UAV Linear Model Predictive Control Using Computer Vision Algorithms

by

Christopher Conrad Surma

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering
University of Alberta

Abstract

This thesis aims to develop a motion control strategy for an Unmanned Aerial Vehicle (UAV) to execute a pursuit algorithm based on a vision based object detection algorithm. This enables a pursuer UAV to follow a target UAV based on images obtained from the onboard camera of the pursuer. The UAV pursuit algorithm is implemented onto a commercially available Parrot AR.Drone 2.0 quadcopter. Two motion control strategies considered for the UAV pursuit algorithm are a Proportional Integral Derivative (PID) Control and a Linear Model Predictive Control (LMPC). The performance of these two control strategies is evaluated based on their performance responding to a step input in each input channel, and tracking a figure-8 flight trajectory. A LMPC strategy was chosen to follow the target drone's trajectory estimated by a vision based object detection algorithm. In order to use a LMPC strategy, a linear state space model was obtained and identified for the Parrot AR.Drone 2.0. The vision based object detection algorithm used for our application is YOLO v2, a single-layer convolutional network which identifies the location of the target drone and returns a bounding box around it in the image frame. Experimental testing proves the proposed UAV pursuit algorithm achieves accurate detection of the target and successfully pursues it using the LMPC motion controller.

Preface

This thesis is an original work by Christopher Conrad Surma. No part of this thesis has been previously published.

Acknowledgements

The author of this thesis would like to thank Dr. Martin Barczyk for his support, guidance and supervision through the course of this thesis. I would like to thank Bingsheng Wei whose work into vision based object detection algorithms enabled this thesis to be produced; Vishek Prasad for his dedication and help in performing experiments, tuning controllers, and assisting me in wrapping up my thesis; Adam Casey for his help in giving me resources to learn C++ and checking my code. Finally, I would like to thank my wife, Ashleigh Surma, for her continued support and love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation of Research

The primary objective of this thesis is to develop a UAV pursuit algorithm which employs a closed-loop control strategy combined with a vision-based object detection algorithm. The resulting design is implemented onto the commercially available Parrot AR.Drone 2.0. The chosen vision based object detection algorithm, YOLO v2, employs a single-layer convolutional neural network (CNN) to detect and locate a target Parrot AR.Drone 2.0 in the image frame of the pursuer [2]. The performance of YOLO v2 is volatile, and requires a pure white background and an absence of any object besides the target drone to function optimally. The outputs of the vision algorithm are used to estimate the trajectory of the target UAV relative to the pursuer. This trajectory is tracked by the pursuer's motion control system, which needs to handle noise and outliers in the reference trajectory plus disturbances from the environment. A Proportional Derivative Integral (PID) controller and a Linear Model Predictive Controller (LMPC) are implemented and their performance is assessed in hardware experiments.

Previous work on this project was carried out in [3] and [4]. The work in [3] used a much simpler and less accurate vision-based algorithm, employed an overly simplistic calculation of the target position, and implemented the target tracking as an open-loop motion control which is unable to reject disturbances. The work in [4] implemented vision-based detection and position estimation of the target drone, but the resulting estimated trajectory was found to be insufficiently accurate based on hardware testing. A secondary objective of this thesis thus became to obtain an accurate estimation of the relative position of the target from the outputs of the YOLO v2 vision algorithm.

1.2 Literature Review

1.2.1 Vicon Motion Capture System Applications

Vicon motion capture systems have been used extensively in biomechanical and robotics applications. Examples of biomechanical applications that have used Vicon motion capture include the performance assessment of a wearable motion sensing suit [5], development of a methodology for tracking foot motions [6], and testing a machine learning algorithm designed for human emotion recognition [7]. In regards to robotic applications, the Vicon motion capture system is typically used as an external sensor in various vehicle control systems and algorithms. Vicon motion capture has been incorporated in quadcopter control used for radio frequency source localization and tracking [8], indoor UAV flight [9], and autonomous obstacle avoidance [10]. Substantial amounts of research have relied on the Vicon motion capture system to measure vehicle position and orientation [11]. The existence of substantial research relying on the Vicon motion capture system illustrates that it is reliable and can provide an accurate measurement of pose of a vehicle.

In addition, the Vicon motion capture system has been used as a ground truth for assessing computer vision algorithms and design of UAV systems. The paper by [12] describes the design of a quadrotor Micro Air Vehicle (MAV) which used computer vision algorithms to operate without the use of an external reference. The computer vision algorithms estimated the position and orientation of the UAV. The accuracy of the resulting poses was compared against the ground truth data obtained from the Vicon motion capture system. A paper by [11] describes a vision-based quadrotor MAV system capable of autonomous mapping and exploring of unfamiliar environments using an on-board camera. The quantitative assessment of the performance of these algorithms was performed by comparing the pose data estimated by these algorithms against data obtained from the Vicon motion capture system.

1.2.2 Control Systems Utilized in Flight Applications

Research into developing dynamics models and motion controllers for flight applications such as quadcopters and helicopters began to flourish during the mid 2000's. Quadcopter applications at this time consisted of using either a PID or LQR motion controller and an Inertial Measurement Unit (IMU) to control the position and orientation of a quadcopter [13, 14, 15, 16, 17, 18]. A forerunner in this research was a group from Stanford that developed STARMAC (Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control) [19, 20]. STARMAC consisted of a set of autonomous quadrotor drones that followed prescribed trajectories using GPS and IMU information and a LQR motion controller [19]. Recent research has involved quadcopters performing formation flight, aggressive acrobatics, autonomous surveillance and path planning [21, 22, 23, 24, 25, 26, 27].

Model Predictive Control (MPC) algorithms began to be developed, simulated, and implemented for helicopter applications during the mid 2000's [28, 29, 30, 31]. Some notable researchers at the time were Dr. Hyounjin Kim and Dr. David Shim from the University of California Berkeley that developed and implemented nonlinear model predictive control algorithms into helicopter applications [30, 32]. In [32], they used a nonlinear model predictive controller to perform vision-based target tracking and stabilization of a helicopter. Incorporating model predictive controllers into quadcopter applications would begin during the early 2010's.

Current research into quadcopter motion control consists of implementing model predictive control algorithms for a variety of purposes. Examples of quadcopter applications that have implemented model predictive control algorithms include trajectory tracking and object avoidance [33], pursuit of a ground vehicle [34], trajectory tracking subject to aerodynamic disturbances [35], and picking up an object and placing it at a set location [36]. Model predictive motion controllers have been successfully implemented into the Parrot AR.Drone 2.0 to perform collision avoidance and trajectory tracking [33, 37, 38]. Model predictive controllers are becoming more prevalent in quadcopter applications due to their ability to deal with the nonlinear and multi-input multi-output (MIMO) dynamics of a quadcopter, take into account the input constraints to avoid saturation of the actuators, plan an optimal path based on future reference information, and be applied to systems that require fast sampling rates [39, 40, 41]. This thesis will attempt to implement PID and LMPC motion control using a commercial Parrot AR.Drone 2.0.

1.2.3 Object Detection and Active-Tracking Algorithms

During the mid 1990's to late 2000's, research into vision-based control of UAV applications began to be investigated. The vision based control of UAV applications consisted of vision-based landing and pursuit. Examples of research conducted during this time consisted of autonomous landing of a UAV [42], landing an UAV with a vision based motion estimation method [43], and vision-based navigation of a UAV in a GPS deprived environment [44]. Notably in [45], a vision-based landing and mapping algorithm was successfully merged with a MPC-based flight controller for a custom built UAV.

Research into vision based control of UAV applications continued into the 2010's. Vision-based object detection algorithms have used computer vision AR tags alongside UAV motion controllers [46, 47, 48, 49, 50, 51]. Many of these applications involve landing a target drone on a AR tag rather than using tags to perform target tracking [47, 48, 50]. An example of research that performed target tracking of a computer vision tag was shown in [52], where a Parrot AR.Drone 2.0 performed target tracking of an AR tag and a SLAM system was implemented to follow a line on the ground using a PID motion controller.

Besides tracking computer vision tags, vision based detection algorithms have been used to perform target tracking of objects such as drones. In [53], a vision based tracking system was used with a UAV to track and hover above an object. A vision based detection algorithm was used to avoid and track aerial vehicles in a dynamic environment using a PD motion controller [54]. A target following algorithm was developed for the Parrot Ar.Drone 1.0 in [55]. The target following algorithm used the color and image moment of the given target to determine its location [55].

The most recent generation of vision based object detection and tracking algorithms employ machine learning algorithms. The DJI Phantom 4 Pro UAV contains a Follow-Me mode that allows the UAV to follow the operator [56]. The Follow-Me mode uses machine learning to identify an object using patterns, colors, position, and scale of a specific object and stores temporary library of images of the object as reference [57]. Research was conducted in UAV pursuit of a target drone in [58]. A pursuer UAV was capable of following a target UAV using a deep convolutional neural network called VGG-M.

1.3 Outline of Thesis

This chapter provides the motivations for undertaking this research, a survey of related literature, and an itemized statement of contributions.

Chapter 2 is a summary of the hardware and software used for this research. The features of the Parrot AR.Drone 2.0 quadcopter used for our experiments are described. The details of the Vicon motion capture system used to provide state information for the control system and a ground truth for verifying the performance of the control and computer vision algorithms are given. A brief description of the WiFi communication channel is provided.

The main piece of software described is the Robot Operating System (ROS), an open-source environment providing communication and interfacing between the UAVs, ground computer and the Vicon motion capture system. ROS provides access to various libraries and packages used to develop and implement our control and vision-based pursuit algorithms, including `ardrone_autonomy`, `tum_simulator`, and `vicon_bridge` which are described in detail.

Chapter 3 describes the principles of a Proportional Integral Derivative (PID) control and a Linear Model Predictive Control (LMPC) and designs them for our system. The mathematical tools used to describe the pose of an object in space and its tracking error are discussed. A linear state space model of the Parrot AR.Drone 2.0 is derived and its parameters are identified.

Chapter 4 describes the vision-based object detection algorithm that was used in this thesis. A procedure to estimate relative target position from monocular video is discussed and implemented. The performance of the algorithm is extensively tested and quantified in a series of experiments.

Chapter 5 describes the implementation of the motion control designs derived in Chapter 3 on the Parrot AR.Drone 2.0. The experimental procedure to evaluate the performance of each control system is given. The experimental results are analyzed and the performance of each control system is assessed.

Chapter 6 covers the integration of the vision-based object detection algorithm with the LMPC motion controller to create a vision-based pursuit algorithm. The experimental procedure to evaluate the performance of the resulting system is discussed. The experimental results are analyzed and the performance of the system is assessed.

Chapter 7 summarizes the results and findings of this thesis. The limitations of the work performed are discussed and future research directions are suggested.

1.3.1 Statement of Contributions

The following are claimed as research contributions of this thesis:

- Two types of closed-loop motion control strategies, PID and LMPC, were written in C++ and implemented under Robot Operating System (ROS). State information is obtained from a Vicon motion capture system. The control code is modularized such that the only information it requires is the reference (desired) world position and yaw angle of the target Parrot AR.Drone 2.0 UAV.

- The dynamics model of the Parrot AR.Drone 2.0 used by the LMPC design, originally proposed in [33], found to be inaccurate in experimental testing, was corrected and re-identified

- The 3D relative position estimation from 2D bounding boxes proposed in[4]was updated, its parameters were identified, and the resulting performance was assessed. The new algorithm was implemented in ROS and written in C++ code. The depth estimation method now uses a width-based calculation, which is shown to outperform the previous area-based calculation.

- An extensive set of experimental hardware testing was performed to qualify and assess the performance of the detection and tracking control system designs.

Chapter 2

Hardware and Software

2.1 Hardware Background

2.1.1 Parrot AR.Drone 2.0

The Parrot AR.Drone 2.0 is a small and lightweight quadrotor drone (see Figure 2.1) designed for both indoor and outdoor flights. The design of the drone consists of a propeller assembly, central unit and Styrofoam hull. The propeller assembly is made up of carbon-fibre rods that support the motors and are connected to a plastic fiber-reinforced central cross member [3]. The central unit contains the Lithium-Polymer battery, onboard camera, inertial sensors, and processors. The Styrofoam hull protects the body and shields the propellers against contact with obstacles [3].

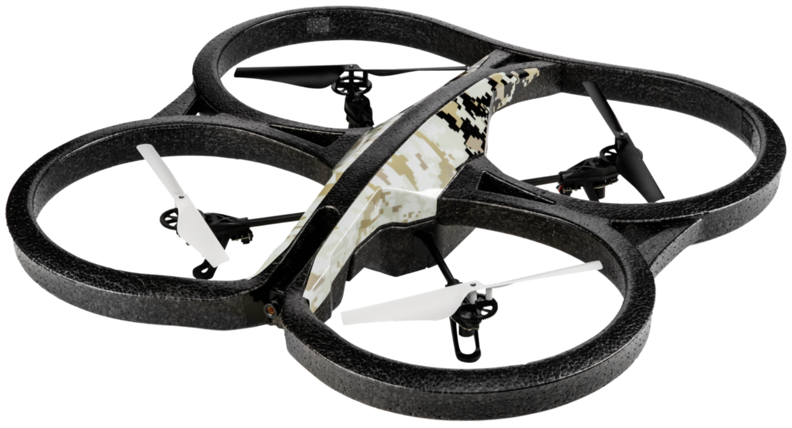


Figure 2.1: Parrot AR.Drone 2.0

The Parrot AR.Drone 2.0 has a battery life of up to 18 minutes depending on how aggressively the UAV is flown, a maximum range of 50 meters and has a 30 fps HD camera that streams at 720p with a 92° diagonal wide-angle lens [59]. The drone is equipped with 4 inrunner brushless DC motors running at 28 500 rpm [59]. The Parrot AR.Drone 2.0 is a second generation drone that succeeds the original Parrot AR.Drone. The camera quality of the Parrot AR.Drone 2.0 was improved from 480p to 720p [60, 59]. The onboard sensors which include a gyroscope and accelerometer were significantly improved. The Parrot AR.Drone 2.0 includes a magnetometer for estimating yaw which is not included in the earlier generation.

The rate gyro outputs raw signals consisting of the angular rates of the UAV with respect to its frame in degrees/s, the acceleration of the IMU is measured in g's, and the magnetometer measures the magnetic field in the body-fixed frame. Using the data from each of these sensors and a sensor fusion algorithm such as an Extended Kalman Filter, the attitude and position of the Parrot AR.Drone 2.0 can be estimated.

The onboard measurements are not used in this thesis due to the extensive work required to fuse information from the onboard sensors to determine the position and attitude of the UAV. In this case, we would need to implement the sensor fusion and control code directly onboard of the drone which requires rewriting the firmware and coding on a low-power processing platform. An alternative approach was used to determine the position and attitude of the UAV and avoid tinkering with the firmware. A Vicon motion capture system was used to determine the attitude and position of the drone rather than the onboard sensors. The motion capture system is described in Section 2.1.2. The Vicon motion capture system was chosen due to being successfully applied in other research groups such as Dr. Raff D'Andrea at ETHZ [25], Dr. Angela Schoellig's Dynamic Systems Lab at the University of Toronto, and Dr. Vijay Kumar's group from the University of Pennsylvania. Future work will involve developing an autonomous UAV which requires implementing a complete sensor fusion, a simultaneous localization and mapping (SLAM) system, and control architecture onboard of the UAV.

2.1.2 Vicon Motion Capture System

This section focuses on how a Vicon motion capture system can be used as a tool to dynamically estimate the position and orientation of a vehicle or object and as a verification tool to assess the performance of a motion controller, vision based object detection algorithm, and a vision based pursuit algorithm. An investigation of the Vicon motion capture system is necessary to determine whether the system can act as a ground truth for the motion controller and verification tool.

2.1.2.1 Motion Capture System Description

Motion capture systems within our application are used to estimate the rigid body position and orientation of objects, specifically UAVs. These systems are useful for providing state information about the Parrot AR.Drone 2.0 which are used to determine the state space model of the UAV and used in the controllers described in Chapter 3 [61].

The Vicon motion capture system uses infrared cameras that strobe infrared light which illuminates passive optical markers coated with a retroflective surface [61]. The Vicon motion capture system consists of 10 cameras that each record the markers. These cameras are used to triangulate the position of the markers in 3D space. A marker that is seen by a camera appears as a cluster of illuminated pixels with a black background. In order to triangulate the position of a marker, at least three cameras must be capable of viewing the marker. The marker position information is passed on to a central computer that runs a software called Tracker 3.

The Vicon motion capture system is capable of estimating the positions and attitudes of rigid bodies, defined by their attached markers, in near real-time. The linear and angular velocity of the rigid body was obtained by performing numerical differentiation on the rigid body estimated pose data combined with a simple low-pass filter described in Section 3.2.3.

2.1.2.2 Vicon Motion Capture System Component Description

The main components of the Vicon motion capture system consist of Vero v2.2 cameras, PoE switch and a computer that runs the Tracker 3 software. Minor components of the motion capture system consist of camera mounts, Ethernet cords, and passive reflective markers. The primary sensing component of the motion capture system are the Vero v2.2 cameras as shown in Figure 2.2.

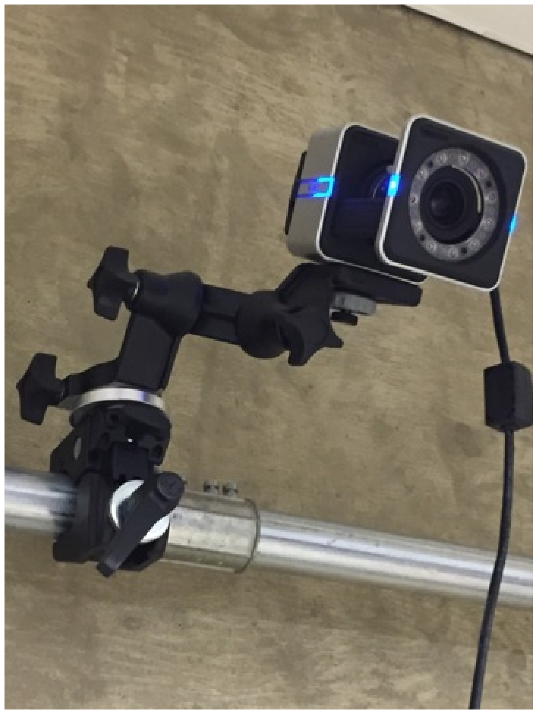


Figure 2.2: Image of the Vero v2.2 camera

The Vero v2.2 infrared camera detects reflective markers and has onboard sensors that detect the camera temperature and detect whether it has been shaken or disturbed. The specifications of the Vero v2.2 cameras are outlined in Table 2.1 .

Table 2.1: Vicon Vero v2.2 Specifications [1]

Camera Parameter	**Value**
Resolution (MP)	2.2 (2048x1088)
Frame Rate FPS (Hz)	330
Lens	6-12 mm Varifocal
Minimum Wide FOV (HxV)	98.1 x 50.1
Camera Latency (ms)	3.6
Strobe	Infrared
Shutter Type	Global
Dimensions (HxWxD) (mm)	83 x 80 x 135
Weight (kg)	0.57

The software that processes the feed from the Vero v2.2 infrared cameras is the Tracker 3 software. The Tracker 3 software is capable of processing data with a latency of 1.5 ms and at more than 500 frames per second [62]. The software allows the user to perform camera adjustments, system calibration, marker tracking, and data export. In addition, the Tracker 3 software performs complex calculations of determining the rigid body poses of tracked objects from 3D marker positions.The rigid body pose data can be exported as .c3d files, .avi files, or .csv file [63]. In addition, the data can be streamed to another computer via a ROS package called `ardrone_autonomy` described in Section 2.2.

2.1.2.3 Vicon Motion Capture System Calibration Process

The camera calibration process involves camera preparation, camera calibration, and setting the volume origin which are detailed below.

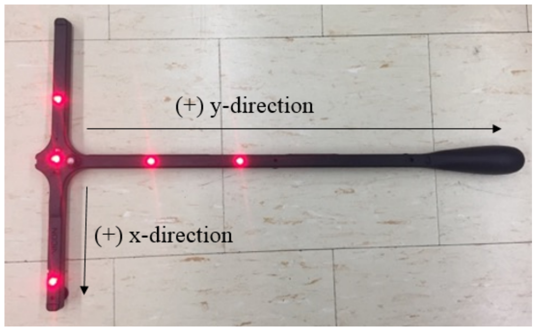


Figure 2.3: Image of the Calibration Wand

The camera preparation involves focusing each camera to identify reflective markers within the capture volume. Each camera's video feed can be monitored through the Tracker 3 software. For optimal performance, the reflective markers need to be clearly visible to each camera. To achieve this, the focus, aperture and focal length rings on each camera are manually adjusted.

Once the camera lenses are adjusted, further optimization of the camera settings can be performed within the Tracker 3 software to improve marker recognition. The gain, sensor threshold, and strobe intensity can be adjusted in the software to improve marker image brightness, determine minimum intensity thresholds, and increase brightness of the strobe, respectively [63]. Once the cameras are focused, the cameras are masked using the Tracker 3 software. When a camera is masked, the Tracker 3 software removes stray reflections (such as reflections from the floor, or other camera strobe lights) and improves the robustness of the camera calibration [64]. Each camera can be masked automatically or manually depending on user preference.

The system calibration is performed with the Tracker 3 software. The camera is calibrated by swinging the calibration wand through the capture volume. The camera captures a user-defined number of frames, typically fifteen hundred. In order to be considered a frame, the camera must capture all five markers of the calibration wand. Once each camera has captured the specified number of frames, the software performs calculations of the positions of each camera relative to each other and determines the image and world error. The image error is root mean square error in pixels which represents the difference between the 2D image of each marker on the camera sensor and the 3D marker location back projected onto the camera sensor while the world error is determined from the image error and the distance of the camera to the center of the control volume in millimeters [mm] [63]. The Tracker 3 software deems a calibration as good if the world error for each camera is approximately 0.5 mm. If any of the cameras report a larger world error, the calibration is considered poor and needs to be redone. Since the typical world error reported by the Vicon motion capture system is approximately 0.5 mm, the error from the Vicon motion capture system is considered to be negligible. Due to the accuracy achieved by this system, the Vicon motion capture system is used as a source of ground truth information.

The camera calibration calculates the relative poses of the cameras with respect to each other. The calibration wand is then used to define the world frame of the capture volume. The calibration wand contains five active LED markers that are picked up by the cameras. The calibration wand is placed flat on the floor in the desired position and orientation of the capture volume world frame [63]. After the origin is set, the Vicon motion capture system is fully calibrated. Any motions by the individual cameras, for instance due to vibrations, will require redoing the entire calibration process.

2.1.3 Wi-Fi

The Parrot AR.Drone 2.0 employs Wi-Fi to communicate to a control device such as a tablet computer or desktop computer. Since the drone is used in an indoor setting, signal interference is low since the distance between the Wi-Fi transmitter and receiver is small and unobstructed. When the drone is used in an outdoor setting, the drone can only operate within a maximum range of 50 m [59] and anything past that range will result in signal loss and/or packet loss due to the AR.Drone 2.0 having a low powered antenna. Signal transmission can be interfered in indoor and outdoor settings if large obstructions are present between the Wi-Fi transmitter and receiver or if too many devices are sharing the Wi-Fi band.

2.2 Software Background

2.2.1 OpenCV

OpenCV (Open source Computer Vision) is a cross-platform library of functions that are used for real-time computer vision applications, written in C/C++ and originally developed by Intel. This library was used to rectify the video feed from the onboard camera of the Parrot AR.Drone 2.0.

2.2.2 CUDA

CUDA is a parallel computing platform developed by NVIDIA that allows running computing applications on graphical processing units (GPUS) [65]. This computing platform was used to run the machine learning framework described in Section 2.2.3 and the object detection algorithm described in Section 2.2.4 in parallel with the motion controllers described in Chapter 3.

2.2.3 Darknet

Darknet is an open source machine learning framework that was written in C and CUDA [66]. This machine learning framework was used to test and train an object detection model described in Section 2.2.4. Additional details about Darknet can be found in [66, 4].

2.2.4 YOLO v2

YOLO (You Only Look Once) v2 is an open source real time object detection system that uses a convolutional neural network (CNN) [2]. YOLO v2 predicts the location of an object in an image and produces a bounding box around that object within the image frame. YOLO v2 considers object detection as a single regression problem and determines the bounding box coordinates and class probabilities based off only one pass through the image [2]. This vision based object detection algorithm is used with conjunction with the motion controllers in Chapter 3 to pursue an object. Additional details about YOLO v2 can be found in [2, 4].

2.2.5 Robot Operating System (ROS)

2.2.5.1 Description

The Robot Operating System (ROS) is an open-source environment running within Linux designed for robotic applications [67]. It contains tools, libraries and services that simplify developing complex robotic systems and communications between multiple processes. While most code for ROS is primarily written in C++ and Python, Java, Lisp, and several other programming languages are also supported [68]. The control algorithms described in this thesis were written in C++. Each new version of ROS is compatible with specific versions of Linux. The version used in this research was the ROS Kinetic Kame distribution that runs in Ubuntu 16.04 (Xenial Xerus). ROS also supports a number of client libraries for C++ and Python. Client libraries that were used include the `tf` library and the `Eigen` library. The `tf` library is used to keep track of coordinate frames of robots while the Eigen library is a C++ template library used for linear algebra.

ROS provides a software infrastructure that allows various modules to work with each other and simplifies communication between them. A robotic system consists of a web of programs called nodes sending messages via transmission lines called topics to one another. Nodes are executable programs that receive and/or send transmissions and perform some type of function. ROS topics transmit information as messages which are classes of data such as IMU readings, 3D point clouds, or velocity commands. These messages are organized and sent from a node via a ROS topic. Topics that are sent from a node are published while topics that are received by a node are subscribed. A node can publish any number of topics and can subscribe any number of topics. A group of nodes can be spanned using a launch file that specifies and configures a cluster of nodes and executes them. All ROS programs are organized into packages which are coherent collection of files, that include both executable and supporting files, which serve a singular purpose [68]. Several ROS packages, client libraries, and software were used throughout the course of this research and are outlined further below. Additional information and details on ROS can be found in [68, 67].

2.2.5.2 `ardrone_autonomy`

`ardrone_autonomy` is a ROS driver for the Parrot AR.Drone 1.0 and 2.0 quadrocopter. This driver is based on the official AR-Drone SDK published by Parrot. This ROS package was initially developed by Mani Monajjemi of Simon Fraser University. This driver allows the user to send commands to the drone, set configuration parameters, and read information from the onboard sensors including the video cameras. Two main functions of this driver that were essential in this research was the capability to fly the drone through ROS and read the camera feed from the onboard cameras. Commands can be sent to the drone that consist of landing, taking off, and flying in a specific direction. Further details about the commands are given in Chapter 3.

The Parrot AR.Drone 2.0 is equipped with two cameras: one camera pointing forwards and one camera pointing downwards. This ROS driver will create and publish two topics that will send camera feed information from both the front and bottom cameras. The video feed from the onboard camera is rectified through a ROS camera calibration module using OpenCV. Details on how to calibrate the cameras can be found in Chapter 4. This rectified camera feed is essential to perform UAV pursuit control which is further explored in Chapter 6.

2.2.5.3 `tum_simulator`

This ROS package contains programs designed to simulate a Parrot AR.Drone 2.0 through a package called Gazebo. Gazebo is a robot physics simulator that is used to quickly test algorithms, design robots, and train AI systems. Gazebo has been previously used to test algorithms implemented on a Parrot AR.Drone 2.0 [69]. The `tum_simulator` package can simulate both the Parrot AR.Drone 2.0 and 1.0 but has been optimized for the Parrot AR.Drone 2.0. It was used to quickly test and design control algorithms, and tune the gains of the motion controllers. The gains of the controllers that were tuned in simulation were further adjusted when flying the hardware Parrot AR.Drone 2.0.

2.2.5.4 `vicon_bridge`

The `vicon_bridge` package is a ROS module that streams data from a Vicon motion capture system. It supports multiple tracked rigid bodies. The `vicon_bridge` package initiates a connection with a host computer running Vicon Tracker. Vicon data is transmitted to another computer as long as the other computer is on the same subnet and the IP address or name of the host computer is known.

The `vicon_bridge` package can be used to shift the frame pose of a rigid body defined in Vicon Tracker. Typically, the tracker software will place the origin at the geometric centre of the markers defining the body. Moving this origin in the tracker software is a tedious procedure. By using the `vicon_bridge` package, the tracked body can be set on top of the world frame of the Vicon motion capture system, and the body-fixed frame can be dynamically set to be aligned with this frame at a specified offset above the floor.

`vicon_bridge` is used to provide state information about the Parrot AR.Drone 2.0 used in motion control algorithms. In addition, it is used as a source of ground truth data for validation experiments.

2.2.6 MATLAB

MATLAB is a numerical computing environment and programming language developed by MathWorks that is written in C, C++, and Java. It provides a large variety of data visualization tools and directly handles matrix and vector mathematical operations. MATLAB contains Simulink which is a graphical programming environment used to model, simulate and analyze dynamical systems and develop control systems for robotic applications [70]. MATLAB has tools for designing model predictive controllers [71] and PID controllers [72]. In addition, MATLAB has tools for running Simulink models and controllers on a Parrot AR.Drone 2.0 [73].

Despite MATLAB having convenient tools for developing control systems for a Parrot AR.Drone 2.0, the PID and LMPC motion controller was developed directly in C++ and implemented in ROS. The disadvantage of this approach was the large time investment spent in learning C++ and Python and implementing PID and LMPC in ROS, and being unable to utilize convenient mathematical tools provided by MATLAB. Conversely the advantage was developing literacy in C++ and Python, in-depth knowledge of how the algorithms work, lower computation time and resource use, and the ability to integrate other ROS packages, libraries, and software such as Darknet which may not be available with MATLAB. MATLAB was thus mainly used to analyze results from experiments and create graphs and schematics in this thesis.

Chapter 3

UAV Control System

This chapter will discuss the design and mathematical principles behind two types of control systems: Proportional Integral Derivative (PID) control and Linear Model Predictive control (LMPC). Section 3.1 will discuss the mathematical tools needed to describe the position and orientation of a rigid body in space. Section 3.2 will describe the plant model of the Parrot AR.Drone 2.0 and how it was obtained. Section 3.3 will describe an alternative way to define yaw which slightly alters the plant model in Section 3.2. Section 3.4 will outline the mathematics and concepts behind PID control while Section 3.5 will outline the mathematics and concepts behind LMPC.

3.1 Mathematical Preliminaries

This section outlines the mathematical tools that were used to build a PID controller described in Section 3.4 and a LMPC system described in Section 3.5. A majority of this section is based off of work done in the Ph.D. thesis [74] and Mec E 651 lectures.

3.1.1 Rotation Matrix

The model of a rigid body in 3D space consists of making use of two coordinate frames: a ground-fixed inertial navigation frame with basis vectors $\{n_1, n_2, n_3\}$ and a body-fixed non-inertial body frame using basis vectors $\{b_1, b_2, b_3\}$. These two coordinate frames are orthonormal and follow the right-hand convention, i.e. $b_1 \times b_2 = b_3$ and $n_1 \times n_2 = n_3$, where $\times$ denotes the $\mathbb{R}^3$ cross-product [74]. The navigation frame is a stationary, inertial frame that serves as the datum that a rigid body will orientate itself with respect to. The body frame is a moving, non-inertial frame situated at the center of mass or geometric center of a rigid body. In order to describe the orientation of a rigid body with respect to the navigation frame, the body frame basis vectors can be expressed in terms of the navigation frame basis vectors as

$$b_1 = (b_1 \cdot n_1)n_1 + (b_1 \cdot n_2)n_2 + (b_1 \cdot n_3)n_3$$

$$b_2 = (b_2 \cdot n_1)n_1 + (b_2 \cdot n_2)n_2 + (b_2 \cdot n_3)n_3$$

$$b_3 = (b_3 \cdot n_1)n_1 + (b_3 \cdot n_2)n_2 + (b_3 \cdot n_3)n_3$$

Concatenating b_1, b_2, and b_3 into a matrix yields the following result.

$$R = \begin{bmatrix} b_1 \cdot n_1 & b_2 \cdot n_1 & b_3 \cdot n_1 \\ b_1 \cdot n_2 & b_2 \cdot n_2 & b_3 \cdot n_2 \\ b_1 \cdot n_3 & b_2 \cdot n_3 & b_3 \cdot n_3 \end{bmatrix}$$

R is known as the rotation matrix. The rotation matrix measures the orientation of the rigid body relative to the navigation frame. As a body rotates, the entries of R change with time which can be denoted as $R = R(t)$.

The rotation matrix is an orthogonal matrix since the columns of R represent the coordinates of b_i in the navigation frame, and $\{b_1, b_2, b_3\}$, are orthonormal. Since R is an orthogonal matrix and $\{b_1, b_2, b_3\}$ obeys the right-handed convention, R has the following properties: $R^{-1} = R^T$ and $\det(R) = +1$. All rotation matrices are a part of a subset of orthogonal matrices known as the *special orthogonal group* $SO(3)$. $SO(3)$ is called the rotation group of $\mathbb{R}^3$ with $R \in SO(3)$ encoding a rotation between two orthonormal frames.

A rotation matrix can be used to determine coordinates of an object in different frames. For a given point p in 3D space, we can describe that point relative to either the navigation frame, $p_N = \begin{bmatrix} p_{N,1} & p_{N,2} & p_{N,3} \end{bmatrix}^T \in \mathbb{R}^3$, or the body frame, $p_B = \begin{bmatrix} p_{B,1} & p_{B,2} & p_{B,3} \end{bmatrix}^T \in \mathbb{R}^3$. The relationship between the two coordinates can be written as follows:

$$\underbrace{\begin{bmatrix} p_{N,1} \\ p_{N,2} \\ p_{N,3} \end{bmatrix}}_{p_N} = \underbrace{\begin{bmatrix} b_1 \cdot n_1 & b_2 \cdot n_1 & b_3 \cdot n_1 \\ b_1 \cdot n_2 & b_2 \cdot n_2 & b_3 \cdot n_2 \\ b_1 \cdot n_3 & b_2 \cdot n_3 & b_3 \cdot n_3 \end{bmatrix}}_{R} \underbrace{\begin{bmatrix} p_{B,1} \\ p_{B,2} \\ p_{B,3} \end{bmatrix}}_{p_B}$$

Since a coordinate can be transformed between any two frames, this can be extended to the case where there are multiple frames which leads to a composition of rotation matrices. Consider a given point p and the frames N, A, and B as shown in Figure 3.1.

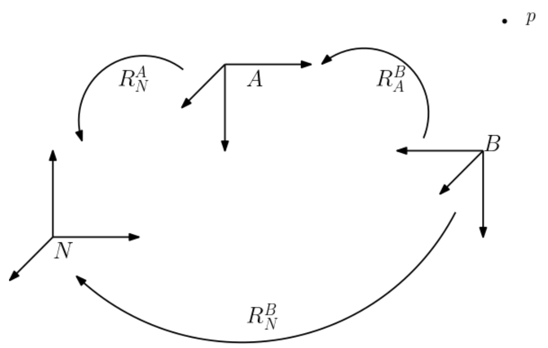


Figure 3.1: Three Frame Diagram

Consider these three changes in coordinates between frames:

$$p_N = R_N^A p_A \tag{3.1a}$$
$$p_A = R_A^B p_B \tag{3.1b}$$
$$p_N = R_N^B p_B \tag{3.1c}$$

By substituting (3.1b) into (3.1a), and comparing it to (3.1c), we find

$$p_N = R_N^A R_A^B p_B = R_N^B p_B$$

Therefore, the $B \to N$ transformation is equivalent to the transformation $B \to A$ followed by the transformation $A \to N$. In addition, the inverse transformation exists such that applying the inverse operation to $B \to A$ will yield $A \to B$ as shown below:

$$p_A = R_A^B p_B \to p_B = \left(R_A^B\right)^T p_A = R_B^A p_A$$

3.1.2 Euler Angles

As shown in [74], the rotation matrix $R \in SO(3)$ can be surjectively parameterized using only three numbers, which we choose as Euler angles. Euler angles are used in a composition of rotations that express $R \in SO(3)$ as a product of three elementary rotations. Composition of rotations was discussed in Section 3.1.1. Consider these three elementary rotations as described in Figure 3.2 each made with respect to frame N.

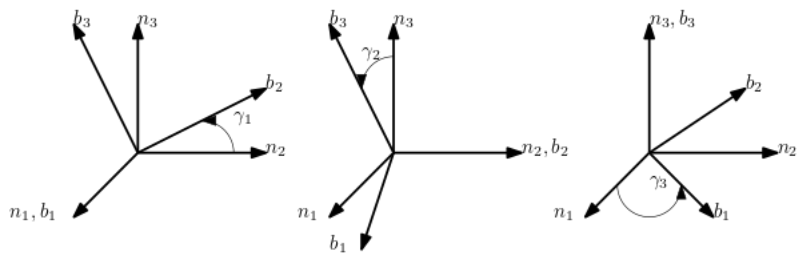


Figure 3.2: Three Elementary Rotations about n_1, n_2, and n_3

Using the definition of R from Section 3.1.1, each elementary rotation can be described as :

$$R_1(\gamma_1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma_1) & -\sin(\gamma_1) \\ 0 & \sin(\gamma_1) & \cos(\gamma_1) \end{bmatrix} \tag{3.2a}$$

$$R_2(\gamma_2) = \begin{bmatrix} \cos(\gamma_2) & 0 & \sin(\gamma_2) \\ 0 & 1 & 0 \\ -\sin(\gamma_2) & 0 & \cos(\gamma_2) \end{bmatrix} \tag{3.2b}$$

$$R_3(\gamma_3) = \begin{bmatrix} \cos(\gamma_3) & -\sin(\gamma_3) & 0 \\ \sin(\gamma_3) & \cos(\gamma_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2c}$$

Each elementary rotation matrix belongs to $SO(3)$. Composing the three rotation matrices involves matrix multiplication which is not commutative. Therefore, there exists many possible sequences of multiplying the three elementary matrices together. A widely used sequence that is used in aerospace and for our application is the roll-pitch-yaw sequence defined by

$$R_N^B = R_3(\psi) R_2(\theta) R_1(\phi) \tag{3.3}$$

where roll ϕ, pitch θ, and yaw ψ are defined as the angle of rotation about the x,y, and z axes, respectively. The difference between how this sequence is used for our application compared to aerospace is that we do not use the north-east-down frame of reference rather the north-west-up frame of reference. Substituting in Equations (3.2a), (3.2b), and (3.2c) into (3.3) and performing the matrix multiplication yields the following result.

$$R_N^B = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi s_\theta & c_\phi c_\theta \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{23} & r_{33} \end{bmatrix} \tag{3.4}$$

where $c = \cos()$ and $s = \sin()$. The roll-pitch-yaw sequence is surjective but not injective onto $SO(3)$, and it's a known fact that any three term parameterization of $SO(3)$ will contain singularities. Further details about singularities in Euler angles are found in [74, 75]. Singularities will be avoided as long as the angles remain within the following bounds.

$$-\frac{\pi}{2} < \theta < \frac{\pi}{2}$$
$$-\pi < \phi < \pi$$
$$-\pi < \psi < \pi$$

Within our application, the drone will not rotate past these bounds for roll and pitch, however, the drone will rotate past the bounds set for yaw. The issues that arise when the yaw is rotated past the bounds is caused by the periodicity of the trigonometric function. The full rotation of the UAV is contained within the bounds of $[-\pi, \pi]$ where the two bounds $-\pi$ and π coincide with each other. Therefore as the UAV rotates past π in the counterclockwise direction, the yaw data retrieved from the Vicon motion capture system will report a negative angle which coincides with the rotation in the clockwise direction. This issue is addressed in Section 3.3.

3.1.3 Unit Quaternions

Unit quaternions are a parameterization of a 3D rotation using 4 variables. Information about rotations of objects in ROS are reported as (unit) quaternions. Quaternions are used to determine the rotation matrix between the navigation frame and the body frame, measure the rotation between two frames, and to obtain Euler angles which are used in our control systems.

A quaternion $r \in \mathbb{H}$ is defined as

$$r = r_0 + r_1\boldsymbol{i} + r_2\boldsymbol{j} + r_3\boldsymbol{k}$$

where $(r_0, r_1, r_2, r_3) \in \mathbb{R}^4$ and $\mathbb{H}$ is four-dimensional vector space over the reals with the basis vectors $(1, \boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}) \in \mathbb{H}$. Another way to express a quaternion is as $r = (r_0, \overrightarrow{r})$ where $r_0 \in \mathbb{R}$ is the scalar component of the quaternion and $\overrightarrow{r} = (r_1, r_2, r_3) \in \mathbb{R}^3$ is the vector component of the quaternion.

The rotation matrix can be obtained from quaternions and the details of determining the rotation matrix from quaternions can be found in [74, 76]. The relationship between quaternions and a rotation matrix is shown in the expression below.

$$R = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} \tag{3.5}$$

Further details about quaternions can be found in [74, 76].

3.2 Parrot AR.Drone 2.0 Control Model

3.2.1 Parrot AR.Drone 2.0 Internal Control Model Description

In order to implement a Linear Model Predictive motion controller for the Parrot AR.Drone 2.0, the dynamics of the plant need to be known. The Parrot AR.Drone 2.0 is known to employ an internal controller to stabilize the attitude and velocity of the vehicle. Details of this architecture are discussed in [60], however, the actual parameter values of the internal controller are not given. Several papers have modelled the dynamics of the stabilized Parrot AR.Drone 2.0 [33, 77]. This thesis will propose an alternative dynamics model of the stabilized Parrot AR.Drone 2.0 extended from [33].

The dynamics model of the Parrot AR.Drone 2.0 was conceived by considering the UAV in 3D space as seen in Figure 3.3.

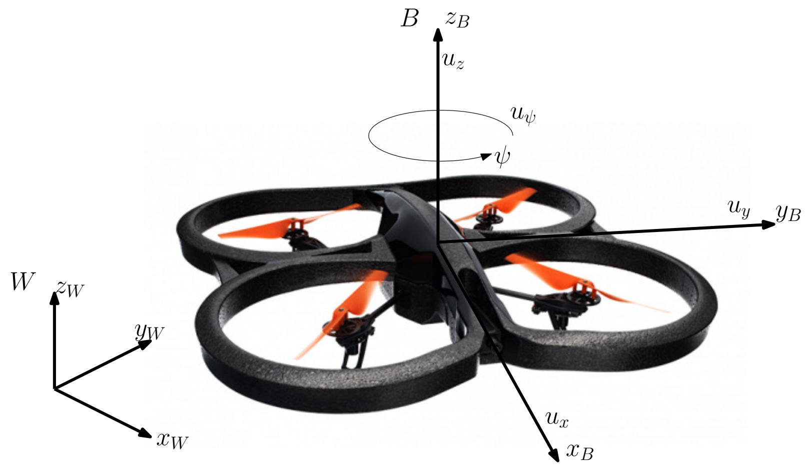


Figure 3.3: Parrot AR.Drone 2.0 Vehicle Coordinate Frame

The subscript b refers to the body fixed frame of the UAV located at its center of mass, and the subscript w refers to the global world coordinate frame which coincides with the origin of the Vicon motion capture system. Therefore, x_w,y_w, and z_w are the coordinates of the UAV's center of mass (CM) in the world coordinate frame while x_b, y_b, and z_b are the coordinates in the body fixed frame. The parameters ϕ, θ, and ψ are the Euler angles of roll, pitch and yaw of the UAV's body fixed frame, respectively. The assumed dynamics model of the system consists of the following first-order differential equations.

$$\ddot{x}_b(t) = a_\alpha \ \dot{x}_b(t) + b_\alpha \ u_x(t) \tag{3.6a}$$
$$\ddot{y}_b(t) = a_\beta \ \dot{y}_b(t) + b_\beta \ u_y(t) \tag{3.6b}$$
$$\ddot{z}_b(t) = a_\gamma \ \dot{z}_b(t) + b_\gamma \ u_z(t) \tag{3.6c}$$
$$\dot{\psi}(t) = a_\psi \ \psi(t) + b_\psi \ u_\psi(t) \tag{3.6d}$$

The drone can be commanded to translate in every direction and can also rotate about the z_B axis as shown in Figure 3.3. The control inputs of the Parrot AR.Drone 2.0 consist of the decimal percentage of the maximum pitch and roll angles u_x and u_y, respectively, the decimal percentage of the maximum vertical acceleration u_z, and the decimal percentage of the maximum rotational velocity u_ψ [77, 78]. The control inputs are bounded as $u_{x,y,z,\psi} \in [-1, 1]$. The parameters $a_{\alpha,\beta,\gamma,\psi}$ and $b_{\alpha,\beta,\gamma,\psi}$ are identified from several experiments described in Section 3.2.2. It should be noted that the entire system has 6 degrees of freedom and is being controlled by 4 control inputs. This means that the system is underactuated.

The proposed dynamics model assumes that the transitional control inputs $u_{x,y,z}$ directly affects the acceleration of the Parrot AR.Drone 2.0 along its body fixed frame axes, and the control input u_ψ directly affects the angular velocity of the Parrot AR.Drone 2.0. Previous research that have used the Parrot AR.Drone 2.0 have stated that the control input u_z relates to the vertical velocity rather than the vertical acceleration of the drone [33]. This thesis suggests that the control input u_z relates to the vertical acceleration of the drone which is supported by results shown in Section 3.2.2.

