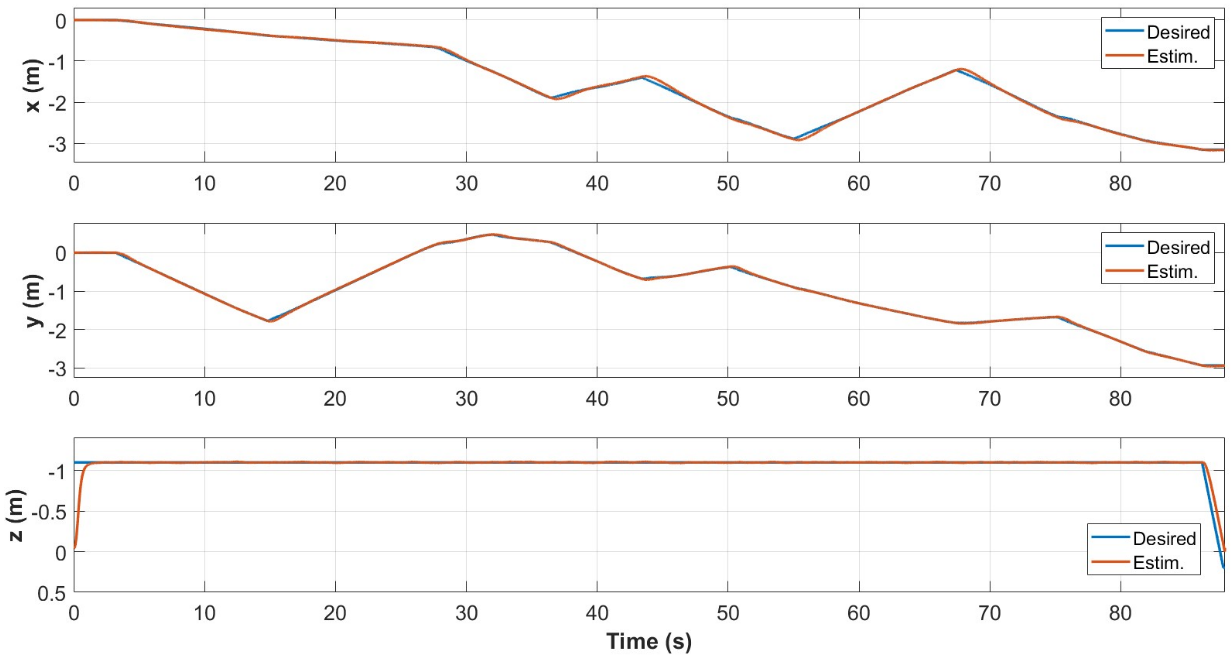


Figure D.67: Position w/ ISMC control during the third test with max speed set to 0.0008 m/sample.

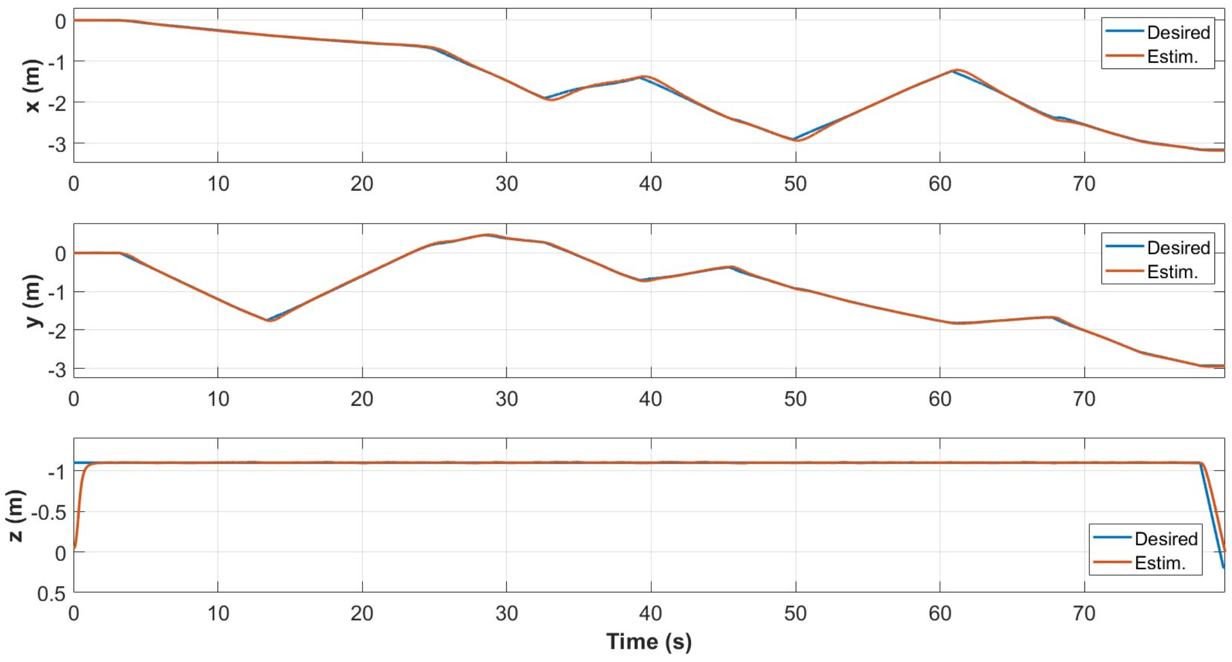


Figure D.68: Position w/ ISMC control during the third test with max speed set to 0.0009 m/sample.

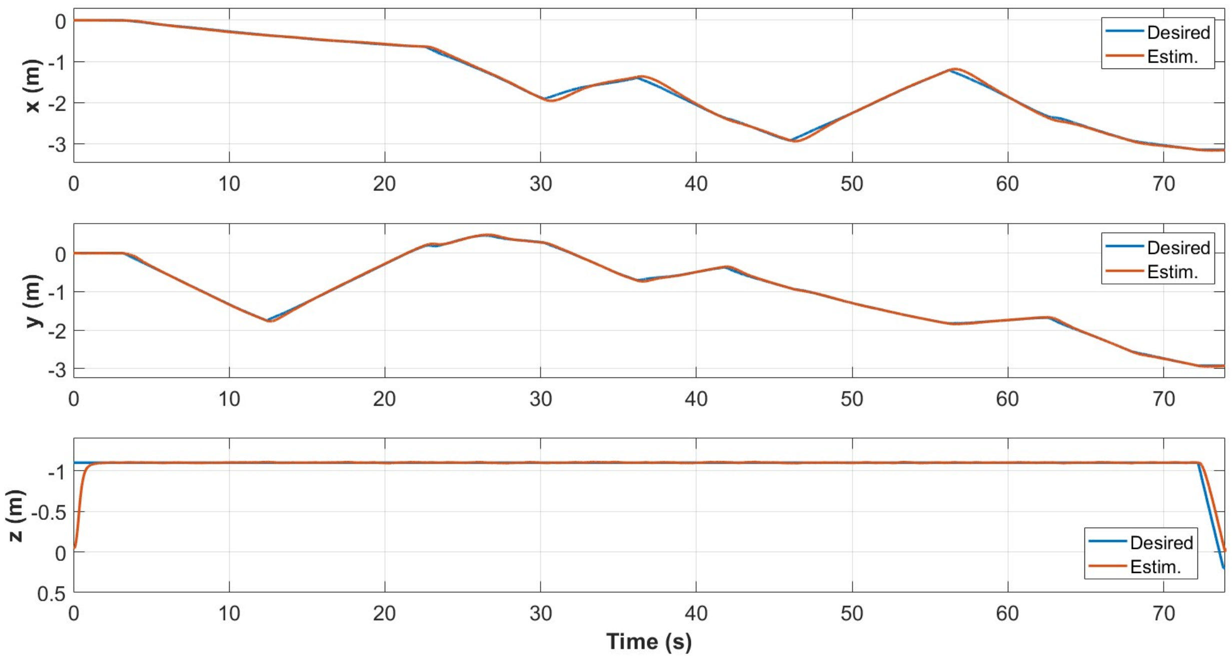


Figure D.69: Position w/ ISMC control during the third test with max speed set to 0.0010 m/sample.

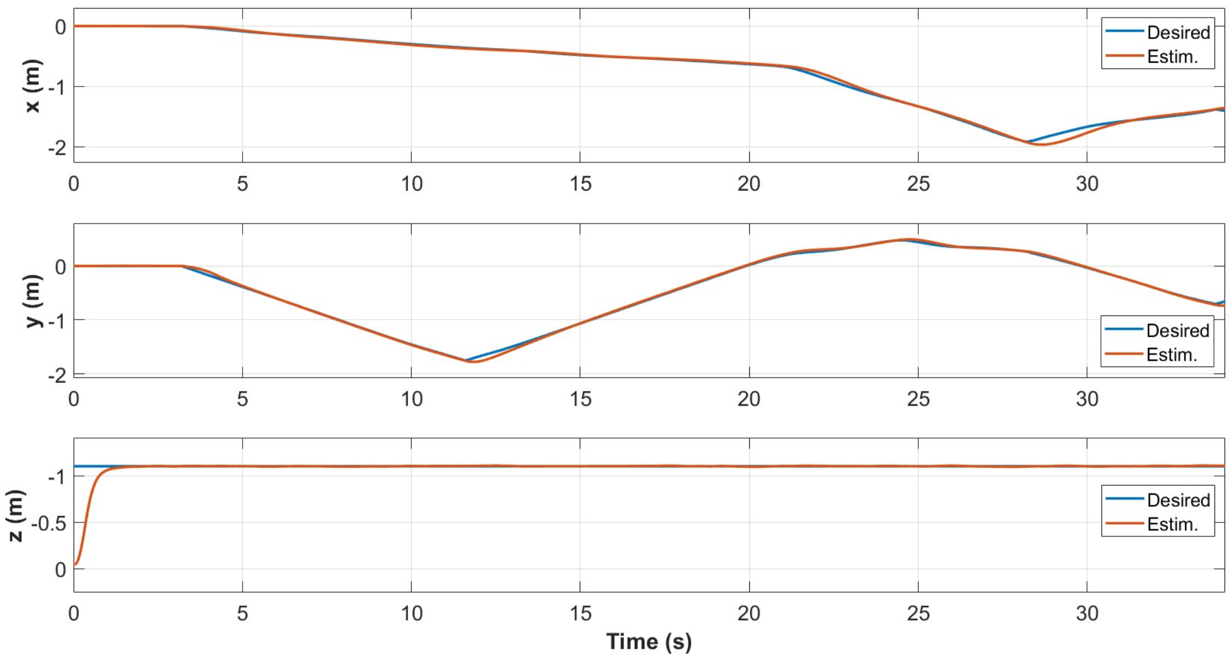


Figure D.70: Position w/ ISMC control during the third test with max speed set to 0.0011 m/sample.

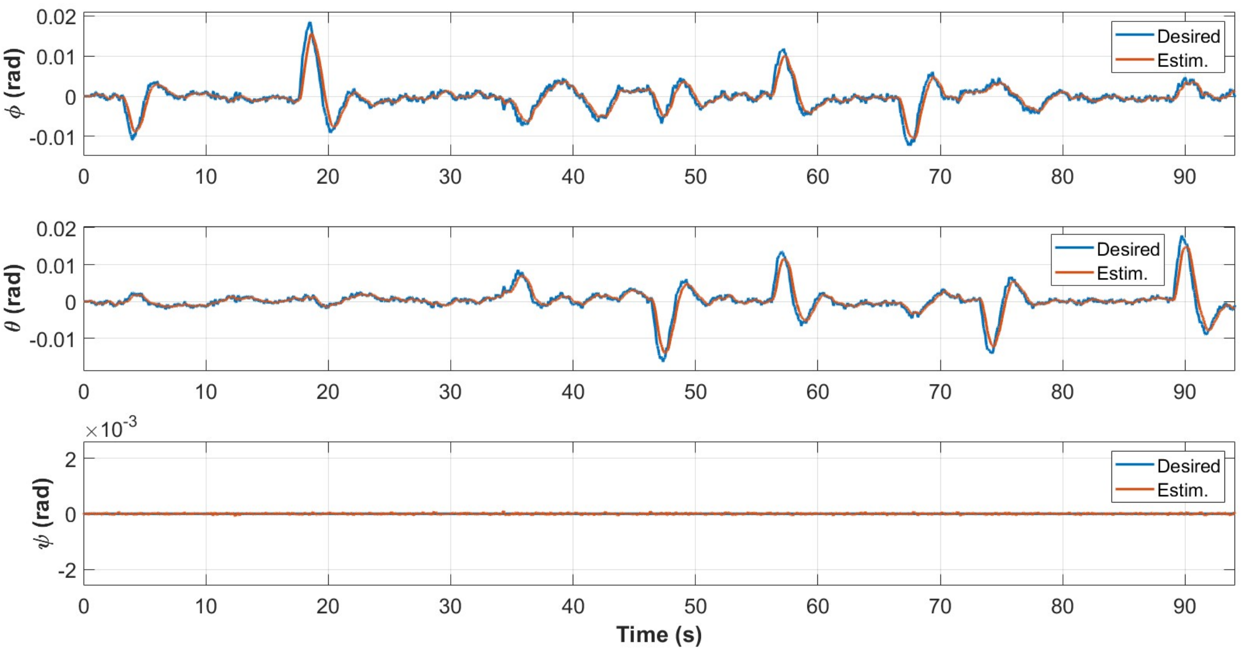


Figure D.71: Attitude w/ PID control during the third test with max speed set to 0.0007 m/sample.

