Deep Learning in Autonomous UAV Pursuit

by

Amir Hossein Ebrahimnezhad

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

Master of Science

Department of Mechanical Engineering
University of Alberta

Abstract

Unmanned aerial vehicles or UAVs have largely become and continue to be an inseparable part of modern warfare, security and surveillance systems, first aid response, aerial cinematography and many other sectors. Therefore, achieving full autonomy for UAVs and drones would ensure mass mobilization and utilization of these devices in large scale applications with more efficiency and precision without the need for deploying extensive human resources. A key aspect of autonomous flights is the capability of performing autonomous pursuits of target UAVs for military and civil purposes. Conventional autonomous pursuit algorithms rely on utilizing LiDAR , radar and ultrasonic sensors or a combination and fusion of these devices. Each of theses sensors come with their disadvantages and shortcoming including a limited range of sight, massive data processing, expense and vulnerability to environment conditions. Thus, this research proposes a new sensor-free and vision-based algorithm for accomplishing fully autonomous UAV pursuit. This algorithm consists of two major parts including control and pose estimation. The flight controllers incorporate a digital four-axis proportional integral derivative (PID) framework and the pose estimator utilizes region-based convolutional neural networks (RCNN) for estimating a 3D bounding box over the target. The 3D bounding box keypoints are then extracted and combined with perspective-n-point (PnP) algorithm for estimating the precise relative pose of the target and pursuing it accordingly.

Preface

This thesis is an original work by Amir Hossein Ebrahimnezhad. The research and experiments were conducted in the Mechatronic Systems Lab at the University of Alberta under the guidance of Dr. Martin Barczyk. Some part of the C++ codes on Vicon and PID control in the developed ROS packages `sphinx_ros` and `anafi_ros` of this research are based on `ar_drone_ros` package developed by Christopher Surma.

Chapters 4 and 5 of this thesis are to be submitted to Robotics and Autonomous Systems journal as "Deep Real-time Pose Estimation and Pursuit of Target UAVs Using Region-based Convolutional Neural Networks and Perspective-n-Point" by Amir Hossein Ebrahimnezhad and Martin Barczyk.

Acknowledgements

Utmost appreciation and gratitude to my distinguished supervisor Dr. Barczyk for his immense and extensive support, supervision, guidance, instruction and invaluable insights throughout my research and study. It has been truly a great honour to be a member of Mechatronic Systems Lab. Special thanks to our summer interns Lucas Gandrey, Debasmita Chatterjee and Aravindh Ganesan for their enthusiasm and dedication on this research, and to my labmates Younes, Shane, Charanjot and Ivy for their encouragement and helpfulness. And last but certainly not least, I would like to thank my parents and my brother for their support and kindness during my education.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Nomenclature

Symbol

$\breve{d}_a$	Estimated Offline Axial Distance
$\breve{x}_a$	Estimated Offline Axial x Position
$\breve{y}_a$	Estimated Offline Axial y Position
$\breve{z}_a$	Estimated Offline Axial z Position
$\hat{d}_a$	Estimated Real-Time Axial Distance
$\hat{x}_a$	Estimated Real-Time Axial x Position
$\hat{y}_a$	Estimated Real-Time Axial y Position
$\hat{z}_a$	Estimated Real-Time Axial z Position
$\mathcal{A}$	Bebop Coordinate Frame
$\mathcal{B}$	Anafi Coordinate Frame
$\mathcal{C}$	Camera Coordinate Frame
$\mathcal{W}$	World Coordinate Frame

Subscripts

ext	Extrinsic
int	Intrinsic
ref	Reference

Abbreviations

ANN Artificial Neural Network.

API Application Programming Interface.

CNN Convolutional Neural Network.

CUDA Compute Unified Device Architecture.

DFT Discrete Fourier Transform.

DNN Deep Neural Network.

DTFT Discrete-Time Fourier Transform.

GPS Global Positioning System.

GPU Graphics Processing Unit.

GUI Graphical User Interface.

IDFT Inverse Discrete Fourier Transform.

IDTFT Inverse Discrete-Time Fourier Transform.

IMU Inertial measurement Unit.

IoU Intersection over Union.

LiDAR Light Detection and Ranging.

LMPC Linear Model Predictive Control.

LQR Linear Quadratic Regulator.

LTI Linear Time-Invariant.

MLP Multilayer Perceptron.

MPC Model Predictive Control.

OKS Object Keypoint Similarity.

PID Proportional, Integral, Derivative.

PLC Programmable Logic Controller.

Radar Radio Detection and Ranging.

R-CNN Region Based Convolutional Neural Network.