In addition, this dynamics model assumes that the system is linear and that the axes are decoupled from each other. This model structure has the advantage of being simpler and easier to implement into a controller, however, may not accurately represent the entirety of the system. The dynamics of the Parrot AR.Drone 2.0 are nonlinear as reinforced by the work done in [33, 77, 78], and future work will investigate implementing a nonlinear controller for the Parrot AR.Drone 2.0. Despite using a simplified dynamics model, Chapters 5 and 6 will show that motion control of the Parrot AR.Drone 2.0 can be achieved.

Since the assumed dynamics model of the Parrot AR.Drone 2.0 is assumed to be linear, a state space model can be determined. A state space model is a mathematical representation of a physical system that consists of input, output and state variables that are related by first-order differential equations. A continuous time state space representation of a physical system can be written as

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

where $x(t)$ is the state vector, $u(t)$ is the control input vector, $y(t)$ is the output vector, A is the state matrix, B is the input matrix, C is the output matrix, and D is the feed forward matrix.

The state vector of the internal controller are found from Equations (3.6a), (3.6b), (3.6c), and (3.6d). The state vector of the state space model is represented by the following expression.

$$x(t) = [x_b(t), y_b(t), z_b(t), \psi(t), \dot{x_b}(t), \dot{y_b}(t), \dot{z_b}(t)]^T \tag{3.7}$$

Where $(x_b(t), y_b(t), z_b(t))$ is the position of the drone in coordinates of the body fixed frame, $(\dot{x}_b(t), \dot{y}_b(t), \dot{z}_b(t))$ is the velocity of the drone in coordinates of the body fixed frame, and $\psi(t)$ is the yaw angle of the drone.

The entries of the control input vector consist of the four control inputs found in Equations (3.6a), (3.6b), (3.6c), and (3.6d) and is represented by the following expression

$$u(t) = [u_x(t), u_y(t), u_z(t), u_\psi(t)] \tag{3.8}$$

Using the state vector expressed in Equation (3.7) and the control input vector expressed in Equation (3.8), the state space model can be constructed and is shown in the expression below.

$$\frac{d}{dt}\begin{bmatrix} x_b(t) \\ y_b(t) \\ z_b(t) \\ \psi(t) \\ \dot{x}_b(t) \\ \dot{y}_b(t) \\ \dot{z}_b(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & a_\psi & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_\alpha & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a_\beta & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_\gamma \end{bmatrix} \begin{bmatrix} x_b(t) \\ y_b(t) \\ z_b(t) \\ \psi(t) \\ \dot{x}_b(t) \\ \dot{y}_b(t) \\ \dot{z}_b(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & b_\psi \\ b_\alpha & 0 & 0 & 0 \\ 0 & b_\beta & 0 & 0 \\ 0 & 0 & b_\gamma & 0 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \\ u_\psi \end{bmatrix}$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_b(t) \\ y_b(t) \\ z_b(t) \\ \psi(t) \\ \dot{x}_b(t) \\ \dot{y}_b(t) \\ \dot{z}_b(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \\ u_\psi \end{bmatrix}$$

3.2.2 Model Parameter Identification

The model parameters are determined by inputting a unit step into each control channel, measuring the response of the drone, and calculating the associated model parameters. The input command is sent to the drone via Wi-Fi. The step duration is 2 s with the step amplitude at a value of either 1 or -1. The position of the drone is found using the Vicon motion capture system, sampling at 100 Hz and the ROS package `vicon_bridge`. The ROS package `vicon_bridge` publishes a ROS topic called `vicon/ARDrone/ARDrone` which contains a `tf` ROS message. The `tf` ROS message reports the position of the drone in the world frame and the quaternion of the drone's attitude. The rotation matrix can be obtained from the quaternion using methods described in Section 3.1.3 and used to map the position from the world frame to the body fixed frame. The drone velocity is determined by using a finite difference method followed by windowing the data with a Hanning 55 dB window. Experiments were run for each input channel to identify the model parameters. The results from those experiments are shown in Figures 3.4, 3.5, 3.6,and 3.7. Note that in the figures below, the control input signal is denoted by u, the filtered measurement response is denoted by the subscript m, and the model prediction response is denoted by the subscript p.

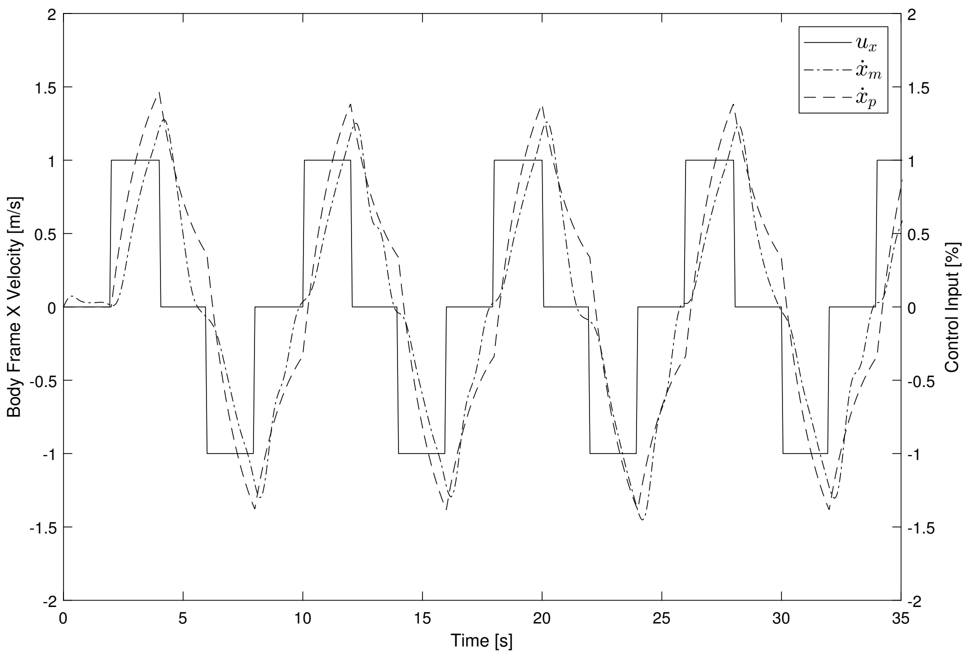


Figure 3.4: Model Parametrization for the X-axis

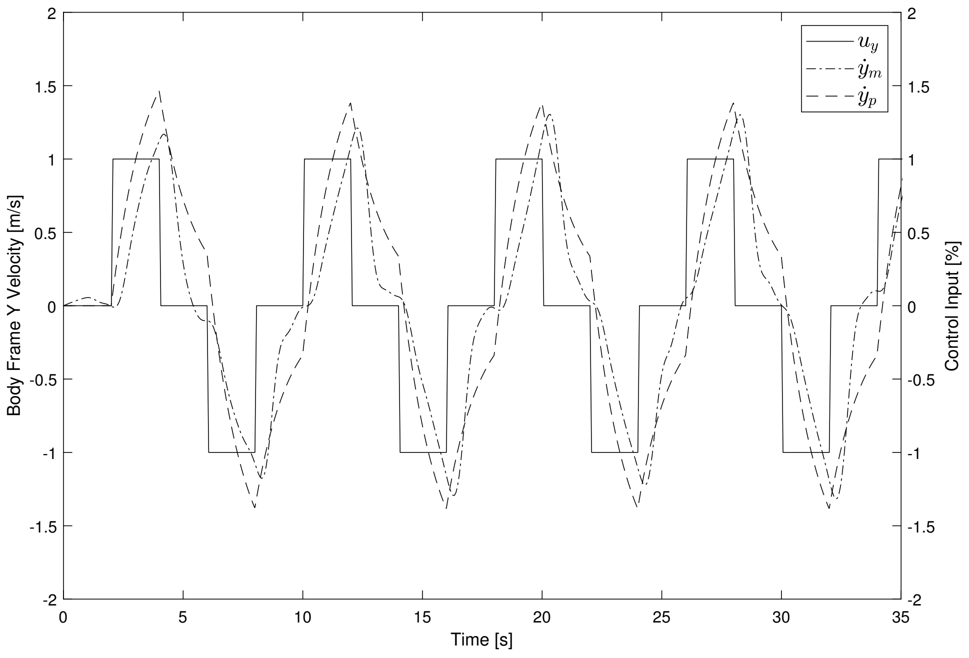


Figure 3.5: Model Parametrization for the Y-axis

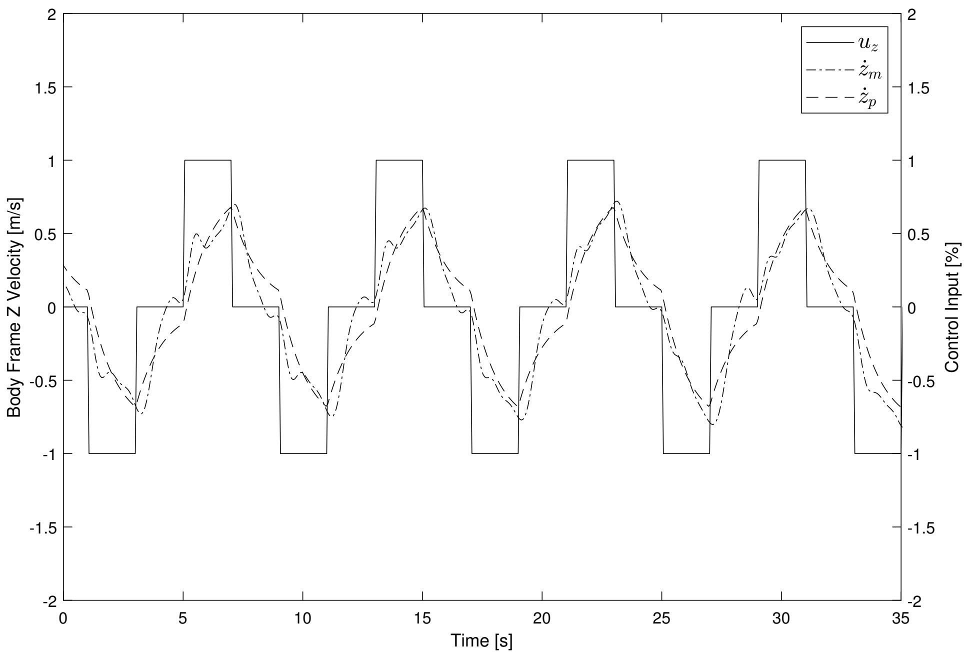


Figure 3.6: Model Parametrization for the Z-axis

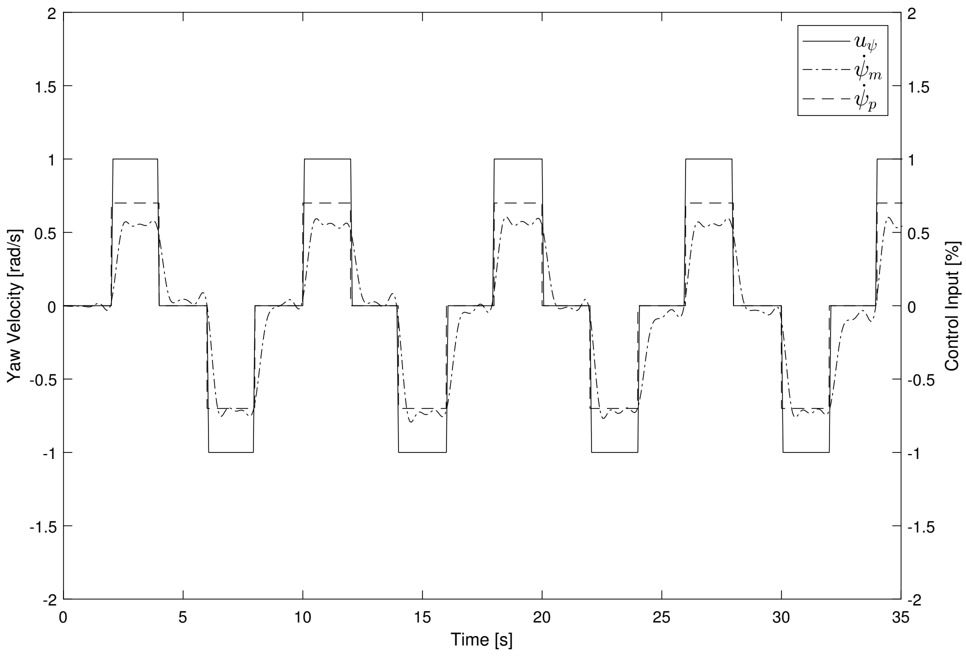


Figure 3.7: Model Parametrization for the Yaw rotation

The model parameters were found by a fitting a straight to the predicted and measured parameters and averaging the values over a number of trails. The chosen model parameters were determined to be $a_\alpha = a_\beta = -0.705$, $a_\gamma = -0.905$, $a_\psi = 0$, $b_\alpha = b_\beta = 1.370$, $b_\gamma = 0.755$, and $b_\psi = 0.7$. For $\dot{x}_b$, $\dot{y}_b$, and $\dot{z}_b$, the results suggest that the chosen model parameters will result in an accurate representation in the real systems behaviour. From Figure 3.6, the data suggests that the control input

u_z affects the acceleration rather than the velocity as suggested in [33]. In addition, as seen in Figure 3.7, the yaw input u_ψ linearly affects the angular velocity in yaw as proposed in [33] and is contrary to the yaw dynamics described in [77]. The dynamics the Parrot AR.Drone are thus described as

$$\begin{bmatrix} \dot{x}_b(t) \\ \dot{y}_b(t) \\ \dot{z}_b(t) \\ \dot{\psi}(t) \\ \ddot{x}_b(t) \\ \ddot{y}_b(t) \\ \ddot{z}_b(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.705 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.705 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.905 \end{bmatrix} \begin{bmatrix} x_b(t) \\ y_b(t) \\ z_b(t) \\ \psi(t) \\ \dot{x}_b(t) \\ \dot{y}_b(t) \\ \dot{z}_b(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.7 \\ 1.37 & 0 & 0 & 0 \\ 0 & 1.37 & 0 & 0 \\ 0 & 0 & 0.755 & 0 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \\ u_\psi \end{bmatrix}$$

3.2.3 Discrete Time Low-Pass Filter Description

Within our application, a low-pass filter is used to filter out noise in our velocity estimates which are used in our LMPC motion controller. The velocity of the drone is determined using a simple numerical differentiation which inherently results in noisy data. The low-pass filter used in our application is a first-order filter. The discretized version of the filter is derived from the transfer function in Equation (3.9),

$$G(s) = \frac{Y(s)}{U(s)} = \frac{1}{\tau s + 1} \tag{3.9}$$

where τ is the time constant in seconds, $U(s)$ is the filter input and $Y(s)$ is the filter output, both in the s domain.

Rearranging Equation (3.9) such that the filter input is on the left-hand side and the filter output is on the right-hand yields the following expression.

$$U(s) = \tau s Y(s) + Y(s) \tag{3.10}$$

Taking the inverse Laplace transform of Equation (3.10) yields the following first-order differential equation in the time domain

$$u(t) = \tau \dot{y}(t) + y(t) \tag{3.11}$$

Take t_k as a point in time indexed by k

$$u(t_k) = \tau \dot{y}(t_k) + y(t_k) \tag{3.12}$$

We approximate the time derivative of the output by the following expression

$$\dot{y}(t_k) = \frac{y(t_k) - y(t_{k-1})}{\Delta t} \tag{3.13}$$

Where $\Delta t = t_k - t_{k-1}$. Substituting Equation (3.13) into Equation (3.12) and rearranging such that $y(t_k)$ is on the left-hand side yields the following expression

$$y(t_k) = \frac{\tau}{\tau + \Delta t} y(t_{k-1}) + \frac{\Delta t}{\tau + \Delta t} u(t_k) \tag{3.14}$$

This can be rewritten in the form

$$y(t_k) = (1 - c_f) y(t_{k-1}) + c_f u(t_k) \tag{3.15}$$

where the filter coefficient c_f is expressed as

$$c_f = \frac{\Delta t}{\tau + \Delta t}$$

The filter input $u(t_k)$ for our application is the position of the drone along a body frame vector at the current time t_k, while $y(t_{k-1})$ is the filtered velocity at the previous time t_{k-1} along the same vector. The positions of the drone along all three body-fixed axes needed to be filtered. In each case, the time constant τ was determined by filtering a series of measured position data along all three directions through MATLAB with different time constants and determining τ. The following table shows the τ values used to filter the body frame positions along each axis.

Table 3.1: Low-Pass Filter Time Constants

Body Frame Velocity	τ [s]
$\dot{x}_b$ [m]	0.027
$\dot{y}_b$ [m]	0.027
$\dot{z}_b$ [m]	0.035

3.3 Geometric Tracking of Yaw

This section reproduces an approach suggested by Dr. Martin Barczyk [79] to overcome the periodicity issue present in the Euler yaw angle by using Lie groups. This method can be implemented in both PID and LMPC motion controllers.

3.3.1 Mathematical Background

Similar to a rotation matrix that represents rotations in $\mathbb{R}^3$, rotations in a plane ($\mathbb{R}^2$) can be described by the rotation matrix

$$R(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix} \tag{3.16}$$

Physically, for a point $p = (x_1, x_2)$ in the cartesian plane and an angle ψ measured counter-clockwise from the positive X axis in the cartesian plane, the product $R(\psi)p$ yields the point p rotated by angle ψ.

In the same way that a rotation matrix in $\mathbb{R}^{3\times3}$ is a member of $SO(3)$, the rotation matrix in Equation (3.16) is an element of $SO(2)$ which is the special orthogonal group of matrices in $\mathbb{R}^{2\times2}$. This can be written as $R(\psi) \in SO(2), \psi \in \mathbb{R}$. Similarly to matrices found in $SO(3)$, matrices in $SO(2)$ have the following properties:

$$R(\psi)^{-1} = R(\psi)^T \qquad \text{and} \qquad \det R(\psi) = +1$$

Using Equation (3.16), you can directly verify that for $R_1, R_2 \in SO(2)$, $R_1R_2 = R_2R_1$. This means that planar rotations are commutative. This is a difference from three-dimensional rotations in $SO(3)$ which are non-commutative.

$SO(2)$ is a Lie group which is associated to a Lie algebra called $so(2)$. Details about Lie algebra and Lie groups can be found in [80] and [81]. The elements of

$so(2)$ are written as

$$so(2) \ni S_2(\delta) = \begin{bmatrix} 0 & -\omega \\ \omega & 0 \end{bmatrix}, \qquad \delta \in \mathbb{R}$$

where by inspection $S_2(\delta)$ is a skew-symmetric matrix which means that $S_2(\delta)^T = -S_2(\delta)$. A Lie algebra like $so(2)$ is also a vector space, and by inspection we see that $so(2)$ is a 1-dimensional space and therefore isomorphic to $\mathbb{R}$. The linear, bijective, and thus globally invertible mapping is

$$H : \mathbb{R} \to so(2), H(\delta) = S_2(\delta)$$

Any Lie group and its Lie algebra are connected by a locally invertible mapping. In the case $SO(2)$, the map $so(2) \to SO(2)$ is the exponential

$$\exp : so(2) \to SO(2), \qquad \exp S_2(\delta) = e^{S_2(\delta)} \tag{3.17}$$

While the inverse map $SO(2) \to so(2)$ is called the logarithm

$$\log SO(2) \to so(2), \qquad \log R = S_2(\operatorname{atan2}(R^{21}, R^{11})) \tag{3.18}$$

Equation (3.17) states that taking the matrix exponential of the skew-symmetric matrix S_2 results in a planar rotation matrix. Conversely, Equation (3.18) states that the rotation angle of an $SO(2)$ matrix can be extracted and used to form the corresponding $so(2)$ entry.

Using Equation (3.16), the dynamics of $R(\psi) \in SO(2)$ can be directly computed to be

$$\begin{aligned} \frac{d}{dt}\begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} &= \begin{bmatrix} -\sin\psi(\dot\psi) & -\cos\psi(\dot\psi) \\ \cos\psi(\dot\psi) & -\sin\psi(\dot\psi) \end{bmatrix} \\ &= \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix}\begin{bmatrix} 0 & -\dot\psi \\ \dot\psi & 0 \end{bmatrix} \qquad \implies \dot R = RS_2(\dot\psi) \end{aligned}$$

3.3.2 Redefined Yaw Error

The error between desired and actual yaw angle is written as $e_\psi = \psi_d - \psi$. The problem is that because ψ_d is obtained from an atan2 function, it exhibits jump discontinuities of $\pm 2\pi$ for a reference trajectory with sufficiently large turns. These discontinuities transfer into e_ψ which may cause the closed-loop control system to react violently and go unstable.

In order to address the issue of discontinuities, yaw error is redefined. Denoting the desired yaw angle as ψ_d and actual (measured) yaw angle as ψ, and from the $SO(2)$ matrices $R_d := R(\psi_d)$ and $R := R(\psi)$, the error matrix can be redefined as

$$R_e := R_d R^T \tag{3.19}$$

By construction, $R_e \in SO(2)$ and $\psi_d = \psi \implies R_d = R \implies R_e = I_{2\times 2}$ i.e. a zero yaw error corresponds to an identity error matrix. Because of the commutativity of $SO(2)$, the error matrix can be equivalently written as $R_e = R^T R_d$.

The rotation angle corresponding to R_e is given by

$$\varepsilon_\psi := H^{-1}\left[\log R_e\right] = \operatorname{atan2}\left(R_e^{21}, R_e^{11}\right) \tag{3.20}$$

It can be directly verified that ε_ψ physically represents $\psi_d \ominus \psi$, the shorter of the two angles between ψ to ψ_d on a circle. Thus ε_ψ can be directly used for a PID-style yaw control law.

3.3.2.1 Application to Parrot AR.Drone 2.0 State Space Model

The redefined yaw error described in Section 3.3.2 can be applied to the state dynamics of the Parrot AR.Drone 2.0 which is used to derive the state space model. As shown in Section 3.2.2, the yaw dynamics of the quadcopter UAV are modelled as

$$\dot{\psi} = a_\psi \psi + b_\psi u_\psi$$

The two constant parameters a_ψ and b_ψ were found in Section 3.2.2, and the parameter $a_\psi = 0$ which means that the yaw dynamics can be simplified to

$$\dot{\psi} = b_\psi u_\psi$$

Note a positive yaw input u_ψ causes the yaw angle ψ of the quadcopter to increase, and vice-versa.

For the LMPC controller, take ε_ψ as a state of the system. As shown below, the linearized dynamics of ε_ψ are governed by

$$\dot{\varepsilon}_\psi = -b_\psi u_\psi \tag{3.21}$$

For practical implementation, the value of state ε_ψ is calculated by employing Equation (3.19) and Equation (3.20) in sequence, where ψ is the measured yaw of the drone, and ψ_d is the desired yaw of the drone. Then, by using $\varepsilon_{\psi,d} = 0$ in the reference trajectory of the closed-loop control system, the quadcopter's yaw angle ψ will track the desired value ψ_d generated by the trajectory generator. Therefore, the Parrot AR.Drone 2.0 state space dynamics can be redefined as:

$$\begin{bmatrix} \dot{x}_b(t) \\ \dot{y}_b(t) \\ \dot{z}_b(t) \\ \dot{\epsilon_\psi}(t) \\ \ddot{x}_b(t) \\ \ddot{y}_b(t) \\ \ddot{z}_b(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.705 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.705 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.905 \end{bmatrix} \begin{bmatrix} x_b(t) \\ y_b(t) \\ z_b(t) \\ \epsilon_\psi(t) \\ \dot{x}_b(t) \\ \dot{y}_b(t) \\ \dot{z}_b(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.7 \\ 1.37 & 0 & 0 & 0 \\ 0 & 1.37 & 0 & 0 \\ 0 & 0 & 0.755 & 0 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \\ u_\psi \end{bmatrix}$$

To prove Equation (3.21), start with the $SO(2)$ dynamics $\dot{R} = RS_2(\dot{\psi})$, which leads to

$$\frac{d}{dt}[R] = RS_2(b_\psi u_\psi)$$

Transposing both sides,

$$\frac{d}{dt}\left[R^T\right] = -S_2(b_\psi u_\psi)R^T$$

Right-multiplying by R_d (taken as a constant, i.e. not a function of time)

$$\frac{d}{dt}\left[R^T R_d\right] = -S_2(b_\psi u_\psi)R^T R_d$$

Since $R_e = R_d R^T = R^T R_d$,

$$\frac{d}{dt}[R_e] = -S_2(b_\psi u_\psi)R_e$$

Using Equation (3.17)

$$\frac{d}{dt}\left[e^{S_2(\varepsilon_\psi)}\right] = -S_2(b_\psi u_\psi)e^{S_2(\varepsilon_\psi)}$$

Evaluating the time derivative,

$$S_2(\dot{\varepsilon}_\psi)e^{S_2(\varepsilon_\psi)} = -S_2(b_\psi u_\psi)e^{S_2(\varepsilon_\psi)}$$

We now introduce linearizing approximations. The matrix exponential is defined as

$$e^{S_2(\varepsilon_\psi)} = I + S_2(\varepsilon_\psi) + \frac{1}{2!}\left[S_2(\varepsilon_\psi)\right]^2 + \cdots$$

but by assuming $\varepsilon_\psi < 1$ (angular error $\leq 60°$), we can *neglect* $\varepsilon_\psi^2, \varepsilon_\psi^3, \ldots$ as close to zero. Thus

$$S_2(\dot{\varepsilon}_\psi)\left(I + S_2(\varepsilon_\psi)\right) \approx -S_2(b_\psi u_\psi)\left(I + S_2(\varepsilon_\psi)\right)$$

which expands to

$$S_2(\dot{\varepsilon}_\psi) + S_2(\dot{\varepsilon}_\psi)S_2(\varepsilon_\psi) = -S_2(b_\psi u_\psi) - S_2(b_\psi u_\psi)S_2(\varepsilon_\psi)$$

The second and fourth terms above expand to quadratic (nonlinear) terms in $\dot{\varepsilon}_\psi\varepsilon_\psi$ and $b_\psi u_\psi \varepsilon_\psi$ respectively. In order to obtain a linear equation, we need to neglect these. Assuming this holds, we have

$$S_2(\dot{\varepsilon}_\psi) \approx -S_2(b_\psi u_\psi)$$

And matching the arguments of S_2 (equivalently applying H^{-1} to both sides) yields

$$\dot{\varepsilon}_\psi = -b_\psi u_\psi$$

as claimed in (3.21).

3.4 Proportional Integral Derivative Control

A PID controller is a common type of controller employed in industry. A PID controller is a feedback system that consists of manipulating an input variable up until a state variable reaches a desired setpoint. The input of the controller is the error $e = r - x$ between the reference (also known as the setpoint), r, and the measured state, x. The controller parameters are the proportional gain k_p, the integral gain k_i, and the derivative gain k_d. The control action u is computed as

$$u(t) = k_p e + k_i \int_0^t e(\tau)d\tau + k_d \frac{de}{dt}$$

where $u(t)$ is the sum of the proportional action, the integral action and the derivative action. The transfer function of the PID controller is given by

$$C(s) = k_p + \frac{k_i}{s} + k_d s$$

The corresponding closed-loop is shown below:

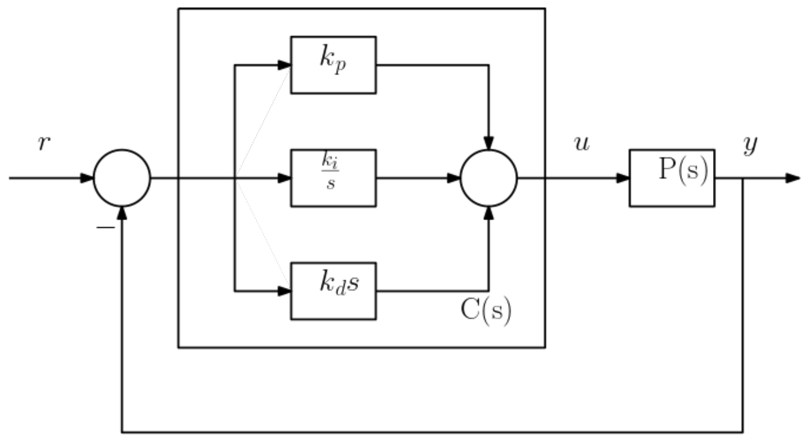


Figure 3.8: PID Closed Feedback System

Where $C(s)$ is the transfer function of the PID controller and $P(s)$ is the transfer function of the plant. The proportional gain, k_p, is related directly to the error between the setpoint and the state variable. As described in [82], as the proportional gain increases the response speed increases, however, the system also becomes more oscillatory. Pure proportional control may have a steady state error to reference step inputs. There are situations where a pure proportional control has no steady state error such as in the case of a Type 1 plant. Errors associated with proportional control can be eliminated using integral control. The integral gain, k_i, as long as it is nonzero will eliminate steady state step tracking error. The integral gain needs to be tuned to eliminate step tracking error in a reasonable time without going unstable. In order to ensure that the system response is damped, the derivative gain, k_d is used. Note that larger values of k_d will result in a slower system response and will amplify noise. In addition, the derivative control is cascaded with a low-pass filter to prevent introducing impulses into the plant. The combination of all three actions can result in a stable closed loop system that reaches the desired setpoint in an appropriate amount of time.

3.5 Linear Model Predictive Control With Reference Tracking

Model Predictive Control (MPC) is a class of control algorithms which compute an input by utilizing a linear or nonlinear process model to optimize an open loop quadratic cost function subject to constraints over a future time horizon [39]. All classes of MPC have three common components as mentioned: a prediction model, cost function, and constraints. MPC controllers model the future inputs that will result in a system reaching a desired a reference state. The following schematic illustrates this point.

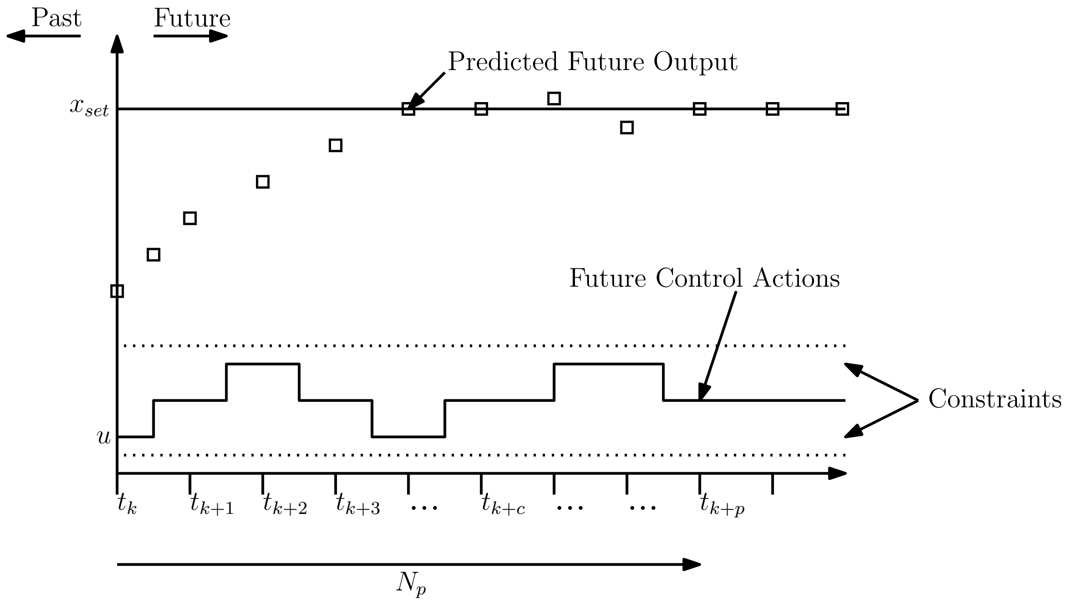


Figure 3.9: MPC Control Strategy

The predicted future outputs for a prediction horizon N_p are predicted at each instant t_k using the prediction model up until the instant t_{k+p}. The predicted outputs depend on the previous inputs and outputs into the system as well as the future control actions. The future control actions are calculated by optimizing a cost function to keep the control process as close to the reference position x_{set} and considering constraints to the system. The control signal u at the time t_{k+1} is obtained from the future control actions and sent to the plant of a system.

The MPC controller used in this thesis will not be subjected to constraints and the addition of constraints is left for future work. The predication model and cost function of the proposed linear model predictive controller are described in the following sections.

3.5.1 Prediction Model

The prediction model is used to calculate the output of a process at future instants in time. Some examples of prediction models are impulse responses, transfer functions or state space representations. We are using the discrete-time formulation of an MPC rather than the continuous time formulation. We are using the discrete form of a MPC since the system that we are controlling obtains digital signals that sample information at discrete points in time. Therefore, the continuous state space space model described in Section 3.2.1 is converted into a discrete state space model using techniques described in [83]. In this case, we consider the discrete state space model of a dynamic system described as the following set of equations.

$$x_{k+1} = Ax_k + Bu_k \tag{3.22}$$

$$y_k = Cx_k + Du_k \tag{3.23}$$

We will assume that the model predictive controller will be a full state feedback controller or in other words $C = I$, where I is the identity matrix. This assumption is valid for systems where the full state can be measured which is for our case due to the Vicon motion capture system. The goal for the control system is to minimize the error, e_k, between the desired reference trajectory,r_k, and the current position and yaw of the object, x_k.

$$e_k = r_k - x_k \to 0 \tag{3.24}$$

In order to ensure that the error reaches a steady state of approximately zero, an integral control is required. This involves redefining the input as follows:

$$u_k = \triangle u_k + u_{k-1} \tag{3.25}$$

Where the input, u_k, is the sum of the the previous input, u_{k-1}, and the difference between the inputs, $\triangle u_k$. Substituting Equation (3.25) into Equation (3.22) and setting u_{k-1} as a state and $\triangle u$ as the input, we form the following state model.

$$\begin{bmatrix} x_{k+1} \\ u_k \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} + \begin{bmatrix} B \\ I \end{bmatrix} \triangle u_k \tag{3.26}$$

$$y_k = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} \tag{3.27}$$

Equations (3.26) and (3.27) can be written as

$$\tilde{x}_{k+1} = \overline{A}\tilde{x}_k + \overline{B} \triangle u_k$$

$$y_k = x_k = \overline{C}\tilde{x}_k$$

Where the matrices of the new state model are

$$\overline{A} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix}, \overline{B} = \begin{bmatrix} B \\ I \end{bmatrix}, \overline{C} = \begin{bmatrix} C & 0 \end{bmatrix}$$

and the new state vector is defined as

$$\tilde{x}_k = \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} \tag{3.28}$$

3.5.2 Cost Function

In order to obtain the optimum steps to approach a desired set-point value a cost function as described below is used,

$$J = \frac{1}{2}\sum_{k=0}^{N} [e_{t+k}^T Q e_{t+k}] + \frac{1}{2}\sum_{k=0}^{N-1} [\Delta u_{t+k}^T R \Delta u_{t+k}]$$

Note that the matrices Q and R represent weights used in the control system. The matrix Q is a positive-definite matrix used to weigh the importance of the tracking error, and R is a positive semi-definite matrix used to weight the importance of the control input. Substituting (3.24) into the above equation and expanding out, we find:

$$J = \sum_{k=0}^{N} \left[\underbrace{\frac{1}{2} r_{t+k}^T Q r_{t+k}}_{constant} - r_{t+k}^T Q \bar{C} \tilde{x}_{t+k} + \frac{1}{2} \tilde{x}_{t+k}^T \bar{C}^T Q \bar{C} \tilde{x}_{t+k} \right] + \frac{1}{2}\sum_{k=0}^{N-1} [\Delta u_{t+k}^T R \Delta u_{t+k}]$$

With this idea in mind, consider the following vector representations:

$$r = \begin{bmatrix} r_t \\ r_{t+1} \\ \vdots \\ r_{t+N-1} \end{bmatrix} \qquad \overline{x}_{t+1} = \begin{bmatrix} \widetilde{x}_{t+1} \\ \widetilde{x}_{t+2} \\ \vdots \\ \widetilde{x}_{t+N} \end{bmatrix} \qquad \Delta u = \begin{bmatrix} \Delta u_t \\ \Delta u_{t+1} \\ \vdots \\ \Delta u_{t+N-1} \end{bmatrix}$$

Therefore, using the above definitions, and substituting them into the cost function described in the equation above, we find:

$$J = \frac{1}{2}\overline{x}_{t+1}^T \overline{Q}\,\overline{x}_{t+1} - r^T \overline{T}\,\overline{x}_{t+1} + \frac{1}{2}\Delta u^T \overline{R} \Delta u \tag{3.29}$$

where the matrices $\overline{Q}$, $\overline{T}$, and $\overline{R}$ are defined as following:

$$\overline{Q} = \begin{bmatrix} \widetilde{C}^T Q \widetilde{C} & 0 & \cdots & 0 \\ 0 & \widetilde{C}^T Q \widetilde{C} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \widetilde{C}^T Q \widetilde{C} \end{bmatrix} \overline{T} = \begin{bmatrix} Q\widetilde{C} & 0 & \cdots & 0 \\ 0 & Q\widetilde{C} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & Q\widetilde{C} \end{bmatrix} \quad \overline{R} = \begin{bmatrix} R & 0 & \cdots & 0 \\ 0 & R & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & R \end{bmatrix}$$

Equation (3.29) depends on the state dynamics of the system for each future time step and is described by the following expression:

$$\overline{x}_{t+1} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & \widetilde{A} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \widetilde{A} & 0 \end{bmatrix} \overline{x}_{t+1} + \begin{bmatrix} \widetilde{B} & 0 & \cdots & 0 \\ 0 & \widetilde{B} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \widetilde{B} \end{bmatrix} \Delta u + \begin{bmatrix} \widetilde{A} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \overline{x}_t \tag{3.30}$$

By rearranging Equation (3.30) such that the equation is expressed in terms of Δu and $\widetilde{x}_t$, we find the following:

$$\begin{aligned}
\widetilde{x}_{t+1} &= \widetilde{A}\widetilde{x}_t + \widetilde{B}\Delta u_t \\
\widetilde{x}_{t+2} &= \widetilde{A}\widetilde{x}_{t+1} + \widetilde{B}\Delta u_{t+1} = \widetilde{A}^2\widetilde{x}_t + \widetilde{A}\widetilde{B}\Delta u_t + \widetilde{B}\Delta u_{t+1} \\
\widetilde{x}_{t+3} &= \widetilde{A}\widetilde{x}_{t+2} + \widetilde{B}\Delta u_{t+2} = \widetilde{A}^3\widetilde{x}_t + \widetilde{A}^2\widetilde{B}\Delta u_t + \widetilde{A}\widetilde{B}\Delta u_{t+1} + \widetilde{B}\Delta u_{t+2} \\
&\vdots \\
\widetilde{x}_{t+N} &= \widetilde{A}\widetilde{x}_{t+N-1} + \widetilde{B}\Delta u_{t+N-1} = \widetilde{A}^N\widetilde{x}_t + \widetilde{A}^{N-1}\widetilde{B}\Delta u_t + \widetilde{A}^{N-2}\widetilde{B}\Delta u_{t+1} + \cdots + \widetilde{B}\Delta u_{t+N}
\end{aligned}$$

Therefore, forming these results into matrix form, we obtain the following expression:

$$\overline{x}_{t+1} = \overline{\overline{A}}\overline{x}_t + \overline{\overline{B}}\Delta u \tag{3.31}$$

where the matrices $\overline{\overline{A}}$ and $\overline{\overline{B}}$ are defined as the following:

$$\overline{\overline{A}} = \begin{bmatrix} \widetilde{A} \\ \widetilde{A}^2 \\ \widetilde{A}^3 \\ \vdots \\ \widetilde{A}^N \end{bmatrix} \qquad \overline{\overline{B}} = \begin{bmatrix} \widetilde{B} & 0 & 0 & \cdots & 0 \\ \widetilde{A}\widetilde{B} & \widetilde{B} & 0 & \cdots & 0 \\ \widetilde{A}^2\widetilde{B} & \widetilde{A}\widetilde{B} & \widetilde{B} & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \widetilde{A}^{N-1}\widetilde{B} & \widetilde{A}^{N-2}\widetilde{B} & \cdots & \widetilde{A}\widetilde{B} & \widetilde{B} \end{bmatrix}$$

Then substituting Equation (3.31) into Equation (3.29) yields the following results:

$$\begin{aligned}
J &= \frac{1}{2}\left(\overline{\overline{A}}\,\overline{x}_t + \overline{\overline{B}}\Delta u\right)^T \overline{Q}\left(\overline{\overline{A}}\,\overline{x}_t + \overline{\overline{B}}\Delta u\right) - r^T\overline{T}\left(\overline{\overline{A}}\,\overline{x}_t + \overline{\overline{B}}\Delta u\right)\\
J &= \frac{1}{2}\overline{\overline{A}}^T\overline{x}_t^T\overline{Q}\;\overline{\overline{B}}\Delta u + \underbrace{\frac{1}{2}\overline{\overline{A}}^T\overline{x}_t^T\overline{Q}\;\overline{\overline{A}}\,\overline{x}_t}_{\text{constant}} + \frac{1}{2}\overline{\overline{B}}^T\Delta u^T\overline{Q}\;\overline{\overline{A}}\,\overline{x}_t + \frac{1}{2}\overline{\overline{B}}^T\Delta u^T\overline{Q}\;\overline{\overline{B}}\Delta u\\
&\quad - \underbrace{r^T\overline{T}\;\overline{\overline{A}}\,\overline{x}_t}_{\text{constant}} - r^T\overline{T}\;\overline{\overline{B}}\Delta u + \frac{1}{2}\Delta u^T\overline{R}\Delta u\\
J &= \frac{1}{2}\Delta u^T\left(\overline{\overline{B}}^T\overline{Q}\;\overline{\overline{B}} + \overline{R}\right)\Delta u + \begin{bmatrix}\overline{x}_t^T & r^T\end{bmatrix}\begin{bmatrix}\overline{\overline{A}}^T\overline{Q}\;\overline{\overline{B}}\\ -\overline{T}\;\overline{\overline{B}}\end{bmatrix}\Delta u
\end{aligned}$$

This finally leads to the following expression:

$$J = \frac{1}{2}\Delta u^T\overline{\overline{G}}\Delta u + \begin{bmatrix}\overline{x}_t^T & r^T\end{bmatrix}\overline{\overline{H}}^T\Delta u \tag{3.32}$$

where the matrices $\overline{\overline{G}}$ and $\overline{\overline{H}}$ are defined as the following:

$$\overline{\overline{G}} = \overline{\overline{B}}^T\overline{Q}\;\overline{\overline{B}} + \overline{R} \qquad\qquad \overline{\overline{H}} = \begin{bmatrix}\overline{\overline{A}}^T\overline{Q}\;\overline{\overline{B}} & -\overline{T}\;\overline{\overline{B}}\end{bmatrix}$$

In order to obtain the optimum path, the cost function needs to be minimized. This is done by taking the Jacobian of the cost matrix and equating it zero to find the minimum. This is represented in the following expression:

$$\nabla J = \overline{\overline{G}}\Delta u + \overline{\overline{H}}\begin{bmatrix}\overline{x}_t\\ r\end{bmatrix} = 0 \tag{3.33}$$

Rearranging Equation (3.33) such that Δu is on the left-hand side yields the following:

$$\Delta u = -\overline{\overline{G}}^{-1}\overline{\overline{H}}\begin{bmatrix}\overline{x}_t\\ r\end{bmatrix} \tag{3.34}$$

Substituting Equation (3.34) into Equation (3.31) will solve for $\overline{x}_{t+1}$. Take the first entry of $\overline{x}_{t+1}$ is the $\widetilde{x}_{t+1}$ term used for the future prediction of the state. As described in Equation (3.28), the $\widetilde{x}_{t+1}$ term consists of the future prediction of the state x_{t+1} and the input required to achieve the predicted state. Note that the previous input is stored.

Chapter 4

Vision-Based Detection Tracking Algorithm

This chapter describes a method to estimate the relative position of an object in the video frame from bounding box information. The bounding box information was obtained from a vision based object detection algorithm called YOLO v2. The performance of YOLO v2 for detecting the Parrot AR.Drone UAV was discussed in the MSc thesis of Bingsheng Wei [4], however the proposed equations to estimate the position of the target relative to the follower were found to perform poorly in experimental testing. An updated approach to estimate position from a monocular camera was suggested by Dr. Martin Barczyk in [84] and used in the present thesis. This approach is outlined in Sections 4.1, 4.2, 4.3, and 4.4.

Section 4.1 describes the camera model used for the Parrot AR.Drone 2.0's on-board video camera. Section 4.2 outlines the imaging model that maps the object in 3D space to the 2D image plane. The camera calibration is described in Section 4.3. The relative position estimation is derived in Section 4.4. Finally, Section 4.5 looks into the performance of YOLO v2 in estimating the relative position of an object.

4.1 Camera Model

We assume that the physics of the monocular camera can be described by a thin lens model. A thin lens model assumes that all rays that enter the lens parallel to the optical axis intersect at a single focal point. The thin lens model assumes that all rays that enter the lens through its optical center are undeflected. The following diagram illustrates the thin lens model in 2D.