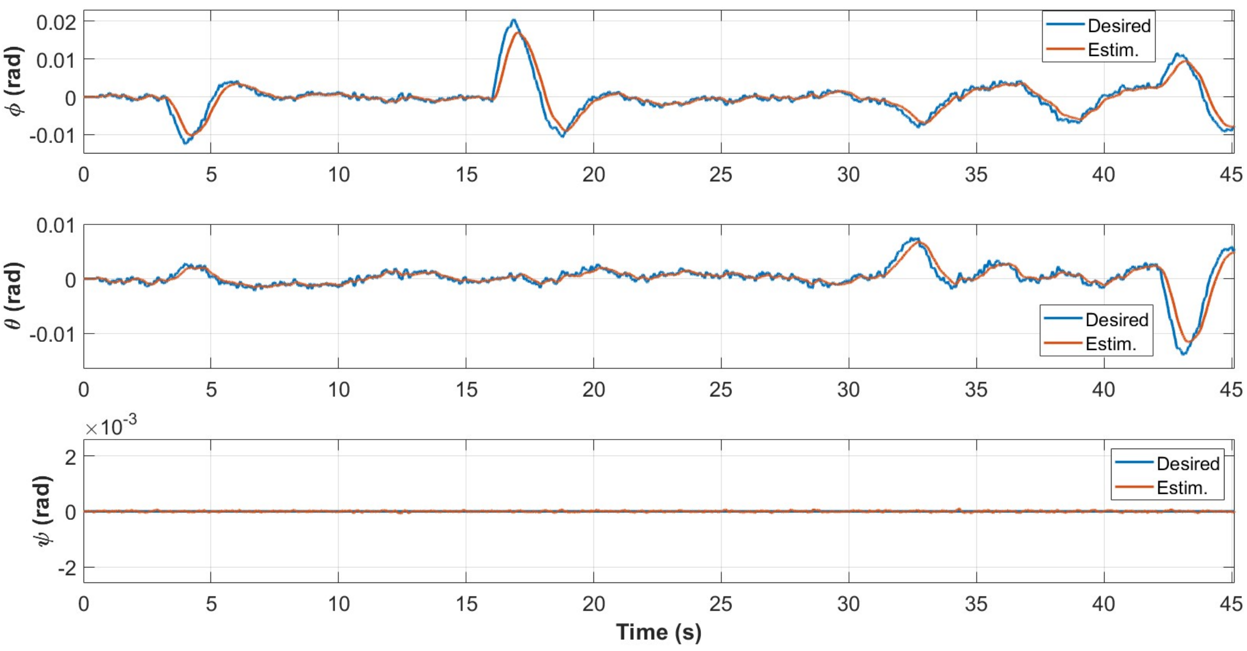


Figure D.72: Attitude w/ PID control during the third test with max speed set to 0.0008 m/sample.

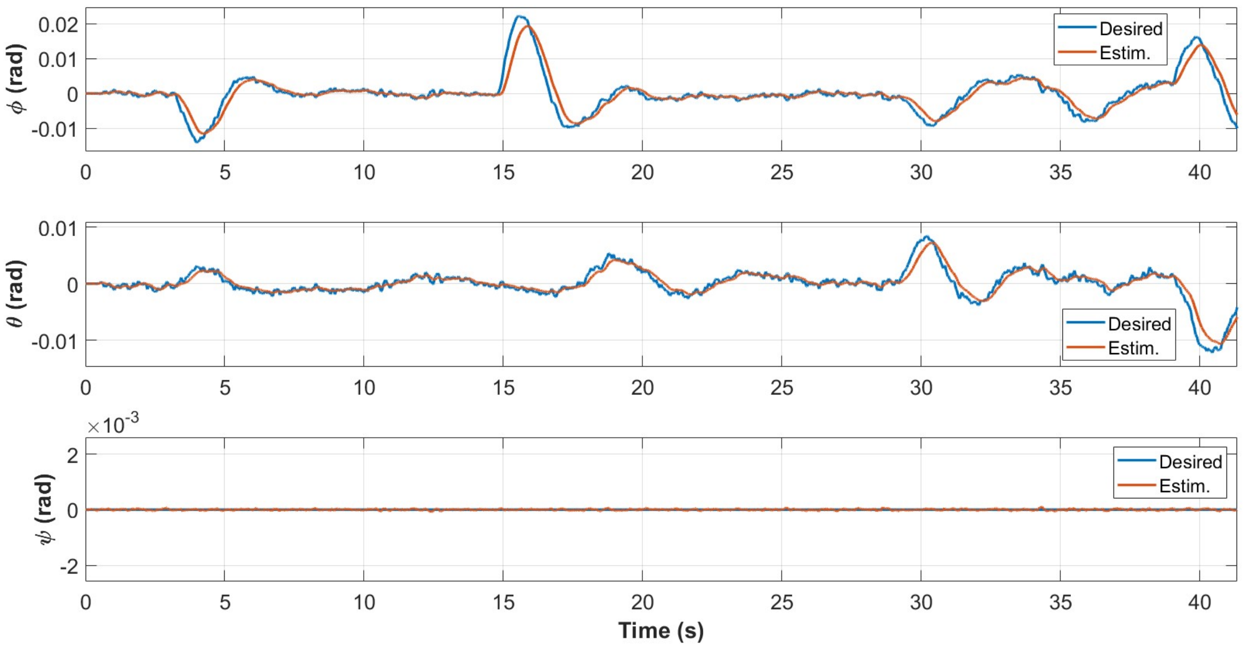


Figure D.73: Attitude w/ PID control during the third test with max speed set to 0.0009 m/sample.

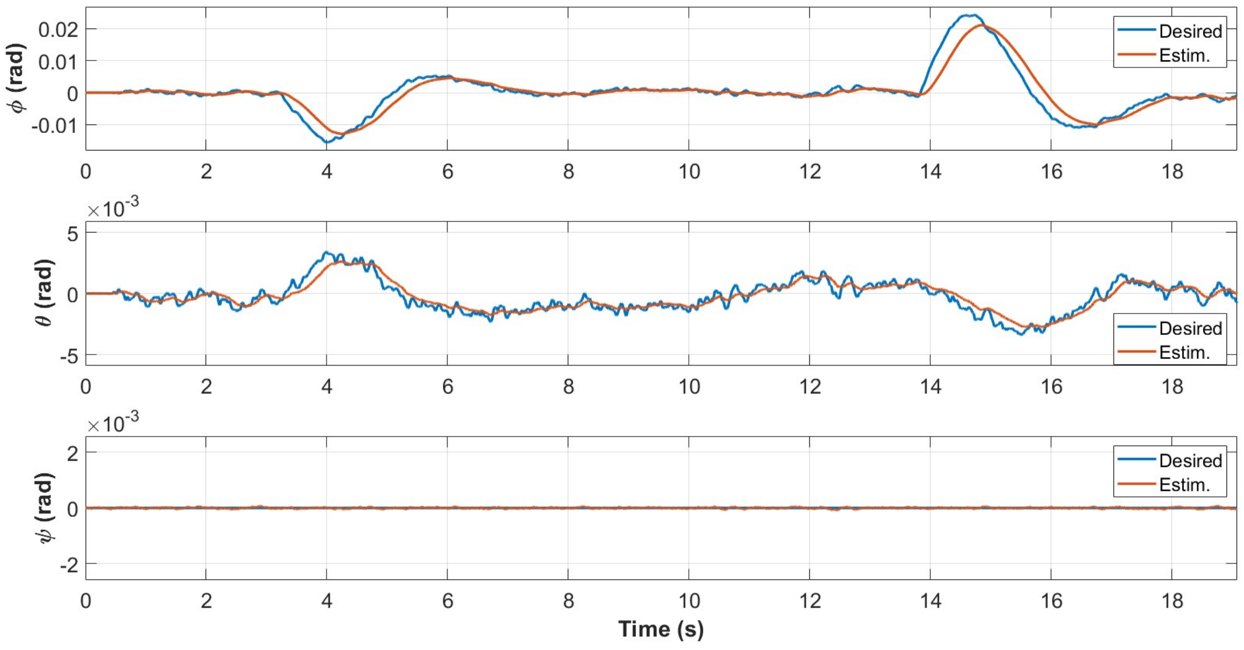


Figure D.74: Attitude w/ PID control during the third test with max speed set to 0.0010 m/sample.

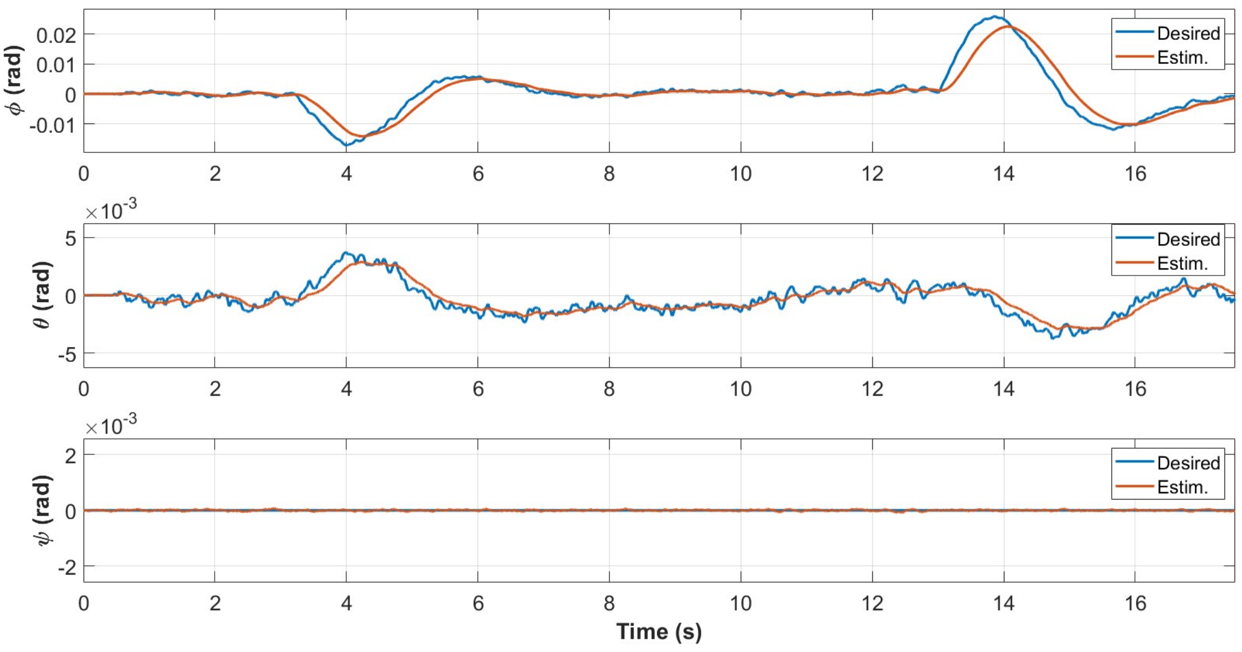


Figure D.75: Attitude w/ PID control during the third test with max speed set to 0.0011 m/sample.