ResNet Residual Neural Network.

RoI Region of Interest.

ROS Robot Operating System.

SDK Software Development Toolkit.

UAV Unmanned Aerial Vehicle.

Chapter 1

Introduction

1.1 Background

Around 555 millions years ago, during the Ediacaran period the first creature with vision appeared on earth [1]. They were called trilobite and soon their offsprings dominated nature due to their visual superiority. The evolution of vision in animals resulted in the creation of a collection of clustered neurons to process the input data of the photosensitive cells, later this clustered organ became what we know today as brain [2].

Throughout millions of years of evolution, both brain and and visual organs have been evolved in a way to enable animals to better perceive and monitor the surrounding environment. A key ability to better perceive the surroundings is the ability to detect objects and creatures as well as obtaining an estimation of their pose and relative distance. This ability has enabled predators to better target and hunt other animals.

Inspired by nature, scientists and engineers have been trying to implement the same functionalities of natural vision in industrial and domestic applications. In fact, the focal domain of interest of imitating natural image processing lays in field of Robotics where the main purpose is to develop robots to automatically perform the required tasks which were previously performed manually by animals or humans.

One of the pillars of autonomy is depth and pose estimations of objects and en-

tities in the surrounding environments. In fact, all animals have some level of pose estimation to perform the necessary tasks like grabbing tools, food or hunting a prey.

From robotics perspective, depth and pose estimation is a vital element for achieving full autonomy in autonomous vehicles. Considering the autonomous cars, it is needed for the car to simultaneously obtain the pose and depth of all the considerable objects including humans, vehicles, obstacles, etc. Therefore, only by obtaining the proper estimation of all important objects around the vehicle can we control the autonomous vehicle in a safe and reliable way.

1.2 Motivation and Challenge

For enabling the autonomous machine to interact with the environment, we need to equip the machine with sensors. Sensors are devices that translate one signal to another and are used to measure the required variables from the environment.

After the invention of laser in 1960 [3], light-based distance sensors were then developed to measure the distances by emitting light. Subsequently, by fusing the new invention with the principles of radar, LiDAR was thereafter invented in 1980s.

LiDAR is in fact a remote-sensing technique for obtaining the distance of surrounding entities in the environment [4]. It utilizes Laser and radar principles to obtain the distance of the objects by emitting and absorbing light beams. Subsequently, the data is collected and represented as a point cloud map.

A common solution for the autonomous pose estimation in robotics is to fuse LiDAR data with the camera vision. In this algorithm, the objects are first detected and classified using deep learning algorithms and then by projecting the LiDAR point cloud data into the camera, the distance of each pixel on the frame could be retrieved [5].

Although this method has been widely used recently in self-driving cars to fulfill autonomy in cars, it has its own obstacles and challenges. LiDAR is relatively expensive and requires huge amount of processing power and is eventually dependent

on visual data for interpreting the point cloud data.

During Tesla Autonomy Day on 22nd of April 2019, Elon Musk pointed out that "LiDAR is lame, it's expensive and unnecessary and once you solve the vision it's worthless"[6]. In addition to the expense of LiDAR , it has more challenges in aerial robotics. Most prominently, due to its relative high weight, it would cause instability and would affect the agility and maneuverability of the aerial vehicles. Therefore, it is needed to develop a LiDAR -free method to obtain autonomy and pose estimation in aerial vehicles and robots.

1.3 Literature Review

Artificial intelligence and neural networks have been a focal point of research during the past five decades and is an ever-growing filed of research. Neural networks have become the key element in intelligent robotics and are used for image processing, motion control, path planning and other aspects of autonomous robots including autonomous cars, drones, etc. In this section, we shall investigate and review the recent publications on object detection, pose estimation and data extraction using convolutional neural networks, region-based neural networks and residual neural networks.

1.3.1 Object Detection

The first step to obtain the pose of an object, is to detect the object itself visually. To do so, we need to utilize a computational tool called the artificial neural networks. An artificial neural network is a computational tool inspired by the biological neural networks in humans and other animals [7]. An artificial neural network is a network of clustered neurons, each equipped with weight and an activation function [7].