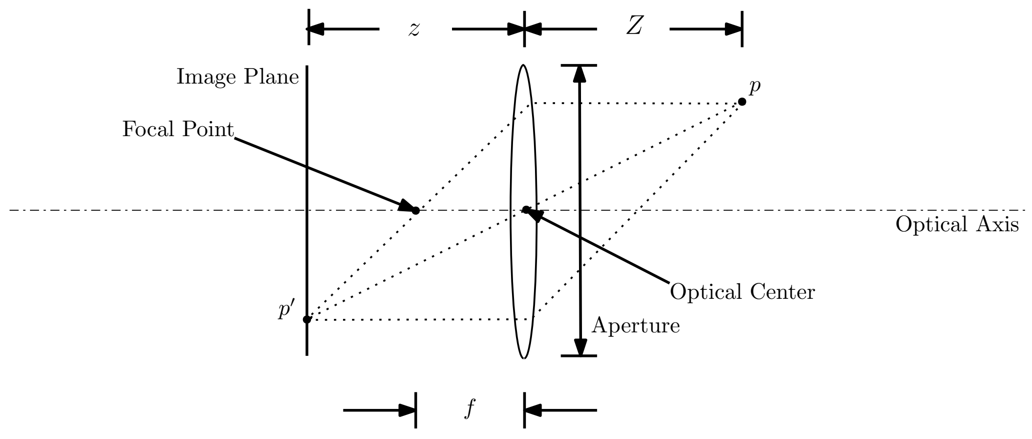


Figure 4.1: Thin Lens Model

The point p located at a distance Z from the lens is imaged onto a point p' on the image plane located a distance z from the lens. The quantity f refers to the focal length of the camera. The aperture determines the amount of rays that pass through the lens. Using similar triangles in Figure 4.1 yields the fundamental equation of the thin lens.

$$\frac{1}{f} = \frac{1}{Z} + \frac{1}{z} \tag{4.1}$$

This model can be further simplified by taking the aperture of the thin lens as approaching zero. In this case, a ray can only pass onto the image plane through the optical center of the lens. This type of thin lens model is referred to as the pinhole camera model as shown in Figure 4.2.

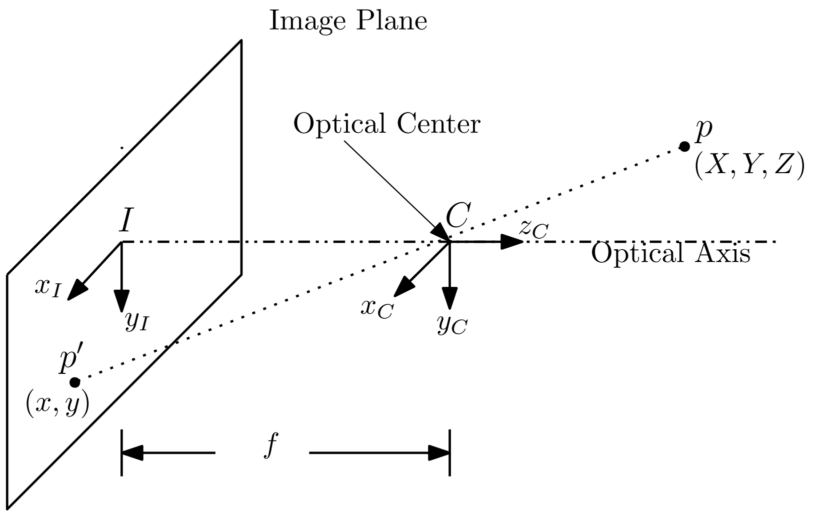


Figure 4.2: Pinhole Camera Model

In a pinhole camera model, the distance from the lens to the imaging plane is the focal length of the camera since a ray that enters the lens along the optical axis must pass through the focal point. From Figure 4.2, the frame C is the camera lens-fixed reference frame whose origin is at the optical center of the thin lens while the frame I is on the image plane and has an origin that lies at the intersection of the optical axis with the image plane.

Let (X, Y, Z) denote the coordinates of p with respect to frame C and let (x, y) denote the coordinates of the point image p' with respect to frame I. Using similar triangles leads to the following expression for (x, y):

$$x = -f\frac{X}{Z} \qquad\qquad y = -f\frac{Y}{Z} \tag{4.2}$$

The negative sign reflects the fact that the image is projected onto the image plane, which in a digital camera corresponds to the CCD sensor array, is an upside-down, mirror-flipped version of the scene in front of the camera. This is intrinsically compensated in the firmware of the camera such that the reported image correctly renders the scene. This effect can be mathematically modelled by assuming that the image plane is in front of the lens as illustrated in Figure 4.3.

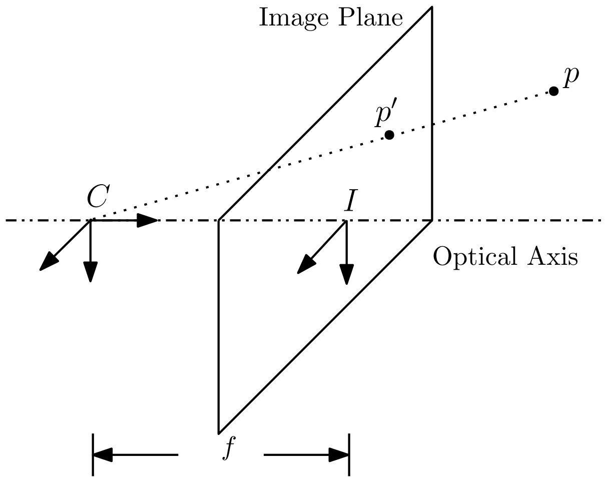


Figure 4.3: Inverted Camera Model

By taking X, Y, Z as the coordinates of p with respect to frame C, coordinates (x, y) of the point p' on the image frame I can be rewritten as

$$x = f\frac{X}{Z} \qquad\qquad y = f\frac{Y}{Z} \tag{4.3}$$

The above model is only used to obtain the sign correspondence between the 3D world coordinates and the 2D image plane coordinates and does not represent the physical setup of the camera.

4.2 Imaging Model

In Section 4.1 we obtained Equation (4.3) which maps the 3D point (X, Y, Z) to its corresponding 2D point (x, y) in the image plane. This mapping can be written in homogeneous coordinates as shown in the following expression

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \\ 1 \end{bmatrix} = \tfrac{1}{Z} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \tfrac{1}{Z} \underbrace{\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{K_f} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\Pi_0} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.4}$$

Note that (X, Y, Z), (x, y), and f are given in units of meters [m]. Since a digital camera is being used, the parameters of the model that transform the coordinates and focal length from units of meters to units of pixels (px) need to be found. This transformation can be performed with the following steps. The dimensions in meters need to be transformed into px such that

$$\begin{aligned} x_s &= s_x x \\ y_s &= s_y y \end{aligned} \tag{4.5}$$

where s_x, s_y are scaling factors in units of px/m. If $s_x = s_y$ then the pixels are square. The scaled coordinates are then transformed into the image frame I as

$$\begin{aligned} x' &= x_s + c_x \\ y' &= y_s + c_y \end{aligned} \tag{4.6}$$

where (c_x, c_y) are the coordinates of the optical axis with respect to the digital image frame, I, in units of px. Combining the two transformations in Equation (4.5) and Equation (4.6), we find

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} s_x & 0 & c_x \\ 0 & s_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K_s} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.7}$$

Combining Equation (4.4) with Equation (4.7) yields the following expression

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \frac{1}{Z} \underbrace{\begin{bmatrix} s_x f & 0 & c_x \\ 0 & s_y f & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\Pi_0} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.8}$$

where K is the intrinsic camera matrix whose entries are determined by the optical properties of the camera and are fixed. The matrix K is an upper triangular matrix which means that it is invertible provided all of its diagonal entries are non-zero. Note that we can define the focal lengths in the X- and Y-direction of the image frame as $f_x = s_x f$ and $f_y = s_y f$, respectively, in units of px. In addition, we can define a projection matrix $P = K\Pi_0$ such that

$$P = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\Pi_0} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.9}$$

4.3 Camera Calibration

The image model described in Section 4.2 is an idealized model that does not take into account distortion effects which warp the image in a number of ways. The Parrot AR.Drone 2.0 is equipped with an onboard wide angle monocular camera that has a barrel distortion. A number of models are available to remove distortion from an image. The default model used by the ROS image pipeline which employs the OpenCV library is the plumb bob model [85]. This model inputs a distorted image in image plane frame I coordinates (x_d, y_d) and outputs a corrected image in the image plane frame I coordinates (x, y) by the following calculations

$$\begin{aligned} r^2 &= x_d^2 + y_d^2 \\ x &= x_d(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x_d y_d + p_2(r^2 + 2x_d^2) \\ y &= y_d(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_2 x_d y_d + p_2(r^2 + 2y_d^2) \end{aligned} \tag{4.10}$$

where k_1, k_2, k_3 model the effect of the radial distortion created by a wide angle lens, and p_1, p_2 model the tangential distortion created by the lens plane not being perfectly parallel to the imaging plane. Since the distortion model is formulated in I frame coordinates, both (x, y) and (x_d, y_d) are in SI units.

The parameters k_1, k_2, k_3, p_1, p_2 in the distortion model can be found by performing a camera calibration. The camera calibration is performed using a ROS package called `camera_calibration` that uses a OpenCV camera calibration module [86]. This camera calibration involves employing a checkerboard with known dimensions, taking a series of pictures of this checkerboard in different poses, and performing model fitting to obtain numerical values for the calibration parameters. The checkerboard can contain an arbitrary number ≥ 3 of squares with arbitrary dimensions. In order to perform a better calibration, the checkerboard needs have a large number of squares with larger dimensions. In this thesis, a 10 x 10 checkerboard with dimensions of 10 cm were used.

Once the camera calibration is found, the ROS image pipeline for both monocular and stereo cameras can be conceptually described by the diagram below, based on a diagram found in [87].

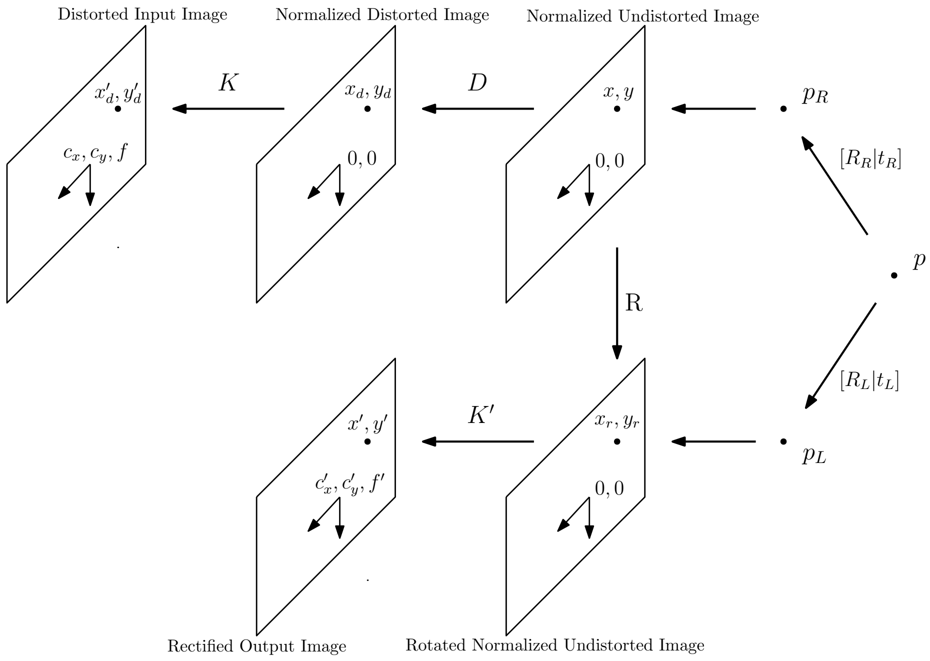


Figure 4.4: Camera Calibration Schematic

For a monocular camera setup, ROS will read in the raw, distorted video reported by the camera. This raw camera feed is transformed through the inverse of the intrinsic camera matrix K^{-1} to convert the distorted digital image coordinates (x'_d, y'_d) into the normalized image plane frame I coordinates (x_d, y_d). These converted coordinates (x_d, y_d) are run through the plumb bob distortion model in Equation (4.10) with parameter values stored in matrix D which result in the corrected coordinates (x, y) in the image plane frame I. These corrected pixel coordinates (x, y) represent the projection of a 3D point p_R onto the image plane frame. The 3D point p_R is determined from a 3D point p subjected to a rotation matrix R_R and a translational offset t. In a monocular camera the point p_R represents a 3D point viewed by the camera lens. We do not consider the point p_L since this corresponds to the view of p from the left camera which doesn't exist in this case.

The corrected coordinates (x, y) are run through a rectification matrix R which in the case of a monocular camera is an identity matrix. The final step is for the post-rectification coordinates (x_r, y_r) to be run through the projection matrix P, which for monocular cameras is $P = \begin{bmatrix} K' & e_3 \end{bmatrix}$ where $e_3 = [0 \quad 0 \quad 1]^T$. This yields (x', y') which is the undistorted version of the image captured by the onboard camera. Further details about how to run a camera calibration for a stereo camera set-up can be found in [87, 86].

Following the steps outlined in [86] will result in a calibration file which is stored within a directory in ROS. An example of the .yaml file is shown in Appendix A. This file contains the intrinsic camera matrix K, the distortion coefficients D, the rectification matrix R, and the projection matrix P. Several camera calibrations were performed and averaged out to obtain the calibration parameters which are found in Appendix A. Note that the .yaml file contains two intrinsic matrices: the camera matrix K and the projection matrix P. The camera matrix K describes the intrinsic parameters of the camera if the distortion is not removed while the left-hand subset of the projection matrix P describes the intrinsic parameters of the camera once the image correction has been applied.

4.4 Object Position Estimation from a Monocular Camera

In order to estimate the position of an object relative to the pursuer UAV from monocular camera images the following assumptions need to hold:

- A vision based object detection algorithm using a well-trained convolutional neural network (CNN) provides accurate rectangular bounding boxes of the target UAV.
- The physical dimensions of the target UAV are precisely known.
- The visible width and height of the target UAV stay approximately the same for the duration of the experiment

The first assumption can be reasonably held as long as the vision based object detection algorithm is sufficiently trained to provide accurate information. The validity of this assumption is explored in [4]. The second assumption is reasonable if the dimensions of the target UAV are well known. Therefore, our depth estimation can not be used to detect distance of different drones and is restricted to the Parrot AR.Drone 2.0. The third assumption is more restrictive, and requires that the target UAV maintain a yaw angle close to 0 relative to the pursuer UAV and be approximately level with it. An approach to removing this assumption would be to train the CNN to classify different orientations of the target UAV, and use this information to dynamically assign a visible width and height. However, this requires additional work that is not covered in this thesis and is left for future work.

Within Equation (4.8), taking (x', y') as the pixel coordinates of the corrected image and extracting parameters f_x,f_y,c_x and c_y from projection matrix P, the following expressions are obtained

$$\begin{aligned} x' &= f_x \frac{X}{Z} + c_x \\ y' &= f_y \frac{Y}{Z} + c_y \end{aligned} \tag{4.11}$$

YOLO v2 provides rectangular bounding box information in the form of the vector $(x'_{ul}, y'_{ul}, x'_{lr}, y'_{lr})$, where (x'_{ul}, y'_{ul}) are the digital video frame coordinates of the upper-left corner of the bounding box and (x'_{lr}, y'_{lr}) are the lower-right corner coordinates of the bounding box, all in units of px. The width of the bounding box is $w' = x'_{lr} - x'_{ul}$ and the height of the bounding box is $h' = y'_{lr} - y'_{ul}$.

Let X_{ul} and X_{lr} denote the camera lens fixed frame C coordinates of the points in 3D space corresponding to the upper left and bottom right pixel of the bounding box. Under the assumption that the target UAV has a small yaw angle with respect to the pursuer UAV such that the yaw angle ψ is close to 0 radians, the following expression is found:

$$\begin{aligned} w' &= x'_{lr} - x'_{ul} \\ w' &= f_x \frac{X_{lr}}{Z} + c_x - f_x \frac{X_{ul}}{Z} - c_x \\ w' &= \frac{f_x}{Z}(X_{lr} - X_{ul}) \end{aligned} \tag{4.12}$$

Note that $X_{lr} - X_{ul} = W$ is the true frontal width of the target drone which is known from its geometry. Rearranging Equation (4.12) such that the depth Z_w (determined using only width information) can be solved, the following equation is found:

$$Z_w = f_x \frac{W}{w'} \tag{4.13}$$

Using the same procedure as described above except with height information h', the depth Z_h is determined using only height information.

$$\begin{aligned} h' &= y'_{lr} - y'_{ul} \\ h' &= f_y \frac{Y_{lr}}{Z} + c_y - f_y \frac{Y_{ul}}{Z} - c_y \\ h' &= \frac{f_y}{Z}(Y_{lr} - Y_{ul}) \end{aligned} \tag{4.14}$$

note that $Y_{lr} - Y_{ul} = H$ is the true frontal height of the target drone, which is known. Rearranging Equation (4.14) such that we solve for the depth Z_h, the following expression is found:

$$Z_h = f_y \frac{H}{h'} \tag{4.15}$$

The results from Equation (4.13) and Equation (4.15) should be identical. However in practice, the errors in the bounding box estimated dimensions w' and h' will affect the computed depth, and since the resolution along the x' and y' directions is typically different, these errors will not affect the depth estimation in the same way. Two methods are proposed to mitigate this issue.

The first method is to calculate the geometric mean of Equation (4.13) and Equation (4.15). The geometric mean $\overline{x} = \sqrt{x_1 x_2}$ is used instead of the arithmetic mean $\overline{x} = (x_1 + x_2)/2$ that is used in [4] to deal with the different numerical ranges along the x' and y' directions. The resulting expression for Z_g is

$$Z_g = \sqrt{Z_w Z_h} \tag{4.16}$$

The second method is to completely discard the depth calculated by Equation (4.15) and only consider the depth calculated by Equation (4.13). This method considers the fact that the width direction has twice the number of pixels than the height direction (640 and 360, respectively). The larger number of pixels in the width direction yields a larger pixel density in width which provides more accurate estimations of depth and less sensitivity to random errors. Therefore the errors in w' have a smaller impact on the estimate of the depth Z than the errors in h'. In addition, the width calculation is invariant to the target flying above or below the level of the pursuer.

Both of these methods are investigated and assessed in Section 4.5 and the method that yields the minimum amount of error is used in the vision based pursuit algorithm.

Once the value of Z is obtained, the (X, Y) coordinates can be obtained. Inverting Equation (4.11), we obtain the following expressions

$$\frac{X}{Z} = \frac{x' - c_x}{f_x} \qquad \frac{Y}{Z} = \frac{y' - c_y}{f_y} \tag{4.17}$$

The digital image frame coordinates of the midpoint of the bounding box are obtained using the following expressions

$$x'_m = \frac{x'_{ul} + x'_{lr}}{2} \qquad y'_m = \frac{y'_{ul} + y'_{lr}}{2} \tag{4.18}$$

Combining Equation (4.17) and Equation (4.18), the coordinates of the midpoint of the bounding box can be determined by the following expression

$$X_m = Z\frac{x'_m - c_x}{f_x} \qquad Y_m = Z\frac{y'_m - c_y}{f_y} \tag{4.19}$$

4.5 Bounding Box Vision Detection Algorithm Analysis

The performance and limitations of the bounding box vision detection algorithm need to be understood in order to properly implement it into a pursuit algorithm. An analysis of the performance of YOLO v2 compared to other vision based object detection algorithms was done in [4]. YOLO v2 was found to be faster than other vision-based object detection algorithms while still maintaining good performance. However, YOLO v2 requires a white background to operate properly. Despite YOLO v2's limitations, it was chosen to perform position estimation of an object due to its faster computation time.

The depth estimation of an object from a 2D bounding box was derived and the RMS error of the estimated depth was approximately ± 12.86 cm when the drone translated purely axially relative to the pursuer[4]. The depth estimation in [4] was determined by using a arithmetic mean of the two depth calculations found in Equations (4.13) and (4.15). This method incorporates Z_h which exhibits higher errors than Z_w due to the smaller resolution in height compared to width.

In order to prevent confusion between the global axes and the camera image frame, the convention for global axes is based off of Figure 3.3 such that the global X, Y, and Z axes are referred to as x_w, y_w, and z_w, respectively. The depth of the object is referred to as Z which is the Z-position of the object in the camera image frame C as shown in Figure 4.2.

We wish to test depth estimations based on either a geometric mean or solely the width based depth Z_w. The mathematics behind these two methods were described in Section 4.4. In order to evaluate the performance of these two alternative methods, two tests are proposed.

The first test looks at the performance of YOLO v2 as the drone moves along the global X-axis x_W of the Vicon motion capture system. The global X-axis x_W and Y-axis y_W are based off the global frame as shown in Figure 3.3. The target drone will initially sit at a distance of approximately 2 m away from the pursuer drone along the global X-axis x_W . The drone running YOLO v2 is systematically moved from one relative X-position to another in increments of 0.5 m while holding the global Y-position and yaw orientation roughly constant at 0 m and 0 radians, respectively. This test is meant to provide insight on how the depth estimation performs at various depths.

The second test looks at the performance of YOLO v2 as the drone moves along the global Y-axis y_W of the Vicon motion capture system as shown in Figure 3.3. The pursuer drone will maintain a distance of approximately 2 m away along the global X-axis from the target and move along y_w from one position to another in increments of 0.25 m while holding a constant yaw orientation. This second test is meant to provide insight on the accuracy of the depth estimations when the drone is not located parallel to the object.

These two tests will only provide insight on how depth estimation is affected along x_W and $y_W$and which depth estimation method has better performance. These two tests will not provide an in depth performance analysis of YOLO v2 and are only meant to provide expectations on how the bounding box algorithm will perform in hardware testing.

4.5.1 Depth Estimation Performance along the Global X-Axis

The first test consists of moving the pursuer drone along the global X-axis x_W while maintaining a Y-position of 0 m and a yaw orientation of 0 radians. The pursuer drone begins at a position of 0.5 m away from the target and is moved by increments of 0.5 m away from it. When the pursuer reaches its new relative position, it holds this position for a duration of about 10 seconds. This is meant to provide an adequate number of data samples to average the estimates of depth of the object at that position. The position of the pursuer drone and the target drone is obtained using the Vicon motion capture system. The following table outlines the average depth error and the root mean square error which are calculated using equations described in Appendix G.

Table 4.1: Relative Positions for Moving along the Global X-Axis

Relative Position [m]	Z_w		Z_g	
	e_{Z_w} [m]	RMS [m]	e_{Z_g} [m]	RMS [m]
0.5 m	0.053	0.003	0.184	0.003
1.0 m	0.031	0.003	0.083	0.004
1.5 m	−0.041	0.004	0.099	0.010
2.0 m	−0.009	0.008	0.116	0.007
2.5 m	−0.017	0.013	0.069	0.007
3.0 m	0.033	0.017	0.092	0.013

Where Z_w is the depth determined using the width-based depth estimation method described in Section 4.4, Z_g is the depth determined using the geometric average based depth estimation method, e_{Z_w} is the average relative error for the width-based depth estimation, and e_{Z_g} is the average relative error for the geometric average based depth estimation.

These values were extracted from position data that is shown in plots in Appendix B. The average relative error from a distance of 0.5 m to 3.0 m appears to be roughly 0.04 m for the width based depth estimation while the geometric average based depth estimation had an average relative error of roughly 0.11 m. The results from this test suggest that the width based depth estimation provides a better performance for the pursuit algorithm as long as the pursuer drone remains parallel to the target drone along the X-axis.

4.5.2 Depth Estimation Performance along the Global Y-axis

This test consists of moving the pursuer drone along the global Y-axis y_W while maintaining a X-position of 2 m and a yaw orientation of 0 radians. The drone begins at a position of 0 m in the Y-axis and is moved by increments of 0.25 m to the left and right from the object. When the drone running YOLO v2 is located at its new relative position, it will maintain its position for a duration of 10 seconds. The following table outlines the average relative error for each position and the root mean square error.

Table 4.2: Relative Positions for Movements in the Global Y-axis

Relative Position [m]	Z_w		Z_g	
	e_{Z_w} [m]	RMS [m]	e_{Z_g} [m]	RMS [m]
-1.0 m	-0.018	0.011	0.192	0.017
-0.75 m	-0.013	0.009	0.223	0.005
-0.5 m	-0.013	0.012	0.248	0.017
-0.25 m	0.043	0.008	0.203	0.015
0.0 m	0.016	0.007	0.153	0.017
0.25 m	-0.082	0.007	0.109	0.005
0.5 m	-0.219	0.009	-0.016	0.015
0.75 m	-0.439	0.007	-0.267	0.017
1.0 m	-0.270	0.005	-0.274	0.013

The column headings in Table 4.2 were defined in Section 4.5.1. The results from Table 4.2 demonstrate some issues that arise when using the assumption that the height and width of the 2D bounding box correspond to the physical dimensions of the UAV. From -1.0 m to 0 m, the width based depth estimation provides an accurate estimation of the depth of the object while the geometric average based depth estimation reports large errors. However, when the pursuer drone moves in the positive Y-direction, the relative error using the width based depth estimation drastically increases. Upon observation and investigating the size of the bounding boxes at these locations, the width and height of the bounding boxes did not accurately encompass the target drone in the image frame. This resulted in large errors in the width based depth estimation for the positions of 0.25 m and 1.0 m. At the 0.5 m position, the inaccurate width and height of the bounding box coincidentally managed to provide a small relative error using the geometric depth estimation. When the position moved to 1.0 m, the geometric depth estimation resumed reporting a large error.

4.5.3 Bounding Box Object Detection Performance Summary

In summary, the bounding box object detection algorithm YOLO v2 can be used to determine the relative position of an object using 2D bounding box information, however, it is important to determine how to estimate the depth of this object. Based off of the results in Section 4.5.1 and 4.5.2, the width based depth estimation was found to provide better performance based on experimental testing at varying positions along the X (depth) and Y (lateral) axes.

In order to evaluate whether the performance of the width based depth estimation in the two experiments is adequate, we need to understand the errors in depth that result from having an inaccurate bounding box. The width based depth estimation error per pixel is the error in depth associated with a single pixel deviation from the theoretical width of the bounding box. The depth error per pixel is calculated by determining the width of a bounding box at a specific depth as shown:

$$w_i = \frac{W\ f_x}{Z_i}$$

Where W and f_x are described in Section 4.4,Z_i is the depth of the object in meters, and w_i is the width of the bounding box that would achieve the depth Z_i in pixels. Let us consider the case if the width w_i was a single pixel larger, then the new depth that we would calculate is found to be:

$$Z_{i+1} = \frac{W\ f_x}{w_i + 1}$$

The depth error can be determined by taking the difference between the new depth Z_{i+1} and the original depth Z_i and multiplying the difference by two to determine the depth error for ± 1 pixel as shown in the expression below:

$$\Delta Z = 2\left(Z_i - Z_{i+1}\right)$$

The depth error per pixel increases the further away the object is located and the predicted relationship is shown in Figure 4.5.

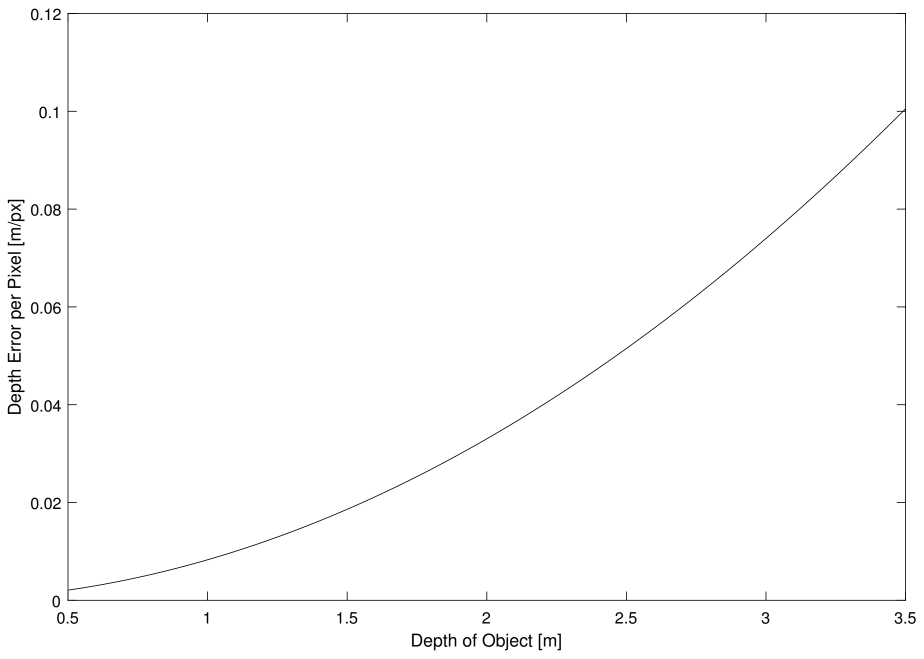


Figure 4.5: Relationship between Depth Error per Pixel and Depth of Object

From Figure 4.5, the depth error per pixel at 2 m, is approximately 0.03 m/px. The average relative error found in Section 4.5.1 was found to be roughly 0.04 m for the width based depth estimation which indicates that the bounding box information was accurate. In addition, the performance of the width based depth estimation along the global Y-axis in Section 4.5.2 was also accurate up to a point. The results in Section 4.5.2 show that if the YOLO v2 is not properly trained, large errors will arise in the depth estimation the further the object is located. This is reinforced by the calculations in Figure 4.5 which indicate that if object is located further away then small errors in the bounding box will result in larger depth estimation errors.

It should be noted that a test along the global Z-axis z_W and yaw rotation was not conducted and is left for future work. However, despite not running tests along the global Z-axis and yaw rotation, certain insights into the performance of the bounding box algorithm can be obtained from the mathematical principles discussed in Section 4.4. The width based depth estimations should not be adversely affected to movement along the Z-axis as long as the target drone remains parallel to the pursuer drone. However, the yaw rotations will have a large impact on the depth estimation. For instance the Parrot AR.Drone 2.0 has a width and depth of 52 cm, and a height of 13 cm. When it faces away from the target drone, its visible width and height are known to be 52 cm by 13 cm. However if it yaws by 45°, its visible width changes to $(52^2 + 52^2)^{1/2} = 73.5$cm, a 41% increase which thus violates the assumptions in Section 4.4 [84]. In light of this, the control algorithm will attempt to keep the pursuer drone maintain the same Z-position and orientation as the target drone.

It is important to note that the tests performed in Chapter 4 involved a static pursuer drone, meaning motion blur did not occur. Motion blur is relevant since the camera used has a low resolution and will be attempting to follow a moving target. An investigation into the effects of motion blur and mitigating its error contribution is left for future work. However, the hardware testing in Chapter 6 involves both the target and pursuer flying, and the full system is shown to perform well under those conditions.

Chapter 5

Control System Implementation and Assessment

5.1 Modular Control System Design

5.1.1 Modular PID Design

The PID motion control is designed as a ROS node that subscribes ROS topics related to the world position and orientation of the controlled drone and the desired world position and yaw angle while publishing input commands to the driver of the drone. This process is outlined in the following control diagram found in Figure 5.1.

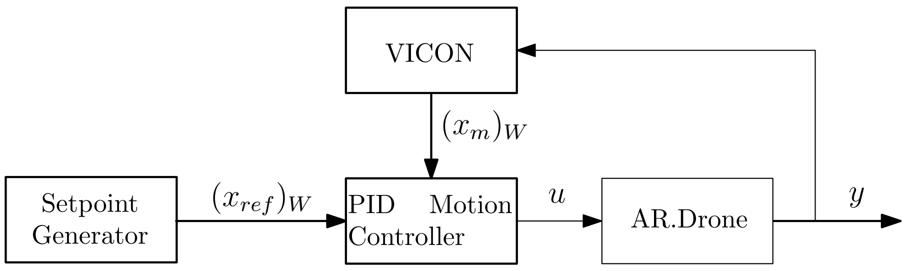


Figure 5.1: Modular PID Control Diagram

The desired state of the drone,$(x_{ref})_W$,is published from a ROS node and consists of the desired world position and yaw of the drone. The desired trajectories that the drone performed for this thesis are outlined in Section 5.2. The current world position of the drone,$(x_m)_W$, is measured by the Vicon motion capture system and uses the ROS package `vicon_bridge` as described in Section 2.2. The desired and current pose are subscribed into a ROS node that performs PID motion control and was written in C++. The mathematical principles of the PID controller were described in Section 3.4. This PID controls the drone along the $(x, y, z)_b$ directions relative to the body frame of the drone as well as the yaw angle ψ. This results in tuning the gains K_p,K_i, and K_d for each direction and the yaw rotation to a total of 12 gains that need to be tuned. The PID motion controller will determine the control inputs, u, that are fed into the driver of the Parrot AR.Drone 2.0. This driver is the ROS package `ardrone_autonomy` as mentioned in Section 2.2.5.2 and is used to communicate with the drone via WiFi. The drone will receive the command inputs and orientate and fly itself to a new pose, y. This new pose is measured by the Vicon motion capture system and fed back into the PID motion controller.

5.1.2 Modular LMPC Design

The LMPC motion control is designed to be a ROS node that subscribes ROS topics related to the world position and orientation of the controlled drone and the desired world position and yaw angle while publishing commands to the driver of the drone. This process is outlined in the following control diagram found in Figure 5.2.

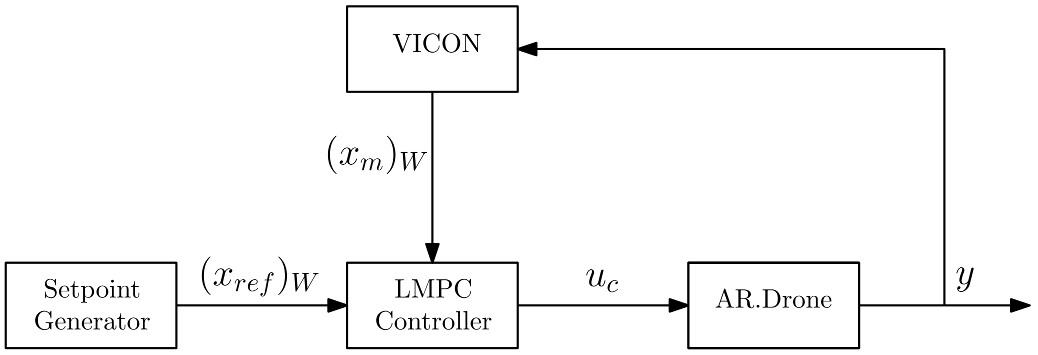


Figure 5.2: Modular LMPC Control Diagram

This control diagram is similar to the control diagram for the PID controller found in Figure 5.1. The major difference between these two controllers is that the LMPC control system uses the state space model of the Parrot AR.Drone 2.0 as described in Chapter 3 and controls both the position and velocity of the drone with respect to the body frame of the drone and the yaw rotation of the drone. Therefore, the desired state of the drone,$(x_{ref})_W$, is not just its pose but also its velocity in each direction. Similarly, the current state of the drone,$(x_m)_W$,is not just the current pose of the drone but also includes the current velocity of the drone. The Vicon motion capture system captures the pose of the drone which means that the velocity is calculated using numerical methods and filtered using a simple real-time low pass filter as described in Section 3.2.3. The LMPC motion controller thus stores the previous pose information in order to calculate the velocity of the drone. The LMPC motion controller will determine the optimum control inputs,u, that are fed into the driver of the AR.Drone. These control inputs are stored in the LMPC motion controller since the LMPC uses the previous control input to determine the optimum control input to achieve the desired setpoint. The drone will receive the command inputs and fly itself to a new pose,y. This new pose is determined by the Vicon motion capture system and fed back into the LMPC motion controller.

5.2 Experimental Procedure

In order to properly assess the performance of both control systems, each will be tested by having the Parrot AR.Drone perform two types of maneuvers. The first test will involve how the drone responds to a reference that is a step input in each direction and the yaw rotation. The second test will evaluate how each control system performs when all 6 degrees of freedom are changing when a drone is performing a Figure-8 maneuver.

5.2.1 Step Responses

In order to evaluate the performance of each of the control systems, each system will be subjected to a sequence of step reference inputs in each direction and the yaw rotation axis. This experiment will assess the performance of the controller in each direction and the yaw rotation, independently. The control system gains are tuned for each direction to obtain the optimum performance. The sequence of steps are described as

$$r_{x,y,z,\psi} = \begin{cases} r_{low}, & 30n \leq t \leq 15 + 30n, \\ r_{mid}, & 15 + 60n \leq t \leq 30 + 60n, \quad n = 0, 1, 2... \\ r_{high}, & 45 + 60n \leq t < 60 + 60n, \end{cases}$$

where t is time in seconds, and n is an integer value. The values for r_{low}, r_{mid}, and r_{high} for each direction and yaw rotation is described in Table 5.1.

Table 5.1: Step Values for Each Direction and Yaw Rotation

Step Direction/Rotation	r_{low}	r_{mid}	r_{high}
Global X-Direction, X_w [m]	-1	0	1
Global Y-Direction, Y_w [m]	-1	0	-1
Global Z-Direction, Z_w [m]	0.75	1.25	1.75
Global Yaw Rotation, ψ_w [rad]	-1.57	0	1.57

5.2.2 Figure 8 Trajectory

The UAV will follow a figure 8 trajectory to assess how each control system handles inputs along each direction and the yaw rotation all at once. These experiments are meant to test the capability of the drone to perform intricate manoeuvres. The gains will be tuned to obtain the optimum performance for each type of control system.

The drone will start from an initial hover of $[(x_0)_w, (y_0)_w, (z_0)_b] = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$ with an attitude of $[\phi_0, \theta_0, \psi_0] = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$ and follow a Figure-8 trajectory in space that is described by the following parametric curves

$$\begin{aligned} (x_{ref})_w(t) &= (l_m/2)\text{sin}(4\pi t/t_c) \\ (y_{ref})_w(t) &= l_M\text{sin}(2\pi t/t_c) \\ (z_{ref})_w(t) &= (h/2)\text{sin}(\pi t/t_c) + z_m \end{aligned}$$

where $(x_{ref})_w(t)$, $(y_{ref})_w(t)$, and $(z_{ref})_w(t)$ are the reference positions in the world frame of reference with respect to time, l_M and l_m are the major and minor diameters of each lobe in metres, respectively, h is the total vertical height of the trajectory, z_m is the hover position of the drone and height that the drone oscillates around, all in metres, and t_c is the time required to complete one full circuit in seconds. The constants associated with the target trajectory are outlined in Table 5.2 below:

Table 5.2: Figure-8 Trajectory Properties

Minor Diameter, l_m [m]	1
Major Diameter, l_M [m]	1.5
Vertical Height, h [m]	0.5
Initial Z-Position, z_m [m]	2
Circuit Time, t_c [s]	30

The reference yaw angle is found using the following expression

$$\psi_{ref}(t) = \text{atan2}[(d/dt)(x_{ref})_w(t), (d/dt)(y_{ref})_b(t)]$$

which points the vehicle in the direction of travel, and where the time derivatives can be determined analytically by differentiating the parametric curves.

5.3 PID Motion Controller Assessment

5.3.1 Step Response using a PID Motion Controller

The Parrot AR.Drone 2.0 performed a step response in each direction and in yaw using a PID motion controller. The system was subjected to a sequence of step reference inputs in one direction or rotation while holding all other directions and/or rotation steady in a user-specified position. Each system was tuned to minimize the rise time, T_{rise}, and fall time, T_{fall}, and reach the reference step input. The PID tuning parameters for the step responses are found in Appendix C. The step responses for each direction and rotation are shown in Figures 5.3, 5.4, 5.5, and 5.6.

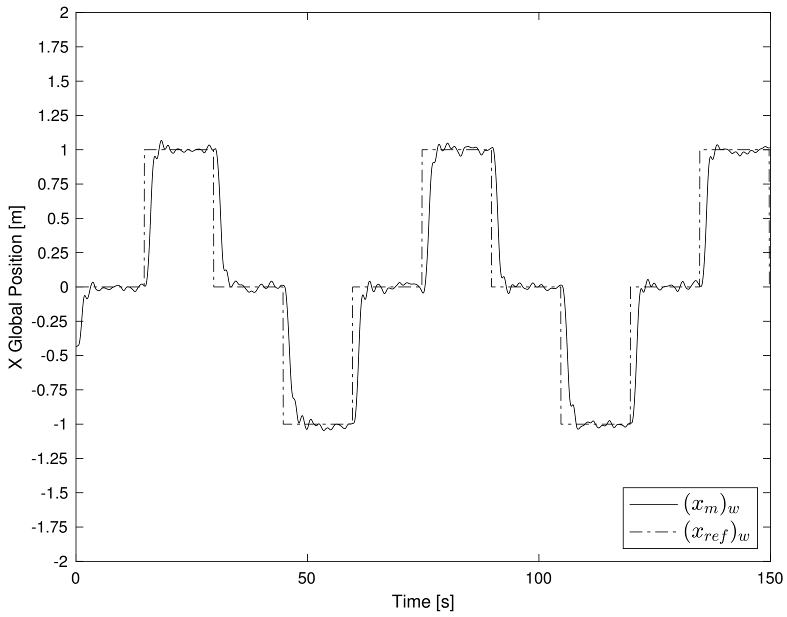


Figure 5.3: Step Response in the X-axis using PID Control

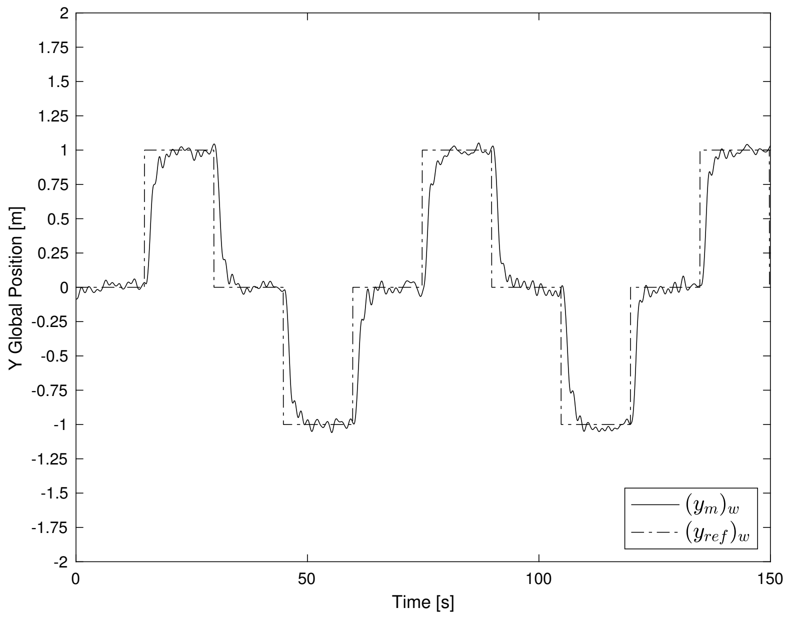


Figure 5.4: Step Response in the Y-axis using PID Control

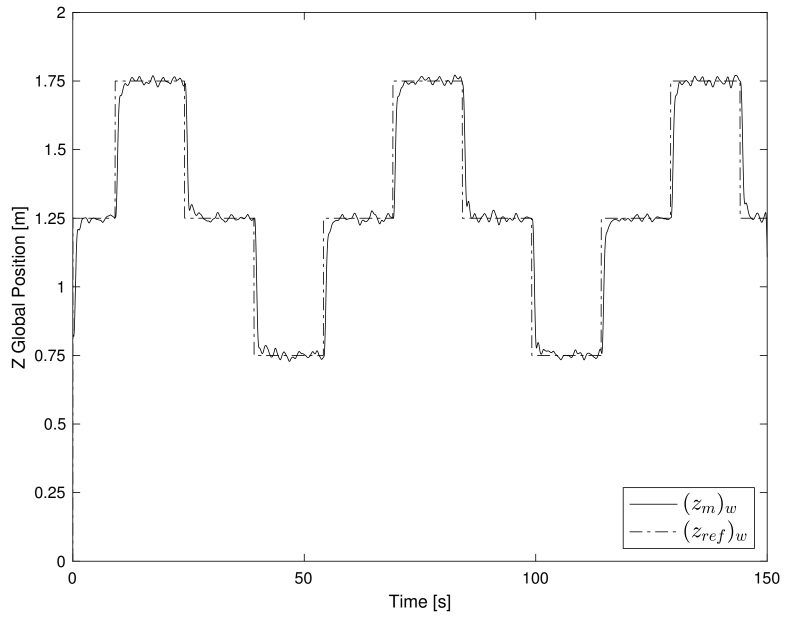


Figure 5.5: Step Response in the Z-axis using PID Control

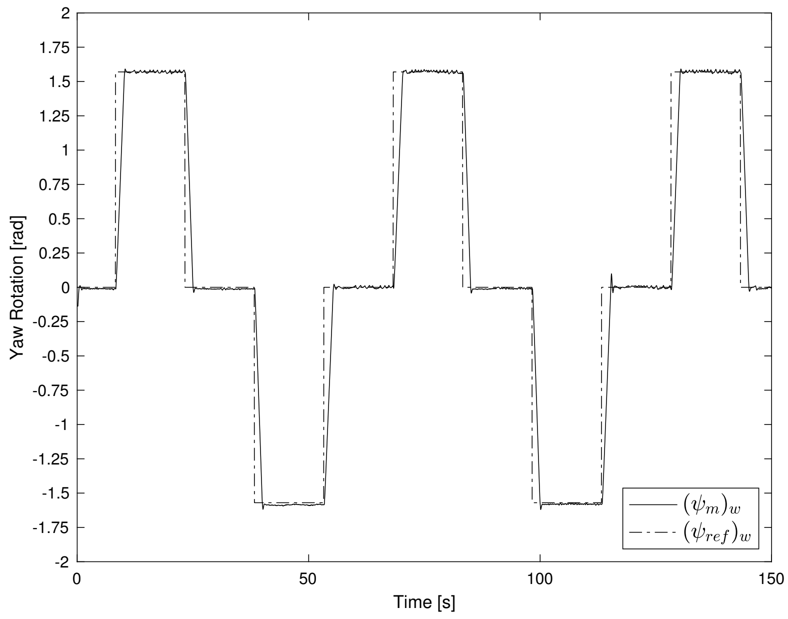


Figure 5.6: Step Response in Yaw using PID Control

In Figures 5.3, 5.4, 5.5, and 5.6, the parameter $((x, y, z, \psi)_m)_w$ denotes the measured global position in either the X-,Y-, or Z-direction while the parameter $((x, y, z, \psi)_{ref})_w$ denotes the desired global position in either X-,Y-, or Z-direction. Figures that show how the system maintained a desired position in the other directions and/or in yaw can be found in Appendix C. The following tables summarizes the performance of the drone in tracking a step response in each direction and in yaw as well as the rise and fall time for each step response.