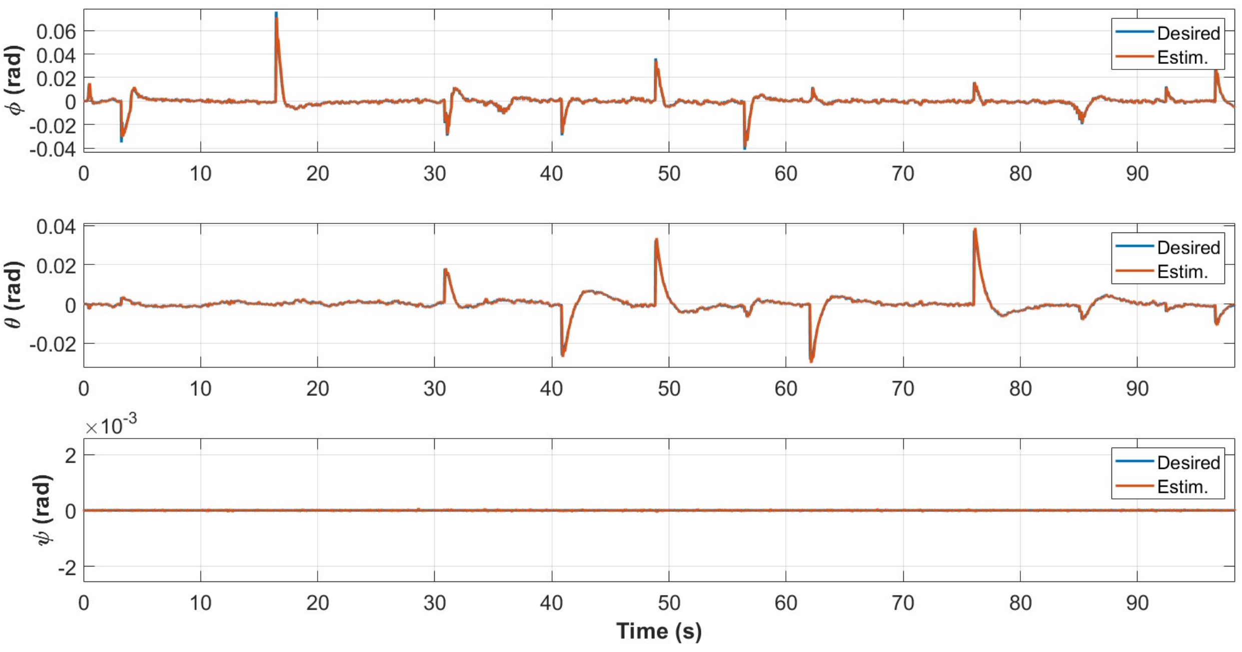


Figure D.76: Attitude w/ ISMC control during the third test with max speed set to 0.0007 m/sample.

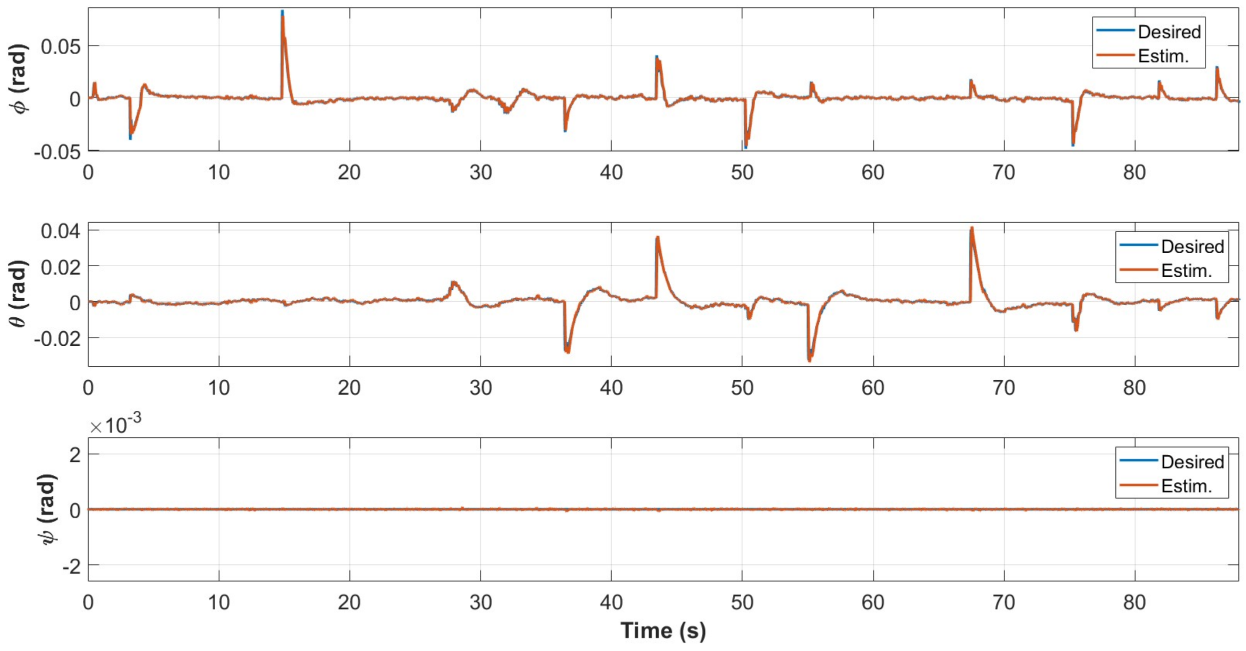


Figure D.77: Attitude w/ ISMC control during the third test with max speed set to 0.0008 m/sample.

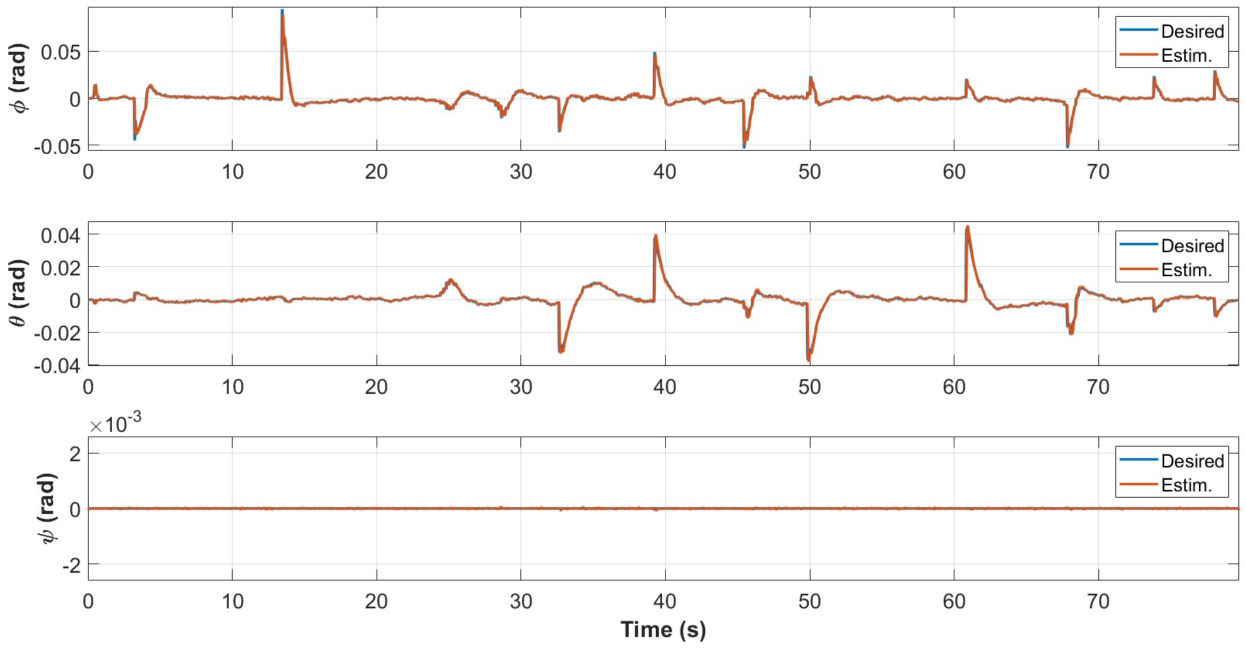


Figure D.78: Attitude w/ ISMC control during the third test with max speed set to 0.0009 m/sample.

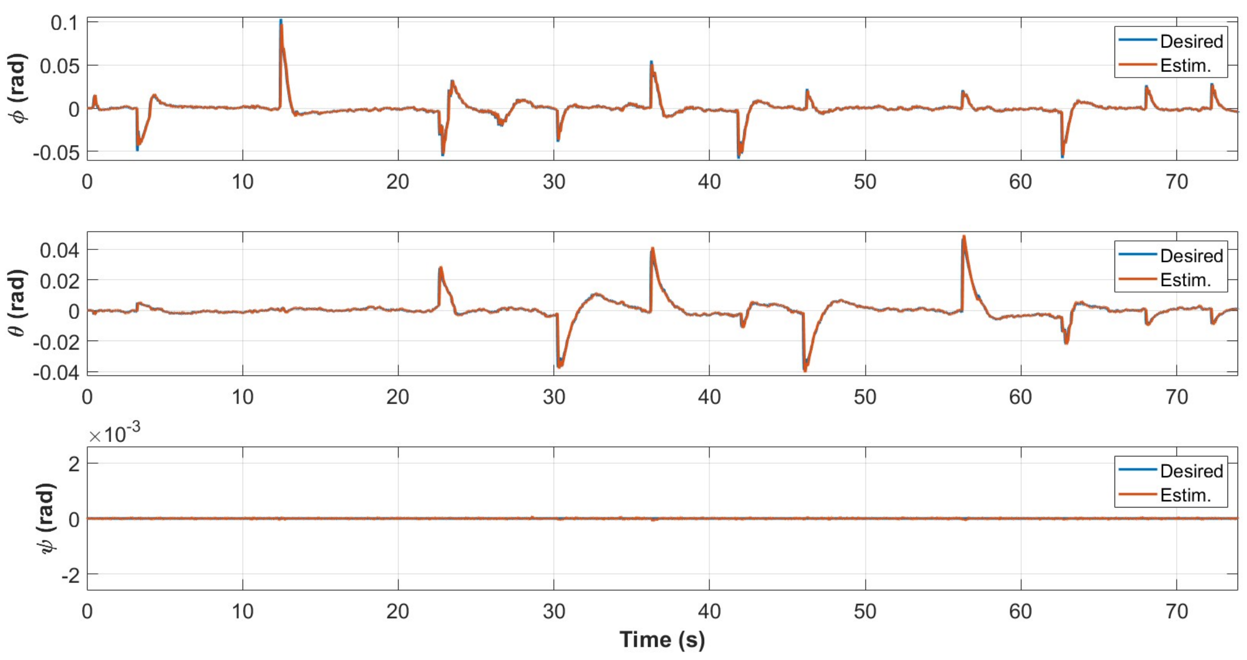


Figure D.79: Attitude w/ ISMC control during the third test with max speed set to 0.0010 m/sample.

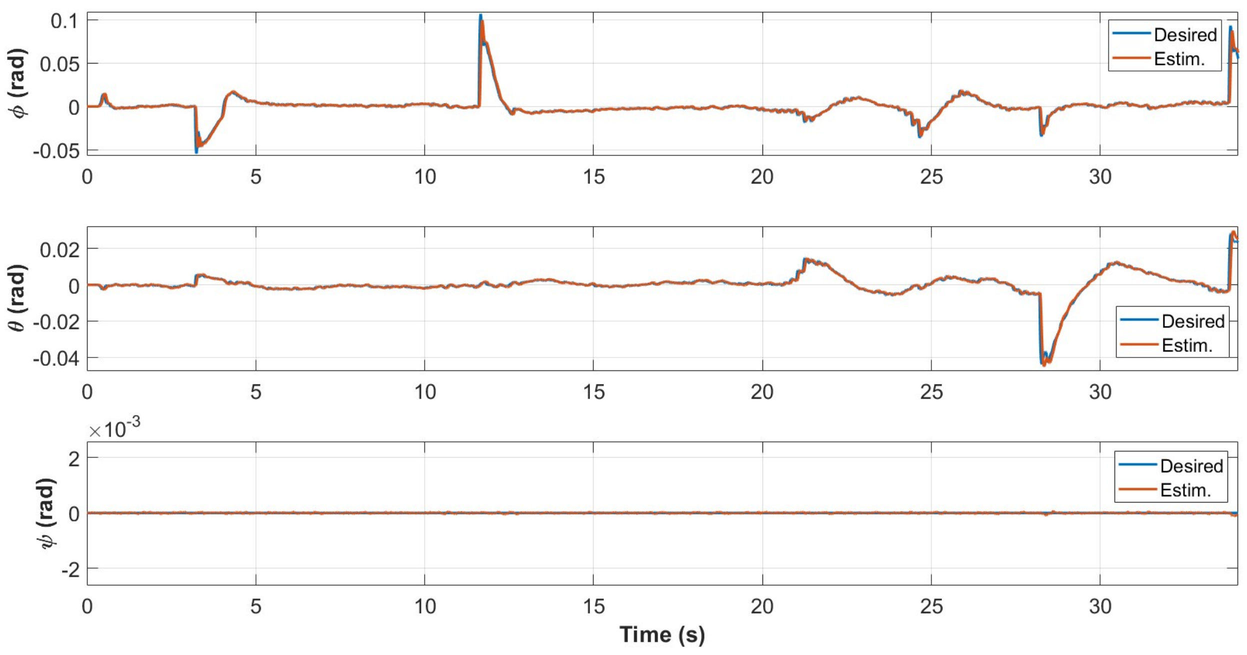


Figure D.80: Attitude w/ ISMC control during the third test with max speed set to 0.0011 m/sample.

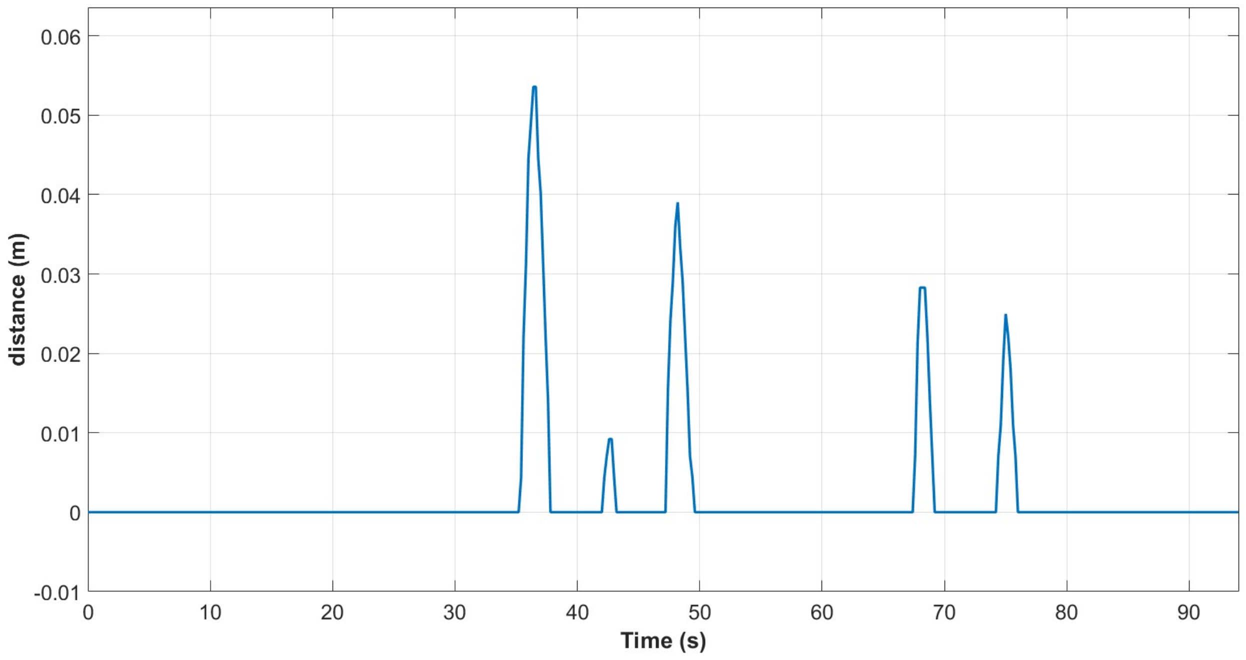


Figure D.81: Center displacement w/ PID control during the third test with max speed set to 0.0007 m/sample.

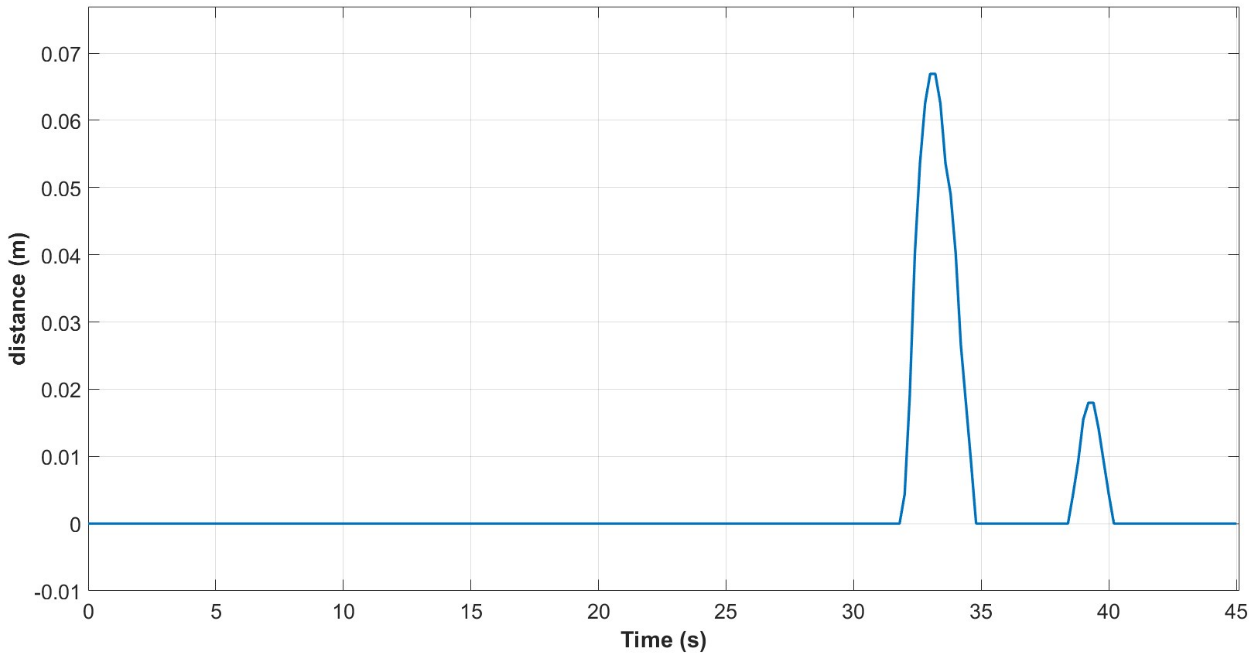


Figure D.82: Center displacement w/ PID control during the third test with max speed set to 0.0008 m/sample.

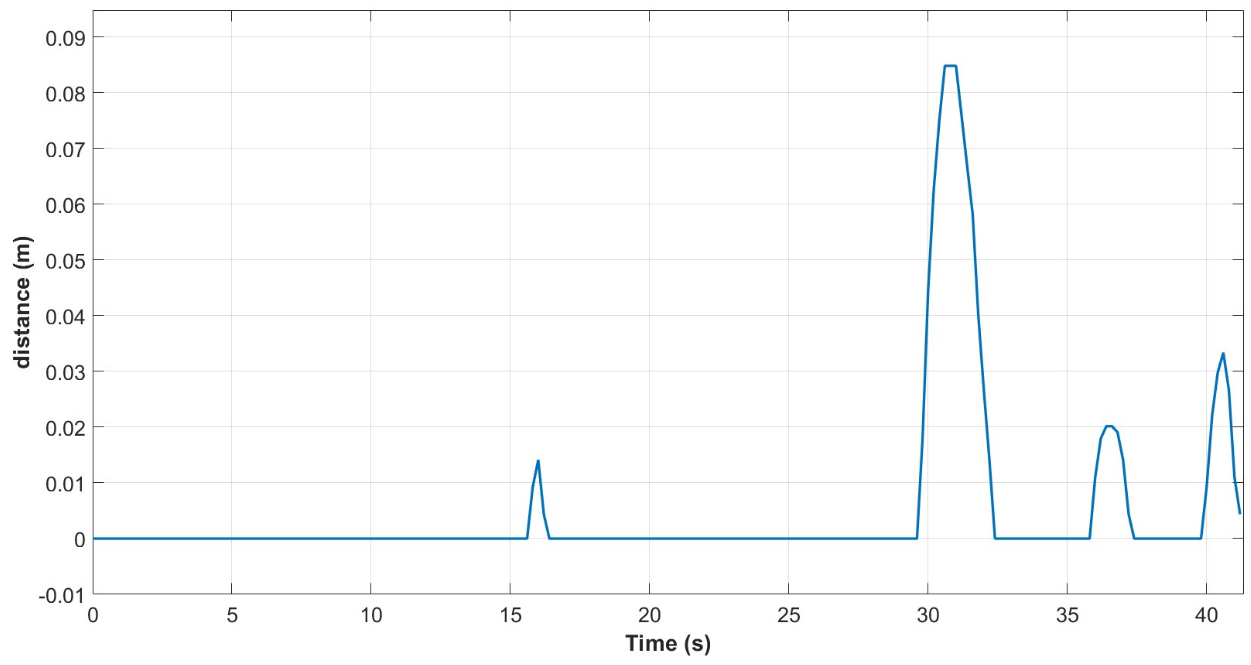


Figure D.83: Center displacement w/ PID control during the third test with max speed set to 0.0009 m/sample.

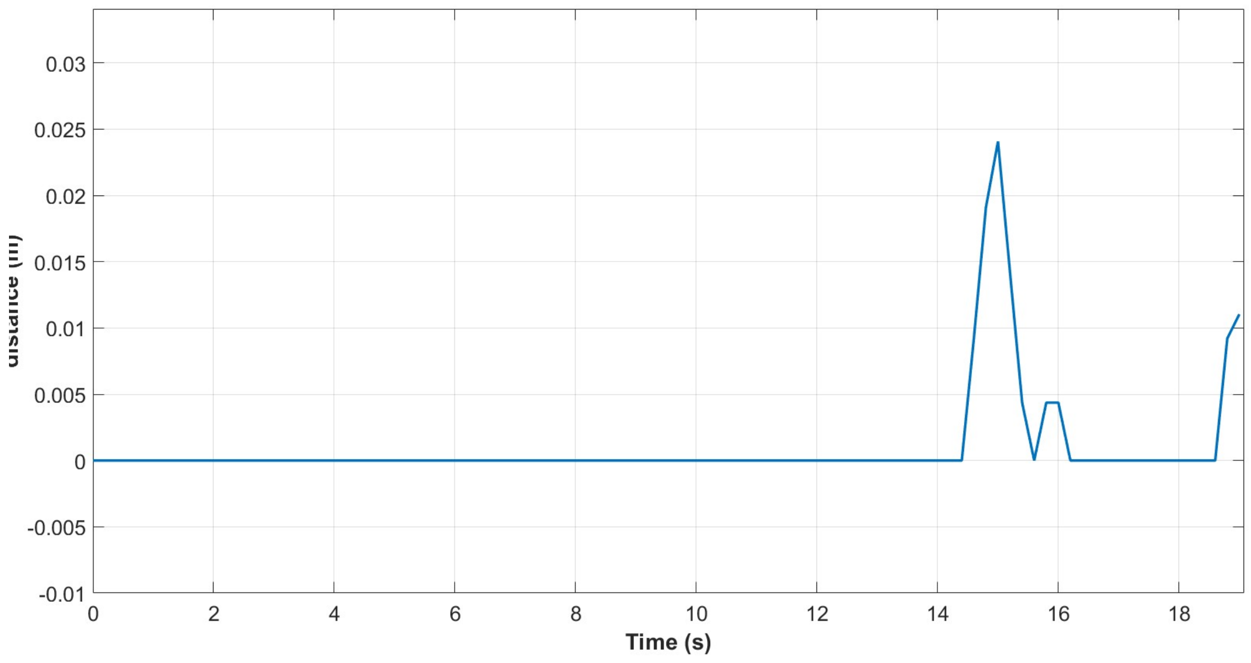


Figure D.84: Center displacement w/ PID control during the third test with max speed set to 0.0010 m/sample.