Table 5.3: Performance Summary for Step Responses in Each Direction and Yaw

Parameter	**Reference Values**	**Averaged Measured Values**
Step Response along the Global X-Axis		
x_{low} [m]	-1	-1.0062 ± 0.0202
x_{mid} [m]	0	0.0030 ± 0.0212
x_{high} [m]	1	-1.0062 ± 0.0202
$(y_m)_w$ [m]	0	-0.0035 ± 0.0643
$(z_m)_w$ [m]	1	1.0063 ± 0.0214
$(\psi_m)_w$ [rad]	0	-0.0653 ± 0.0679
Step Response along the Global Y-Axis		
y_{low} [m]	-1	-1.0044 ± 0.0276
y_{mid} [m]	0	-0.0032 ± 0.0357
y_{high} [m]	1	0.9851 ± 0.0281
$(x_m)_w$ [m]	0	0.0001 ± 0.0227
$(z_m)_w$ [m]	1	0.9824 ± 0.0144
$(\psi_m)_w$ [rad]	0	-0.0400 ± 0.0723
Step Response along the Global Z-Axis		
z_{low} [m]	0.75	0.7550 ± 0.0128
z_{mid} [m]	1.25	1.2501 ± 0.0142
z_{high} [m]	1.75	1.7467 ± 0.0161
$(x_m)_w$ [m]	0	0.0062 ± 0.0337
$(y_m)_w$ [m]	0	0.0058 ± 0.0686
$(\psi_m)_w$ [rad]	0	-0.0186 ± 0.0570
Step Response in Yaw		
ψ_{low} [rad]	-1.571	-1.5834 ± 0.0061
ψ_{mid} [rad]	0	-0.0051 ± 0.0076
psi_{high} [rad]	1.571	1.5706 ± 0.0087
$(x_m)_w$ [m]	0	-0.0038 ± 0.0838
$(y_m)_w$ [m]	0	-0.0074 ± 0.0832
$(z_m)_w$ [m]	1	0.9734 ± 0.0397

Table 5.4: Rise Time and Fall Time for Step Responses in Each Direction and Yaw

Step Response	T_{rise} [s]	T_{fall} [s]
X-Direction	2.6233	3.1499
Y-Direction	3.4136	3.5102
Z-Direction	1.0233	0.9500
Yaw	1.8500	1.4200

Note that $(x, y, z, \psi)_{low,mid,high}$ denotes the step input in a direction or rotation as well as whether the step input is the lowest, middle, or highest step value. The reference values are the desired position or rotation of the system. The measured values include the averaged steady state of the system and the standard deviation from the averaged steady state. The rise time T_{rise} and fall time T_{fall} is from the initial time a system responds to the step input to the first time it takes the system to reach at least 5% of the reference value. These time values are not settling times but rather an indication of how quickly the system attempts to reach steady state.

As seen from Table 5.3, the system was capable of closely tracking step input along all four axes. The average steady state step response standard deviation in the X- and Y- translation axes deviated approximately 0.02 m, and in yaw deviated approximately 0.007 rads. The system was also capable of achieving the reference position for the axes and/or rotation that were not subjected to a step input. However, the deviations in these other directions or rotation were larger compared to the deviations in the steady state step input. This is apparent in the results of the step response in yaw. From Table 5.3, the standard deviation of the steady state in the off-axis directions is around 0.08 m. From Figure 5.6, it is observed that the system reaches a step input in a relatively linear fashion and maintains a steady state close to the reference value. The results from the step response experiment in yaw suggest that the system can achieve an accurate steady state yaw, however, at the expense of the performance of the system maintaining a position in the X- and Y-direction.

With the current tuning setup, the rise time and fall time was the least in the step response in the Z-direction and the most in the Y-direction. This suggests the gains for the Y-direction may need to be changed to reduce the rise and fall time of the step input response.

5.3.2 Figure 8 Trajectory Response

The Parrot AR.Drone 2.0 performed a Figure-8 maneuver using a PID controller. The PID tuning parameters for performing a Figure-8 are shown in Appendix C. The reference trajectory of the Parrot AR.Drone is shown in Section 5.2.2. Figure 5.7 shows a 3D representation of how the Parrot AR.Drone 2.0 followed the reference trajectory, where $(p_{ref})_W$ is the reference position with respect to the world frame and $(p_m)_W$ is the measured position with respect to the world frame.

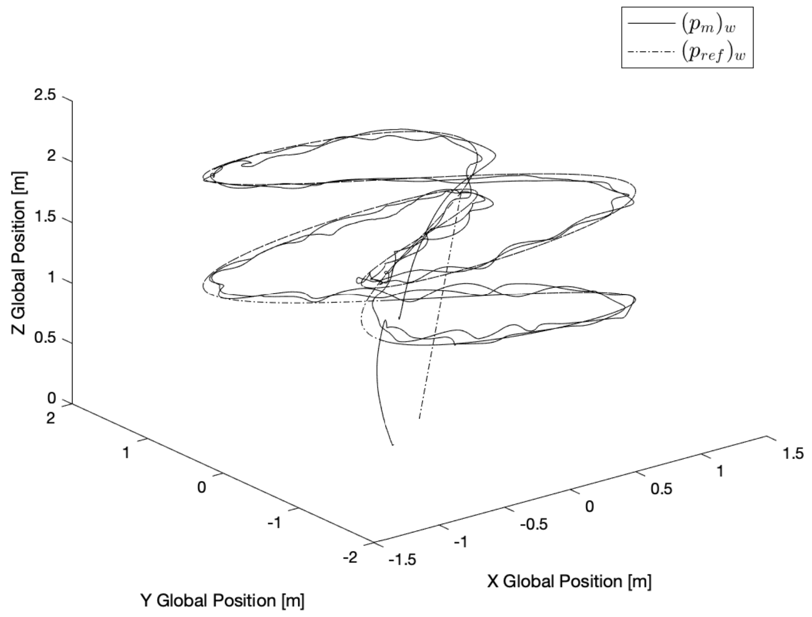


Figure 5.7: 3D plot of the Figure-8 Trajectory of the drone using PID Control

In order to evaluate the PID controller performance, the average tracking error and root mean square error (RMS) is determined for each direction and for the yaw orientation. The average tracking error and root mean square error are calculated using equations described in Appendix G.The resulting values are given in Table 5.5.

Table 5.5: Average Tracking Error and RMS for PID Controlled Figure-8 Trajectory Response

Parameter	**Average Tracking Error**	**RMS Error**
Position		
Global X-Position [m]	-0.0311	0.2022
Global Y-Position [m]	-0.0178	0.1877
Global Z-Position [m]	-0.0029	0.0280
Translational Velocity		
Global X-Velocity [m/s]	-0.0020	0.1498
Global Y-Velocity [m/s]	0.0077	0.1440
Global Z-Velocity [m/s]	0.0004	0.0743
Orientation		
Global Yaw [rad]	-0.0366	0.1887

The velocity of the UAV in each direction was determined using numerical differentiation and filtered using a simple low pass filter described in Section 3.2.3. The position tracking error suggests that the UAV flew on average in the same trajectory as the reference trajectory,however the position and velocity RMS error in the X- and Y-direction were large values of approximately 0.19 m and 0.14 m/s, respectively. As shown in Figure 5.8 and in Figure 5.9, the UAV was unable to follow the reference trajectory when the reference trajectory was ascending to its peak and descending from its peak.

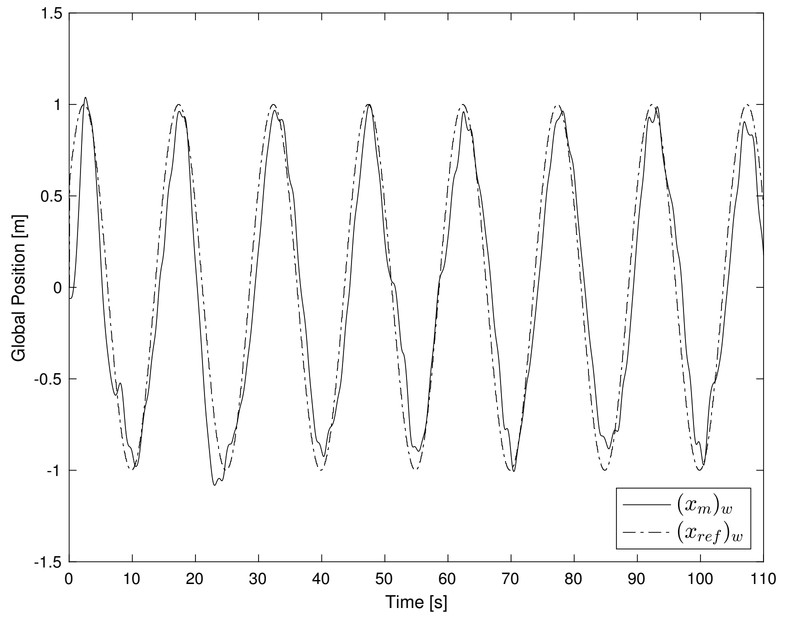


Figure 5.8: PID Figure-8 Trajectory Response in the X-axis

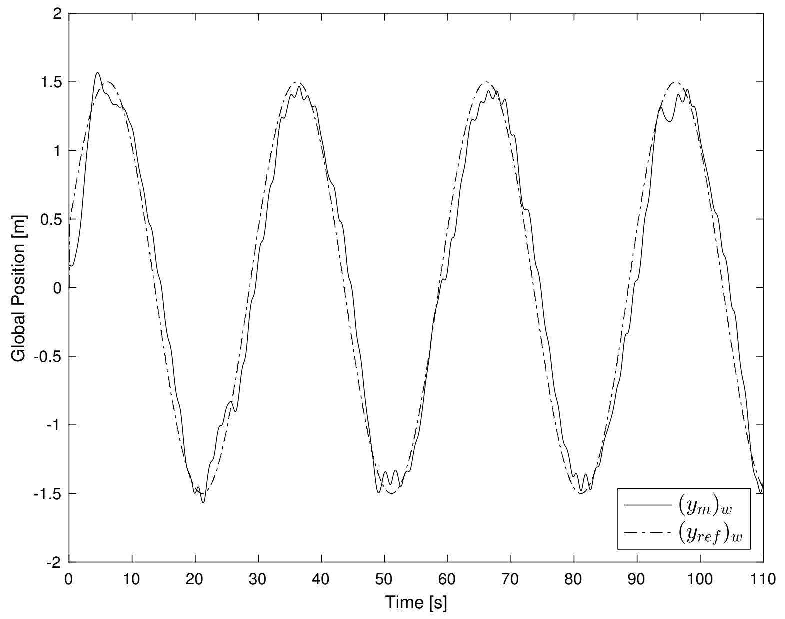


Figure 5.9: PID Figure-8 Trajectory Response in the Y-axis

This point is further reinforced by the fact that the RMS error values in yaw were roughly 0.1887 rads or approximately 11°. This suggest that the system had difficulty turning in the bends of the figure 8 trajectory. The larger RMS values are likely due to the fact that the system is attempting to track a sinusoidal reference. In order to get perfect asymptotic tracking, a model of the sinusoidal function needs to be embedded within the controller. However, this can't be done, since the motion controller does not know beforehand the dimensions and period of the Figure-8 trajectory. There is a limit to the performance that is achievable in this case. Possible ways to reduce the RMS error is either tuning the controller to be more aggressive or incorporating velocity control into the PID controller. The velocity controller would consist of receiving reference velocities from a trajectory generator and having the PID controller attempt to track the reference velocities. This would be an inner-outer loop control with velocity loop as the outer control loop and the position control as the inner control loop.

The position and velocity tracking error and RMS error in the Z-direction was minimal since the reference trajectory for the Z-direction did not require the UAV to perform an aggressive movement in the Z-direction.

5.4 LMPC Motion Controller Assessment

5.4.1 LMPC Step Response Assessment

The Parrot AR.Drone 2.0 performed a step response in each direction and in yaw using a LMPC motion controller. The system was subjected to the same sequence of step reference inputs as the PID motion controller. Similarly to the experiments done for the PID controller,the system was tuned to minimize the rise time, T_{rise}, and fall time, T_{fall}, and track the reference step input. The LMPC tuning parameters for the step responses are found in Appendix D. The step responses for each direction and rotation are shown in Figures 5.3, 5.4, 5.5, and 5.6.

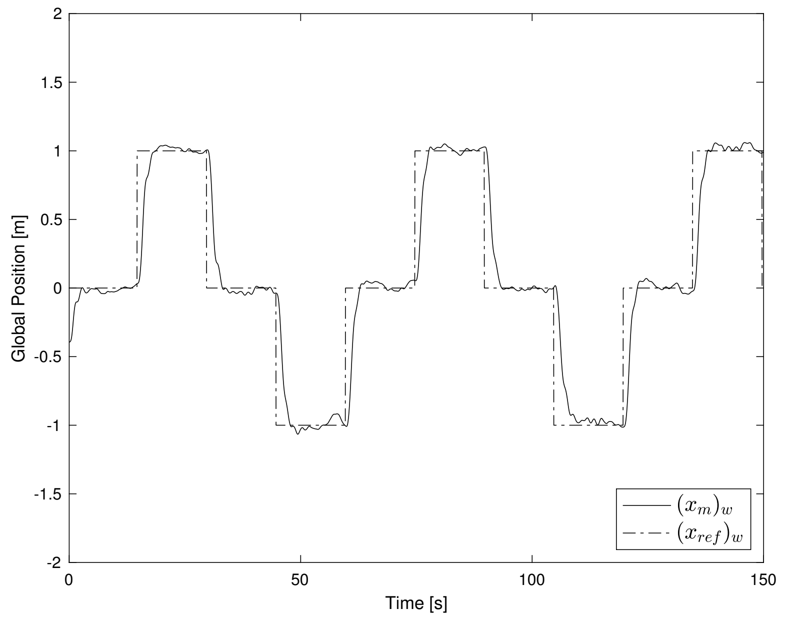


Figure 5.10: Step Response in the X-axis using LMPC Control

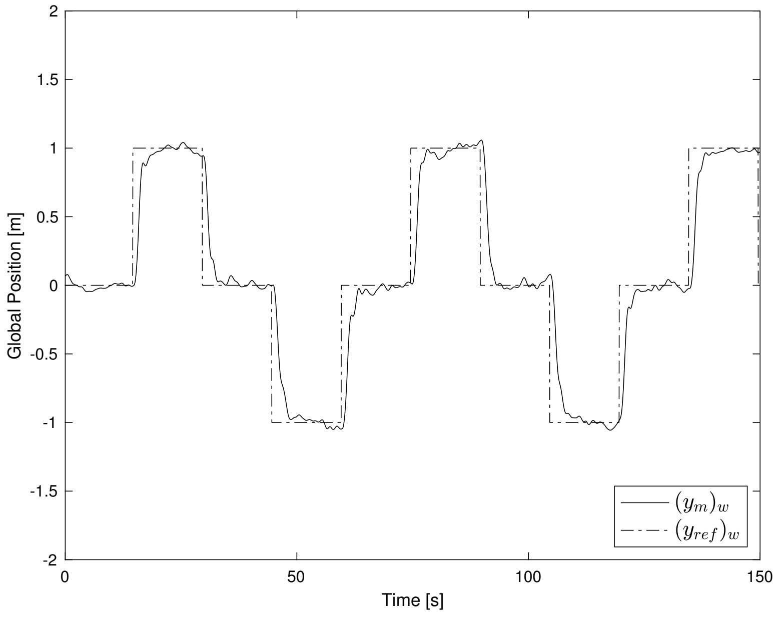


Figure 5.11: Step Response in the Y-axis using LMPC Control

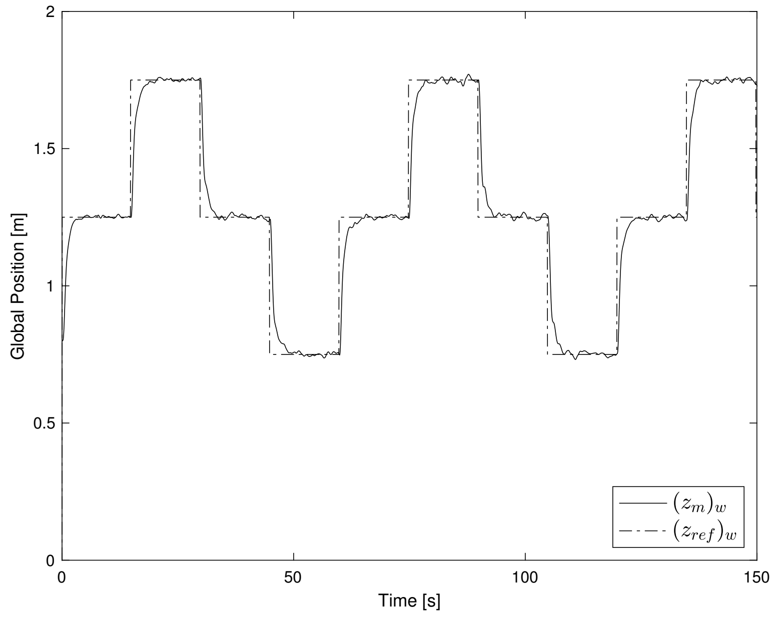


Figure 5.12: Step Response in the Z-axis using LMPC Control

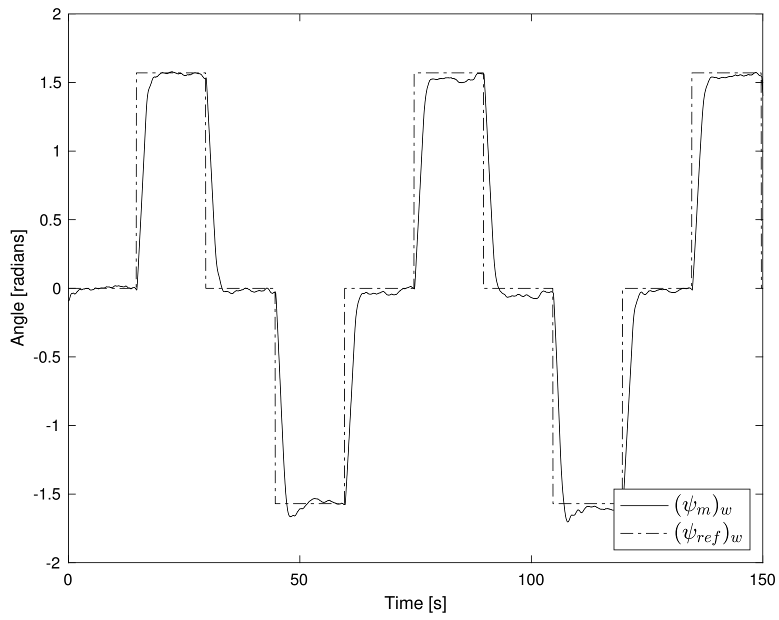


Figure 5.13: Step Response in Yaw using LMPC Control

In Figures 5.10, 5.11, 5.12, and 5.13, the parameter $((x, y, z, \psi)_m)_w$ denotes the measured global position in either the X-,Y-, or Z-direction while the parameter $((x, y, z, \psi)_{ref})_w$ denotes the desired global position in either X-,Y-, or Z-direction. Figures that show how the system maintained a desired position in the other directions and/or in yaw can be found in Appendix D. The following tables summarizes the performance of the drone in tracking a step response in each direction and in yaw as well as the rise and fall time for each step response.

Table 5.6: LMPC Performance Summary for Step Responses in Each Direction and Yaw

Parameter	**Reference Values**	**Averaged Measured Values**
Step Response along the Global X-Axis		
x_{low} [m]	-1	-0.9869 ± 0.0280
x_{mid} [m]	0	0.0009 ± 0.0208
x_{high} [m]	1	1.0146 ± 0.0208
$(y_w)_s$ [m]	0	-0.0111 ± 0.0501
$(z_w)_s$ [m]	1	1.0023 ± 0.0272
$(\psi_w)_s$ [rad]	0	-0.0307 ± 0.0596
Step Response along the Global Y-Axis		
y_{low} [m]	-1	-1.0055 ± 0.0293
y_{mid} [m]	0	-0.0058 ± 0.0255
y_{high} [m]	1	0.9852 ± 0.0249
$(x_w)_s$ [m]	0	0.0122 ± 0.0365
$(z_w)_s$ [m]	1	0.9931 ± 0.0238
$(\psi_w)_s$ [rad]	0	-0.0184 ± 0.0298
Step Response along the Global Z-Axis		
z_{low} [m]	0.75	0.7533 ± 0.0089
z_{mid} [m]	1.25	1.2496 ± 0.0130
z_{high} [m]	1.75	1.7438 ± 0.0158
$(x_w)_s$ [m]	0	0.0069 ± 0.0354
$(y_w)_s$ [m]	0	-0.0079 ± 0.0238
$(\psi_w)_s$ [rad]	0	-0.0211 ± 0.0264
Step Response in Yaw		
ψ_{low} [rad]	-1.571	-1.5962 ± 0.0319
ψ_{mid} [rad]	0	-0.0273 ± 0.0166
psi_{high} [rad]	1.571	1.5438 ± 0.0153
$(x_w)_s$ [m]	0	-0.0016 ± 0.0845
$(y_w)_s$ [m]	0	-0.0131 ± 0.0900
$(z_w)_s$ [m]	1	1.0012 ± 0.0175

Table 5.7: Rise Time and Fall Time for Step Responses in Each Direction and Yaw

Step Response	T_{rise} [s]	T_{fall} [s]
X-Direction	2.9567	3.4800
Y-Direction	3.5600	3.8566
Z-Direction	1.8233	2.3867
Yaw	2.0032	1.7100

The parameters in Figure 5.6 are the same parameters found in Figure 5.3 and are described in Section 5.3.1. As seen from Table 5.6, the system was capable of maintaining a steady state step input close to the reference values with little deviation. The average steady state step input in each of the three axes deviated approximately 0.02 m and in yaw deviated approximately 0.021 rads. These values are close to values found in the PID motion controller experiments. These results suggest that the performance of the PID and LMPC controller are about the same. However, the performance of the PID and LMPC controller can be improved with more extensive tuning.

Similarly to the PD experiments, the system was capable of stabilizing the reference position for the axes and/or rotation that were not subjected to a step input. However, the deviations in these other directions or rotation were larger compared to the deviations in the steady state step input. Both the PID and LMPC motion controller had larger deviations in the X- and Y-directions in the yaw step response. As seen from Table 5.6, the standard deviation of the steady state in the X- and Y-direction is around $0.08 - 0.09$ m. This experiment reinforce the idea that the system can achieve an accurate steady state yaw, however, at the expense of the performance of the system maintaining a position in the X- and Y-direction as mentioned in Section 5.3.1.

5.4.2 Figure 8 Trajectory Response

The Parrot AR.Drone 2.0 performed exactly the same Figure-8 maneuver as described in Section 5.2.2 except with a LMPC controller rather than a PID controller. The LMPC tuning parameters for performing a Figure-8 are shown in Appendix D. Figure 5.14 shows a 3D representation of how the Parrot AR.Drone followed the reference trajectory. Similiarly to Figure 5.7, $(p_{ref})_W$ is the reference position with respect to the world frame and $(p_m)_W$ is the measured position with respect to the world frame.

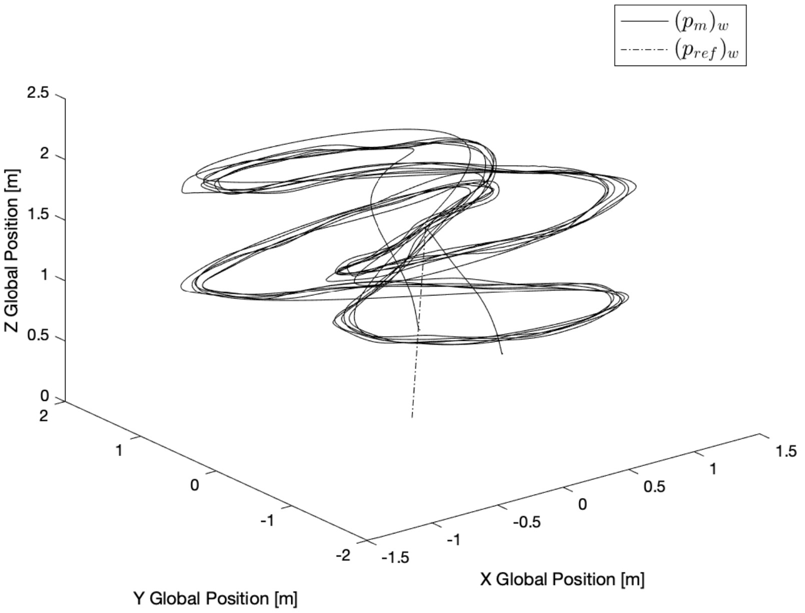


Figure 5.14: 3D plot of the Figure-8 Trajectory of the drone using LMPC Control

In order to evaluate the LMPC controller performance, the average tracking error and root mean square (RMS) error was determined for each direction and for the yaw orientation. Their values are shown in Table 5.8.

Table 5.8: Average Tracking Error and RMS for LMPC Controlled Figure-8 Trajectory Response

Parameter	**Average Tracking Error**	**RMS Error**
Position		
Global X-Position [m]	-0.0060	0.1013
Global Y-Position [m]	0.0071	0.1344
Global Z-Position [m]	-0.0009	0.0421
Translational Velocity		
Global X-Velocity [m/s]	0.0014	0.0881
Global Y-Velocity [m/s]	-0.0007	0.0955
Global Z-Velocity [m/s]	0.0003	0.0421
Orientation Tracking Error		
Global Yaw [rad]	0.0099	0.1465

The position tracking error suggests that the UAV on average tracked the reference trajectory,however the position and velocity RMS error in the X- and Y-direction were large with average values of approximately 0.09 m and 0.07 m/s, respectively.As shown in Figure 5.8 and in Figure 5.9, the UAV experienced difficulty following the reference trajectory when the reference trajectory was ascending to its peak and descending from its peak.

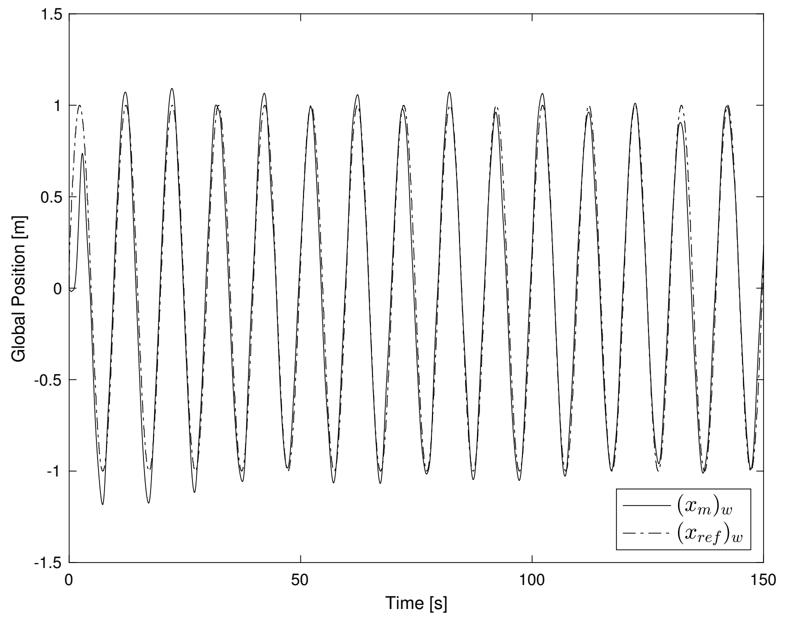


Figure 5.15: LMPC Figure-8 Trajectory Response in the X-axis

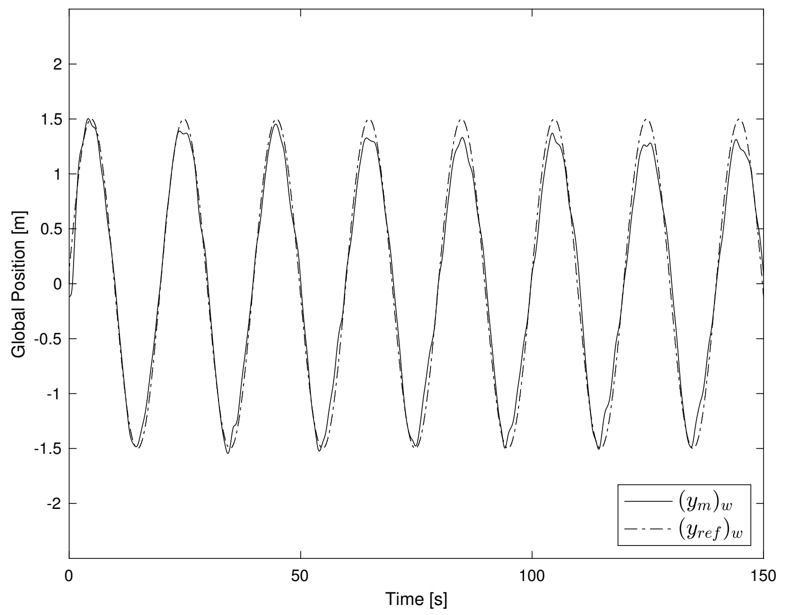


Figure 5.16: LMPC Figure-8 Trajectory Response in the Y-axis

This point is further reinforced by the fact that the RMS error values in yaw were roughly 0.1465 rads or approximately 8.4°. Similarly to the case where a PID motion controller was used, the system had difficulty tracking a sinusoidal reference and a model of the sinusoidal reference needs to be incorporated into the controller in order to perfectly track the Figure-8 maneuver. Overall, when comparing the results from Table 5.5 to Table 5.8, the root mean square error in the X- and Y-direction and in yaw are much lower for the LMPC motion controller compared to the PID motion controller. This suggests that the LMPC motion controller in our case is better than a PID motion controller for following a complex trajectory without any prior knowledge of the trajectory. This result suggests that a LMPC controller would perform well in the case that we are tracking a complex reference such as following a target like a drone. However, the performance of both controllers can be improved with extensive tuning with the possibility of the PID controller outperforming the LMPC controller.

Chapter 6

Vision Based Pursuit Algorithm Implementation and Assessment

This chapter describes the development of two vision based pursuit algorithms that are based on the Vicon motion capture system and the bounding box object detection algorithm from [4]. Section 6.1 describes how the pursuit algorithms were written and implemented into ROS. Section 6.2 details how the pursuit algorithms will be assessed. Section 6.3 details and compares the performance of the pursuit algorithms.

6.1 Pursuit Algorithm Implementation

The pursuit algorithm consists of determining the relative position and orientation of a target drone and moving the pursuer drone to track a global position and orientation relative to this target drone. There are two versions of the pursuit algorithms which are based on the Vicon motion capture system and the bounding box object detection algorithm, respectively. The version of the pursuit algorithm that uses the Vicon motion capture system is capable of determining the relative position and orientation of the target drone. The pursuit algorithm that uses the bounding box object detection algorithm is only capable of determining the relative position of the target drone and can not obtain the orientation of the target drone. The bounding box pursuit algorithm will only work if the assumption that both the pursuer drone and target drone have the same global orientation holds. For both pursuit algorithms, the desired velocity of the pursuer drone for each direction is specified to be 0 m/s. This type of pursuit will emphasize maintaining a hover position once the target drone has reached a hovering state rather than effectively reacting to the movement of a target. This is due to the fact that the data that is retrieved from the bounding box vision algorithm is noisy and can't be used to effectively to replicate the velocity of the target drone. Tracking the reference velocity trajectory of the target drone which can be predicted from a kinematic model could be done as future work.

In the case that we are using the Vicon motion capture system, the following schematic in Figure 6.1 illustrates the situation where we have full information about the position and orientation of the target and pursuer drone. In this schematic, we are attempting to determine the desired global position and orientation of the pursuer drone.

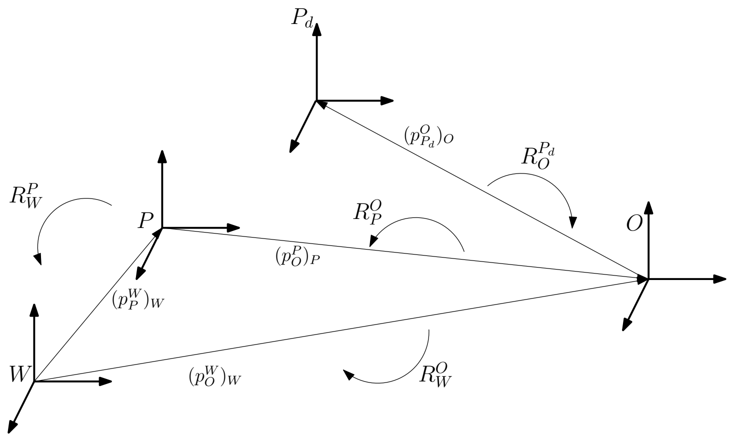


Figure 6.1: Coordinate Frames used in Vicon-based Pursuit Algorithm

In Figure 6.1 W is the world frame, P is the pursuer drone body frame, O is the target drone body frame, and P_d is the desired body frame for the pursuer drone. The vector $(p^W_P)_W$ is the position of the origin of the pursuer body frame relative to the origin of the world frame in the basis of the world frame, the vector $(p^P_O)_P$ is the position of the origin of the target body frame relative to the origin of the pursuer body frame in the basis of the pursuer body frame, the vector $(p^W_O)_W$ is the position of the origin of the target body frame relative to the origin of the world frame in the basis of the world frame, and the vector $(p^O_{P_d})_O$ is the position of the origin of the desired pursuer body frame relative to the origin of the target body frame in the basis of the target body frame. The rotation matrix R^P_W maps the pursuer body frame P to the world frame W, the rotation matrix R^O_W maps the target body frame O to the world frame W, and the rotation matrix R^O_P maps the target body frame O to the pursuer body frame P. The rotation matrix $R^{P_d}_O$ is equivalent to the identity matrix since the desired body frame P_d has the same orientation as the target body frame O.

The orientation of the desired pursuer body frame P_d is obtained from the rotation matrix R^O_W. The pursuer drone is attempting to have the same orientation as the target drone and the orientation of the target drone can be determined from the Vicon motion capture system. Based off the schematic shown in Figure 6.1, there are two methods to determine the desired vector $(p^W_{P_d})_W$ which is the position of the origin of the desired pursuer body frame relative to the origin of the world frame in the basis of the world frame.

The first method is to determine the desired world position based on the position of the pursuer drone. This method is similar to the method used to determine the desired global position for the bounding box pursuit algorithm which uses a camera feed and vision based object detection algorithm to determine the relative position of an object to the camera. The desired global position is determined using mathematical principles described in Chapter 3 and is expressed by the following equation.

$$(p^W_{P_d})_W = (p^W_P)_W + R^P_W(p^P_O)_P + R^P_W R^O_P(p^O_{P_d})_O \tag{6.1}$$

The second method is to directly measure the world position of the target drone and add it to the desired reference position relative to the target body frame. This is a simpler method but is only possible with the Vicon motion capture system, which can directly determine the position of both the pursuer and target drone in the world frame. In this case, the desired global position is determined by using the following expression.

$$(p^W_{P_d})_W = (p^W_O)_W + R^O_W (p^O_{P_d})_O \tag{6.2}$$

The first method was used in order to adapt the Vicon pursuit algorithm to the bounding box pursuit algorithm.

The bounding box pursuit algorithm differs from the Vicon pursuit algorithm by the fact that that bounding box object detection algorithm is unable to determine the orientation of an object. The bounding box pursuit algorithm uses YOLO v2 and the monocular camera feed from the pursuer drone to determine the relative position of a target as described in Chapter 4. The schematic in Figure 6.2 illustrates this idea.

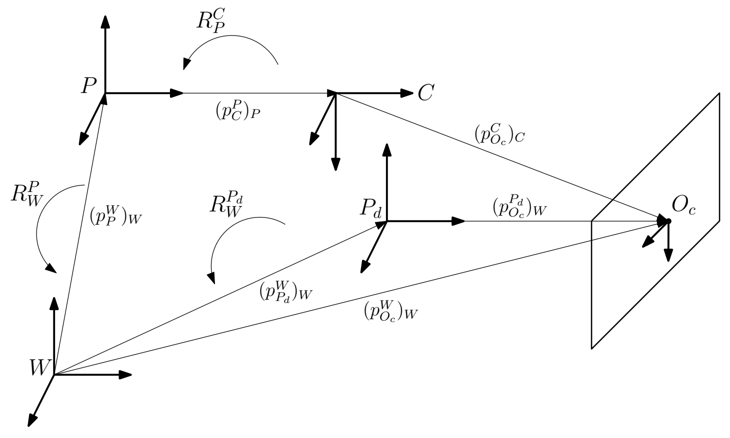


Figure 6.2: Coordinate Frames used in Bounding Box Pursuit Algorithm

Figure 6.2 is similar to Figure 6.1 except that we introduce a new frame C which is the camera frame of the pursuer drone and the frame O no longer exists since the YOLO v2 object detection algorithm does not provide orientation information. Instead we have a point O_c which represents the center of a bounding box which we assume coincides with the center of the target drone. In a similiar fashion as the Vicon pursuit algorithm, we determine the desired position $(p^W_{P_d})_W$ based off the schematic in Figure 6.2, which is expressed by the following equation.

$$(p^W_{P_d})_W = (p^W_P)_W + R^P_W (p^P_C)_P + R^P_W R^C_P (p^C_{O_c})_C - (p^{P_d}_{O_c})_W \tag{6.3}$$

In both pursuit algorithms, the desired position, velocity, and orientation of the pursuer drone is determined and the information is outputted into a ROS topic that is fed into the LMPC motion controller. The control process of both pursuit algorithms are the same except for the sensors that are being used in each system. The control process of the Vicon pursuit algorithm can be generalized by the diagram in Figure 6.3 while the control process of the bounding box pursuit algorithm is shown in Figure 6.4.

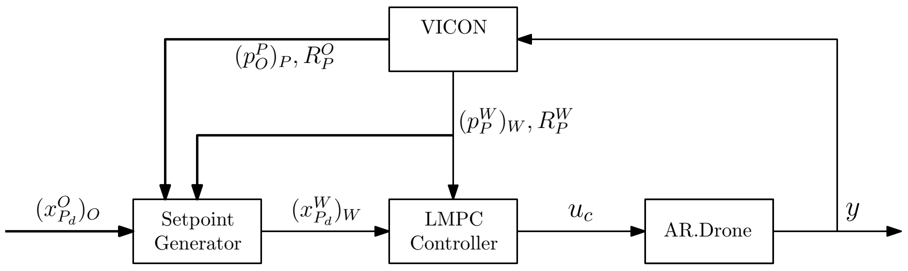


Figure 6.3: Control Process for Vicon Pursuit Algorithm

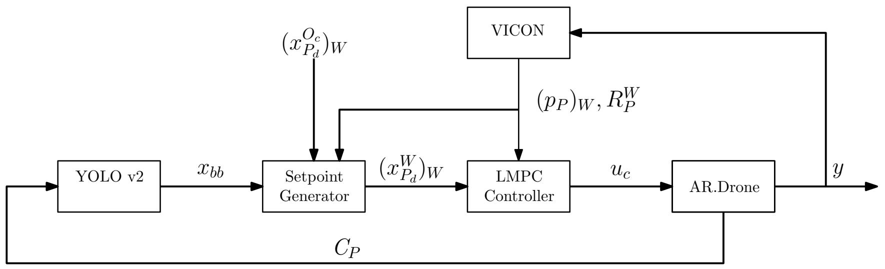


Figure 6.4: Control Process for Bounding Box Pursuit Algorithm

The following parameters are equivalent in Figures 6.3 and 6.4: $(x_{P_d}^W)_W$ is the desired state of the origin of the pursuit drone body frame relative to the origin of the global frame in the basis of the global frame, the parameter u_c is the control input, and parameters y is the state output that consists of the new pose and velocity of the system. In Figure 6.3, the parameter $(x_{P_d}^O)_O$ is the desired state of the origin of the pursuit drone body frame relative to the origin of the target drone body frame in the basis of the target drone body frame.In Figure 6.4, the parameter x_{bb} is a vector that contains the location of the bounding box in the image in units of pixels, C_P is the raw camera feed, the parameter $(x_{P_d}^{O_c})_W$ is the desired offset between the pursuer drone body frame and the point O_c which represents a 3D vector in space that can be adjusted in real -time using a GUI. In Figure 6.3, the parameter $(x_{P_d}^O)_O$ is the desired state between the pursuer and target drone body frame and can be adjusted in real-time using a GUI.

The setpoint generator subscribes to a topic published by the `vicon_bridge` node that is represented by the Vicon block. The `vicon_bridge` node publishes a ROS topic that contains information about the position and orientation of the target drone. The Vicon block is a generalized representation of the `vicon_bridge` ROS package described in Chapter 2 that determines the position of the target and pursuer drone, and the relative position between the two drones for the Vicon pursuit algorithm and determines only the pursuer drone position and orientation for the bounding box pursuit algorithm. The world position of the pursuer drone is published onto a `/tf` topic that is subscribed by the setpoint generator and the LMPC controller. The setpoint generator subscribes to the `/tf` topic that contains the relative position of the pursuer drone relative to the target drone for the Vicon pursuit algorithm. The setpoint generator block is a representation of a C++ code written to calculate and publish the desired state of the pursuer drone in terms of global coordinates. The desired state with respect to the world frame $(x_{P_d}^W)_W$ is published into a custom message and subscribed by the LMPC Controller block. The LMPC Controller block represented C++ code written to determine the command inputs u_c using $(x_{P_d}^W)_W$ obtained from the setpoint generator and $(x_P^W)_W$ obtained from the Vicon motion capture system. The LMPC Controller block stores previous control inputs and positions which are used in the LMPC control algorithm and calculating the velocity of the pursuer drone, respectively. The command inputs are fed into the AR.Drone block that represents the ROS package `ardrone_autonomy` which is a driver for the Parrot AR.Drone 2.0. The AR.Drone block receives command inputs and generates the state y. This state of the pursuer drone is determined by the Vicon motion capture system and fed back into the control system. In the case of the bounding box pursuit algorithm, the raw camera feed C_p which is published by the AR.Drone Block is fed into the YOLO v2 block. The YOLO v2 block publishes the bounding box data x_{bb} which is fed into the setpoint generator block.

6.2 Pursuit Algorithm Assessment

The performance of the pursuit algorithms is assessed by having a pursuer drone follow a target drone that is executing a complex circular trajectory. The target drone will follow the trajectory using a LMPC motion controller whose tuning parameters are shown in Appendix E. The pursuer drone will pursue the drone at a desired relative position and orientation. The pursuer drone will attempt to maintain the same attitude as the target drone and a relative position of $(p_{P_D}^O)_O = \begin{bmatrix} 2 & 0 & 0 \end{bmatrix}$ m with respect to the target drone body frame. This experiment is meant to test the capability of the pursuer drone to follow a target drone performing arbitrary manoeuvres. The gains will be tuned to obtain the optimum performance for each control system. The performance of both pursuit algorithms in following the target drone will be compared and analyzed. Note that each pursuit algorithm does not include any prior knowledge about the trajectory of the target drone. The pursuer drone will maintain a yaw orientation of 0 radians relative to the global frame and with no kinematic model of the target drone will attempt to track a velocity of 0 m/s in each direction.

The target drone will fly a circular trajectory described by the following parametric curves and associated velocities:

$$\begin{aligned} x_{ref}(t) &= r\cos(2\pi t/t_c) + x_m & \qquad (v_x)_{ref}(t) &= -(2\pi r/t_c)\sin(2\pi t/t_c) \\ y_{ref}(t) &= 2r\sin(2\pi t/t_c) + y_m & \qquad (v_y)_{ref}(t) &= (4\pi r/t_c)\cos(2\pi t/t_c) \\ z_{ref}(t) &= r\sin(2\pi t/t_c) + z_m & \qquad (v_z)_{ref}(t) &= (2\pi r/t_c)\cos(2\pi t/t_c) \end{aligned}$$

where $x_{ref}(t)$,$y_{ref}(t)$, and $z_{ref}(t)$ are the reference positions for the target drone with respect to the world frame in meters [m], $(v_x)_{ref}(t)$, $(v_y)_{ref}(t)$, and $(v_z)_{ref}(t)$ are the reference velocities for the target drone with respect to the world frame in meters per second [m/s], r is the radius of the circle that moves in the X-, Y-, and Z-direction, x_m, y_m, z_m is the location of the center of the circle and the height that the drone will rotate about in meters, and t_c is the time required to complete one full circuit in seconds. The constants associated with the target trajectory and velocity are outlined in Table 6.1 below:

Table 6.1: Circular Parametric Curve Properties

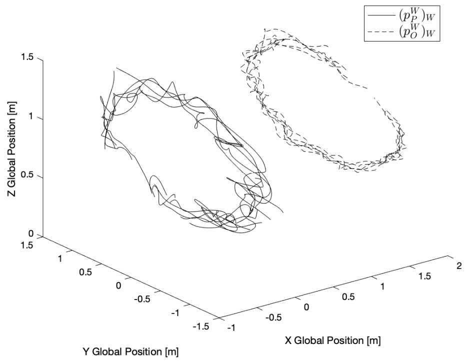
Initial X-Position, x_m [m]	1
Initial Y-Position, y_m [m]	0
Initial Z-Position, z_m [m]	1
Radius, r [m]	1
Circuit Time, t_c [s]	20

The target drone will attempt to maintain an attitude of approximately

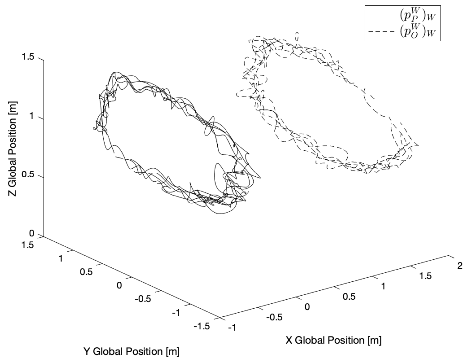
$$[(\phi_W)_{ref}, (\theta_W)_{ref}, (\psi_W)_{ref}] = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

6.3 Pursuit Algorithm Performance

The pursuer drone followed a target drone performing the circular trajectory described in Section 6.2 using both the Vicon pursuit algorithm and the bounding box pursuit algorithm. The tuning parameters for the LMPC motion controllers used in both pursuit algorithms for the target and pursuer drones are shown in Appendix E. The results are shown by the 3D plots in Figure 6.5.



(a) Bounding Box Pursuit Algorithm



(b) Vicon Pursuit Algorithm

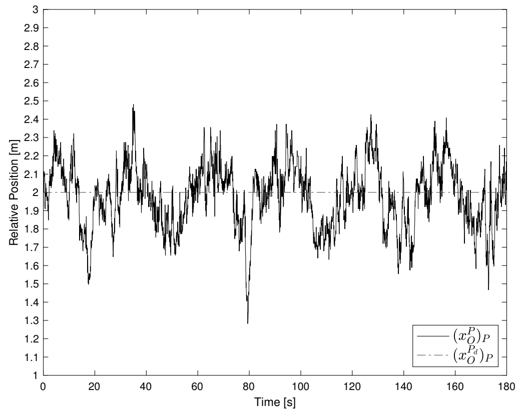
Figure 6.5: 3D plot of Vision Based Pursuit of a Circular Trajectory

where $(p_P^W)_W$ is the position of the pursuer drone with respect to the world frame and $(p_O^W)_W$ is the position of the target drone with respect to the world frame. The performance of the pursuit algorithms is evaluated by determining the relative tracking error and RMS error in the relative position of the pursuer drone to the target drone. The average tracking error and RMS error for each direction are shown in Table 6.2

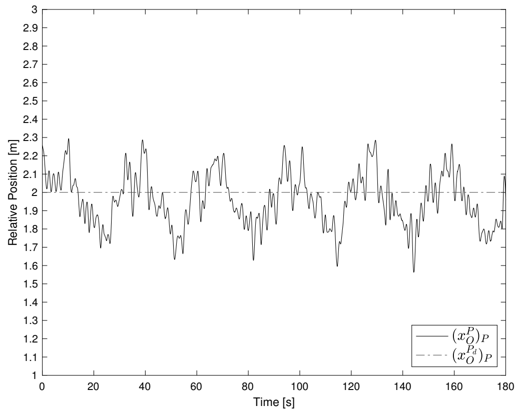
Table 6.2: Average Tracking Errors and RMS for the Pursuit Algorithms Flight Performance

Relative Position [m]	**Vicon**		**Bounding Box**	
	e_v [m]	RMS [m]	e_{bb} [m]	RMS [m]
X-Position [m]	-0.0162	0.1792	-0.0477	0.1456
Y-Position [m]	0.0224	0.2380	-0.0416	0.2421
Z-Position [m]	0.0042	0.0899	0.0096	0.1035

where e_v is the average tracking error of the relative position between the target and pursuer drone when using the Vicon pursuit algorithm and e_{bb} is the average tracking error of the relative position between the target and pursuer drone when using the bounding box pursuit algorithm. On average the pursuer drone managed to maintained a desired relative distance away from the target drone and track its motion using both pursuit algorithms. There exists large deviations in the tracking error in all three directions in each pursuit algorithms. These deviations possibly arise from computational time delays in the closed-loop system or gain tuning. This insight is derived from the fact that the behaviour of the relative position in all three directions is oscillatory as shown in Figures 6.6, 6.7 and 6.8.

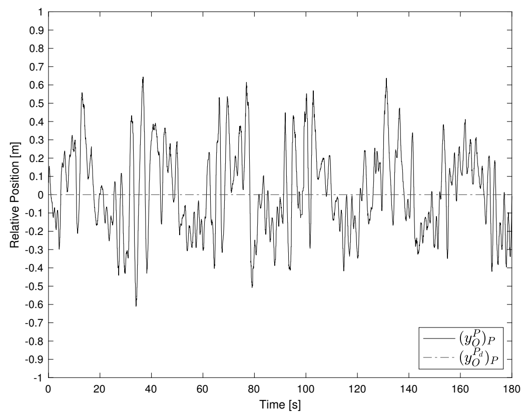


(a) Bounding Box Pursuit Algorithm

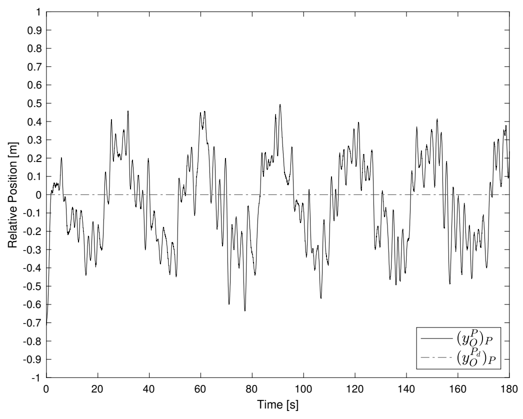


(b) Vicon Pursuit Algorithm

Figure 6.6: Relative X-Position Performance

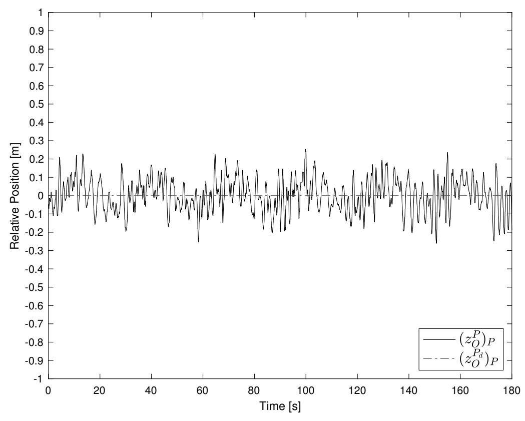


(a) Bounding Box Pursuit Algorithm

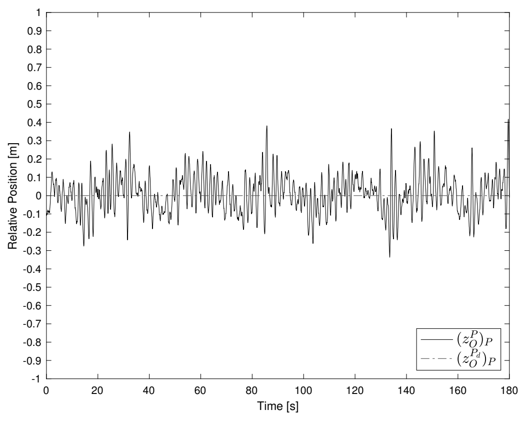


(b) Vicon Pursuit Algorithm

Figure 6.7: Relative Y-Position Performance



(a) Bounding Box Pursuit Algorithm



(b) Vicon Pursuit Algorithm

Figure 6.8: Relative X-Position Performance

These figures suggest that for both pursuit algorithms the gains associated with each direction need to be tuned such that the pursuer drone acts more aggressively in the Y-direction while maintaining stability. The oscillatory response seen in the figures can be dampened by adding a kinematic model of the target drone. An investigation into determining a motion model for the target drone is left for future work.

The performance of the pursuit algorithm system is not optimum due to issues that arise when using linear state space model that does not take into consideration the inter-axis coupling, as well as disturbances acting on the system. A disturbance that was apparent in experiments were wakes and gusts emitted by the target drone's rotary blades disturbing the pursuer drone and affecting its performance. As the target drone increased in height, the pursuer drone located temporally below it was affected by the gusts and wakes. This result suggested that the performance of the pursuit algorithms is related to the proximity of the target drone to the pursuer drone. The closer the pursuer drone is to the target drone, the more pronounced are the disturbance effects from the gusts and wakes. The LMPC motion controller can be designed to be robust to the disturbances, however, at the cost of tracking performance.

In spite of the issues that may contribute to a decline in performance, both pursuit algorithms were capable of having a pursuer drone accurately follow a target drone. The performance of both pursuit algorithms were similar suggesting that given optimal conditions the bounding box pursuit algorithm is on par with the Vicon pursuit algorithm. Both pursuit algorithms experienced similar issues which indicates that issues present in the system are not due to the vision algorithm, but rather an issue in the LMPC controller which can be addressed in future work. Further details about the performance of both pursuit algorithms is outlined in Appendix E and additional information about the evaluating the bounding box pursuit algorithm is outlined in Appendix F .

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis was successful in developing a UAV pursuit algorithm that employs a closed-loop control strategy combined with a vision-based object detection system. The vision based pursuit algorithm allows a pursuer UAV to follow a target UAV at a set standoff distance. The vision based pursuit algorithm incorporated either a PID or LMPC closed-loop motion control and a vision based object detection algorithm called YOLO v2.

The performance of the vision based object detection developed in [4] was assessed. An alternative approach to calculate depth from a monocular camera was devised and successfully implemented in order to improve the tracking performance of the control system.

LMPC and PID motion controllers were developed and implemented on the Parrot AR.Drone 2.0 to perform a variety of maneuvers such as a Figure-8 and a circle trajectory. An uncoupled linear state model of the Parrot AR.Drone 2.0 was devised for the LMPC motion controller. The LMPC motion controller was chosen to be used in the pursuit algorithms due to having less gains to tune, achieving a better performance in following a complex trajectory, and robustness.

A Vicon based pursuit algorithm was implemented and successfully demonstrated a pursuer drone following a target drone. With the success of the Vicon based pursuit algorithm, a vision bounding box pursuit algorithm was developed and successfully implemented. Both pursuit algorithms succeeded in having a pursuer drone follow a target drone performing a complex trajectory. The performance of the bounding box pursuit algorithm under optimal conditions had similar performance to the Vicon pursuit algorithm. This suggests that vision based pursuit relying on a computer vision algorithm like YOLO v2 can result in accurate performance.

7.2 Limitations of Work

While the results of this thesis are promising, some limitations were found in both the vision-based object detection algorithm and the UAV LMPC motion controller:

- The bounding box object detection algorithm only determines a 2D bounding box that describes the location of the object in the image frame. This limits the type of motion control that can be developed since the orientation of the object can not be determined. The bounding box pursuit algorithm can only react and move to a specified distance and can't track the orientation of the object. This causes issues because as the target drone rotates, its projected visible area changes and affects the depth estimation calculation.

- The performance of the Vicon and bounding box pursuit algorithm is predicated on how well the LMPC motion controller was tuned. The current iteration of the LMPC motion controller does not have the ability for real time gain tuning, which makes finding gains a tedious and long process.
- The bounding box pursuit algorithm only works well if the target is over a white background, since the YOLO v2 object detection algorithm has issues tracking an object without a white background and is subject to false positive detections.
- The LMPC motion controller is capable of tracking the position and yaw orientation of the drone as well as the velocity. In the case that we are pursuing a target, we set the desired velocity to 0 m/s since we do not know the motion of the target. By determining a dynamics model of the target, the performance of the pursuit algorithms could be improved.
- The pursuit algorithms contain time delays due to calculations associated with the LMPC motion controller. Modeling this time delay within the LMPC motion controller could improve the performance of the pursuit algorithms.

7.3 Future Work

Further work needs to be performed in order to iterate on the results that are presented in this thesis. Most of the points that are discussed in section will focus on addressing the limitations listed in Section 7.2.

The main work that needs to be performed is determining a method to obtain 3D position and orientation of an object from vision data. This may consist of developing a sophisticated vision based object detection algorithm that uses a monocular camera feed or using alternative hardware such as stereo vision or Lidar. There is currently research being done to determine 3D pose of an object from a 2D bounding box [88]. In addition, stereo vision could be used to determine the 3D pose of an object. In addition, future work can be done to train a network to work around background noise and track a specific object despite the presence of additional objects. If this future work is completed, this can lead to having a pursuer drone effectively follow a target drone in 3D space without the need of a white background.

The LMPC motion controller needs to be adjusted to take into consideration time delays that may arise, and incorporating a motion model of the target drone. An Extended Kalman filter needs to be incorporated to filter noise present in position and orientation information from the Vicon motion capture system, bounding box information from YOLO v2 and pursuer velocity calculated using numerical differentiation. Additional time needs to be invested in tuning the LMPC controller used for the bounding box pursuit algorithm.

Bibliography

[1] VICON. Vero. www.vicon.com/products/camera-systems/vero, 2019.

[2] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[3] Pablo Martinez. Vision-based algorithms for uav mimicking control system. Master's thesis, University of Alberta, 2017.

[4] Bengsheng Wei. Evaluation of object detection algorithms for uav tracking. Master's thesis, University of Alberta, 2019.

[5] Yigit Menguc, Yong-Lae Park, Ernesto Martinez-Villalpando, Patrick Aubin, Miriam Zisook, Leia Stirling, Robert J. Wood, and Conor J. Walsh. Soft wearable motion sensing suit for lower limb biomechanics measurements. In *Proceeding of the IEEE International Conference on Robotics and Automation (ICRA'13)*, pages 5289–5296, May 2013.

[6] Alberto Leardini, Maria Grazia Benedettiand L Berti, D Bettinelli, R Nativo, and Sandro Giannini. Rear-foot, mid-foot and fore-foot motion during the stance phase of gait. *Gait and posture*, 25:453–62, 04 2007.

[7] Asha Kapur, Ajay Kapur, Naznin Virji-Babul, George Tzanetakis, and Peter F. Driessen. Gesture-based affective computing on motion capture data. In *ACII*, 2005.

[8] J. T. Isaacs, F. Quitin, L. R. Garca Carrillo, U. Madhow, and J. P. Hespanha. Quadrotor control for rf source localization and tracking. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 244–252, May 2014.

[9] Shaima Al Habsi. Integration of a vicon camera system for indoor flight of a parrot ar drone. In *2015 10th International Symposium on Mechatronics and its Applications (ISMA)*, pages 1–6, Dec 2015.

[10] L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing. In *2011 IEEE International Conference on Robotics and Automation*, pages 2472–2477, May 2011.

[11] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys. Vision-based autonomous mapping and exploration using

a quadrotor mav. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4557–4564, Oct 2012.

[12] Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In *2011 IEEE International Conference on Robotics and Automation*, pages 2992–2997. IEEE, 2011.

[13] Paul Pounds, Robert Mahony, and Peter Corke. Modelling and control of a large quadrotor robot. *Control Engineering Practice*, 18(7):691–699, 2010.

[14] Paul Pounds, Robert Mahony, and Peter Corke. Modelling and control of a quad-rotor robot. In *Proceedings Australasian Conference on Robotics and Automation 2006*. Australian Robotics and Automation Association Inc., 2006.

[15] Tarek Hamel, Robert Mahony, Rogelio Lozano, and James Ostrowski. Dynamic modelling and configuration stabilization for an x4-flyer. *IFAC Proceedings Volumes*, 35(1):217–222, 2002.

[16] Samir Bouabdallah, Andre Noth, and Roland Siegwart. Pid vs lq control techniques applied to an indoor micro quadrotor. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2451–2456. IEEE, 2004.

[17] Kaan T Oner, Ertugrul Cetinsoy, Mustafa Unel, Mahmut F Aksit, Ilyas Kandemir, and Kayhan Gulez. Dynamic model and control of a new quadrotor unmanned aerial vehicle with tilt-wing mechanism. *World Academy of Science, Engineering and Technology*, 45, 2008.

[18] Ian D Cowling, James F Whidborne, and Alastair K Cooke. Optimal trajectory planning and lqr control for a quadrotor uav. In *International Conference on Control*, 2006.

[19] G. Hoffmann, D. G. Rajnarayan, S. L. Waslander, D. Dostal, J. S. Jang, and C. J. Tomlin. The stanford testbed of autonomous rotorcraft for multi agent control (starmac). In *The 23rd Digital Avionics Systems Conference (IEEE Cat. No.04CH37576)*, volume 2, pages 12.E.4–121, Oct 2004.

[20] Gabriel Hoffmann, Haomiao Huang, Steven Waslander, and Claire Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *AIAA guidance, navigation and control conference and exhibit*, page 6461, 2007.

[21] Martin Saska, Tomas Baca, Justin Thomas, Jan Chudoba, Libor Preucil, Tomas Krajnik, Jan Faigl, Giuseppe Loianno, and Vijay Kumar. System for deployment of groups of unmanned micro aerial vehicles in gps-denied environments using onboard visual relative localization. *Autonomous Robots*, 41(4):919–944, 2017.

[22] Yash Mulgaonkar, Gareth Cross, and Vijay Kumar. Design of small, safe and robust quadrotor swarms. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 2208–2215. IEEE, 2015.

[23] Giuseppe Loianno, Yash Mulgaonkar, Chris Brunner, Dheeraj Ahuja, Arvind Ramanandan, Murali Chari, Serafin Diaz, and Vijay Kumar. A swarm of flying smartphones. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1681–1688. IEEE, 2016.

[24] Shupeng Lai, Kangli Wang, Hailong Qin, Jin Q Cui, and Ben M Chen. A robust online path planning approach in cluttered environments for micro rotorcraft drones. *Control Theory and Technology*, 14(1):83–96, 2016.

[25] Sergei Lupashin, Markus Hehn, Mark W Mueller, Angela P Schoellig, Michael Sherback, and Raffaello D'Andrea. A platform for aerial robotics research and demonstration: The flying machine arena. *Mechatronics*, 24(1):41–54, 2014.

[26] Myungsoo Jun and Raffaello DAndrea. Path planning for unmanned aerial vehicles in uncertain and adversarial environments. In *Cooperative control: models, applications and algorithms*, pages 95–110. Springer, 2003.

[27] Stefania Tonetti, Markus Hehn, Sergei Lupashin, and Raffaello D'Andrea. Distributed control of antenna array with formation of uavs. *IFAC Proceedings Volumes*, 44(1):7848–7853, 2011.

[28] C. L. Castillo, W. Moreno, and K. P. Valavanis. Unmanned helicopter waypoint trajectory tracking using model predictive control. In *2007 Mediterranean Conference on Control Automation*, pages 1–8, June 2007.

[29] Hoam Chung and S Sastry. Autonomous helicopter formation using model predictive control. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 6066, 2006.

[30] H Jin Kim, David H Shim, and Shankar Sastry. Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 5, pages 3576–3581. IEEE, 2002.

[31] E. A. Wan and A. A. Bogdanov. Model predictive neural control with applications to a 6 dof helicopter model. In *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, volume 1, pages 488–493 vol.1, June 2001.

[32] H Jin Kim and David H Shim. A flight control system for aerial robots: algorithms and experiments. *Control engineering practice*, 11(12):1389–1400, 2003.

[33] J. Dentler, S. Kannan, M. A. O. Mendez, and H. Voos. A real-time model predictive position control with collision avoidance for commercial low-cost quadrotors. In *2016 IEEE Conference on Control Applications (CCA)*, pages 519–525, Sep. 2016.

[34] Jaeseung Byun, Karan P Jain, Siddharth H Nair, Haoyun Xu, and Jiaming Zha. Predictive control for chasing a ground vehicle using a uav. *arXiv preprint arXiv:1905.09396*, 2019.

[35] Kostas Alexis, George Nikolakopoulos, and Anthony Tzes. On trajectory tracking model predictive control of an unmanned quadrotor helicopter subject to aerodynamic disturbances. *Asian Journal of Control*, 16(1):209–224, 2014.

[36] G. Garimella and M. Kobilarov. Towards model-predictive control for aerial pick-and-place. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4692–4697, May 2015.

[37] Jan Dentler, Somasundar Kannan, Miguel Angel Olivares Mendez, and Holger Voos. A modularization approach for nonlinear model predictive control of distributed fast systems. In *2016 24th mediterranean conference on control and automation (MED)*, pages 292–297. IEEE, 2016.

[38] Mina Kamel, Michael Burri, and Roland Siegwart. Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles. *IFAC-PapersOnLine*, 50(1):3463–3469, 2017.

[39] E.F. Camacho and C. Bordons. *Model Predictive Control*. Springer, 2004.

[40] Jay H. Lee. Model predictive control: Review of the three decades of development. *International Journal of Control, Automation and Systems*, 9(3):415, Jun 2011.

[41] Cunjia Liu, Wen-Hua Chen, and John Andrews. Piecewise constant model predictive control for autonomous helicopters. *Robotics and Autonomous Systems*, 59(7-8):571–579, 2011.

[42] S. Saripalli, J. F. Montgomery, and G. S. Sukhatme. Vision-based autonomous landing of an unmanned aerial vehicle. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 3, pages 2799–2804 vol.3, May 2002.

[43] Omid Shakernia, Yi Ma, T John Koo, and Shankar Sastry. Landing an unmanned air vehicle: Vision based motion estimation and nonlinear control. *Asian journal of control*, 1(3):128–145, 1999.

[44] M. Blosch, S. Weiss, D. Scaramuzza, and R. Siegwart. Vision based mav navigation in unknown and unstructured environments. In *2010 IEEE International Conference on Robotics and Automation*, pages 21–28, May 2010.

[45] T. Templeton, D. H. Shim, C. Geyer, and S. S. Sastry. Autonomous vision-based landing and terrain mapping using an mpc-controlled unmanned rotorcraft. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1349–1356, April 2007.

[46] A. Lawrence and A. Turchina. Tag tracking in ros. https://github.com /ablarry91/ros-tag-tracking, 2014.

[47] C. Minhua and G. Jiangtao. Mobile robotics lab spring 2015 automatic landing on a moving target. https://robotics.shanghaitech.edu.cn /sites/default/files/files/robot-land-project20report.pdf, 2015.

[48] Anze Rezelj and Danijel Skocaj. Autonomous charging of a quadcopter on a mobile platform. In *Austrian Robotics Workshop*, pages 65–66, 2015.

[49] Shannon Hood. Bird's eye view: Cooperative exploration by ugv and uav. Master's thesis, University of South Carolina, 2017.

[50] Patrick Benavidez, Josue Lambert, Aldo Jaimes, and Mo Jamshidi. Landing of an ardrone 2.0 quadcopter on a mobile base using fuzzy logic. In *2014 World Automation Congress (WAC)*, pages 803–812. IEEE, 2014.

[51] Janis Tiemann and Christian Wietfeld. Scalable and precise multi-uav indoor navigation using tdoa-based uwb localization. In *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–7. IEEE, 2017.

[52] K Boudjit and Chevif Larbes. Detection and implementation autonomous target tracking with a quadrotor ar. drone. In *2015 12th International Conference on Informatics in Control, Automation and Robotics*, volume 2, pages 223–230. IEEE, 2015.

[53] Syaril Azrad, Farid Kendoul, and Kenzo Nonami. Visual servoing of quadrotor micro-air vehicle using color-based tracking algorithm. *Journal of System Design and Dynamics*, 4:255–268, 01 2010.

[54] Sungwook Cho, Sungsik Huh, David Hyunchul Shim, and Hyoung Sik Choi. Vision-based detection and tracking of airborne obstacles in a cluttered environment. *Journal of Intelligent & Robotic Systems*, 69(1-4):475–488, 2013.

[55] JeongWoon Kim, David Hyunchul Shim, and James R Morrison. Tablet pc-based visual target-following system for quadrotors. *Journal of Intelligent & Robotic Systems*, 74(1-2):85–95, 2014.

[56] DJI. Film like a pro: Dji drone activetrack with video tutorials. https://store.dji.com/guides/film-like-a-pro-with-activetrack, 2017.

[57] Korey Smith. Dji phantom 4 released with machine learning. https://myfirstdrone.com/blog/dji-phantom-4-released-with-machine-learning, 2016.

[58] Alexandre Bonnet and Moulay A Akhloufi. Uav pursuit using reinforcement learning. In *Unmanned Systems Technology XXI*, volume 11021, page 1102109. International Society for Optics and Photonics, 2019.

[59] Parrot. Technical specification. www.parrot.com/ca/drones/parrot-ardrone-20-power-edition, 2019.

[60] Pierre-Jean Bristeau, François Callou, David Vissiere, and Nicolas Petit. The navigation and control technology inside the ar. drone micro uav. *IFAC Proceedings Volumes*, 44(1):1477–1484, 2011.

[61] Myer Kutz. *Handbook of Measurement in Science and Engineering, Volume 1.* John Wiley and Sons, 2013.

[62] VICON. Tracker 3. www.vicon.com/file/vicon/tracker-3-11042017-55587.pdf, 2019.

[63] VICON. Vicon tracker user guide. docs.vicon.com/display/Tracker37, 2016.

[64] VICON. Mask unwanted reflections. https://docs.vicon.com/display/Nexus25/Mask+unwanted+reflections, 2019.

[65] NVIDIA. Cuda zone. www.developer.nvidia.com/cuda-zone, 2019.

[66] Joseph Redmon. Darknet: Open source neural networks in c. pjreddie.com/darknet/, 2013–2016.

[67] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[68] Jason M. O'Kane. *A Gentle Introduction to ROS.* Independently published, 2013.

[69] A. Chakrabarty, R. Morris, X. Bouyssounouse, and R. Hunt. Autonomous indoor object tracking with the parrot ar.drone. In *2016 International Conference on Unmanned Aircraft Systems*, pages 25–30, June 2016.

[70] The MathWorks Inc. Simulink. https://www.mathworks.com/products/simulink.html, 2019.

[71] The MathWorks Inc. Model predictive control toolbox. https://www.mathworks.com/products/mpc.html, 2019.

[72] The MathWorks Inc. Pid control with matlab and simulink. https://www.mathworks.com/discovery/pid-control.html, 2019.

[73] The MathWorks Inc. Ar.drone 2.0 support from embedded coder. https://www.mathworks.com/hardware-support/ar-drone.html, 2019. Accessed: 2019-05-30.

[74] Martin Barczyk. Nonlinear state estimation and modeling of a helicopter uav. Master's thesis, University of Alberta, 2012.

[75] M. H. Ang and V. D. Tourassis. Singularities of euler and roll-pitch-yaw representations. *IEEE Transactions on Aerospace and Electronic Systems*, AES-23(3):317–324, May 1987.

[76] J. B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality.* Princeton University Press, 1999.

[77] MA Rosaldo-Serrano and E Aranda-Bricaire. Trajectory tracking for a commercial quadrotor via time-varying backstepping. *IFAC-PapersOnLine*, 51(13):532–536, 2018.

[78] P. Vlez, N. Certad, and E. Ruiz. Trajectory generation and tracking using the ar.drone 2.0 quadcopter uav. In *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*, pages 73–78, Oct 2015.

[79] Martin Barczyk. Geometric tracking of yaw. Private Communications, 2019.

[80] Andrew Baker. *Matrix Groups: an Introduction to Lie Group Theory.* Springer London, 2002.

[81] John M Lee. *Introduction to smooth manifolds*. Springer, 2003.

[82] Karl J Astrom and Tore Hagglund. *PID Controllers - Theory, Design, and Tuning (2nd Edition)*. ISA, 1995.

[83] Jer-Nan Juang and Minh Q. Phan. *Identification and Control of Mechanical Systems*. Cambridge University Press, 2001.

[84] Martin Barczyk. Monocular camera depth estimation. Private Communications, 2019.

[85] C Brown Duane. Close-range camera calibration. *Photogramm. Eng*, 37(8):855–866, 1971.

[86] James Bowman and Patrick Mihelich. camera-calibration. wiki.ros.org/camera-calibration, 2019.

[87] Kelvin Liu. image_pipeline/camerainfo. wiki.ros.org/image_pipeline/CameraInfo, 2016.

[88] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3d bounding box estimation using deep learning and geometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7074–7082, 2017.