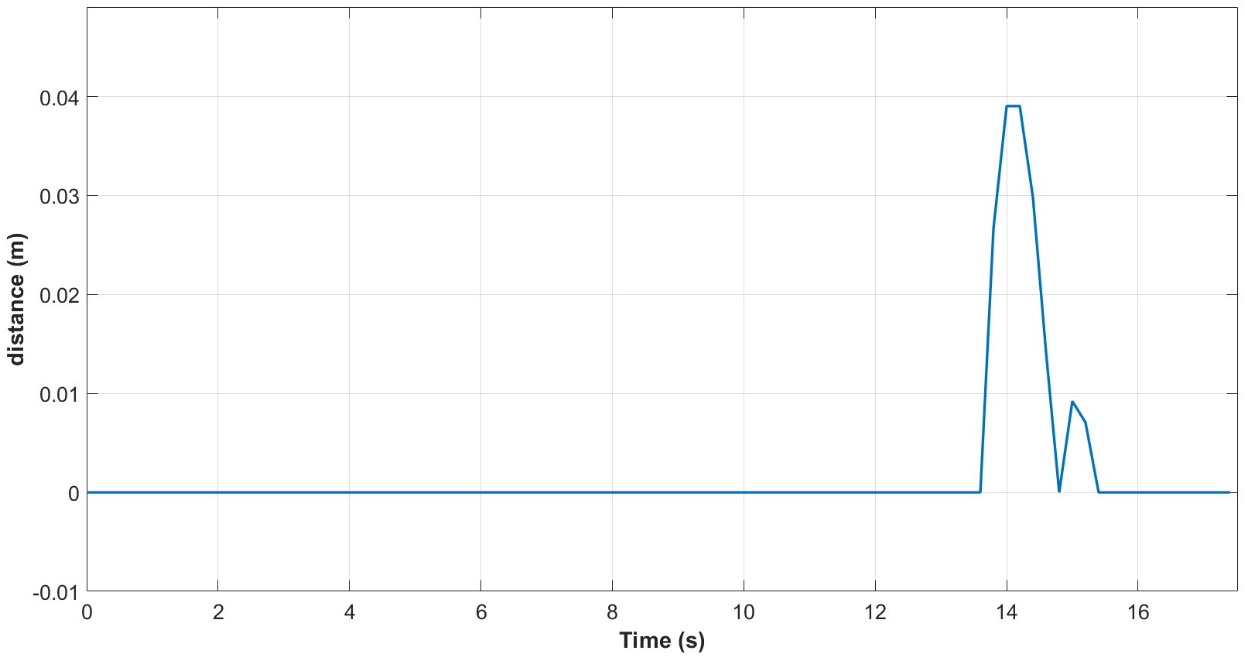


Figure D.85: Center displacement w/ PID control during the third test with max speed set to 0.0011 m/sample.

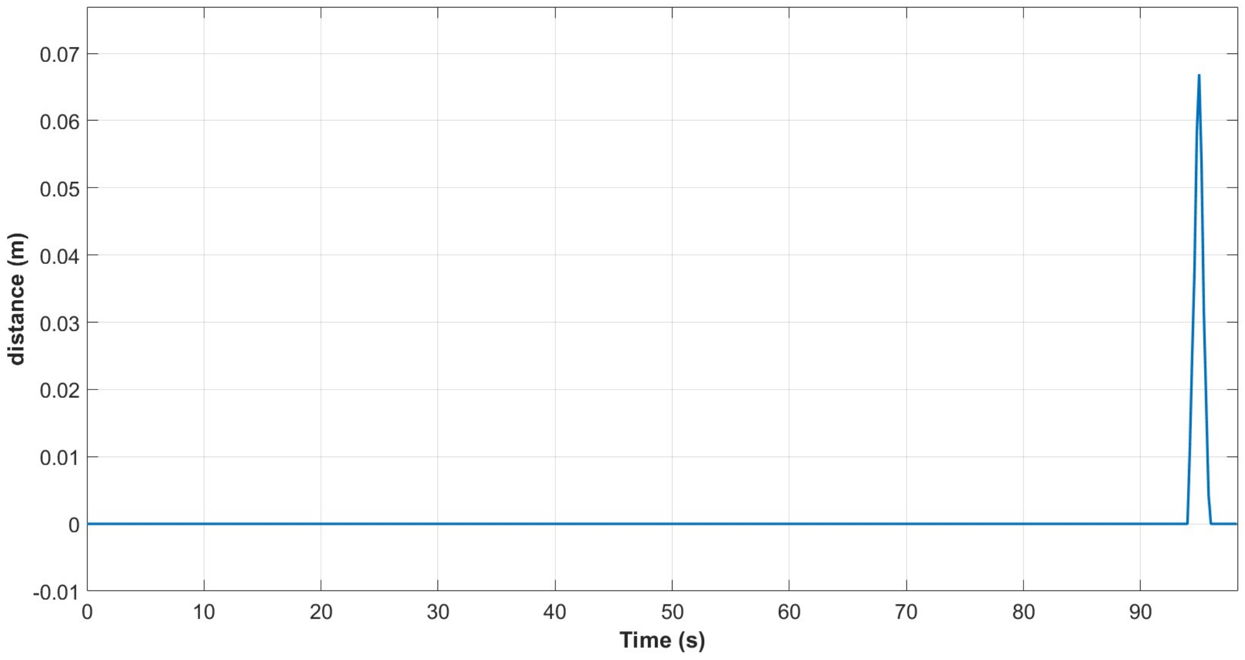


Figure D.86: Center displacement w/ ISMC control during the third test with max speed set to 0.0007 m/sample.

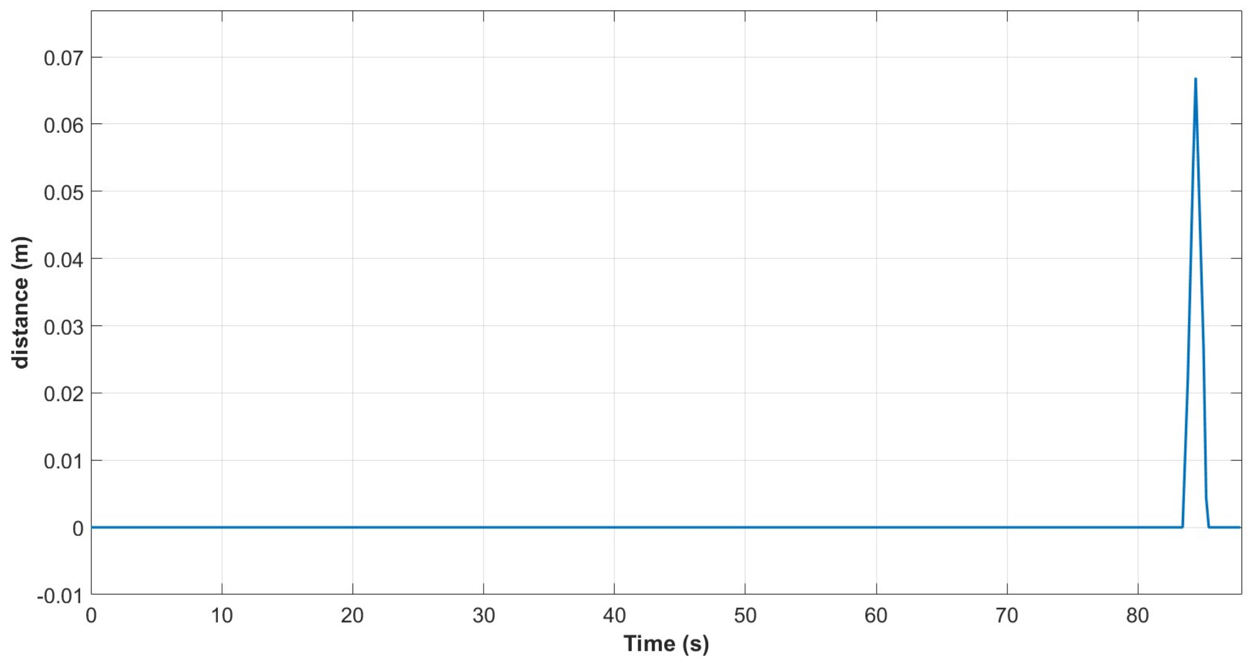


Figure D.87: Center displacement w/ ISMC control during the third test with max speed set to 0.0008 m/sample.

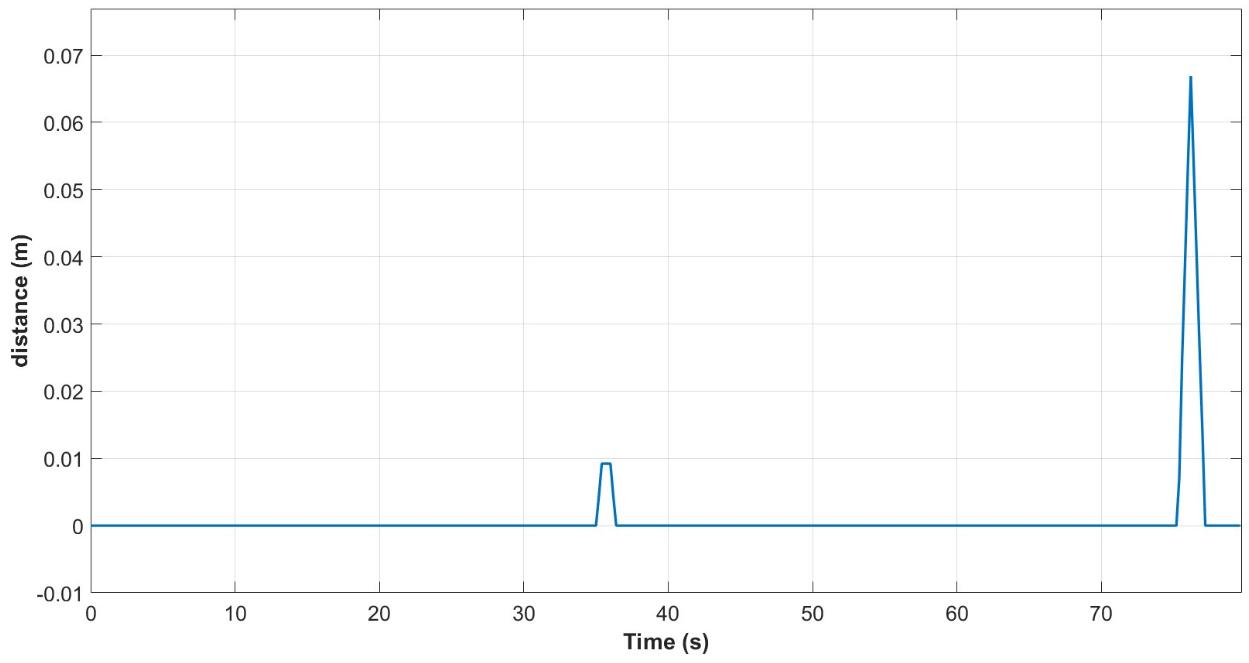


Figure D.88: Center displacement w/ ISMC control during the third test with max speed set to 0.0009 m/sample.

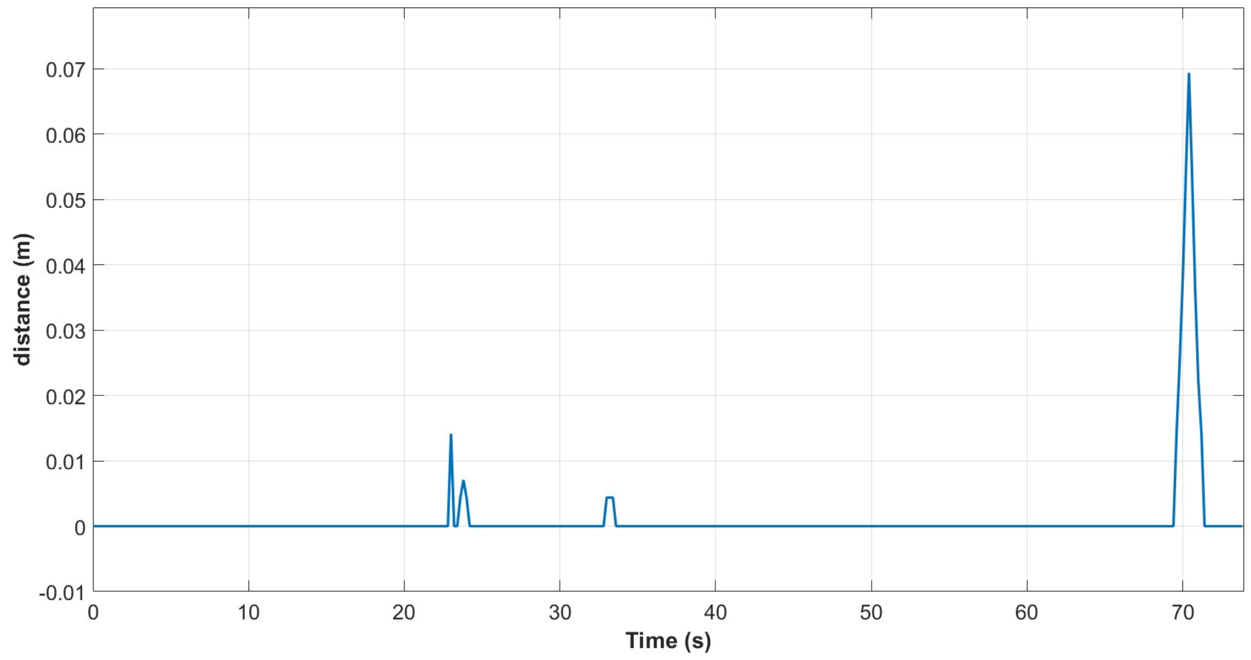


Figure D.89: Center displacement w/ ISMC control during the third test with max speed set to 0.0010 m/sample.