Appendices

Appendix A

Parrot AR.Drone 2.0 Camera Calibration File

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix: K
  rows: 3
  cols: 3
  data: [572.7366231999233, 0, 319.5969119642032, 0,
      ↪ 570.8044531287701, 157.8742799142208, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients: D
  rows: 1
  cols: 5
  data: [-0.5388475156926118, 0.2886187448441893,
      ↪ 0.001056001496586167, -0.003164048988444699, 0]
rectification_matrix: R
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix: P
  rows: 3
  cols: 4
  data: [462.7559204101562, 0, 315.6955710386646, 0, 0,
      ↪ 536.928466796875, 155.340347346857, 0, 0, 0, 1, 0]
```

Appendix B

Bounding Box Detection Uncertainty Data

B.1 Bounding Box X-Position Test Data

Table B.1: Bounding Box X-Position Test Data Summary

Relative X-Position [m]	e_{Z_w} **[m]**	$e_{Z_{geom}}$ **[m]**	X_{Vicon} **[m]**	**Duration** t_e **[s]**
0.5 m	0.0528 ± 0.0025	0.1835 ± 0.0028	0.4948	12.1803
1.0 m	0.0309 ± 0.0028	0.0828 ± 0.0037	0.9923	10.9102
1.5 m	-0.0409 ± 0.0042	0.0985 ± 0.0101	1.4935	10.1401
2.0 m	-0.0094 ± 0.0077	0.1156 ± 0.0071	1.9823	14.1297
2.5 m	-0.0165 ± 0.0126	0.0692 ± 0.0065	2.5013	10.8903
3.0 m	0.0330 ± 0.0170	0.0916 ± 0.0131	2.9945	10.8298

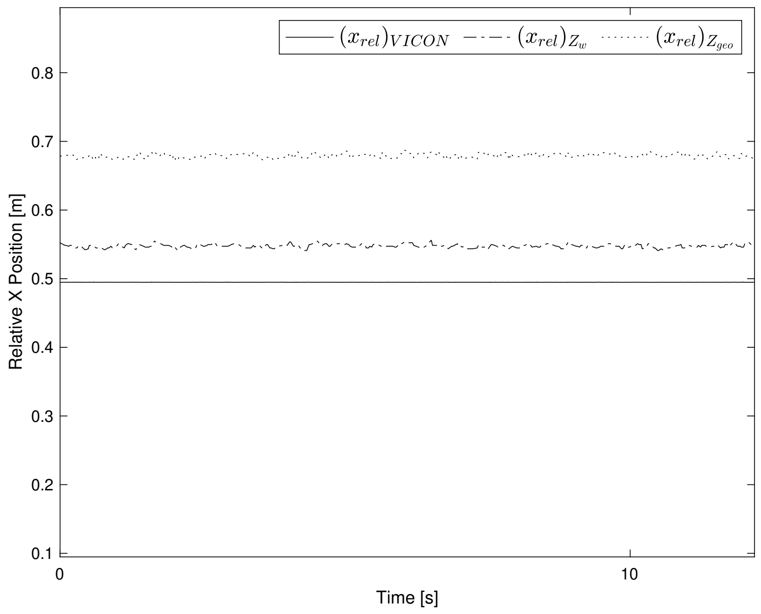


Figure B.1: Depth Estimation Performance at a X-Position of 0.5 m

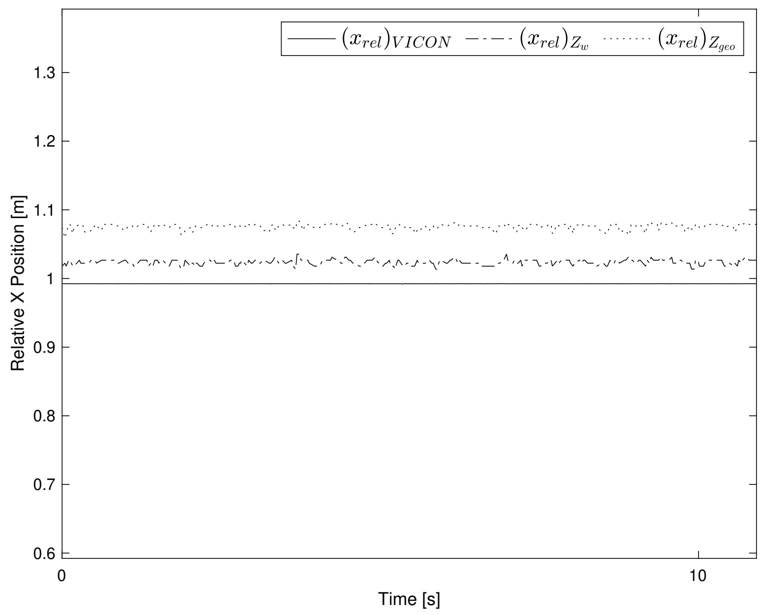


Figure B.2: Depth Estimation Performance at a X-Position of 1.0 m

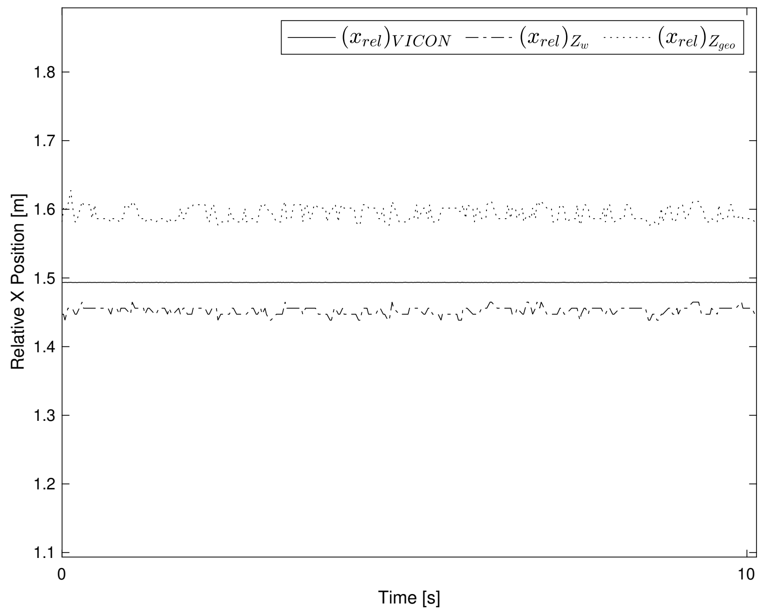


Figure B.3: Depth Estimation Performance at a X-Position of 1.5 m

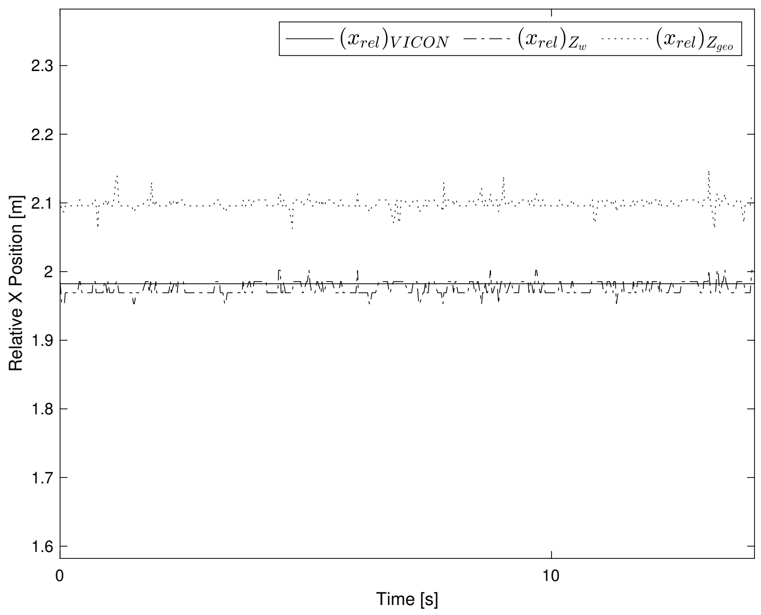


Figure B.4: Depth Estimation Performance at a X-Position of 2.0 m

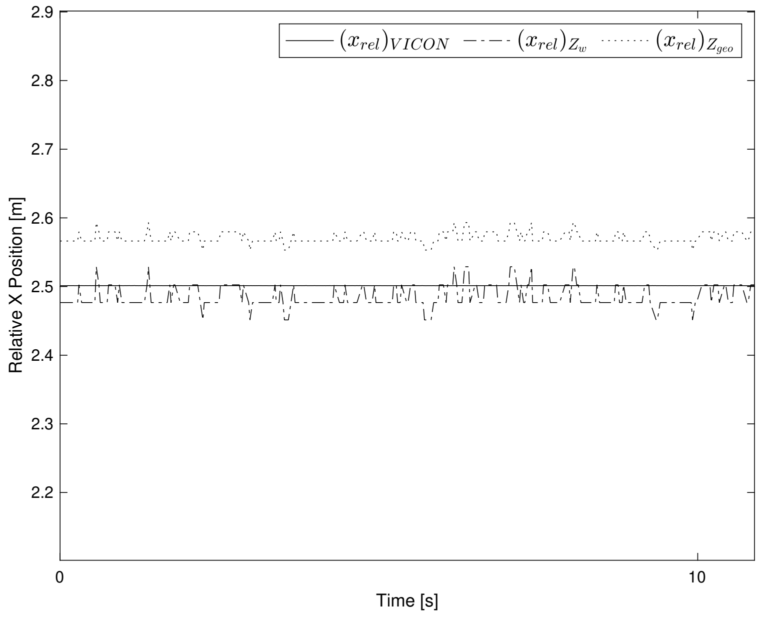


Figure B.5: Depth Estimation Performance at a X-Position of 2.5 m

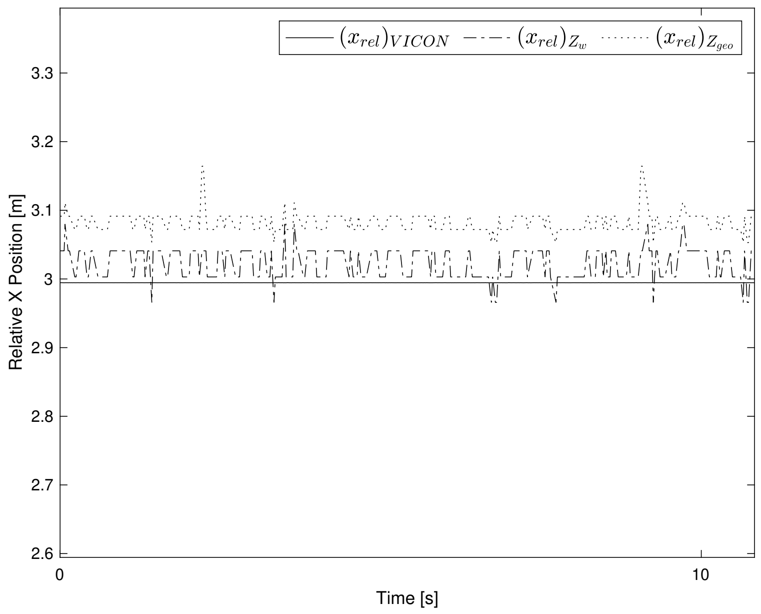


Figure B.6: Depth Estimation Performance at a X-Position of 3.0 m

B.2 Bounding Box Y-Position Test Data

Table B.2: Bounding Box Y-Position Test Data Summary

Relative Y-Position [m]	e_{Z_w} **[m]**	$e_{Z_{geom}}$ **[m]**	Y_{Vicon} **[m]**	**Duration** t_e **[s]**
-1.0 m	-0.0175 ± 0.0114	0.1920 ± 0.0171	-1.0226	11.6199
-0.75 m	-0.0125 ± 0.0093	0.2230 ± 0.0052	-0.7714	11.7498
-0.5 m	-0.0130 ± 0.0123	0.2479 ± 0.0167	-0.5386	11.6303
-0.25 m	0.0432 ± 0.0083	0.2030 ± 0.0146	-0.2773	10.4501
0.0 m	0.0155 ± 0.0073	0.1531 ± 0.0166	0.0237	10.9696
0.25 m	-0.0818 ± 0.0066	0.1087 ± 0.0054	0.2332	11.2899
0.5 m	-0.2189 ± 0.0090	-0.0157 ± 0.0151	0.4777	10.4599
0.75 m	-0.4393 ± 0.0073	-0.2673 ± 0.0172	0.7258	10.4483
1.0 m	-0.2695 ± 0.0054	-0.2737 ± 0.0137	0.9495	11.3478

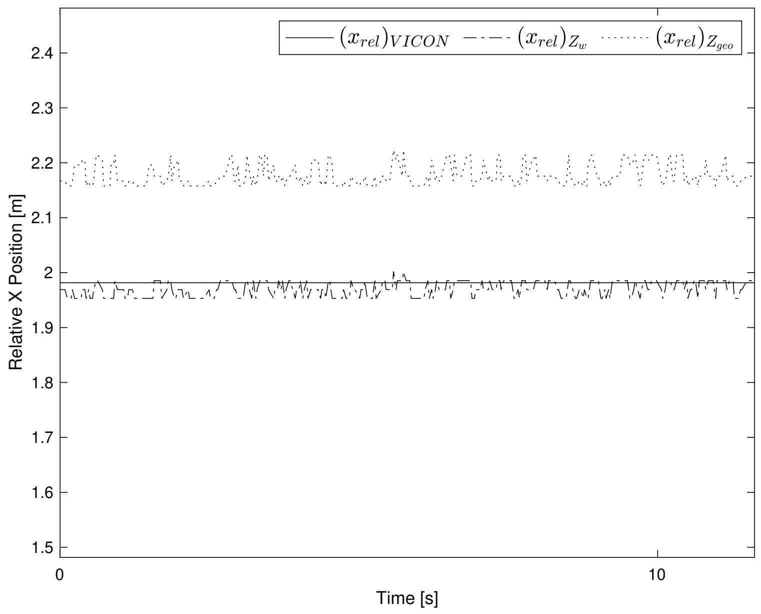


Figure B.7: Depth Estimation Performance at a Y-Position of −1.0 m

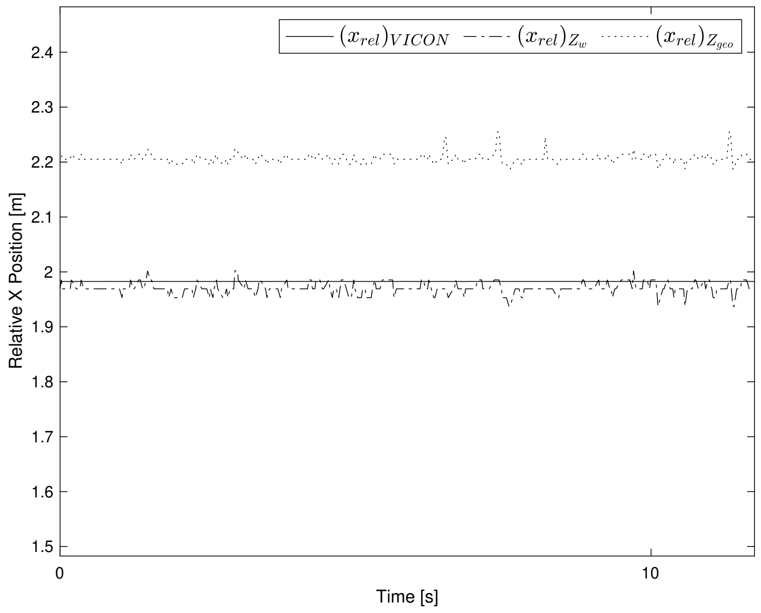


Figure B.8: Depth Estimation Performance at a Y-Position of −0.75 m

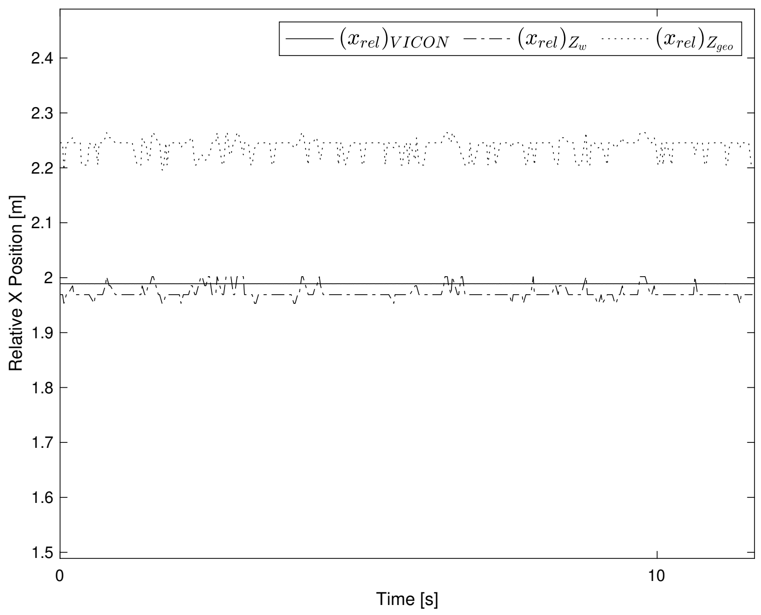


Figure B.9: Depth Estimation Performance at a Y-Position of -0.5 m

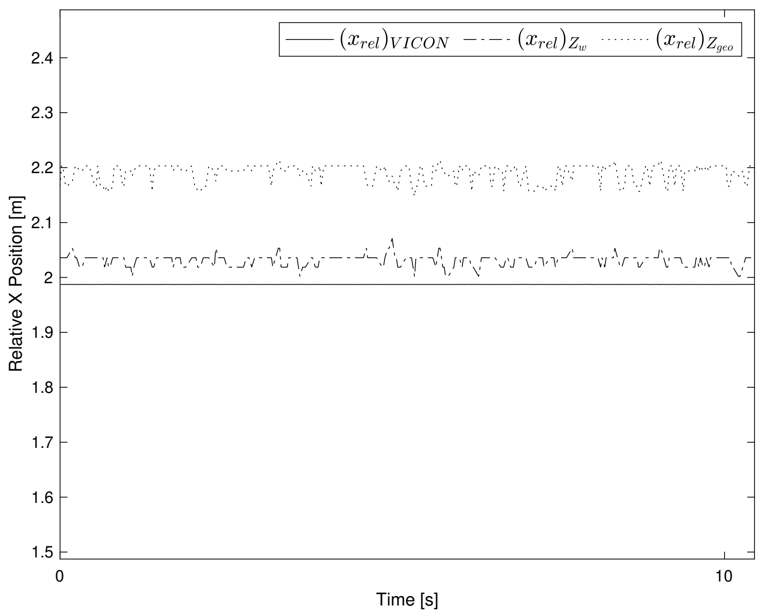


Figure B.10: Depth Estimation Performance at a Y-Position of -0.25 m

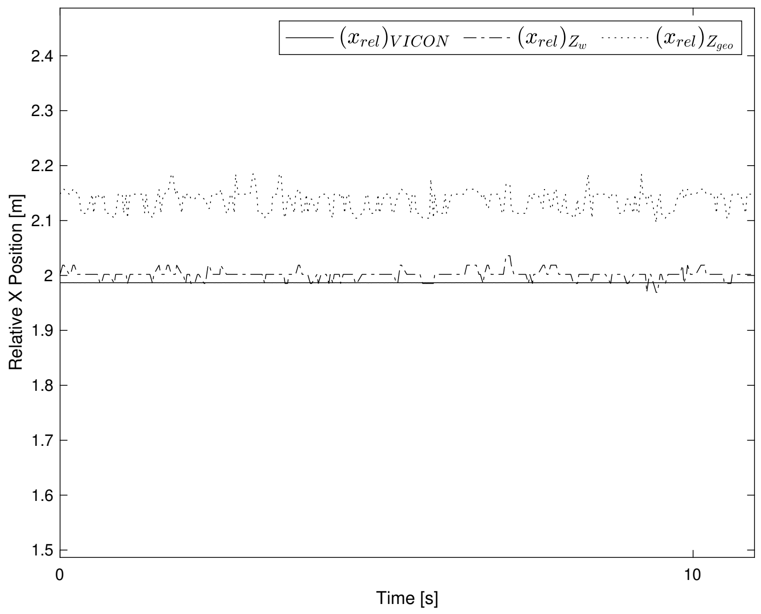


Figure B.11: Depth Estimation Performance at a Y-Position of 0.0 m

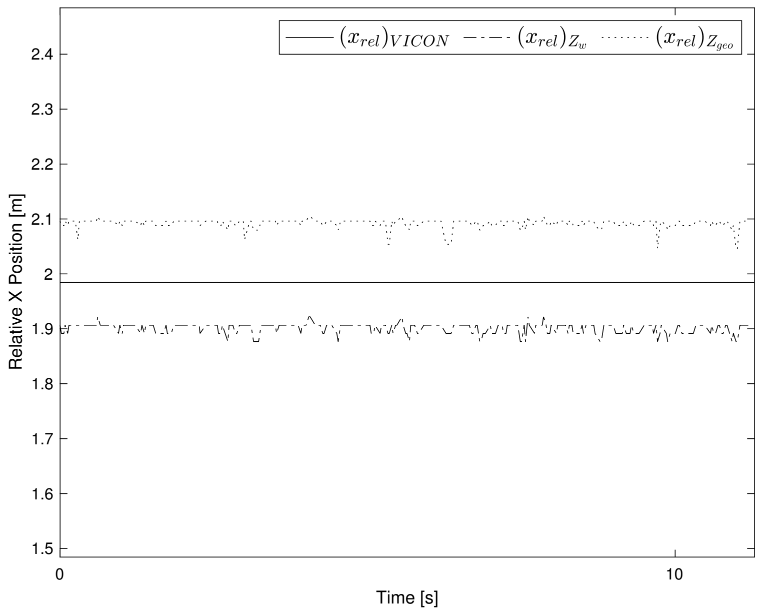


Figure B.12: Depth Estimation Performance at a Y-Position of 0.25 m

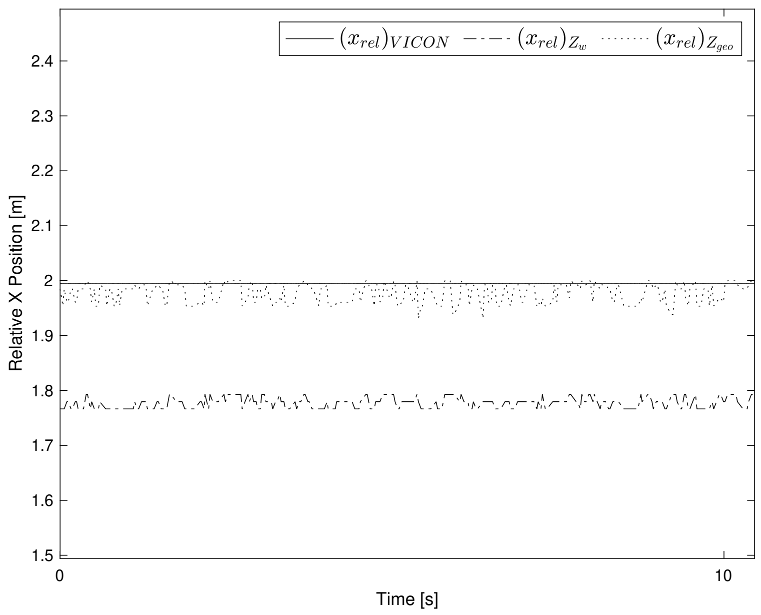


Figure B.13: Depth Estimation Performance at a Y-Position of 0.50 m

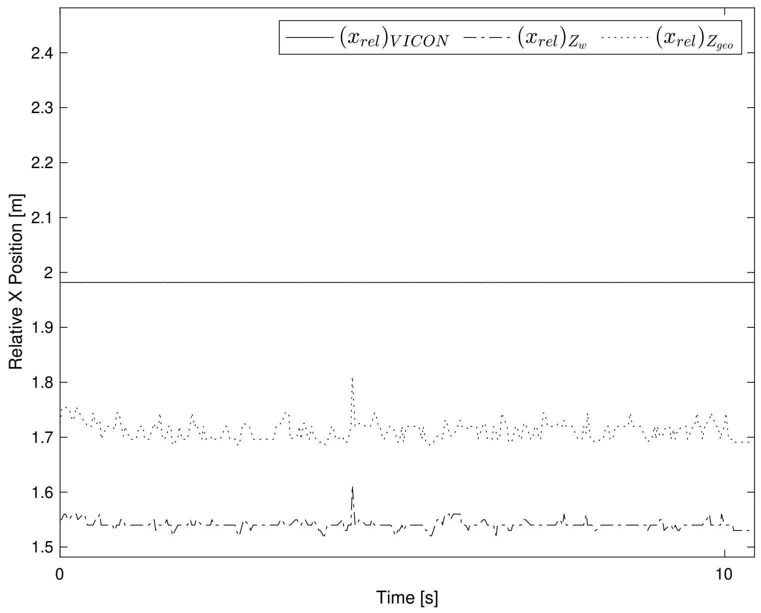


Figure B.14: Depth Estimation Performance at a Y-Position of 0.75 m

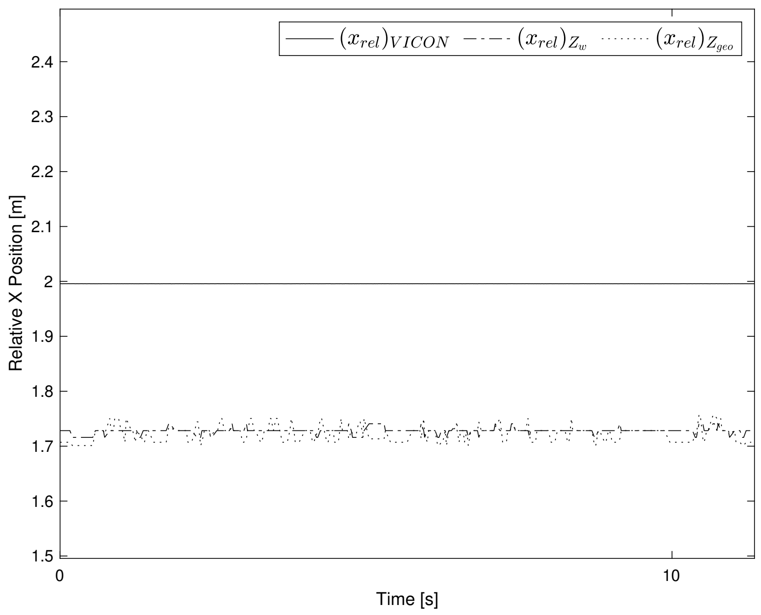


Figure B.15: Depth Estimation Performance at a Y-Position of 1.00 m

Appendix C

PID Assessment Data

C.1 PID Step Response

C.1.1 PID Step Response in the X-Axis

Table C.1: PID Parameters for Step Response in the X-axis

PID Tuning Gains			
Position Gains	K_p $[\frac{1}{m}]$	K_i $[\frac{1}{m*s}]$	K_d $[\frac{s}{m}]$
Body Frame X-Direction, X_B	1.35	0.003	1.50
Body Frame Y-Direction, Y_B	1.00	0.010	0.50
Body Frame Z-Direction, Z_B	1.00	0.010	0.50
Orientation Gains	K_p $[\frac{1}{rad}]$	K_i $[\frac{1}{rad*s}]$	K_d $[\frac{s}{rad}]$
Yaw Rotation, ψ	1.00	0.010	0.75
Additional PID Parameters			
Frequency [Hz]	25		

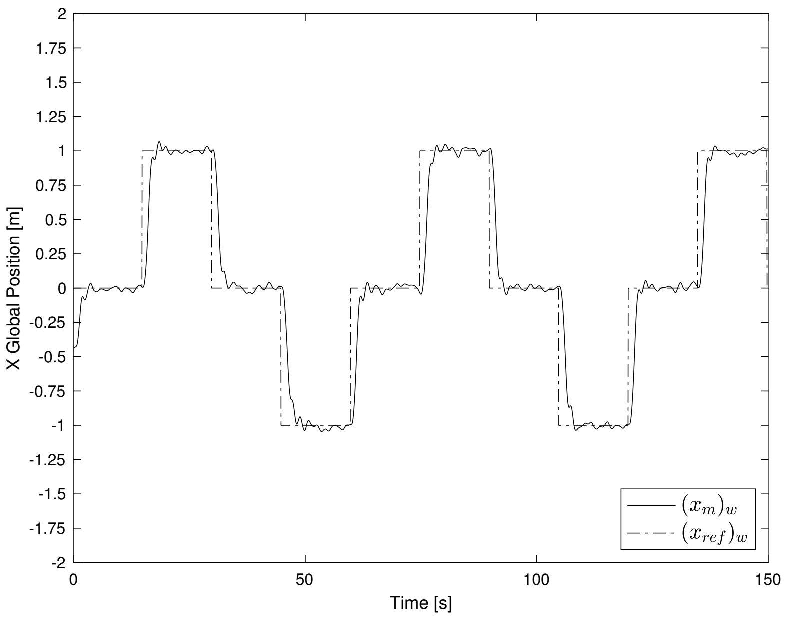


Figure C.1: Global X-Position obtained from PID Step Response in the X-Axis

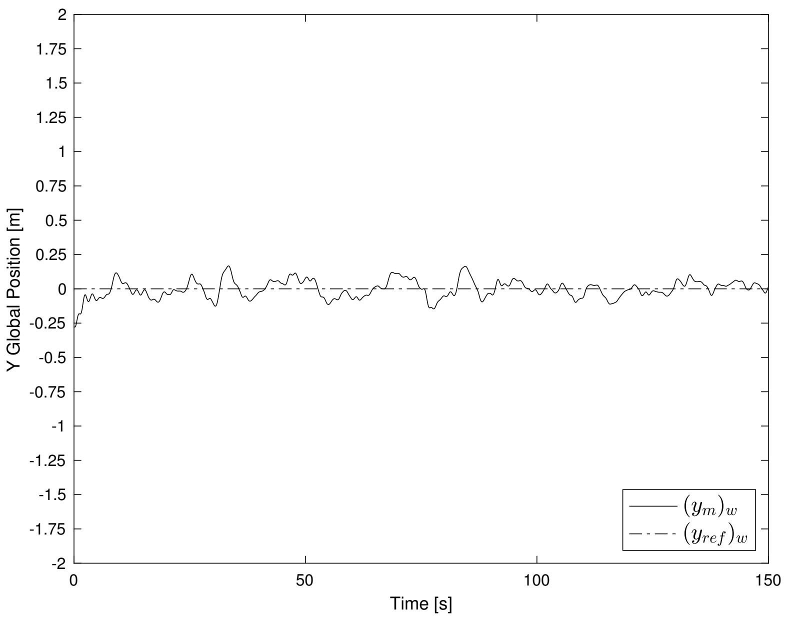


Figure C.2: Global Y-Position obtained from PID Step Response in the X-Axis

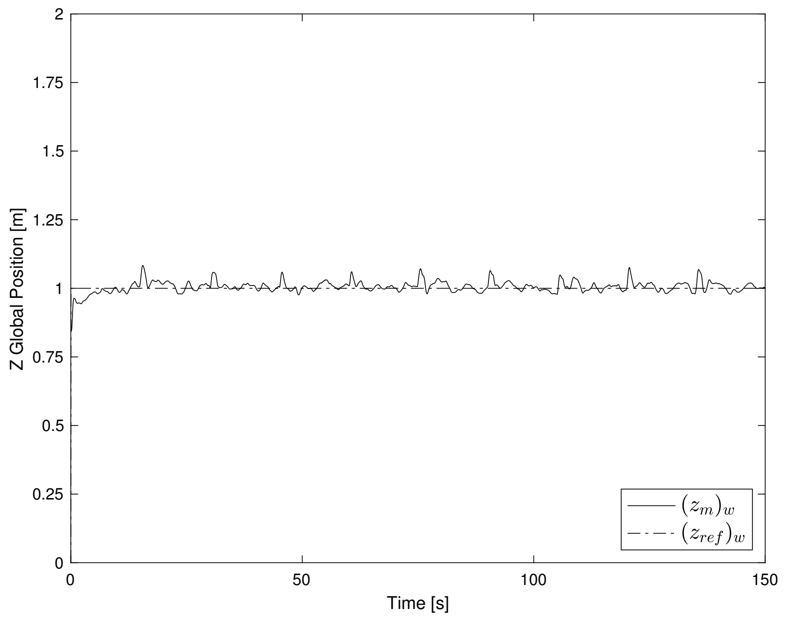


Figure C.3: Global Z-Position obtained from PID Step Response in the X-Axis

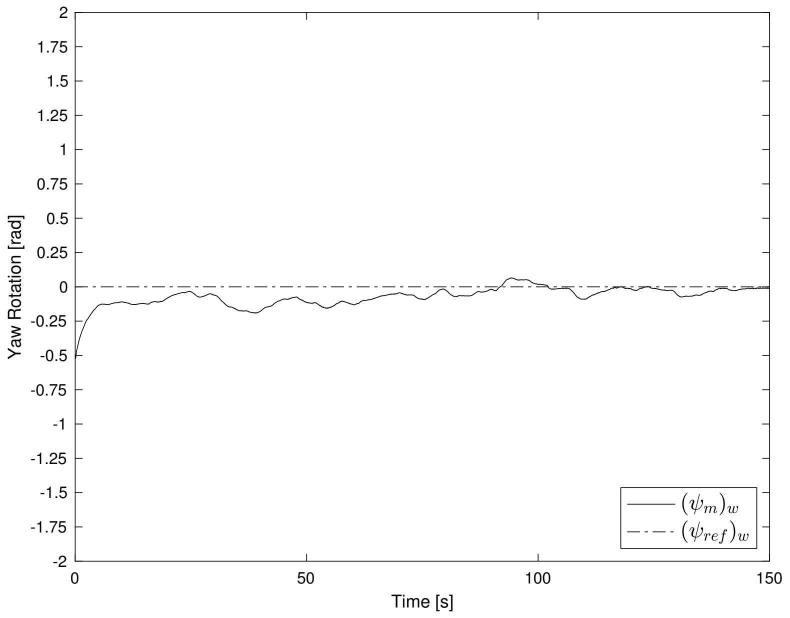


Figure C.4: Yaw obtained from PID Step Response in the X-Axis

C.1.2 PID Step Response in the Y-Axis

Table C.2: PID Parameters for Step Response in the Y-axis

PID Tuning Gains			
Position Gains	$K_p\ [\frac{1}{m}]$	$K_i\ [\frac{1}{m*s}]$	$K_d\ [\frac{s}{m}]$
Body Frame X-Direction, X_B	1.35	0.003	1.50
Body Frame Y-Direction, Y_B	1.45	0.003	1.50
Body Frame Z-Direction, Z_B	1.00	0.010	0.50
Orientation Gains	$K_p\ [\frac{1}{rad}]$	$K_i\ [\frac{1}{rad*s}]$	$K_d\ [\frac{s}{rad}]$
Yaw Rotation, ψ	1.00	0.010	0.75
Additional PID Parameters			
Frequency [Hz]	25		

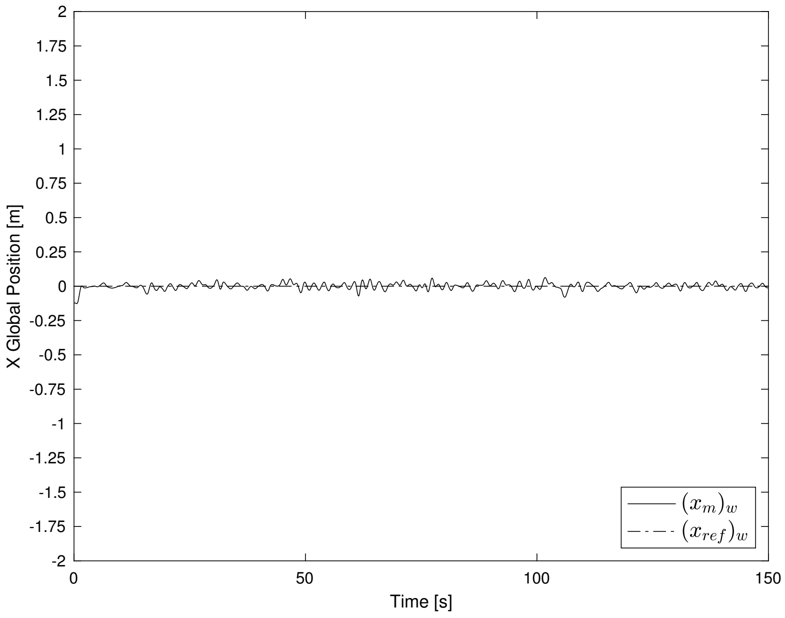


Figure C.5: Global X-Position obtained from PID Step Response in the Y-Axis

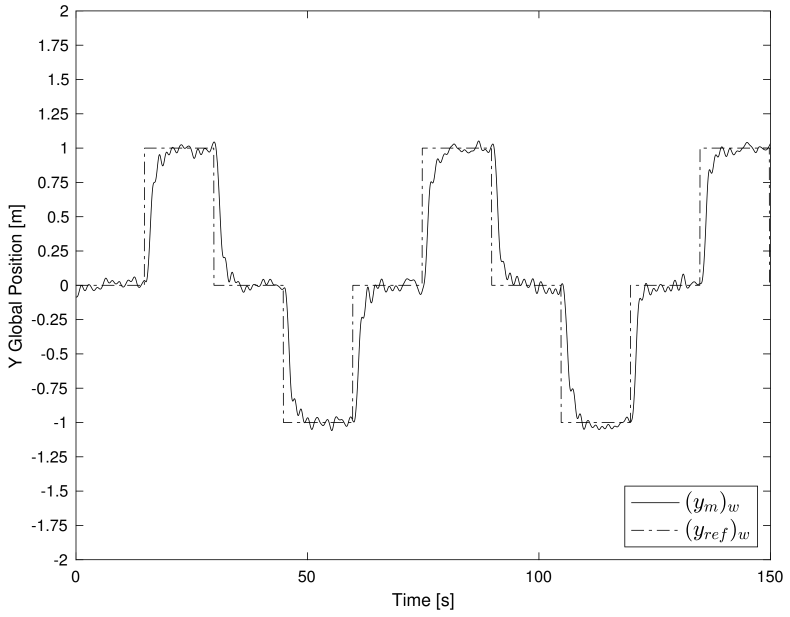


Figure C.6: Global Y-Position obtained from PID Step Response in the Y-Axis

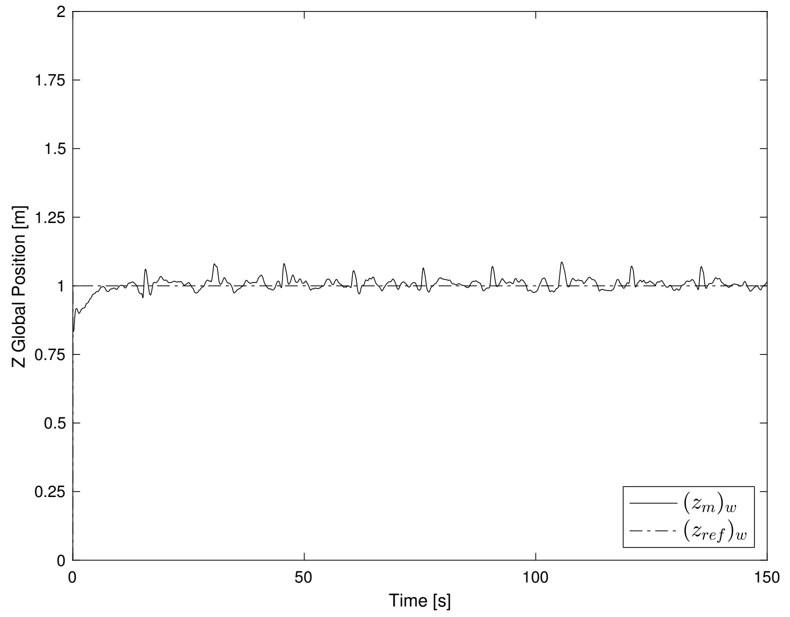


Figure C.7: Global Z-Position obtained from PID Step Response in the Y-Axis

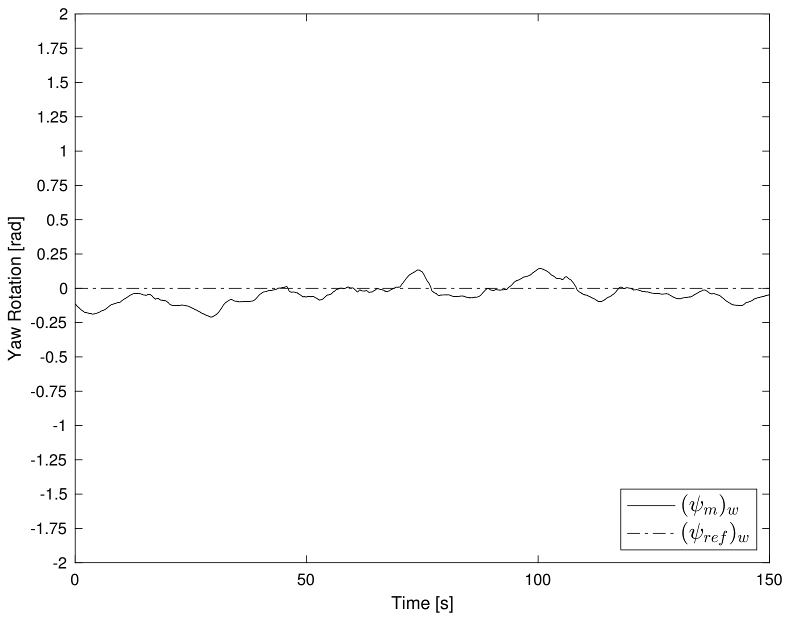


Figure C.8: Yaw obtained from PID Step Response in the Y-Axis

C.1.3 PID Step Response in the Z-Axis

Table C.3: PID Parameters for Step Response in the Z-axis

PID Tuning Gains			
Position Gains	K_p $[\frac{1}{m}]$	K_i $[\frac{1}{m*s}]$	K_d $[\frac{s}{m}]$
Body Frame X-Direction, X_B	1.00	0.010	0.50
Body Frame Y-Direction, Y_B	1.00	0.010	0.50
Body Frame Z-Direction, Z_B	1.10	0.003	3.50
Orientation Gains	K_p $[\frac{1}{rad}]$	K_i $[\frac{1}{rad*s}]$	K_d $[\frac{s}{rad}]$
Yaw Rotation, ψ	1.00	0.010	0.75
Additional PID Parameters			
Frequency [Hz]	25		

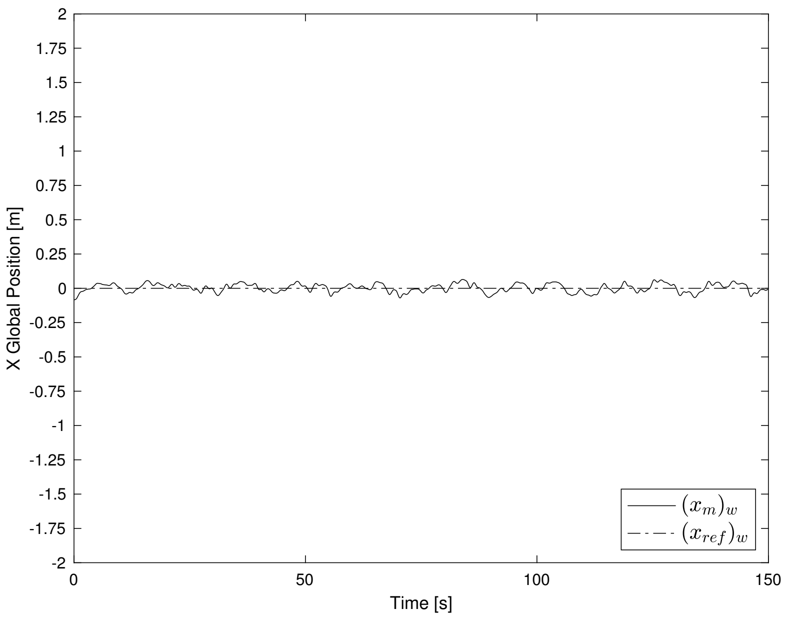


Figure C.9: Global X-Position obtained from PID Step Response in the Z-Axis

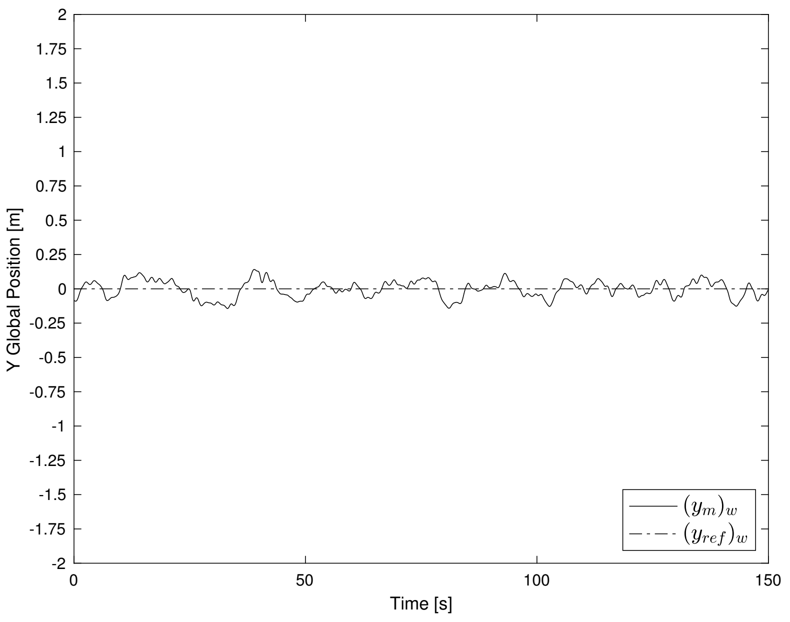


Figure C.10: Global Y-Position obtained from PID Step Response in the Z-Axis

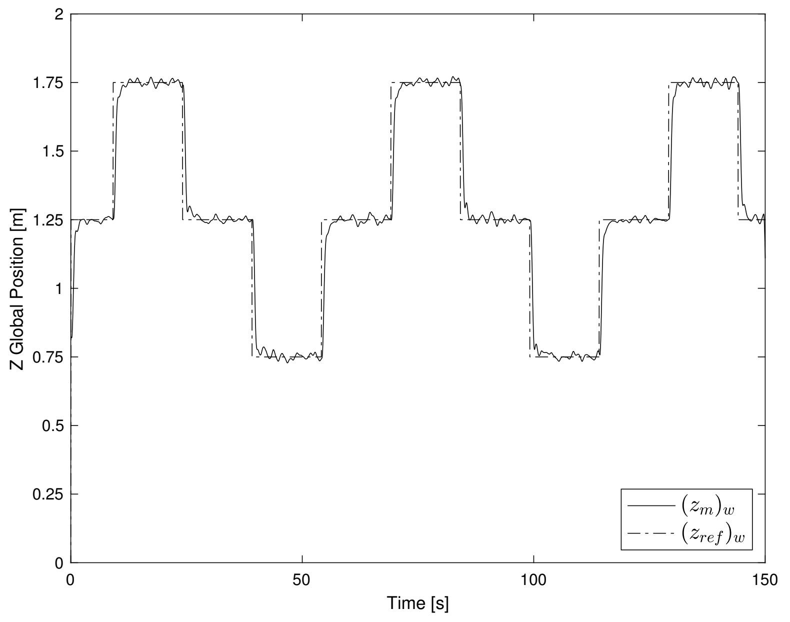


Figure C.11: Global Z-Position obtained from PID Step Response in the Z-Axis

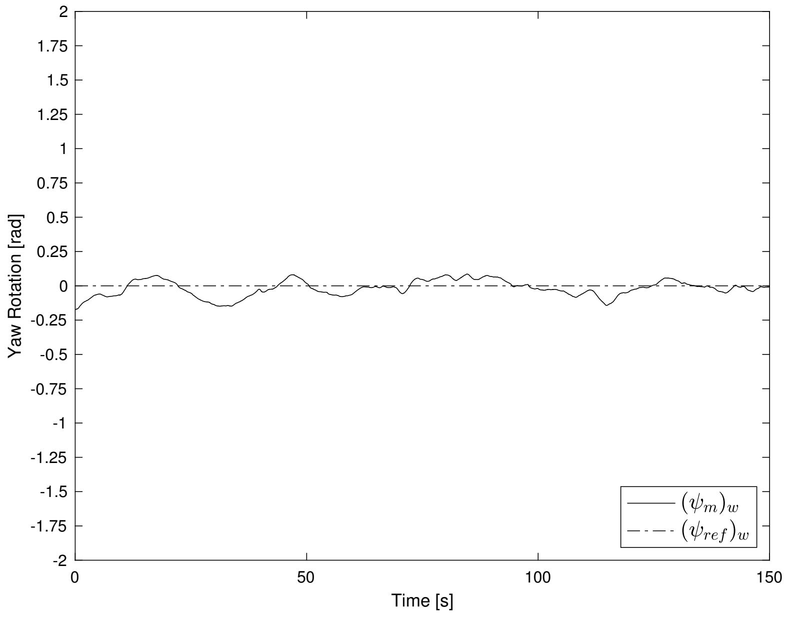


Figure C.12: Yaw obtained from PID Step Response in the Z-Axis

C.1.4 PID Step Response in Yaw Rotation

Table C.4: PID Parameters for Step Response in Yaw

PID Tuning Gains			
Position Gains	K_p $[\frac{1}{m}]$	K_i $[\frac{1}{m*s}]$	K_d $[\frac{s}{m}]$
Body Frame X-Direction, X_B	1.00	0.010	0.50
Body Frame Y-Direction, Y_B	1.00	0.010	0.50
Body Frame Z-Direction, Z_B	1.00	0.010	0.50
Orientation Gains	K_p $[\frac{1}{rad}]$	K_i $[\frac{1}{rad*s}]$	K_d $[\frac{s}{rad}]$
Yaw Rotation, ψ	0.10	0.007	8.50
Additional PID Parameters			
Frequency [Hz]	25		

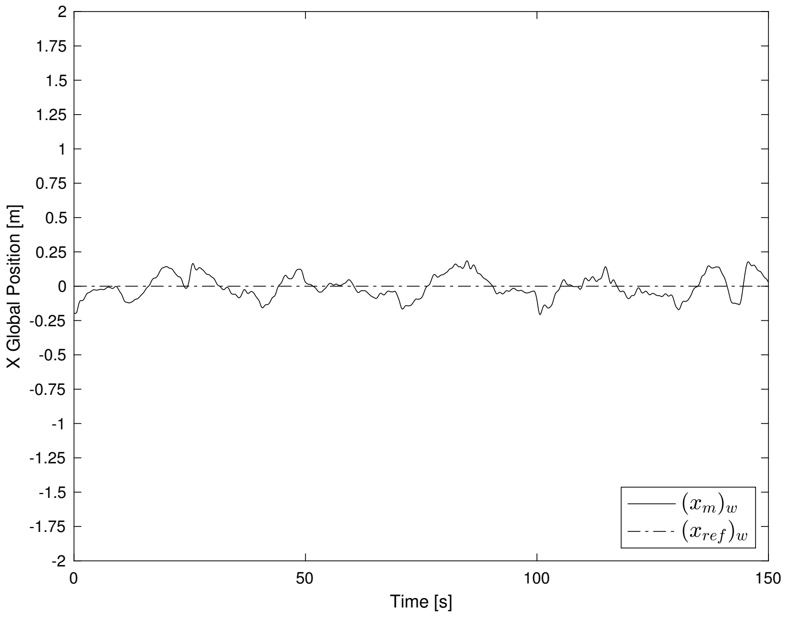


Figure C.13: Global X-Position obtained from PID Step Response in Yaw

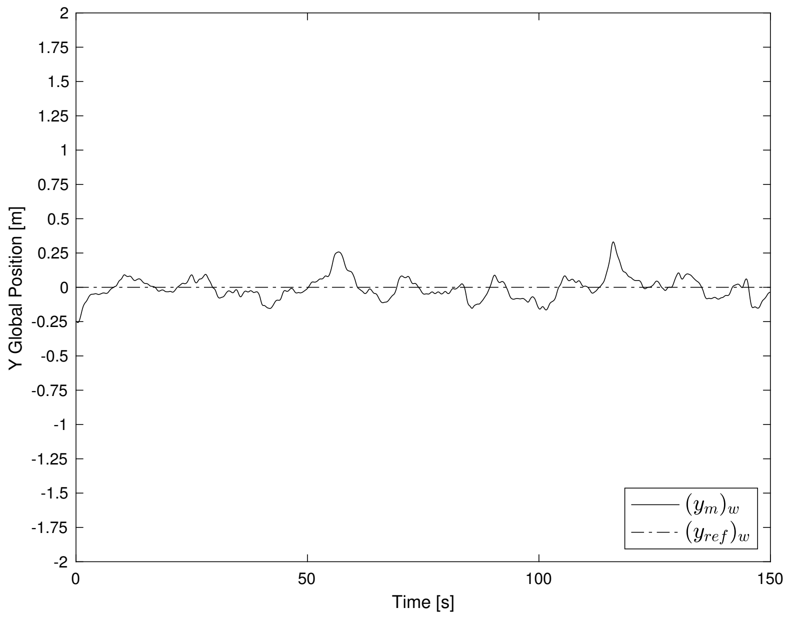


Figure C.14: Global Y-Position obtained from PID Step Response in Yaw

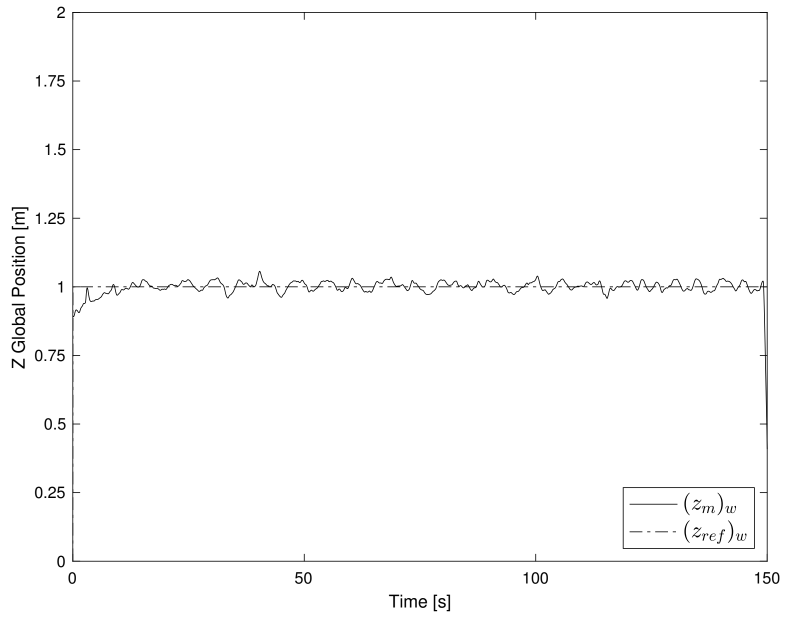


Figure C.15: Global Z-Position obtained from PID Step Response in Yaw

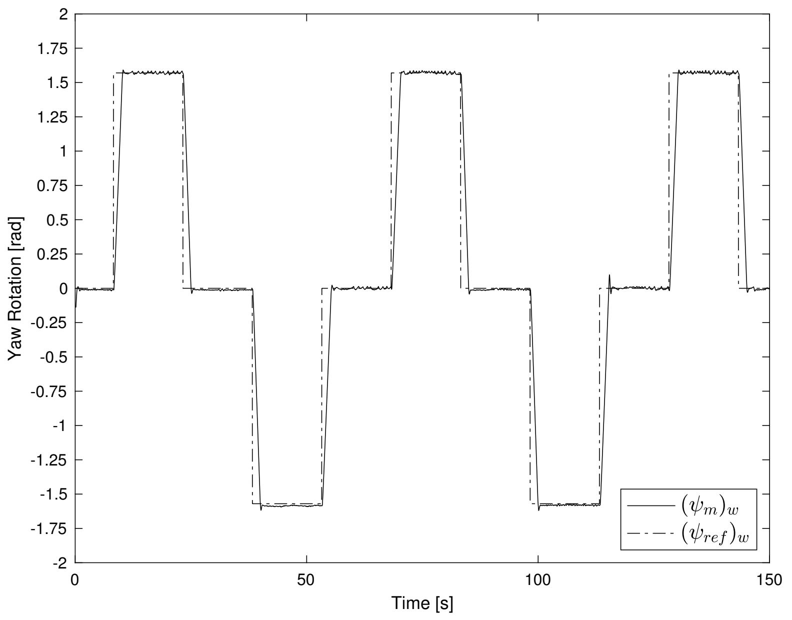


Figure C.16: Yaw obtained from PID Step Response in Yaw

C.2 PID Figure 8 Trajectory Response

Table C.5: PID Parameters for Step Response in the X-axis

PID Tuning Gains			
Position Gains	K_p $[\frac{1}{m}]$	K_i $[\frac{1}{m*s}]$	K_d $[\frac{s}{m}]$
Body Frame X-Direction, X_B	1.35	0.003	1.50
Body Frame Y-Direction, Y_B	1.45	0.003	1.50
Body Frame Z-Direction, Z_B	1.10	0.003	3.50
Orientation Gains	K_p $[\frac{1}{rad}]$	K_i $[\frac{1}{rad*s}]$	K_d $[\frac{s}{rad}]$
Yaw Rotation, ψ	0.10	0.007	8.50
Additional PID Parameters			
Frequency [Hz]	25		

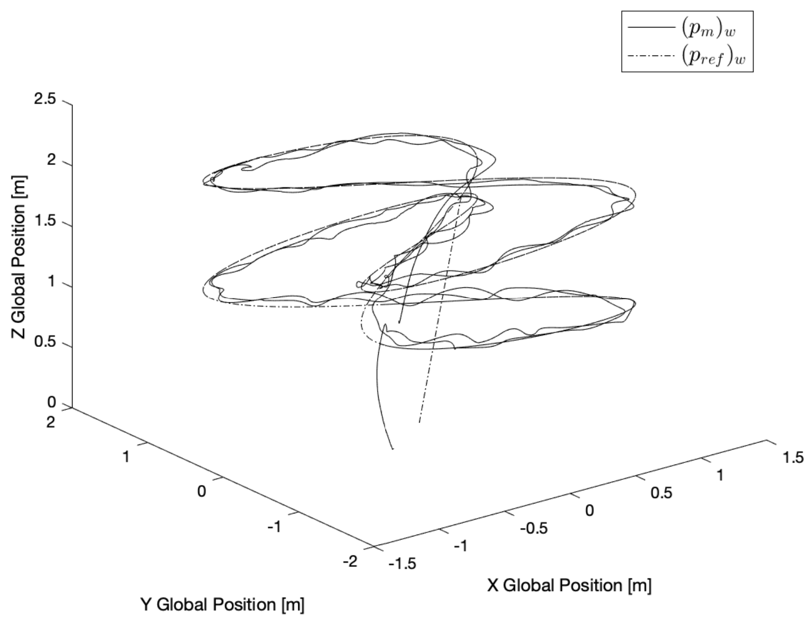


Figure C.17: 3D plot of the PID Figure-8 Trajectory Response

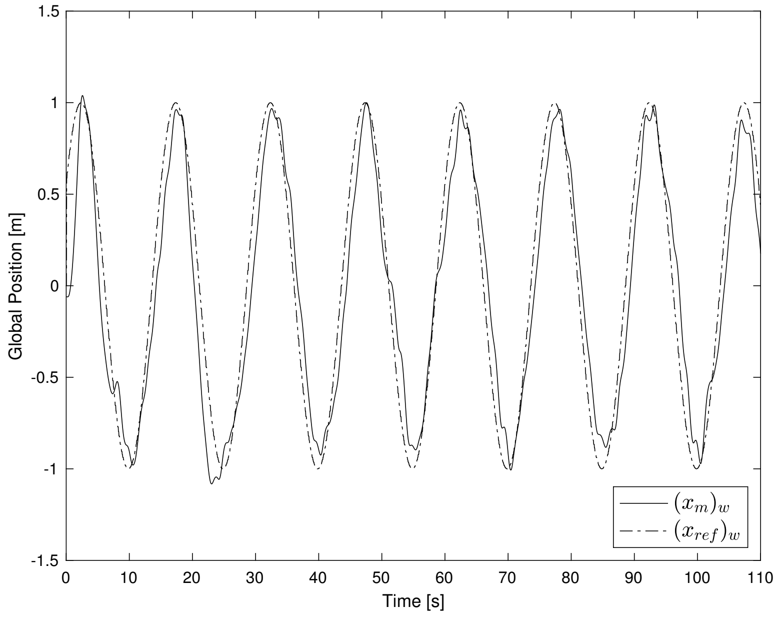


Figure C.18: PID Figure-8 Trajectory Response in the X-axis

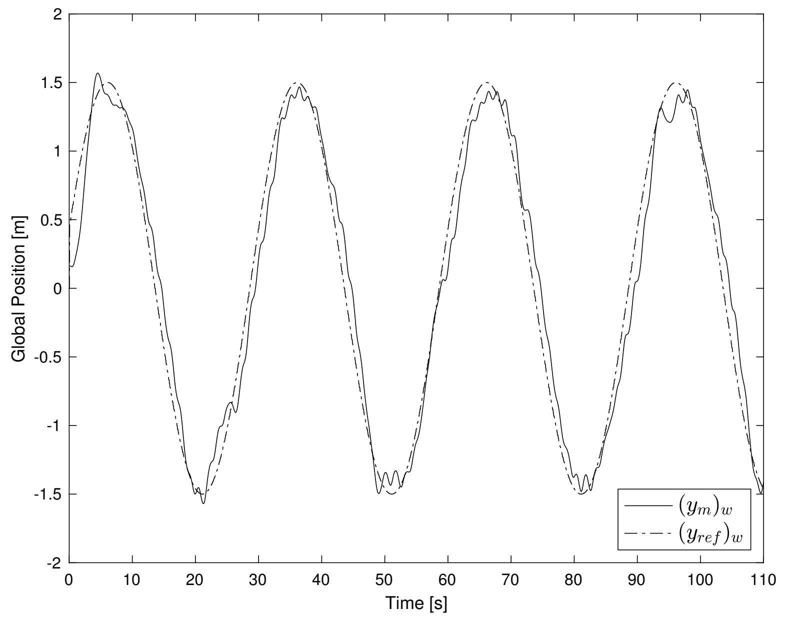


Figure C.19: PID Figure-8 Trajectory Response in the Y-axis

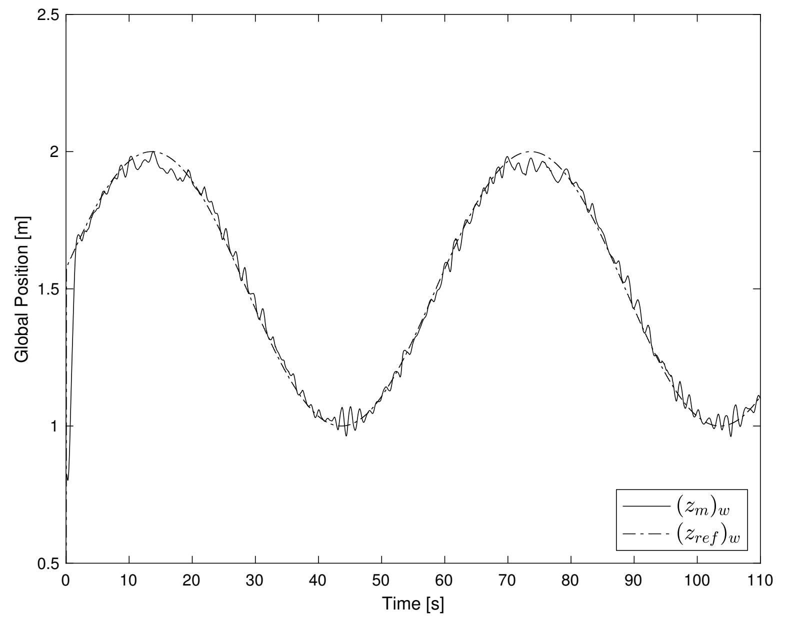


Figure C.20: PID Figure-8 Trajectory Response in the Z-axis

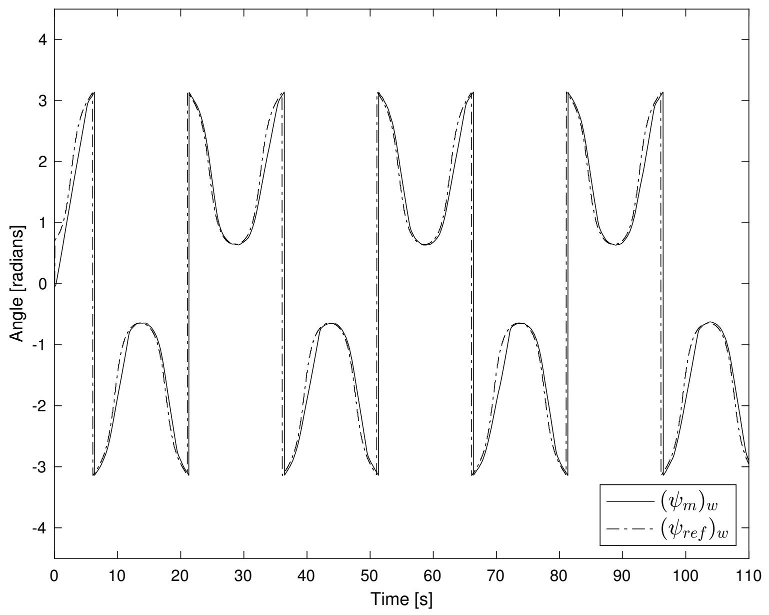


Figure C.21: PID Figure-8 Trajectory Response in Yaw

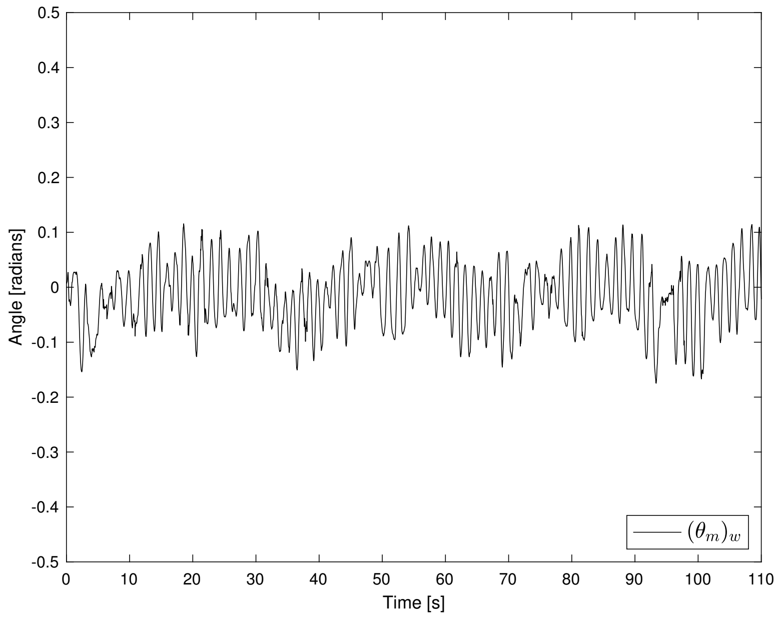


Figure C.22: PID Figure-8 Trajectory Response in Roll

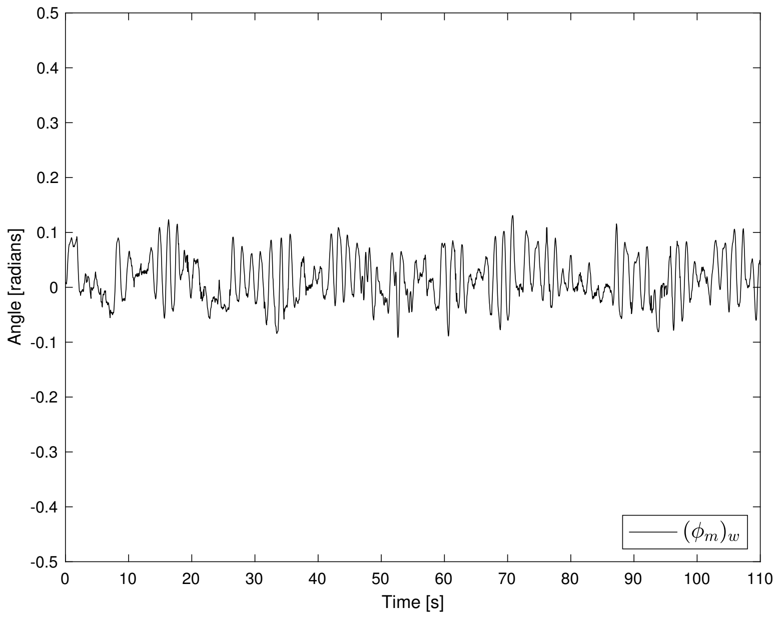


Figure C.23: PID Figure-8 Trajectory Response in Pitch

Appendix D

LMPC Assessment Data

D.1 LMPC Step Response

D.1.1 LMPC Step Response in the X-Axis

Table D.1: LMPC Parameters for Step Response in the X-axis

LMPC Parameter	**Value**
State Weights, Q	$(2.25, 2.25, 1.525, 5, 4.25, 4.25, 1.75)$
Input Weights, R	$(1, 1, 1.25, 1)$
Prediction Horizon, N_p	30
Frequency, f [Hz]	25

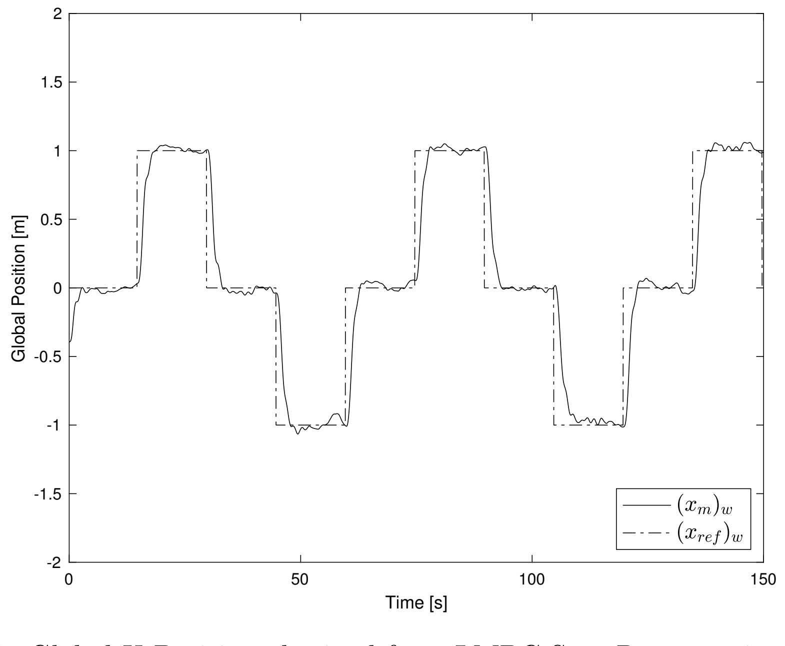


Figure D.1: Global X-Position obtained from LMPC Step Response in the X-Axis

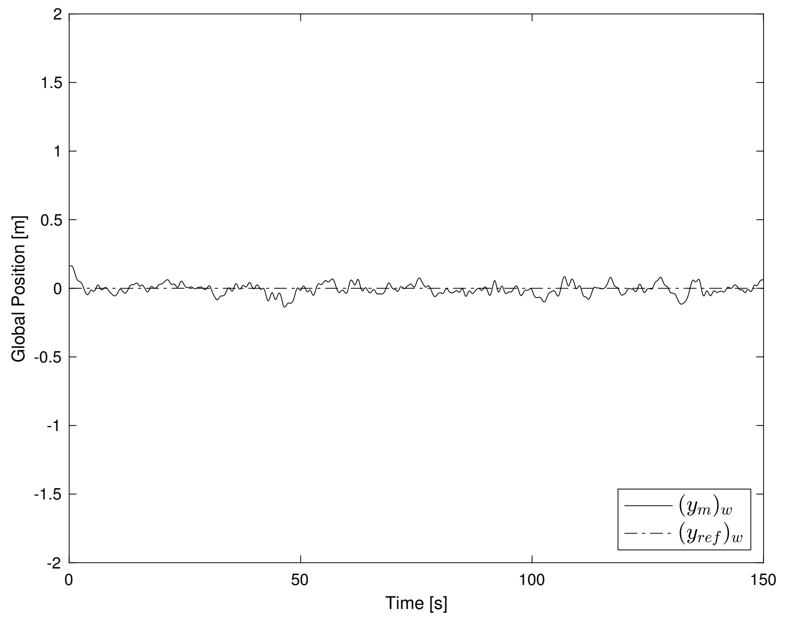


Figure D.2: Global Y-Position obtained from LMPC Step Response in the X-Axis

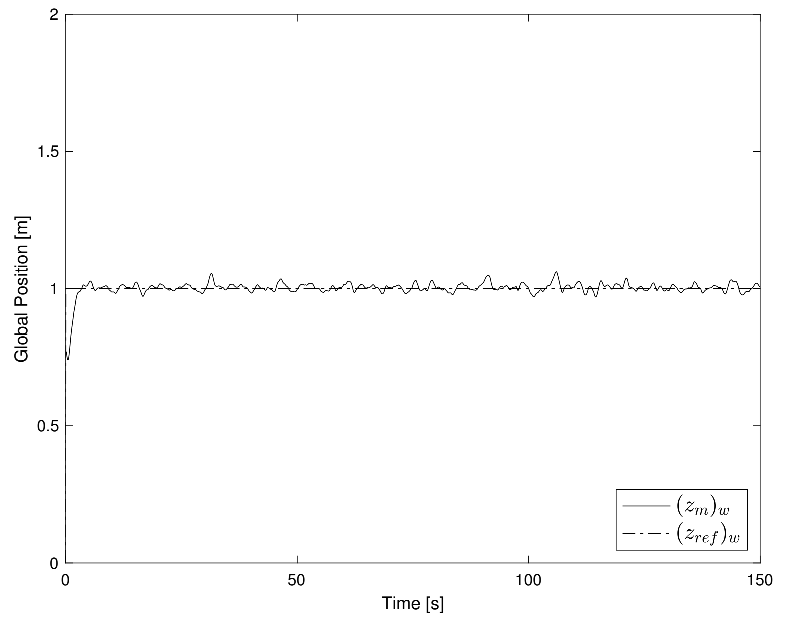


Figure D.3: Global Z-Position obtained from LMPC Step Response in the X-Axis

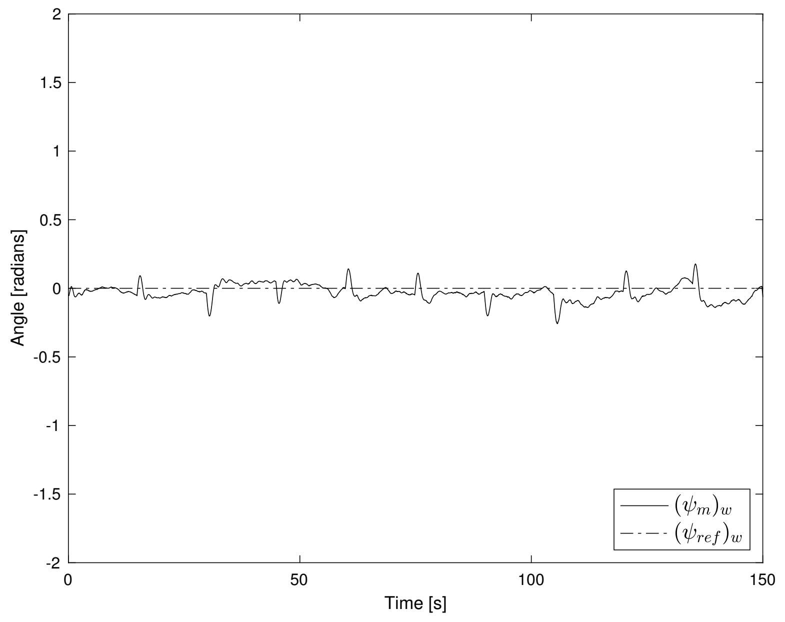


Figure D.4: Yaw obtained from LMPC Step Response in the X-Axis

D.1.2 LMPC Step Response in the Y-Axis

Table D.2: LMPC Parameters for Step Response in the Y-axis

Tuning Parameter	**Value**
State Gain, Q	$(2.25, 2.25, 1.525, 5, 4.25, 4.25, 1.75)$
Input Gain, R	$(1, 1, 1.25, 1)$
Prediction Horizon, N_p	30
Frequency, f [Hz]	25

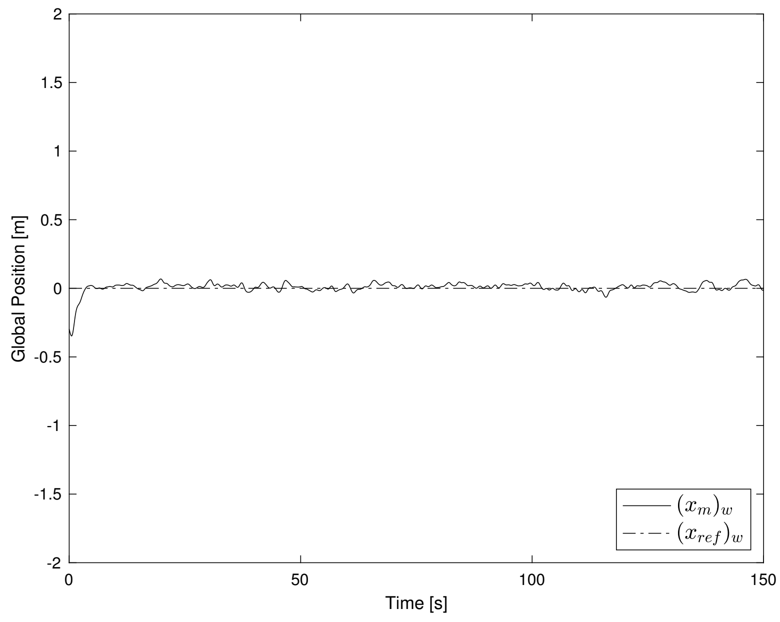


Figure D.5: Global X-Position obtained from LMPC Step Response in the Y-Axis

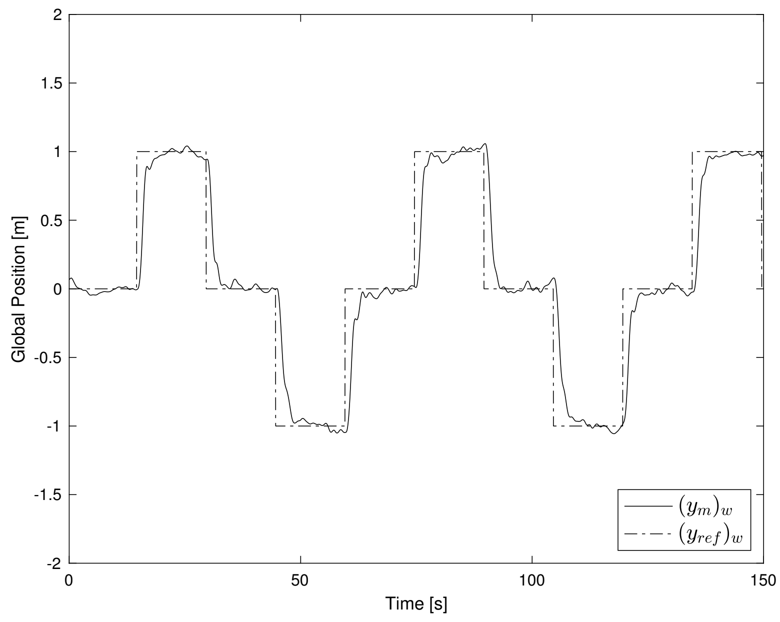


Figure D.6: Global Y-Position obtained from LMPC Step Response in the Y-Axis

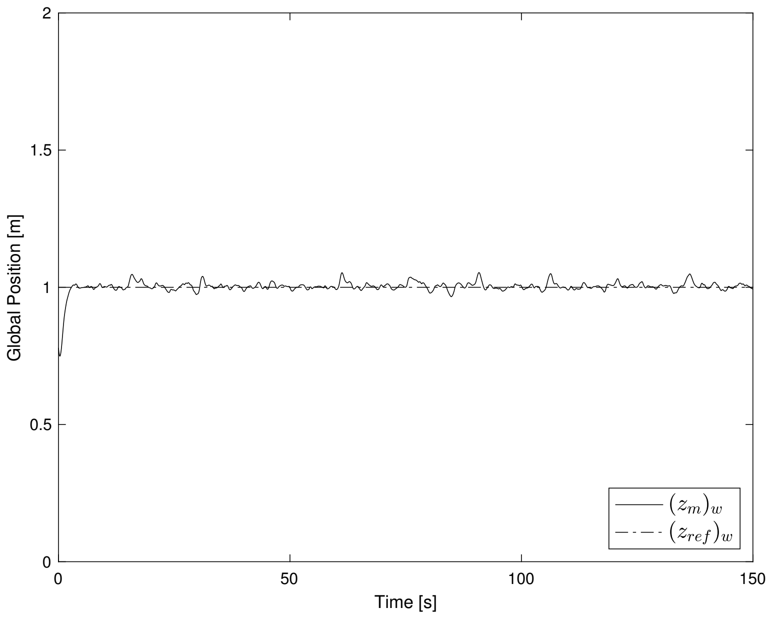


Figure D.7: Global Z-Position obtained from LMPC Step Response in the Y-Axis

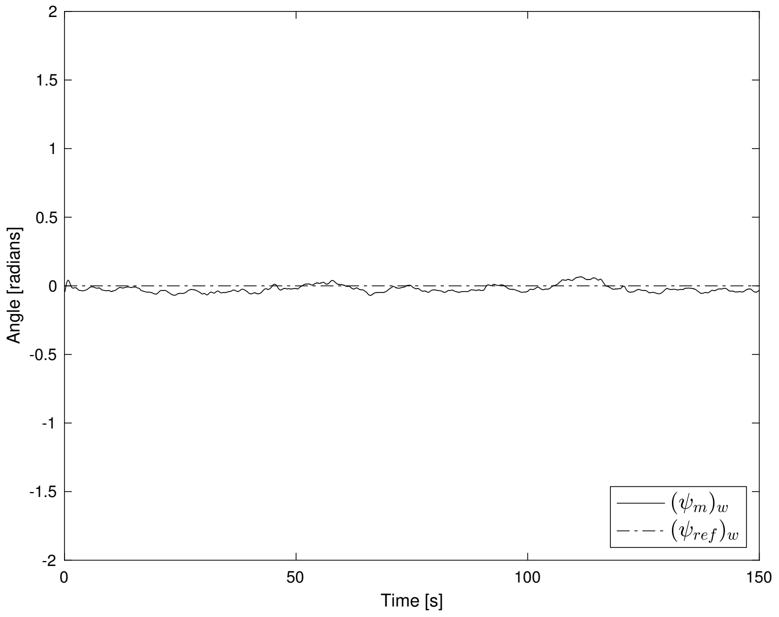


Figure D.8: Yaw obtained from LMPC Step Response in the Y-Axis

D.1.3 LMPC Step Response in the Z-Axis

Table D.3: LMPC Parameters for Step Response in the Z-axis

Tuning Parameter	**Value**
State Gain, Q	$(2.25, 2.25, 3.25, 5, 4.25, 4.25, 1.75)$
Input Gain, R	$(1, 1, 1.25, 1)$
Prediction Horizon, N_p	30
Frequency, f [Hz]	25

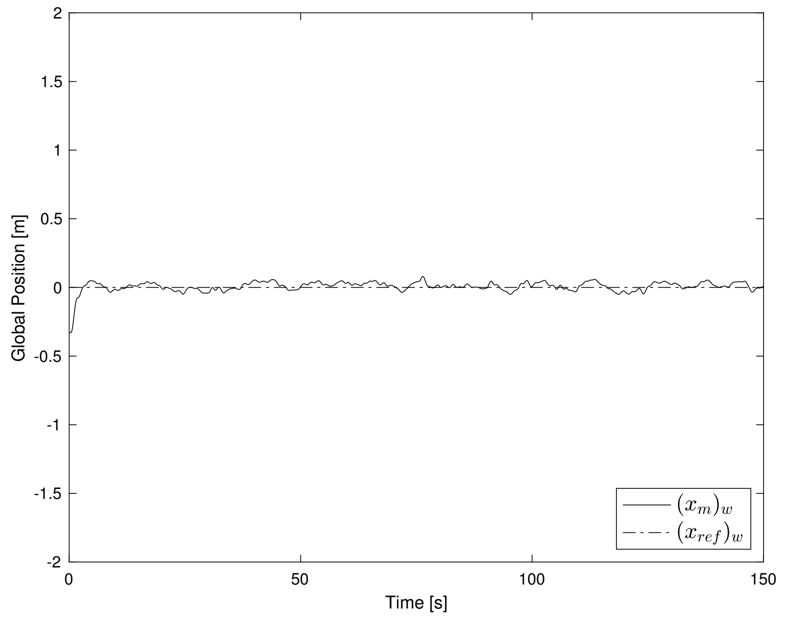


Figure D.9: Global X-Position obtained from LMPC Step Response in the Z-Axis

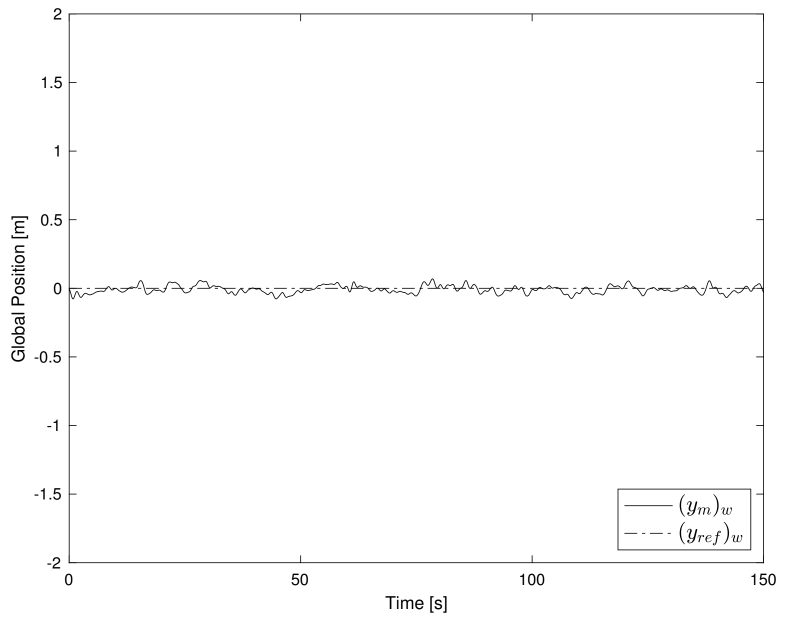


Figure D.10: Global Y-Position obtained from LMPC Step Response in the Z-Axis

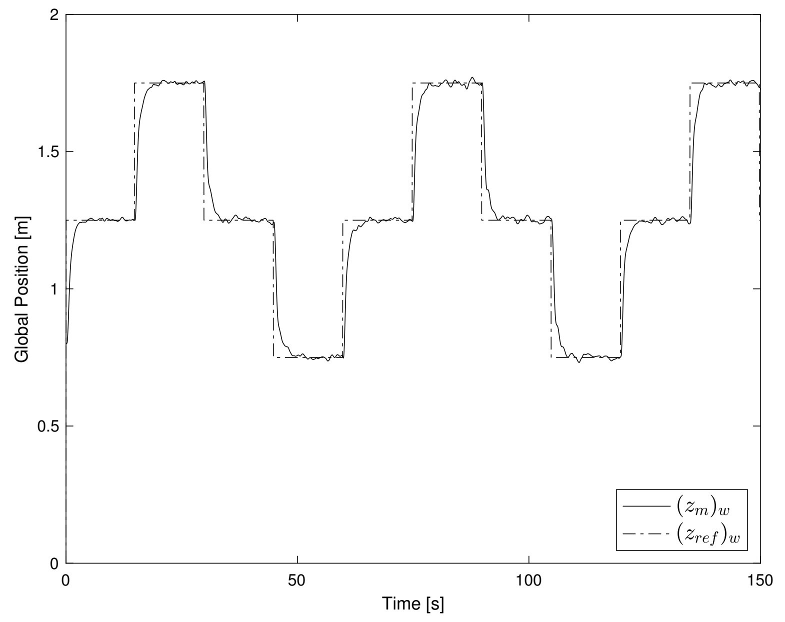


Figure D.11: Global Z-Position obtained from LMPC Step Response in the Z-Axis

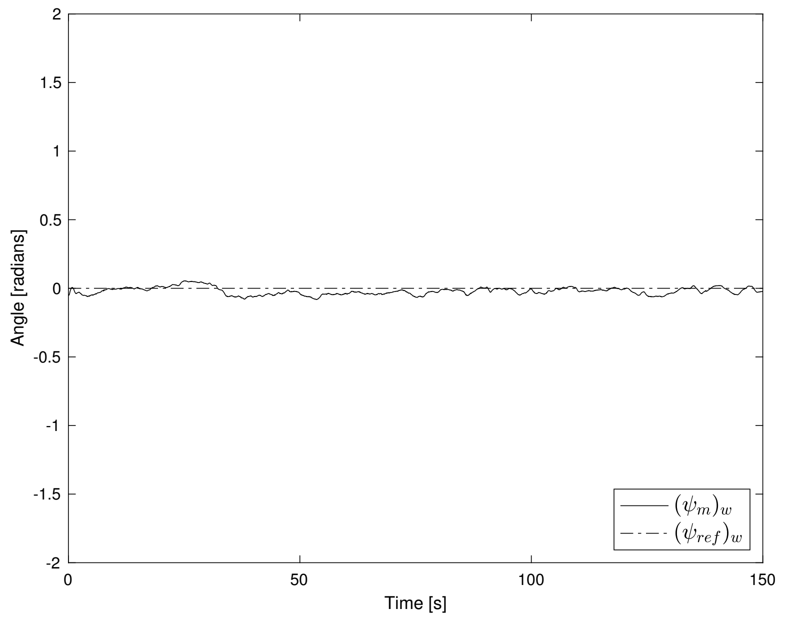


Figure D.12: Yaw obtained from LMPC Step Response in the Z-Axis

D.1.4 LMPC Step Response in Yaw Rotation

Table D.4: LMPC Parameters for Step Response in Yaw

Tuning Parameter	**Value**
State Gain, Q	$(2.25, 2.25, 3.25, 7.5, 4.25, 4.25, 1.75)$
Input Gain, R	$(1, 1, 1.25, 1)$
Prediction Horizon, N_p	30
Frequency, f [Hz]	25

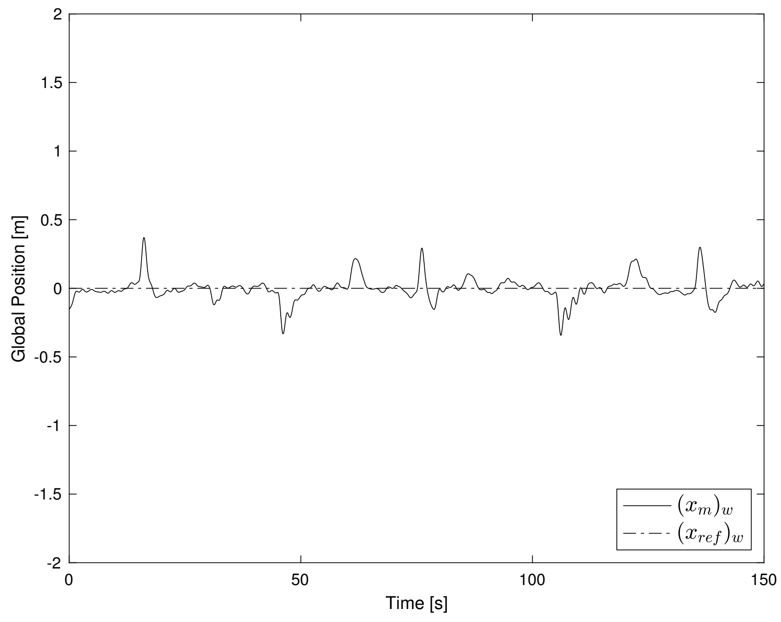


Figure D.13: Global X-Position obtained from LMPC Step Response in Yaw

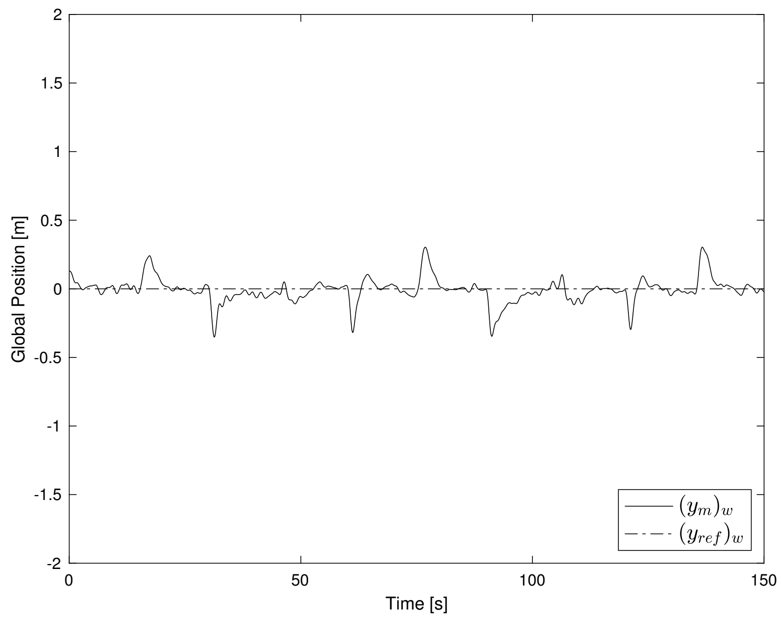


Figure D.14: Global Y-Position obtained from LMPC Step Response in Yaw

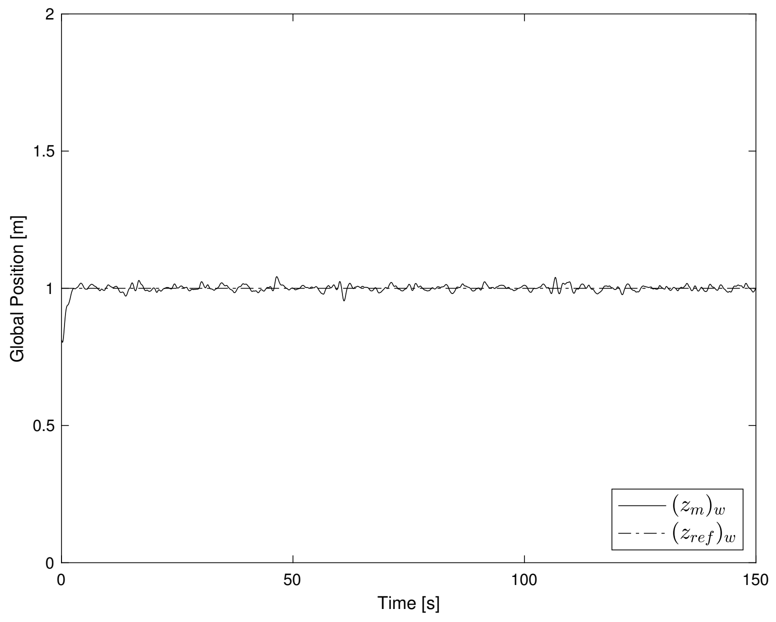


Figure D.15: Global Z-Position obtained from LMPC Step Response in Yaw

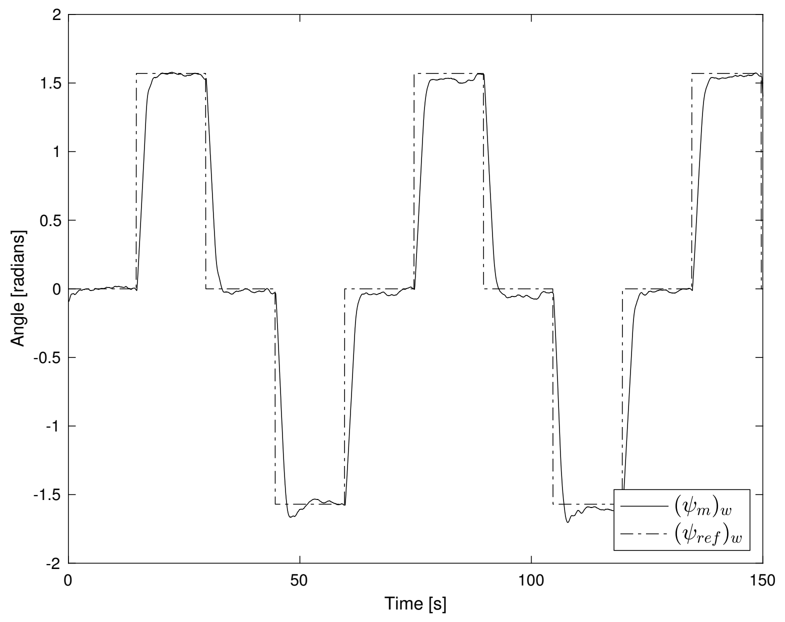


Figure D.16: Yaw obtained from LMPC Step Response in Yaw

D.2 LMPC Figure 8 Trajectory Response

Table D.5: LMPC Parameters for Figure 8 Trajectory

Tuning Parameter	**Value**
State Gain, Q	$(2.25, 2.25, 3.25, 7.5, 4.25, 4.25, 1.75)$
Input Gain, R	$(1, 1, 1.25, 1)$
Prediction Horizon, N_p	30
Frequency, f [Hz]	25

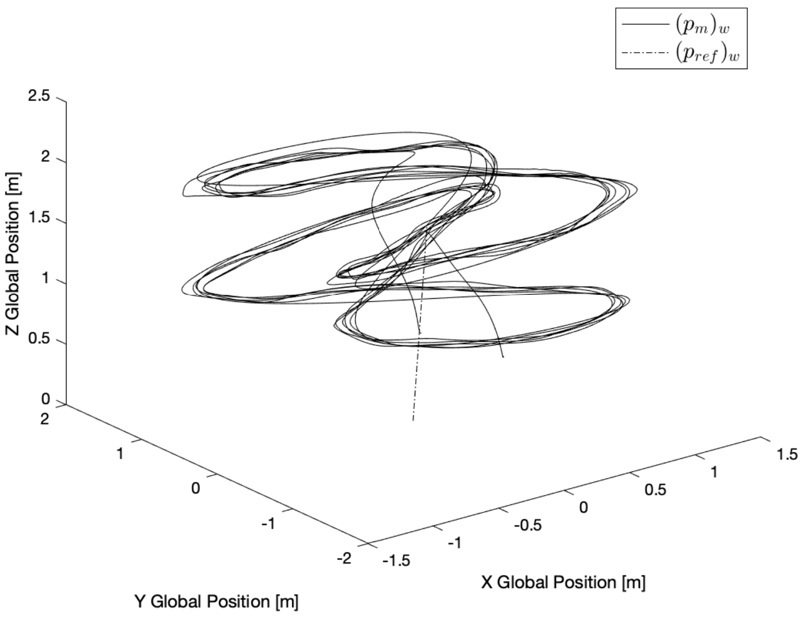


Figure D.17: 3D plot of the LMPC Figure-8 Trajectory Response

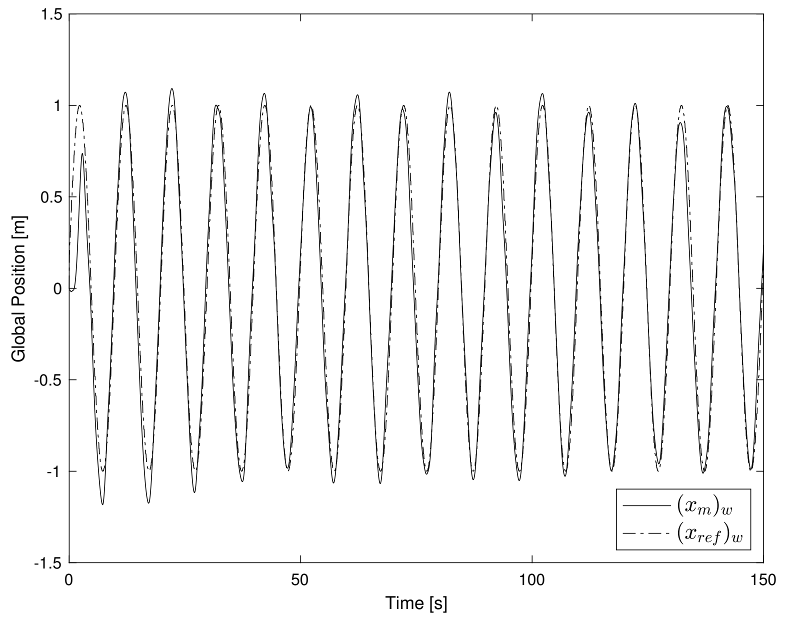


Figure D.18: LMPC Figure-8 Trajectory Response in the X-axis

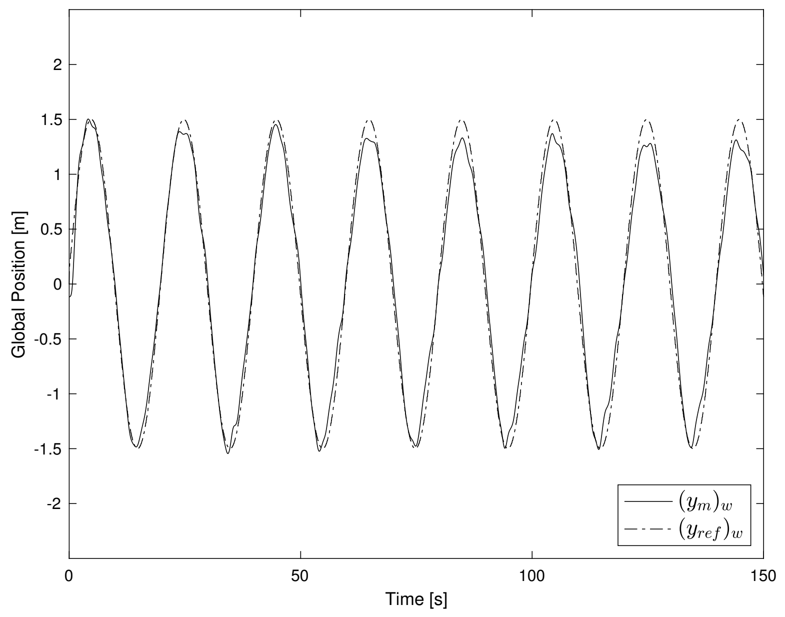


Figure D.19: LMPC Figure-8 Trajectory Response in the Y-axis

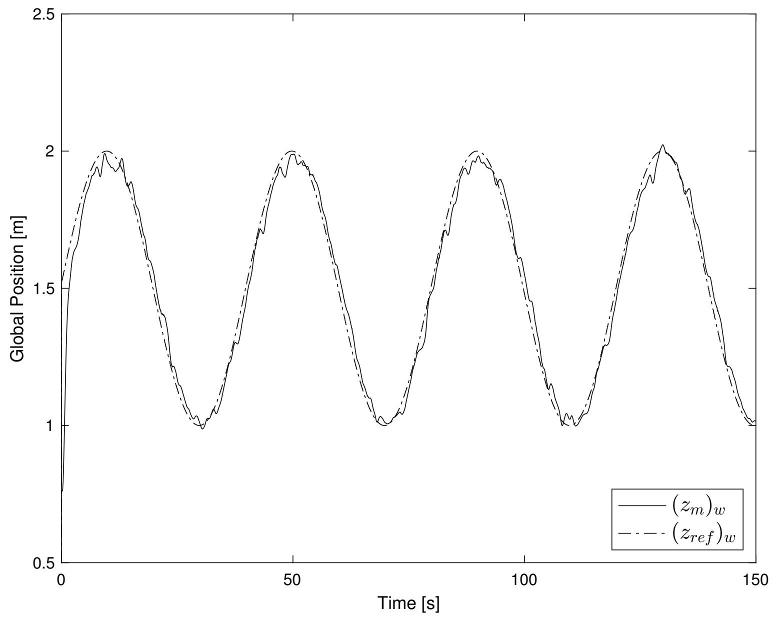


Figure D.20: LMPC Figure-8 Trajectory Response in the Z-axis

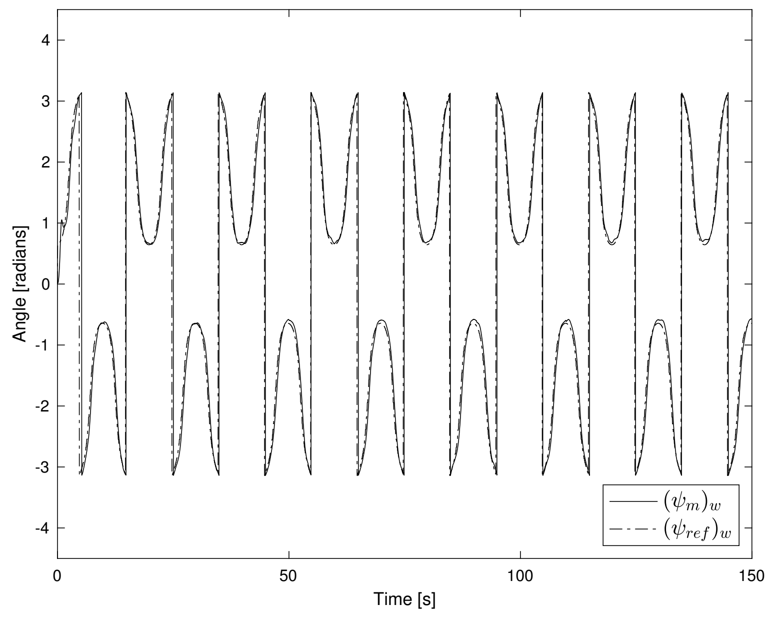


Figure D.21: LMPC Figure-8 Trajectory Response in Yaw

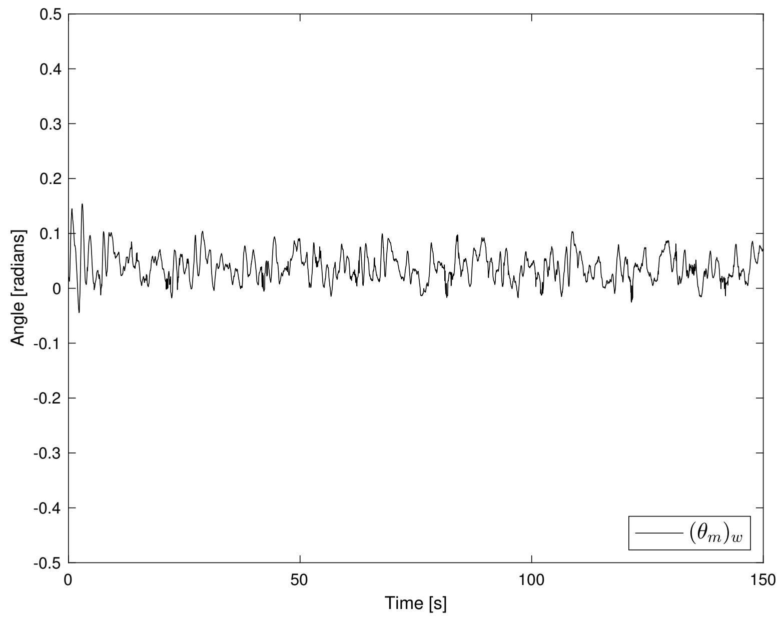


Figure D.22: LMPC Figure-8 Trajectory Response in Roll

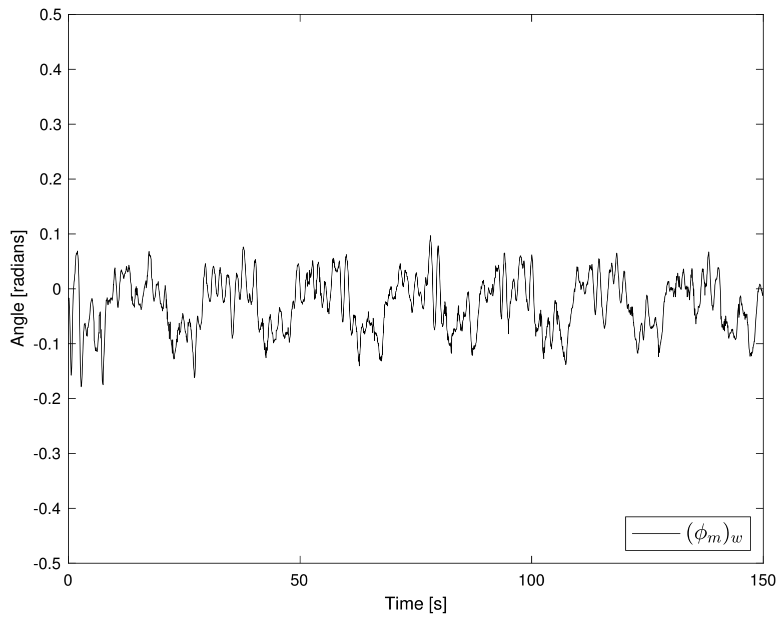


Figure D.23: LMPC Figure-8 Trajectory Response in Pitch

Appendix E

Pursuit Control Data

E.1 Vicon Pursuit Control Data

Table E.1: LMPC Properties and Tuning Parameters for Vicon Pursuit Algorithm

Properties	**Target Drone**	**Pursuer Drone**
State Gain, Q	$(3.25, 3.25, 3.25, 7.5, 5.25, 5.25, 1.75)$	$(2.75, 3.15, 3.25, 7.5, 4.25, 3.75, 1.75)$
Input Gain, R	$(1.25, 1.25, 1, 1.3)$	$(1, 1, 1.25, 1.5)$
Prediction Horizon, N_p	30	30
Frequency, f [Hz]	25	25

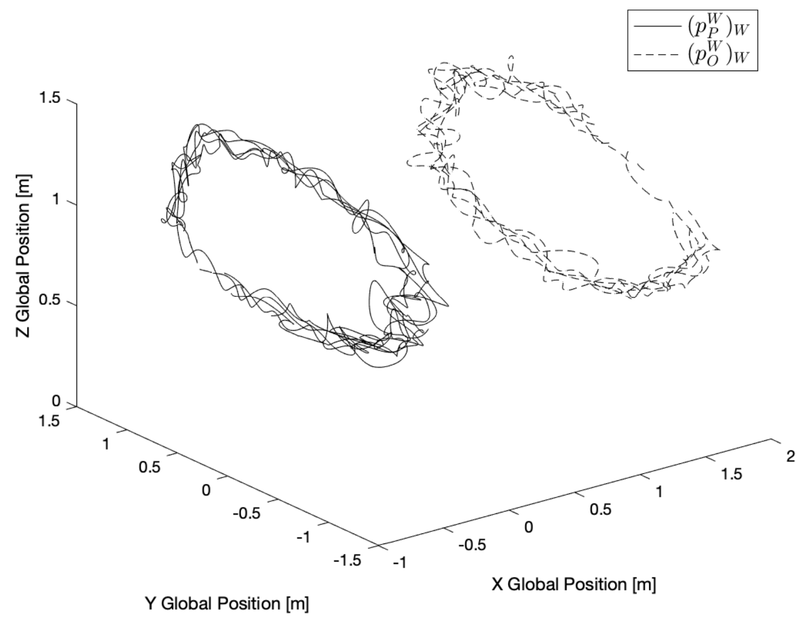


Figure E.1: 3D plot of the Pursuit LMPC Motion Control using Vicon Pursuit Algorithm

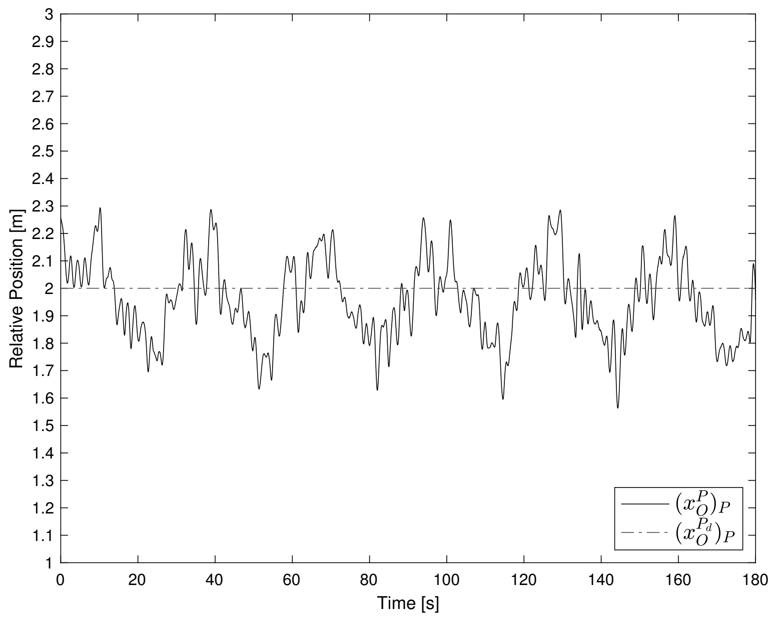


Figure E.2: Relative X-Position using Vicon Pursuit Algorithm

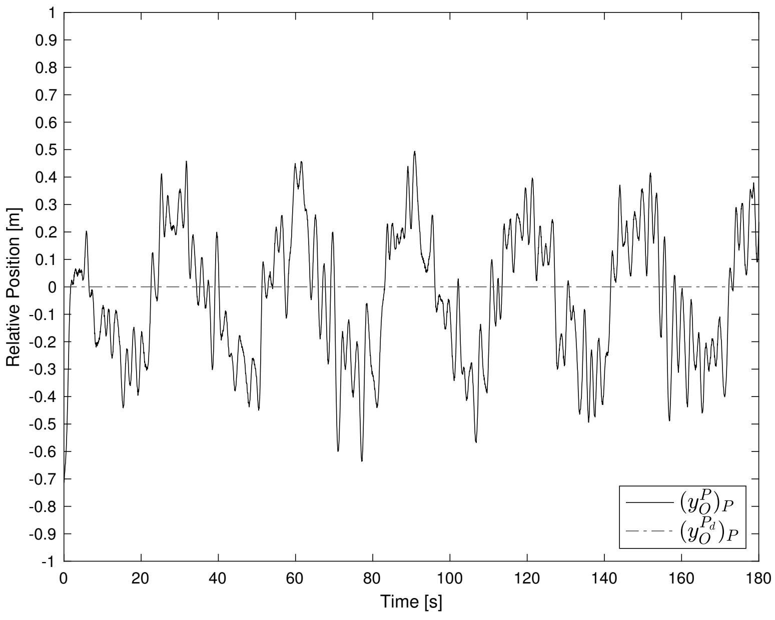


Figure E.3: Relative Y-Position using Vicon Pursuit Algorithm

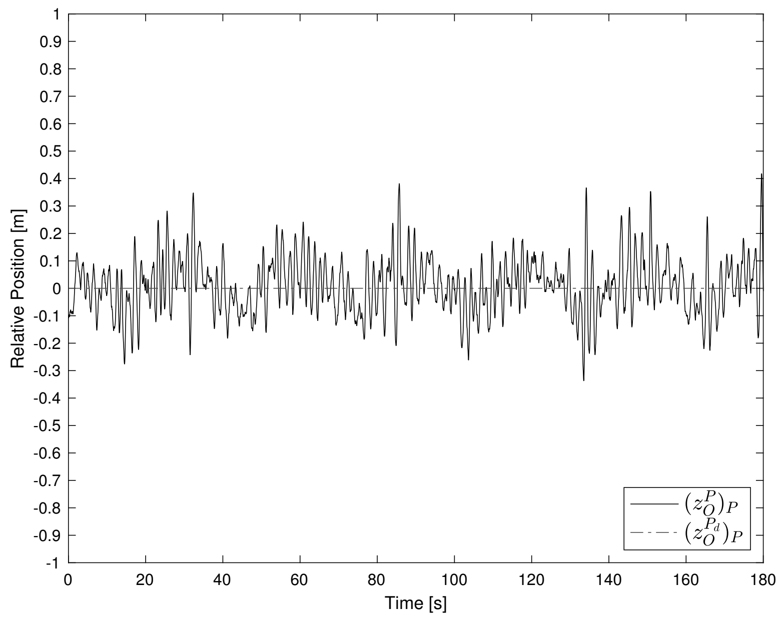


Figure E.4: Relative Z-Position using Vicon Pursuit Algorithm

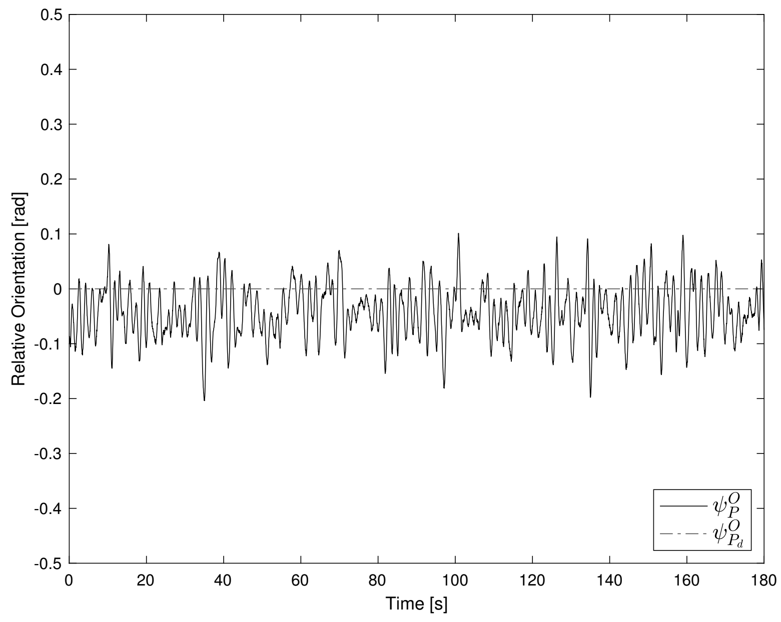


Figure E.5: Relative Yaw Rotation using Vicon Pursuit Algorithm

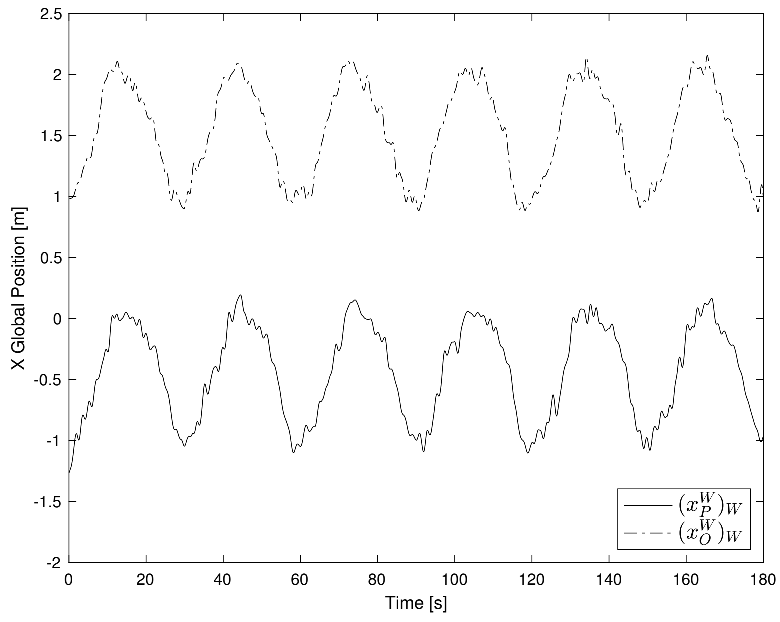


Figure E.6: Global Drone X-Position using Vicon Pursuit Algorithm

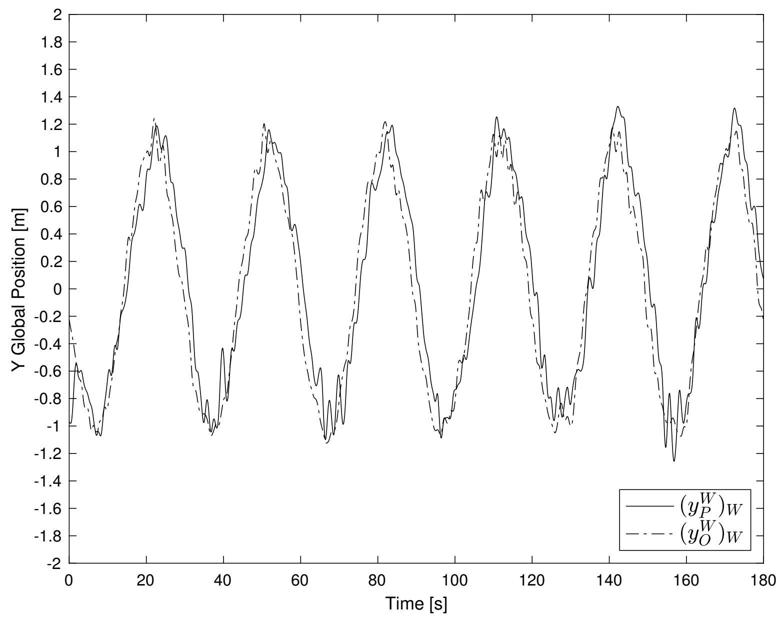


Figure E.7: Global Drone Y-Position using Vicon Pursuit Algorithm

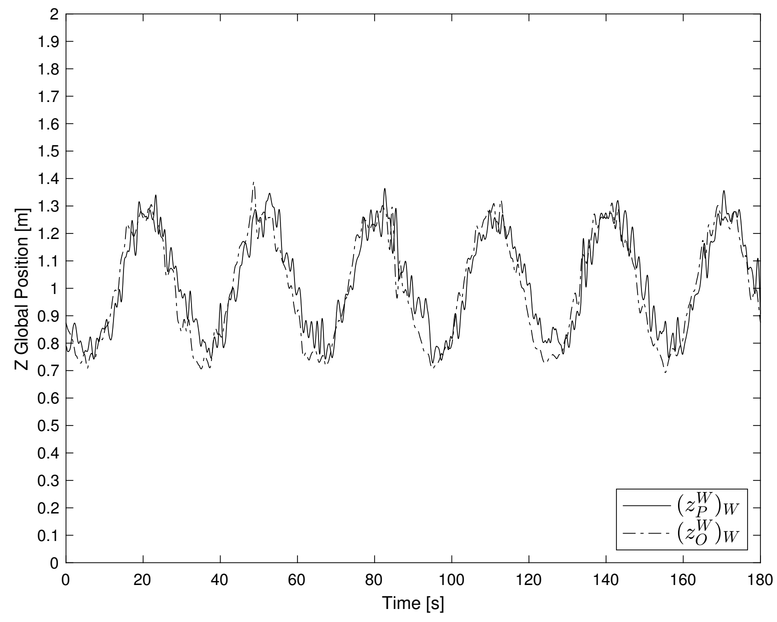


Figure E.8: Global Drone Z-Position using Vicon Pursuit Algorithm

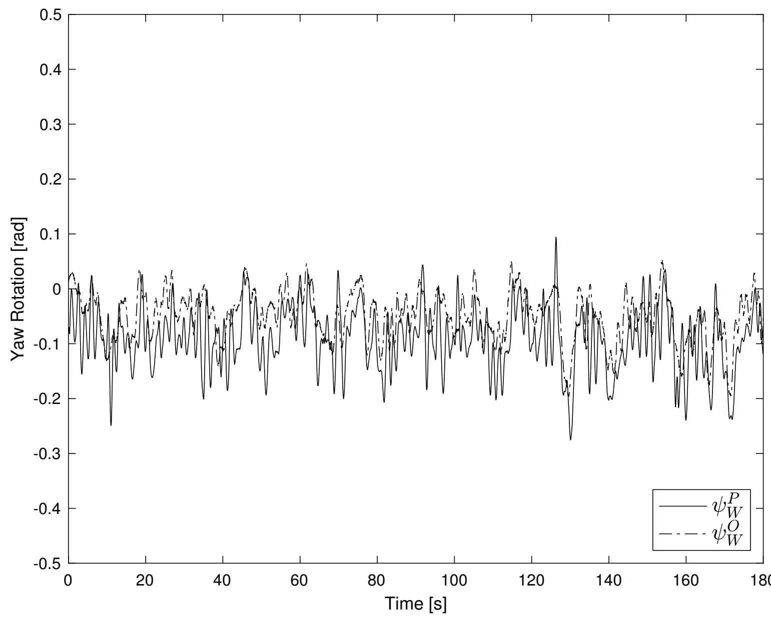


Figure E.9: Global Drone Yaw Rotation using Vicon Pursuit Algorithm

E.2 Bounding Box Pursuit Data

Table E.2: LMPC Properties and Tuning Parameters for Bounding Box

Properties	**Target Drone**	**Pursuer Drone**
State Gain, Q	$(2.75, 3.15, 3.25, 7.5, 4.25, 3.75, 1.75)$	$(3.25, 3.25, 3.25, 7.5, 5.25, 5.25, 1.75)$
Input Gain, R	$(1, 1, 1.25, 1.5)$	$(1.25, 1.25, 1, 1.3)$
Prediction Horizon, N_p	30	30
Frequency, f [Hz]	25	25

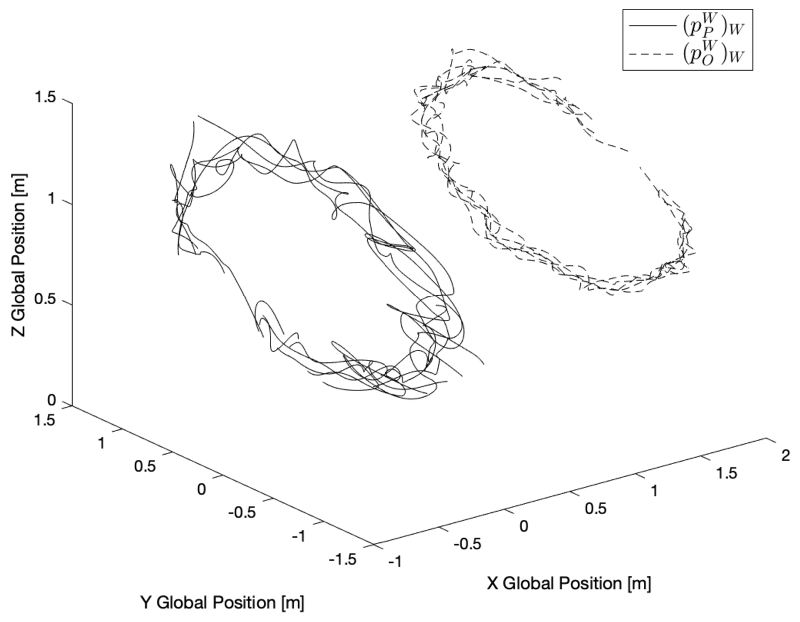


Figure E.10: 3D plot of the Circular Pursuit Motion using Bounding Box Pursuit Algorithm

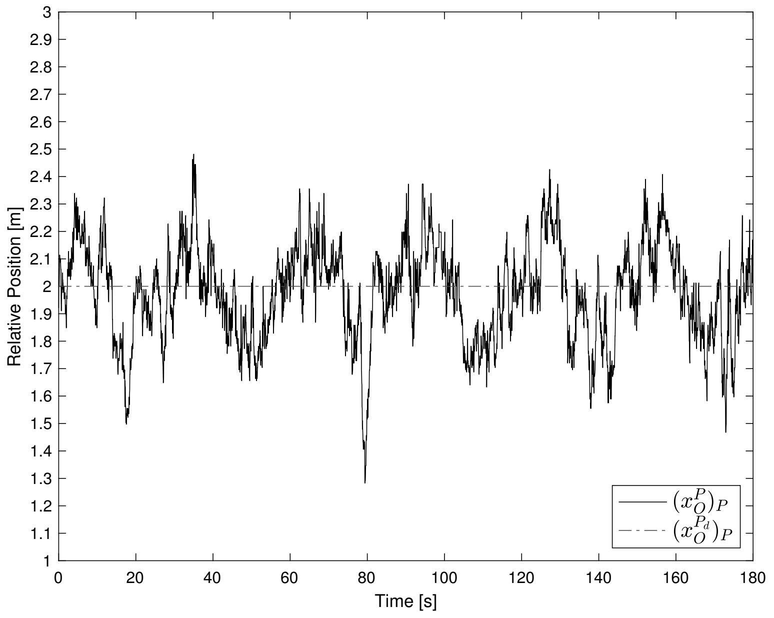


Figure E.11: Relative X-Position using Bounding Box Pursuit Algorithm

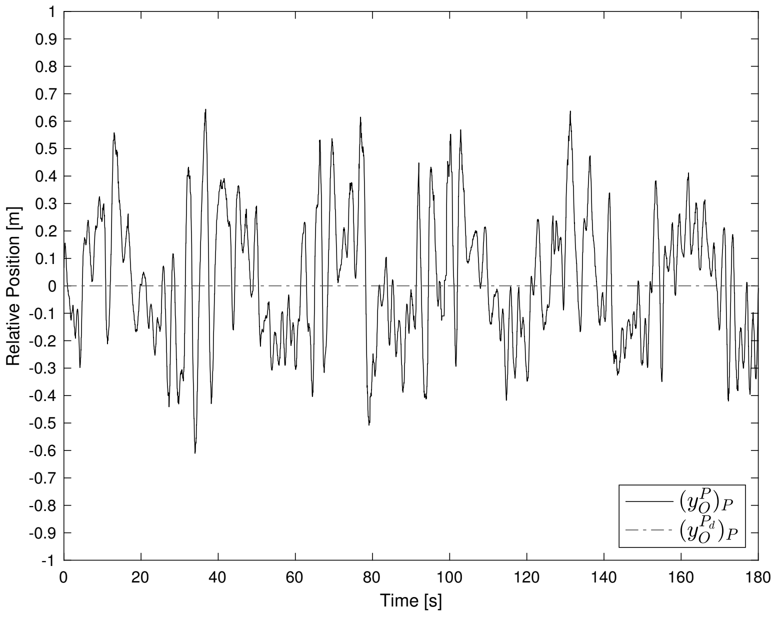


Figure E.12: Relative Y-Position using Bounding Box Pursuit Algorithm

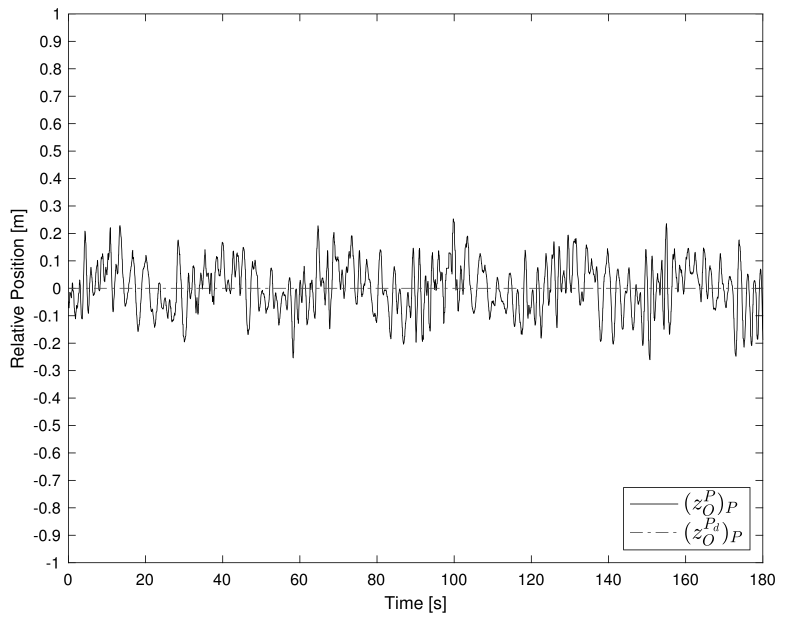


Figure E.13: Relative Z-Position using Bounding Box Pursuit Algorithm

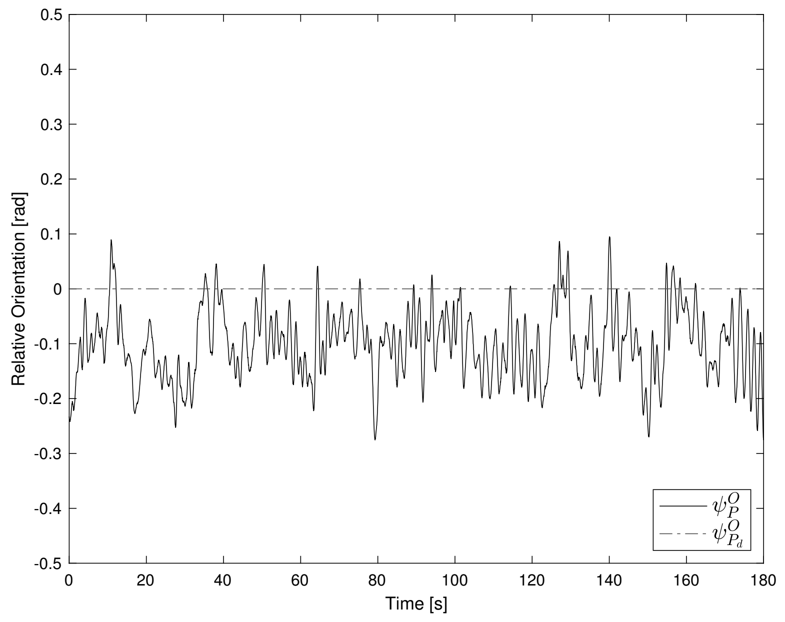


Figure E.14: Relative Yaw Rotation using Bounding Box Pursuit Algorithm

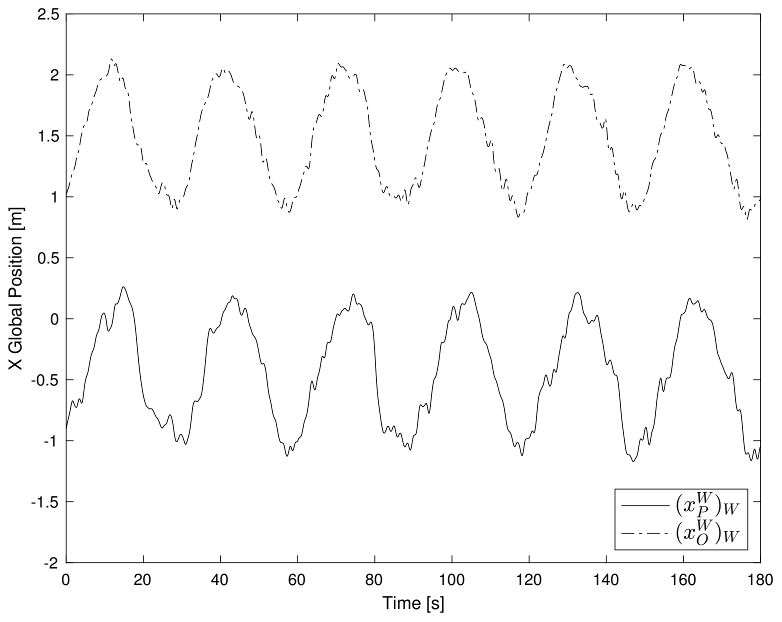


Figure E.15: Global Drone X-Position using Bounding Box Pursuit Algorithm

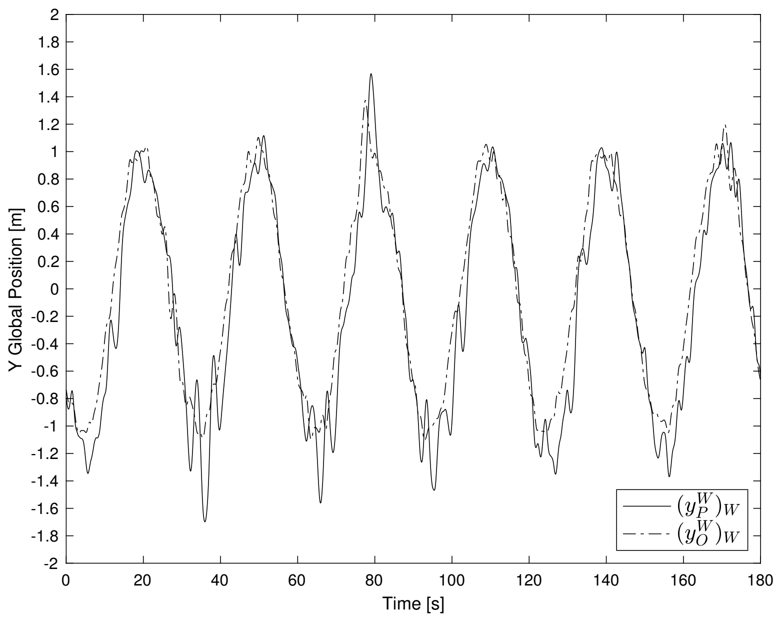


Figure E.16: Global Drone Y-Position using Bounding Box Pursuit Algorithm