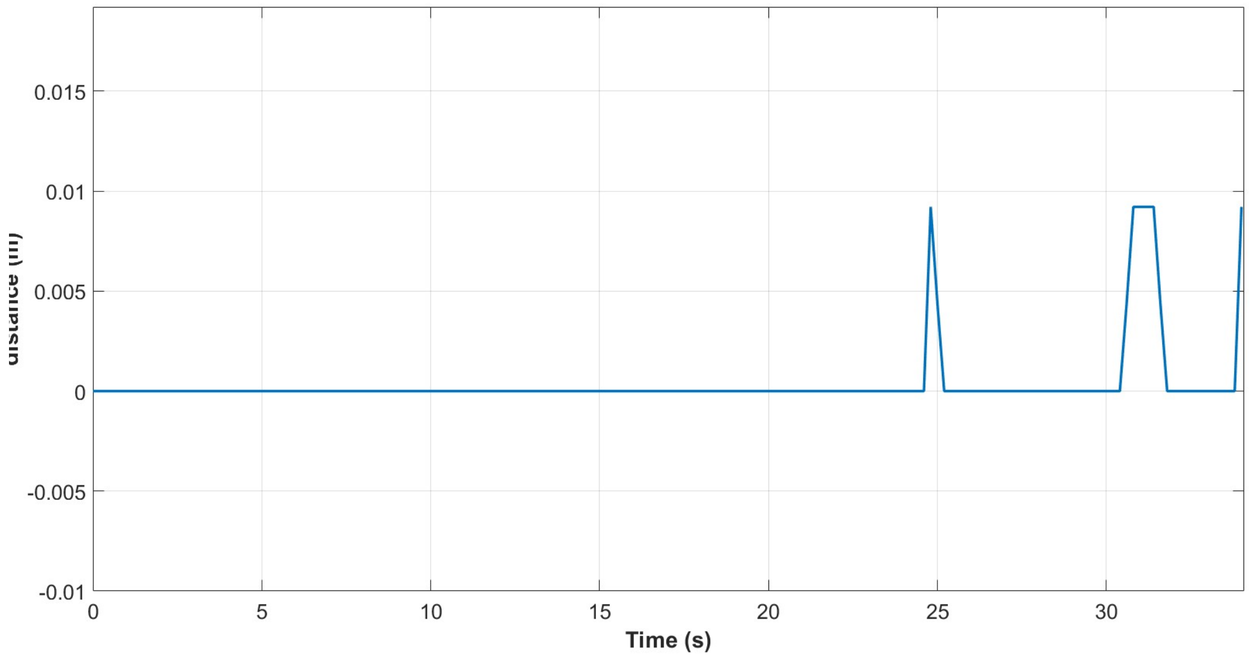


Figure D.90: Center displacement w/ ISMC control during the third test with max speed set to 0.0011 m/sample.

Tables

Max speed	Controller	Track completed?	Sensor time (s)	Run Time (s)	$\frac{\text{Sensor time}}{\text{Run time}}$ (%)
0.0007	PID	No	49.800	94.080	52.9
	ISMC	Yes	50.200	98.330	51.1
0.0008	PID	No	21.400	45.120	47.4
	ISMC	Yes	45.400	87.950	51.6
0.0009	PID	No	17.800	41.340	43.1
	ISMC	Yes	42.800	79.740	53.7
0.0010	PID	No	5.600	19.080	29.4
	ISMC	Yes	40.400	73.945	54.6
0.0011	PID	No	4.800	17.520	27.4
	ISMC	No	17.600	34.080	51.6

Table D.5: Performance, track 3.

					Divided by Run Time	
Max speed	Param.	Index	PID	ISMC	PID	ISMC
0.0007	x	IAE	8.892	1.4831	0.094515	0.015083
		ISE	1.2004	0.056281	0.01276	0.00057237
	y	IAE	10.5247	0.60356	0.11187	0.0061381
		ISE	1.5277	0.011838	0.016238	0.00012039
	z	IAE	1.0378	1.0955	0.011031	0.011142
		ISE	0.47501	0.45761	0.005049	0.0046538
	ϕ	IAE	0.070889	0.07813	0.0007535	0.00079457
		ISE	0.00012907	0.00038614	1.3719e-06	3.927e-06
	θ	IAE	0.065982	0.043039	0.00070134	0.0004377
		ISE	0.00014276	0.00014183	1.5175e-06	1.4424e-06
	ψ	IAE	0.001877	0.0015329	1.9952e-05	1.559e-05
		ISE	5.878e-08	3.7466e-08	6.2479e-10	3.8102e-10
	d	IAE	0.1955	0.061237	0.002078	0.00062277
		ISE	0.0060376	0.002847	6.4175e-05	2.8953e-05

Continued on next page

Table D.6 – continued from previous page

					Divided by Run Time	
Max speed	Param.	Index	PID	ISMC	PID	ISMC
0.0008	x	IAE	2.9587	1.5158	0.065574	0.017235
		ISE	0.37198	0.076493	0.0082442	0.00086973
	y	IAE	6.3238	0.66016	0.14016	0.0075061
		ISE	1.2036	0.01531	0.026675	0.00017407
	z	IAE	0.88522	1.0802	0.019619	0.012282
		ISE	0.47416	0.46368	0.010509	0.0052721
	ϕ	IAE	0.043633	0.077852	0.00096704	0.00088519
		ISE	0.00010565	0.00049208	2.3416e-06	5.595e-06
	θ	IAE	0.02485	0.042291	0.00055075	0.00048085
		ISE	3.8681e-05	0.00015515	8.573e-07	1.7641e-06
	ψ	IAE	0.0008845	0.0013791	1.9603e-05	1.5681e-05
		ISE	2.7608e-08	3.4183e-08	6.1188e-10	3.8866e-10
	d	IAE	0.1331	0.063281	0.0029499	0.00071951
		ISE	0.0061888	0.0029225	0.00013716	3.3229e-05
0.0009	x	IAE	2.9434	1.643	0.071199	0.020604
		ISE	0.40173	0.10006	0.0097176	0.0012548
	y	IAE	6.3773	0.7528	0.15426	0.0094407
		ISE	1.3457	0.020868	0.032552	0.0002617
	z	IAE	0.86882	1.0574	0.021016	0.01326
		ISE	0.47406	0.45879	0.011467	0.0057535
	ϕ	IAE	0.046698	0.078139	0.0011296	0.00097992
		ISE	0.00013702	0.00059814	3.3146e-06	7.5011e-06
	θ	IAE	0.023463	0.0435	0.00056756	0.00054552
		ISE	2.943e-05	0.00019669	7.119e-07	2.4666e-06
	ψ	IAE	0.00081197	0.0012631	1.9641e-05	1.584e-05
		ISE	2.5347e-08	3.1678e-08	6.1314e-10	3.9727e-10
	d	IAE	0.20117	0.07435	0.0048661	0.00093241
		ISE	0.011033	0.0030643	0.00026687	3.8428e-05

Continued on next page

Table D.6 – continued from previous page

					Divided by Run Time	
Max speed	Param.	Index	PID	ISMC	PID	ISMC
	x	IAE	0.74985	1.7772	0.0393	0.024034
		ISE	0.039864	0.11726	0.0020893	0.0015858
	y	IAE	3.2938	0.84048	0.17263	0.011366
		ISE	0.75661	0.027619	0.039655	0.0003735
	z	IAE	0.7738	1.0493	0.040555	0.014191
		ISE	0.47345	0.45967	0.024814	0.0062164
	ϕ	IAE	0.026706	0.083497	0.0013997	0.0011292
		ISE	0.00010546	0.00078213	5.527e-06	1.0577e-05
	θ	IAE	0.0068511	0.044671	0.00035907	0.00060412
		ISE	3.9226e-06	0.0002316	2.0559e-07	3.132e-06
	ψ	IAE	0.00034871	0.0011901	1.8276e-05	1.6094e-05
		ISE	1.0232e-08	3.053e-08	5.3627e-10	4.1287e-10
	d	IAE	0.018876	0.073863	0.00098929	0.00099889
		ISE	0.00028653	0.0030377	1.5017e-05	4.1081e-05
0.0011	x	IAE	0.71348	0.62637	0.040723	0.01838
		ISE	0.039756	0.03756	0.0022692	0.0011021
	y	IAE	3.2087	0.49648	0.18314	0.014568
		ISE	0.80394	0.019058	0.045887	0.00055921
	z	IAE	0.76869	0.54892	0.043875	0.016107
		ISE	0.47342	0.35126	0.027022	0.010307
	ϕ	IAE	0.027603	0.044592	0.0015755	0.0013084
		ISE	0.00011653	0.00063927	6.6515e-06	1.8758e-05
	θ	IAE	0.006804	0.018443	0.00038836	0.00054116
		ISE	4.2322e-06	8.2389e-05	2.4156e-07	2.4175e-06
	ψ	IAE	0.00030983	0.00052974	1.7685e-05	1.5544e-05
		ISE	8.8758e-09	1.3862e-08	5.0661e-10	4.0675e-10
	d	IAE	0.032998	0.012755	0.0018835	0.00037428
		ISE	0.00099667	0.00010481	5.6887e-05	3.0753e-06

Table D.6: IAE and ISE results, track 3.

D.2 MATLAB Code

```
%% startVars.m - Initialize variables
% This script initializes variables and buses required for the model to
% work.

% Copyright 2013-2019 The MathWorks, Inc.

% Register variables in the workspace before the project is loaded
initVars = who;

% Variants Conditions
asbVariantDefinition;
VSS_COMMAND = 0;       % 0: Signal builder, 1: Joystick, 2: Pre-saved data, 3: Pre-saved data in a Spreadsheet
VSS_SENSORS = 1;       % 0: Feedthrough, 1: Dynamics
VSS_ENVIRONMENT = 0;   % 0: Constant, 1: Variable
VSS_VISUALIZATION = 3; % 0: Scopes, 1: Send values to workspace, 2: FlightGear, 3: Simulink 3D.
VSS_VEHICLE = 1;       % 0: Linear Airframe, 1: Nonlinear Airframe.

% Bus definitions
asbBusDefinitionCommand;
asbBusDefinitionSensors;
asbBusDefinitionEnvironment;
asbBusDefinitionStates;

% Enum definitions
asbEnumDefinition;

% Sampling rate
Ts= 0.005;

% Simulation time
TFinal = 1000;

% Geometric properties
thrustArm = 0.10795;
```

```matlab

% Initial conditions
% If they already exist in the workspace, save them to a .mat-file
if exist('init','var')
    save('init.mat','init');
% If they don't, but exist in a .mat-file, load that into the
   workspace
elseif exist('init.mat','file')
    load('init.mat');
% Otherwise, use standard values
else
    init.date = [2017 1 1 0 0 0];
    init.posLLA = [42.299886 -71.350447 71.3232];
    init.posNED = [3.3548 0.6575 -0.046];
    init.vb = [0 0 0];
    init.euler = [0 0 0];
    init.angRates = [0 0 0];
end

% init.date = [2017 1 1 0 0 0];
% init.posLLA = [42.299886 -71.350447 71.3232];
% init.posNED = [3.3548 0.6575 -0.046];
% init.vb = [0 0 0];
% init.euler = [0 0 0];
% init.angRates = [0 0 0];

% Initialize States:
States = Simulink.Bus.createMATLABStruct('StatesBus');
States.V_body = init.vb';
States.Omega_body = init.angRates';
States.Euler = init.euler';
States.X_ned = init.posNED';
States.LLA = init.posLLA;
States.DCM_be = angle2dcm(init.euler(3),init.euler(2),init.euler(1)
   );

% Environment
rho = 1.184;
g = 9.81;
```

```

% Variables
% Load MAT file with model for persistence
load('modelParrot');
% Obtain vehicle variables
vehicleVars;
% Obtain sensor variables
sensorsVars;
% Obtain controller variables
controllerVars;
% Obtain command variables
commandVars;
% Obtain estimator variables
estimatorVars;
% Obtain visualization variables
visualizationFlightGearVars;

% Simulation Settings
takeOffDuration = 1;

%% Custom Variables
% Add your variables here:
% myvariable = 0;

%% For circle detection
% pixel threshold for circle
t_circle = 1000;

%% Corner sensor variables
% width/thickness of corner sensors
w_corner = 2;
% length corner sensors
l_corner = 36;
% distance from center of close corner sensor
d_close = 30;
% distance from center of far corner sensor
d_far = 59;
% threshold for corner detection
t_far = 10;
```

```
t_close = 10;

%% Side sensor variables
% width/thickness of side sensors
w_sides = 2;
% length of front side sensors
l_front = 32;
% length of back side sensors
l_back = 59;
% distance from center of side sensors
d_sides = 16;
% weighting of front sensors and back sensors
m_front = 10;
m_back = 1;

%% Vision mask variables
% tunnel vision width/thickness
w_tunnel = 20; % default: 20
% distance from center to tunnel vision
d_tunnel = 0;
% distance from center of left/right region
d_region = 10;

%% X- and Y-matrices
% Used for direction vector calculations and creation of image
   regions
[X,Y] = meshgrid(nonzeros(60:-1:-60),nonzeros(-80:80));
X = X';
Y = Y';

% Some other variants
% [X2,Y2] = meshgrid(nonzeros(2*60:-2:-2*60),nonzeros(-2*80:2:2*80)
   );
% X2 = X2';
% Y2 = Y2';
%
% Xequal = ones(120,160);
% Xequal(61:end,:) = -1;
%
```

```
% Yequal = ones(120,160);
% Yequal(:,1:80) = -1;
%
% [Xpw2,Ypw2] = meshgrid([2.^(59:-1:0),-2.^(0:59)],[-2.^(79:-1:0)
    ,2.^(0:79)]);
% Xpw2 = Xpw2';
% Ypw2 = Ypw2';
% Xreversed = ones(120,160);
% Xreversed(1:60,:) = X(60:-1:1,:);
% Xreversed(61:end,:) = -1*X(1:60,:);
%
% Yreversed = ones(120,160);
% Yreversed(:,1:80) = Y(:,80:-1:1);
% Yreversed(:,81:end) = -1*Y(:,1:80);

%% Drone speed variables
% Minimum speed
min_speed = 0.0005;
% Maximum speed
max_speed = 0.0011;
% Gain and offset for mapping between pixels and drone speed
max_pixels = min(w_tunnel,20)*(60-d_tunnel);
min_pixels = -2*w_sides*(m_front*l_front+m_back*l_back);
a = (max_speed-min_speed)/(max_pixels-min_pixels);
b = min_pixels-min_speed/a;

%% Control parameters for Sliding Mode Control
%% x-position
lambda_x = 0.6766;
rho_x = 0.2145;
zeta_x = 1.8376;
k_x = 0.01;
epsilon_x = 0.3430;

%% y-position
lambda_y = 0.6766;
rho_y = 0.2145;
zeta_y = 1.8376;
k_y = 0.01;
```

```
epsilon_y = 0.3430;

%% Altitude
lambda_z = 5.6479;
rho_z = 5.3435;
zeta_z = 3.1581;
k_z = 0.01;
epsilon_z = 1.4904;

%% Roll
lambda_phi = 0.8100;
rho_phi = 26.3578;
zeta_phi = 23.7246;
k_phi = 10.1140;
epsilon_phi = 1.5242;

%% Pitch
lambda_theta = 1.1739;
rho_theta = 2.3434;
zeta_theta = 22.2347;
k_theta = 13.7415;
epsilon_theta = 1.3286;

%% Yaw
lambda_psi = 10;
rho_psi = 5;
zeta_psi = 2;
k_psi = 0.01;
epsilon_psi = 0.5;

%%
% Vision Ts
VTs= 40*Ts;

% Register variables after the project is loaded and store the
   variables in
% initVars so they can be cleared later on the project shutdown.
endVars = who;
initVars = setdiff(endVars,initVars);
```

```
clear endVars;

% LocalWords:  myvariable
```

Listing D.1: Matlab-code for initializing variables/constants used by the Flight Control System.

D.3 Simulink Schemes

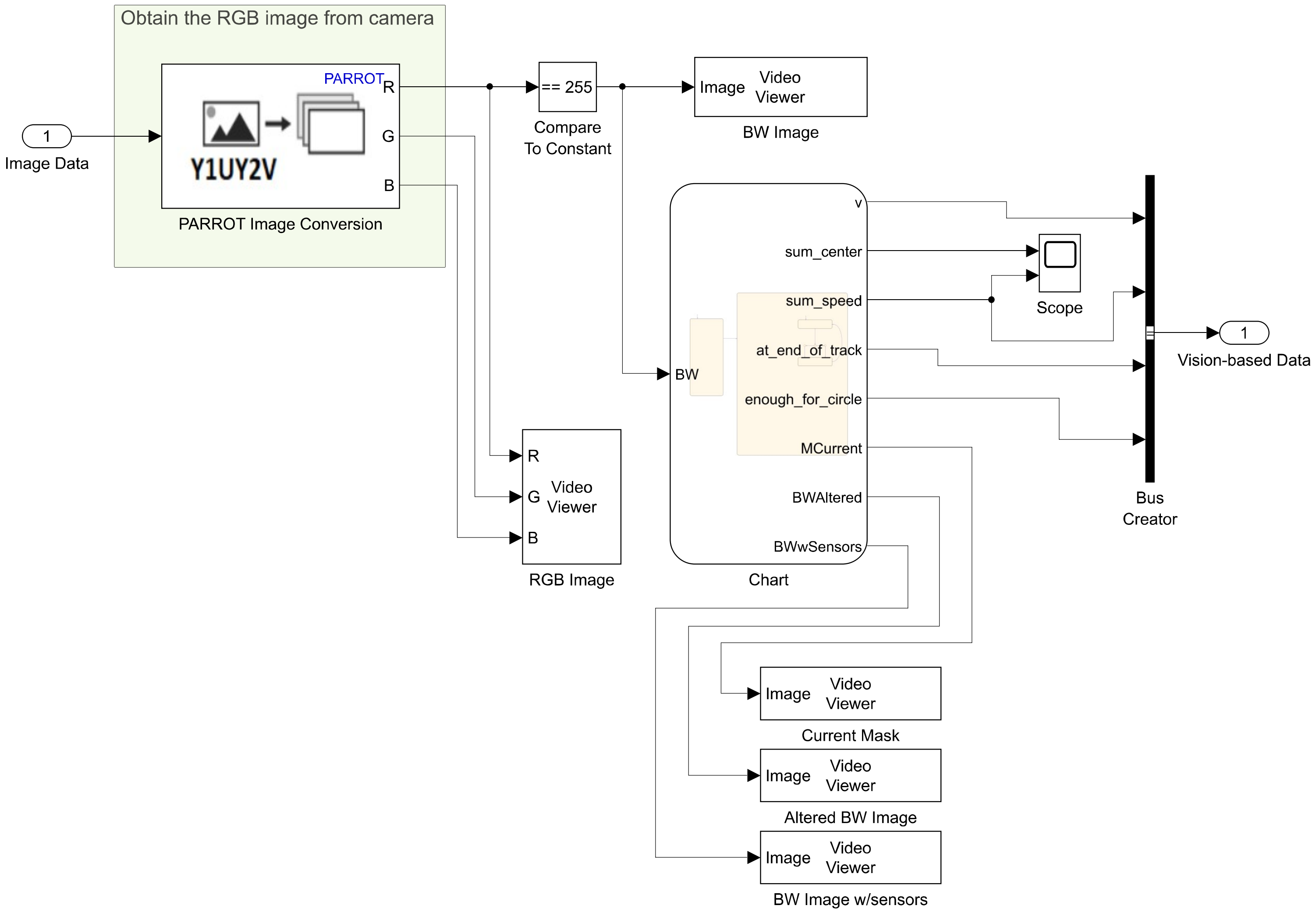
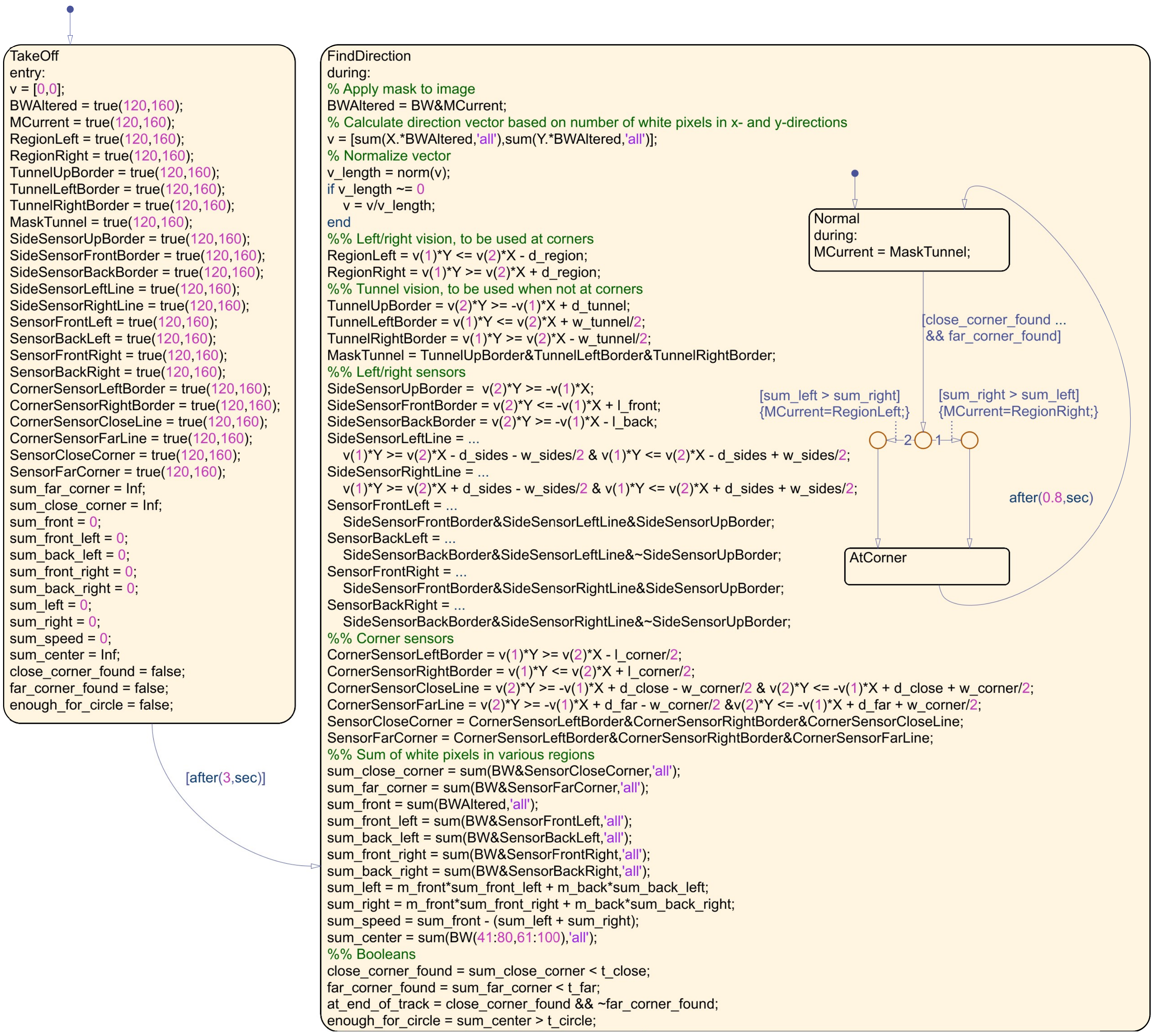