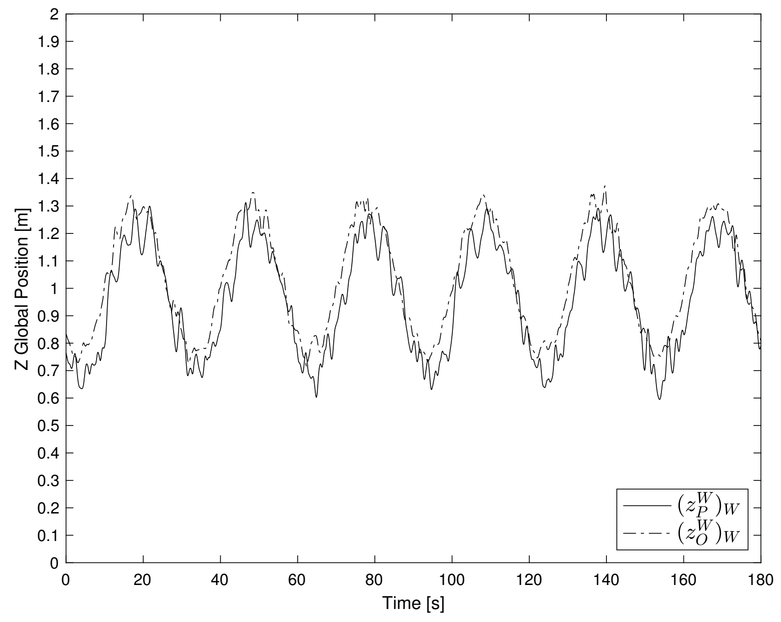


Figure E.17: Global Drone Z-Position using Bounding Box Pursuit Algorithm

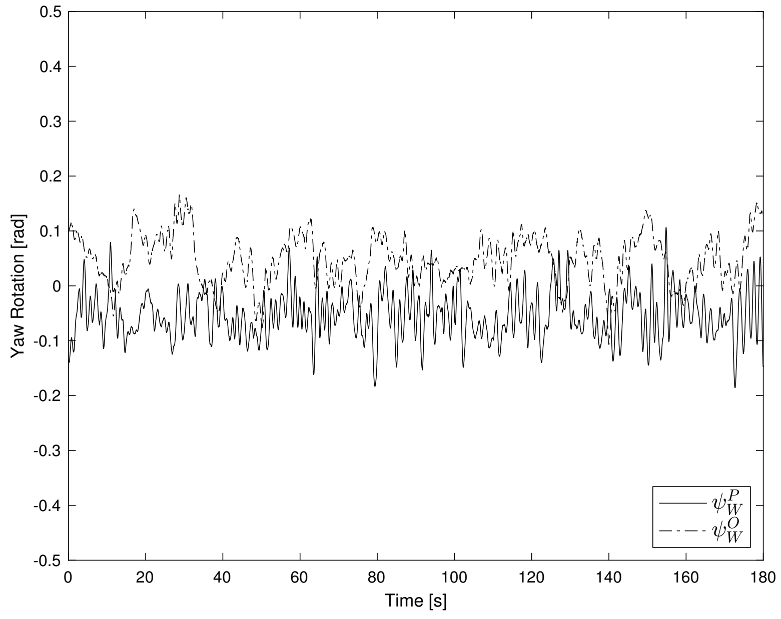


Figure E.18: Global Drone Yaw Rotation using Bounding Box Pursuit Algorithm

Appendix F

Bounding Box Pursuit Algorithm Step Performance

F.1 Step Evaluation of the Bounding Box Pursuit Algorithm

The performance of the bounding box motion control algorithm is evaluated by having a pursuer drone follow a target drone that is subjected to a step input in each direction. The target drone performed a step response using a LMPC motion controller. As shown in Chapter 5, the Parrot AR.Drone 2.0 is capable of performing a step response. The LMPC motion controllers for the pursuer and target drone are tuned for each direction to obtain the optimum performance. This evaluation will assess the performance of the bounding box pursuit algorithm in each direction independently. The step trajectory of the target drone is described by the following expressions.

$$x_{x,y,z} = \begin{cases} x_{low}, & 30n \leq t \leq 15 + 30n, \\ x_{mid}, & 15 + 60n \leq t \leq 30 + 60n, \quad n = 0, 1, 2... \\ x_{high}, & 45 + 60n \leq t < 60 + 60n, \end{cases}$$

where t is time in seconds, and n is an integer value. The values for x_{low}, x_{mid}, and x_{high} for each direction and yaw rotation is described in Table F.1.

Table F.1: Step Values for Each Direction and Yaw Rotation

Step Direction/Rotation	x_{low}	x_{mid}	x_{high}
Global X-Direction, X_W [m]	-1	0	1
Global Y-Direction, Y_W [m]	-1	0	-1
Global Z-Direction, Z_W [m]	0.75	1.25	1.75

F.2 Bounding Box Pursuit Algorithm Step Response Performance

Similarly to Sections 5.4.1 and 5.3.1 in Chapter 5, the target drone performed a step response in each direction and in yaw using a LMPC motion controller while the pursuer drone replicates the motion of the target drone. The target drone was subjected to a sequence of step reference inputs in one direction or rotation while holding all other directions and/or rotation steady in a user-specified position. The pursuer drone without knowing any prior knowledge of the movement of the target drone attempts to replicate its motion. The global world positions of the pursuer and target drone are shown in Figures F.1, F.2 and F.3.

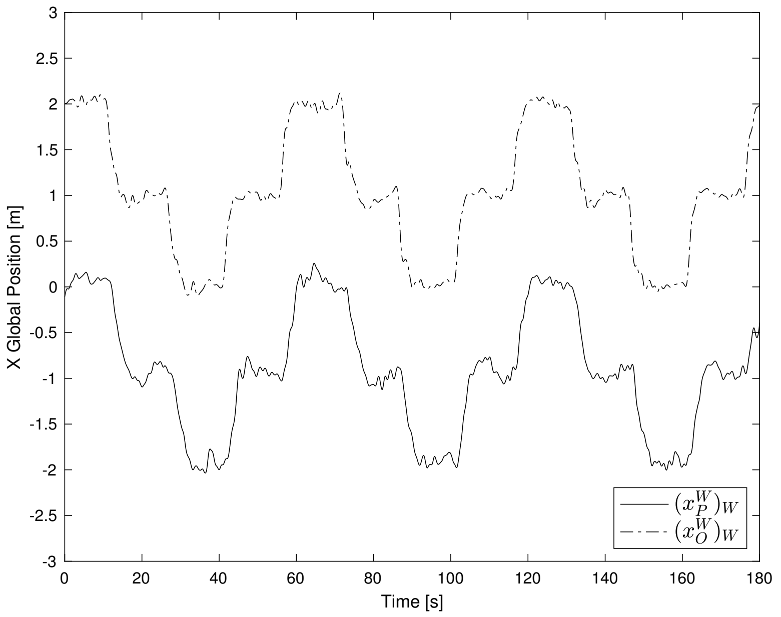


Figure F.1: Global World X-Position of the Pursuer and Target Drone

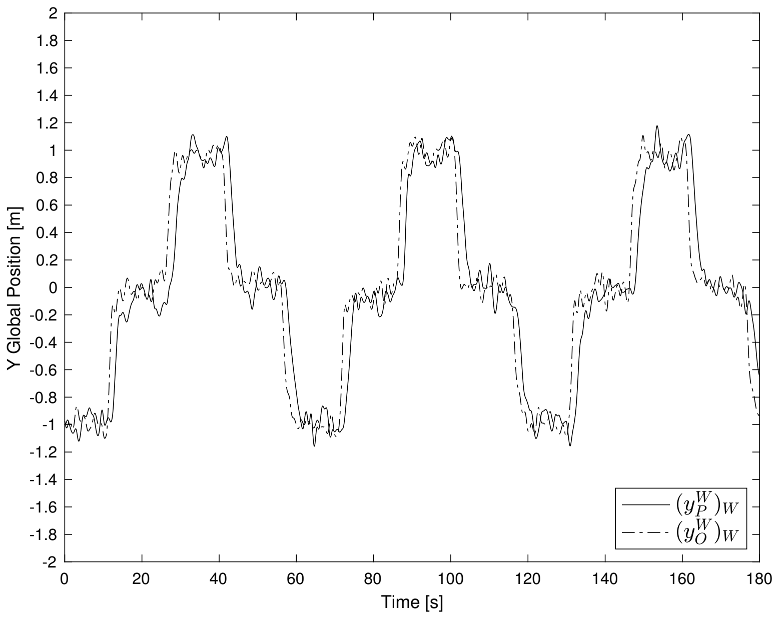


Figure F.2: Global World Y-Position of the Pursuer and Target Drone

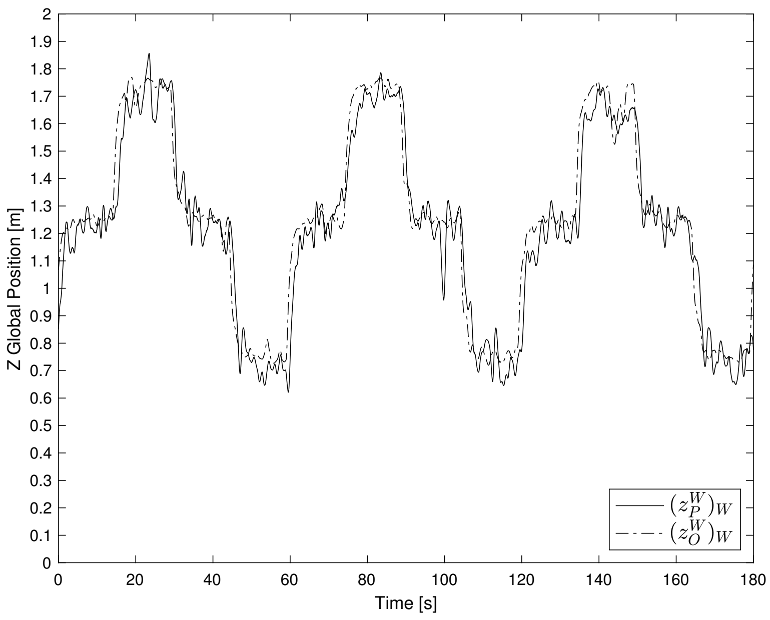


Figure F.3: Global World Z-Position of the Pursuer and Target Drone

Figures F.1, F.2,and F.3 suggest that the pursuer drone followed the motion of the target drone. The following table summarizes the performance of pursuer drone in maintaining a relative distance from the target drone performing a step response.

Table F.2: Pursuer Drone Performance Summary for Tracking a Step Response

Parameter	**Average Tracking Error**	**RMS Error**
Step Response along the Global X-Axis		
$(x_P^O)_O$ [m]	-0.0134	0.2264
$(y_P^O)_O$ [m]	0.0266	0.1399
$(z_P^O)_O$ [m]	-0.0019	0.0877
Step Response along the Global Y-Axis		
$(x_P^O)_O$ [m]	-0.0011	0.1153
$(y_P^O)_O$ [m]	0.0120	0.2669
$(z_P^O)_O$ [m]	-0.0019	0.0801
Step Response along the Global Z-Axis		
$(x_P^O)_O$ [m]	-0.0409	0.1004
$(y_P^O)_O$ [m]	0.0440	0.1407
$(z_P^O)_O$ [m]	-0.0081	0.1189

Based on the average tracking errors in each case being relatively small, the pursuer drone was capable of following the target drone at a desired relative position away from the target drone. When the RMS errors in Table F.2 and Figures F.1, F.2,and F.3 are taken into consideration, the data suggest that the system experienced significant noise in each case. This noise is arises from a number of factors that involve the performance of YOLO v2, the presence of disturbances, possible time delays, and camera feed drop outs.

From Table F.2, the RMS error in the relative X-direction and relative Y-direction were relatively high in the cases where the pursuer drone was following a target drone performing a step response in the global X- and Y-direction, respectively. The higher RMS errors in this case were due to the pursuer drone not reacting quickly enough to the movement of the target drone. It was found that the pursuer drone was capable of maintaining a reference distance away from the target drone when it was hovering. The performance of the pursuer drone in reacting to the target drone could be improved with tuning, developing a kinematic model of the target drone, or considering the effects that time delay may have on the system. Optimally tuning the system, developing a kinematic model of the target drone using the YOLO v2 object detection algorithm is left for future work, and exploring the effects of time delay are left for future work.

It was observed during the course of the experiment, the gusts and wakes from the target drone were disturbing the motion of the pursuer drone. These disturbances appeared to have impacted the pursuer drone more when following the target drone along the global X direction than when following the target drone along the global Y direction. In addition, when the target drone flew above the pursuer drone, the gusts emitted by the target drone disturbed the pursuer drone and swayed the drone in the X- and Y-direction. These disturbances may partly explain the RMS errors found in the directions that were set a constant reference value.

Another possible contributor to higher RMS errors is the inaccuracies in the bounding box information. From Section 4.5.2, the lateral position estimation was not perfect which introduces a systematic error into the system. This suggests that the YOLO v2 can provide poor data that affects the performance of the control system. In addition, as seen in Section, the further away the pursuer drone is to the target drone, the errors in the bounding box information become exponential. Since the position of the target drone is dependent on the depth estimation as shown in Section 4.4, depth estimation from an inaccurate bounding box data leads to poorer performance in pursuit algorithm. Future work could consist of training YOLO v2 further or implementing a Extended Kalman Filter on the bounding box information.

Despite the number of issues that are present within the system, the pursuit drone is capable of maintaining a relative position away from a target drone performing a step response.

Appendix G

Experimental Error Equations

The performance of the motion controllers described in Chapter 3 are evaluated based on their ability to follow a desired reference. A method to assess the motion controllers' ability to follow a desired reference is by determining the average tracking error and the root mean square error.

The tracking error within the context of control is the difference between the reference input state and the measured output state at a given time. The average tracking error is found by averaging out the tracking error at each given time. The average tracking error is calculated using Equation (G.1) as shown below.

$$e_{ave} = \frac{\sum_{k=1}^{N}(x_{ref} - x_m)}{N} \tag{G.1}$$

Where e_{ave} is the average tracking error, x_{ref} is the reference state, and x_m is the measured state.

The root mean square (RMS) error is the standard deviation of the tracking error and measures the spread of the tracking error. The RMS error is calculated using Equation (G.2) as shown below.

$$RMS = \sqrt{\frac{\sum_{k=1}^{N}(x_{ref} - x_m)^2}{N}} \tag{G.2}$$