Figure D.91: Image Processing System.



```
TakeOff
entry:
v = [0,0];
BWAltered = true(120,160);
MCurrent = true(120,160);
RegionLeft = true(120,160);
RegionRight = true(120,160);
TunnelUpBorder = true(120,160);
TunnelLeftBorder = true(120,160);
TunnelRightBorder = true(120,160);
MaskTunnel = true(120,160);
SideSensorUpBorder = true(120,160);
SideSensorFrontBorder = true(120,160);
SideSensorBackBorder = true(120,160);
SideSensorLeftLine = true(120,160);
SideSensorRightLine = true(120,160);
SensorFrontLeft = true(120,160);
SensorBackLeft = true(120,160);
SensorFrontRight = true(120,160);
SensorBackRight = true(120,160);
CornerSensorLeftBorder = true(120,160);
CornerSensorRightBorder = true(120,160);
CornerSensorCloseLine = true(120,160);
CornerSensorFarLine = true(120,160);
SensorCloseCorner = true(120,160);
SensorFarCorner = true(120,160);
sum_far_corner = Inf;
sum_close_corner = Inf;
sum_front = 0;
sum_front_left = 0;
sum_back_left = 0;
sum_front_right = 0;
sum_back_right = 0;
sum_left = 0;
sum_right = 0;
sum_speed = 0;
sum_center = Inf;
close_corner_found = false;
far_corner_found = false;
enough_for_circle = false;
```

```
FindDirection
during:
% Apply mask to image
BWAltered = BW&MCurrent;
% Calculate direction vector based on number of white pixels in x- and y-directions
v = [sum(X.*BWAltered,'all'),sum(Y.*BWAltered,'all')];
% Normalize vector
v_length = norm(v);
if v_length ~= 0
   v = v/v_length;
end
%% Left/right vision, to be used at corners
RegionLeft = v(1)*Y <= v(2)*X - d_region;
RegionRight = v(1)*Y >= v(2)*X + d_region;
%% Tunnel vision, to be used when not at corners
TunnelUpBorder = v(2)*Y >= -v(1)*X + d_tunnel;
TunnelLeftBorder = v(1)*Y <= v(2)*X + w_tunnel/2;
TunnelRightBorder = v(1)*Y >= v(2)*X - w_tunnel/2;
MaskTunnel = TunnelUpBorder&TunnelLeftBorder&TunnelRightBorder;
%% Left/right sensors
SideSensorUpBorder =  v(2)*Y >= -v(1)*X;
SideSensorFrontBorder = v(2)*Y <= -v(1)*X + l_front;
SideSensorBackBorder = v(2)*Y >= -v(1)*X - l_back;
SideSensorLeftLine = ...
   v(1)*Y >= v(2)*X - d_sides - w_sides/2 & v(1)*Y <= v(2)*X - d_sides + w_sides/2;
SideSensorRightLine = ...
   v(1)*Y >= v(2)*X + d_sides - w_sides/2 & v(1)*Y <= v(2)*X + d_sides + w_sides/2;
SensorFrontLeft = ...
   SideSensorFrontBorder&SideSensorLeftLine&SideSensorUpBorder;
SensorBackLeft = ...
   SideSensorBackBorder&SideSensorLeftLine&~SideSensorUpBorder;
SensorFrontRight = ...
   SideSensorFrontBorder&SideSensorRightLine&SideSensorUpBorder;
SensorBackRight = ...
   SideSensorBackBorder&SideSensorRightLine&~SideSensorUpBorder;
%% Corner sensors
CornerSensorLeftBorder = v(1)*Y >= v(2)*X - l_corner/2;
CornerSensorRightBorder = v(1)*Y <= v(2)*X + l_corner/2;
CornerSensorCloseLine = v(2)*Y >= -v(1)*X + d_close - w_corner/2 & v(2)*Y <= -v(1)*X + d_close + w_corner/2;
CornerSensorFarLine = v(2)*Y >= -v(1)*X + d_far - w_corner/2 &v(2)*Y <= -v(1)*X + d_far + w_corner/2;
SensorCloseCorner = CornerSensorLeftBorder&CornerSensorRightBorder&CornerSensorCloseLine;
SensorFarCorner = CornerSensorLeftBorder&CornerSensorRightBorder&CornerSensorFarLine;
%% Sum of white pixels in various regions
sum_close_corner = sum(BW&SensorCloseCorner,'all');
sum_far_corner = sum(BW&SensorFarCorner,'all');
sum_front = sum(BWAltered,'all');
sum_front_left = sum(BW&SensorFrontLeft,'all');
sum_back_left = sum(BW&SensorBackLeft,'all');
sum_front_right = sum(BW&SensorFrontRight,'all');
sum_back_right = sum(BW&SensorBackRight,'all');
sum_left = m_front*sum_front_left + m_back*sum_back_left;
sum_right = m_front*sum_front_right + m_back*sum_back_right;
sum_speed = sum_front - (sum_left + sum_right);
sum_center = sum(BW(41:80,61:100),'all');
%% Booleans
close_corner_found = sum_close_corner < t_close;
far_corner_found = sum_far_corner < t_far;
at_end_of_track = close_corner_found && ~far_corner_found;
enough_for_circle = sum_center > t_circle;
```

```
Normal
during:
MCurrent = MaskTunnel;
```

```
AtCorner
```

Figure D.92: Image Processing System Stateflow Chart.

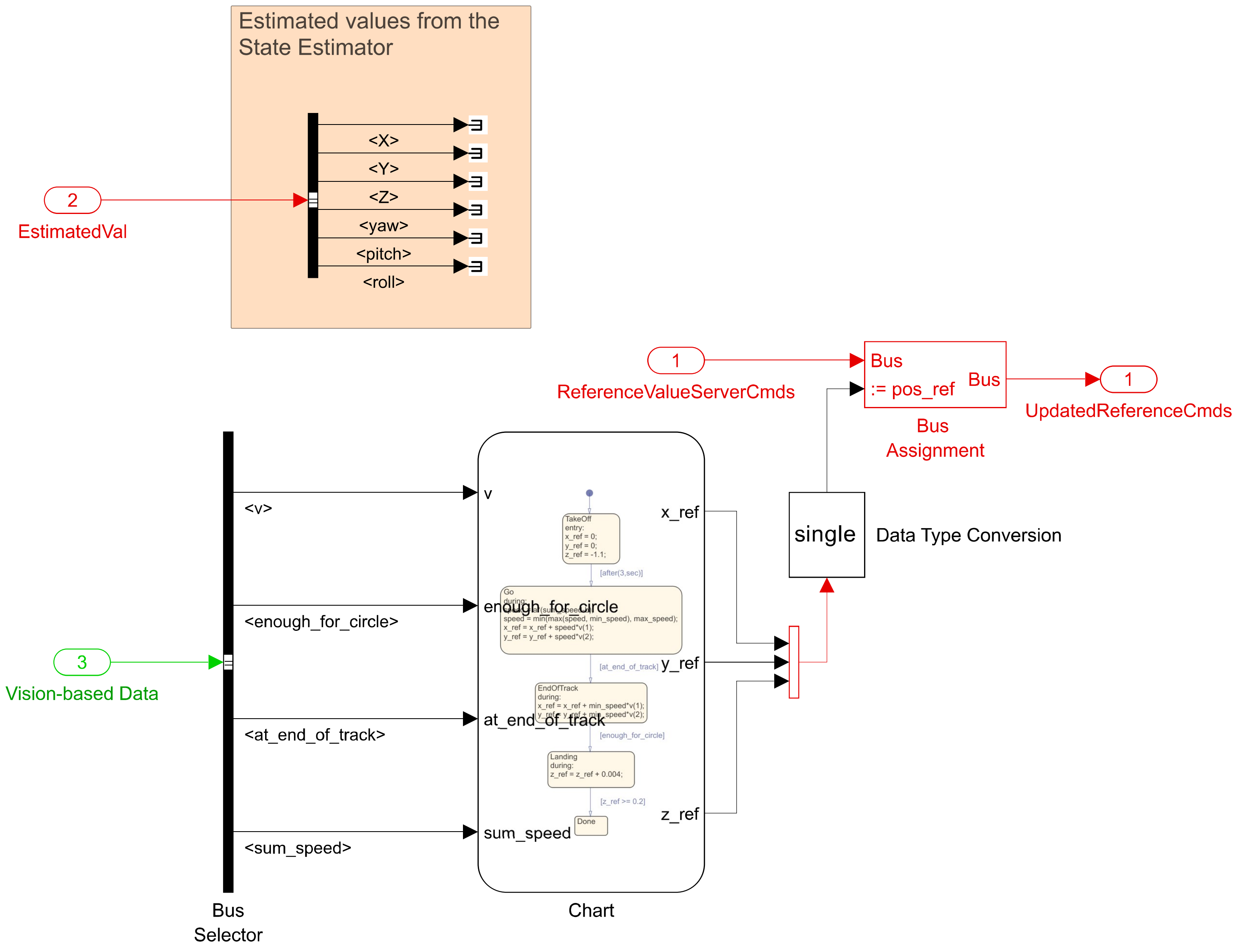


Figure D.93: Path Planning.

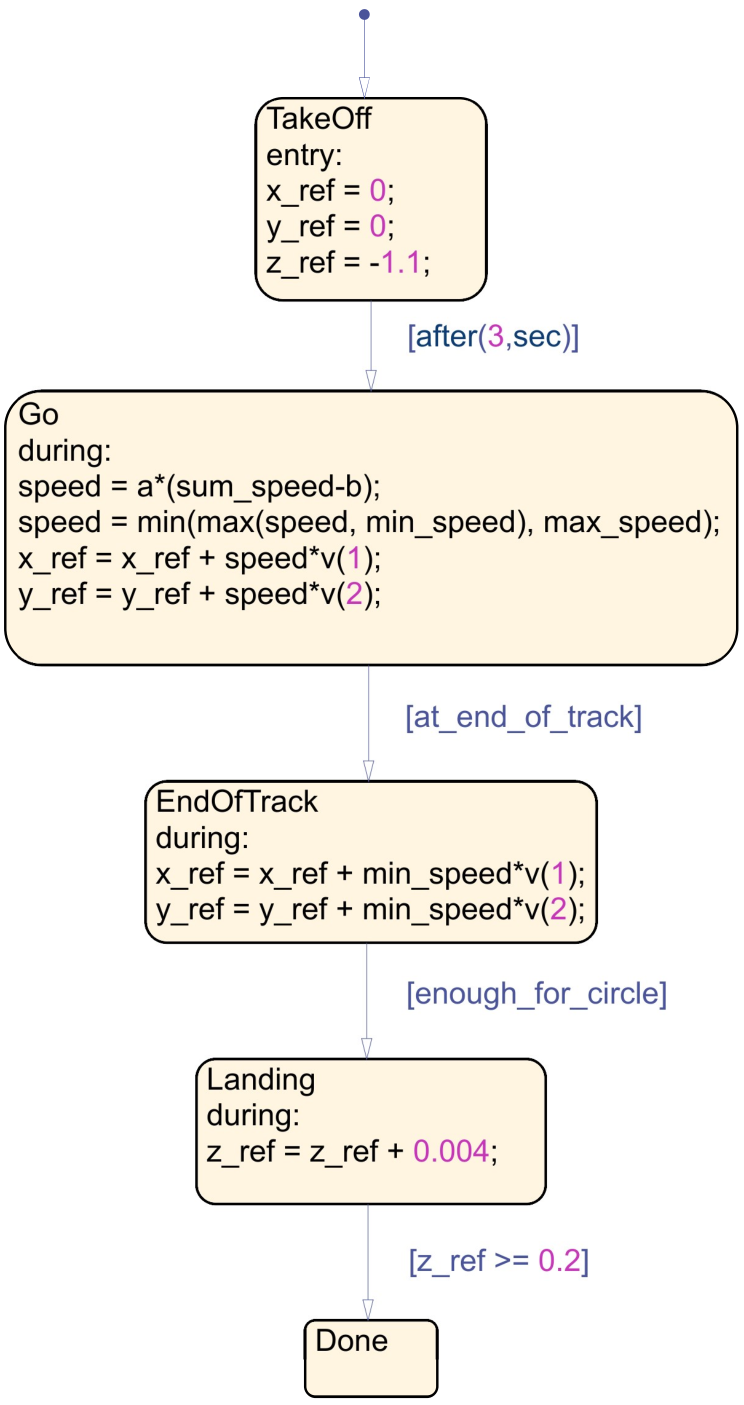


Figure D.94: Path Planning Stateflow Chart.