An artificial neural network with two hidden layers is represented in Figure 1.1. As it is shown, the network has four inputs and two outputs. By providing the proper training data, the network would be enabled to translate the input vector to that of intended output. Increasing the number of hidden layers and hidden neurons could

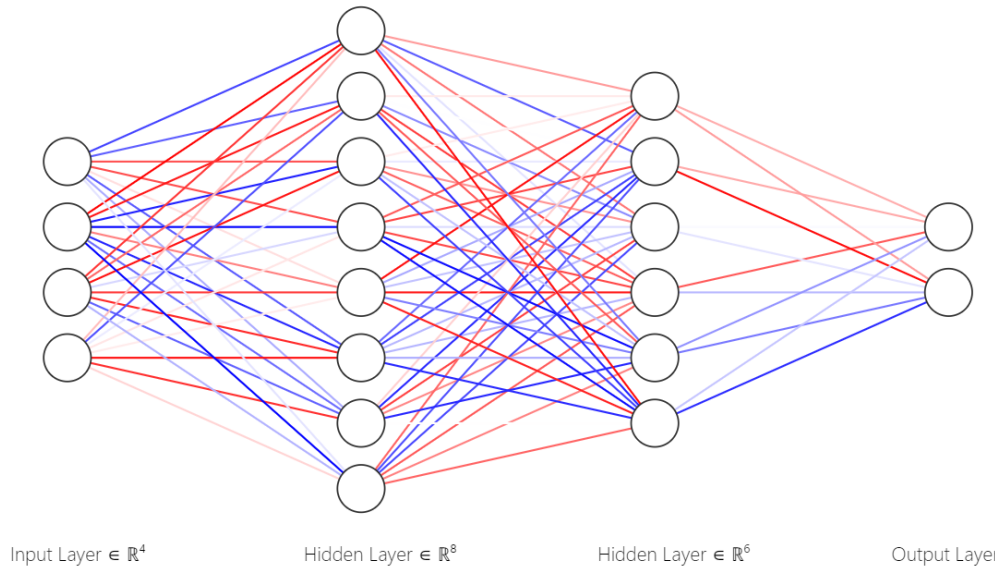


Figure 1.1: Schematic representation of an artificial neural network with two hidden layers [8]

increase the accuracy of the model but it would require more processing power at the time.

Using the proper input and corresponding labeled output data, the network could be trained using gradient descent method to tune the weights in order enable the network to perform the required calculations. This method is called supervised learning as opposed to the unsupervised learning which is based on using untagged output data. Supervised learning is commonly used for speech analysis, image and speech recognition, whereas unsupervised learning is used for picture imagination, video generation and speech synthesis [9].

As it is shown in Figure 1.2, a convolutional network could consist of different layers including convolution, max-pooling and dense layers. The purpose of applying convolution in neural networks is to extract the required feature from the input image. This takes considerably less amount of computation than resizing the input image into a vector and applying it into a fully connected neural network.

This unique feature of convolutional networks has paved the way for the design

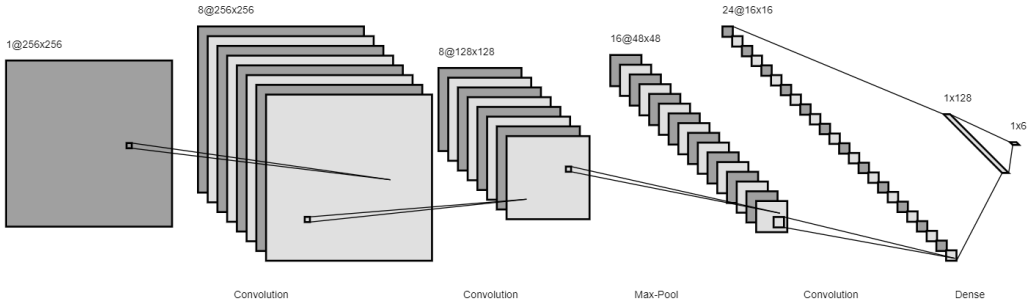


Figure 1.2: Schematic representation of a convolutional neural network [8]

and utilization of more complex network for more advanced image processing task including object detection.

After years of advancements in object detection area using convolutional neural networks, the two prominent CNN object detection architectures are region-based methods including R-CNN and fast R-CNN and regression/classification based methods including YOLO and DSSD as it is shown in Figure 1.3 [10][11][12][13].

Single stage object detection methods are based on applying one-step regression on the whole input frame rather than segmenting the input frame into separate regions. Through YOLO , the input frame is first processed through CNN layers and then feature map is extracted and therefore fed into fully connected layers for obtaining classes and bounding boxes. Single-step methods have higher speed compared to two-step methods, although they are less precise and cannot extract more detailed features including the keypoints and masks [12] [14].

On the contrary to the single-step object detection methods, the two-step algorithms first divide the input image into different RoIs (region of interests) and then would perform the object detection and classification separately on each RoI [14].

Although performing two-step object detection requires more processing time and power, it is much more accurate and has less localization accuracy compared to single-

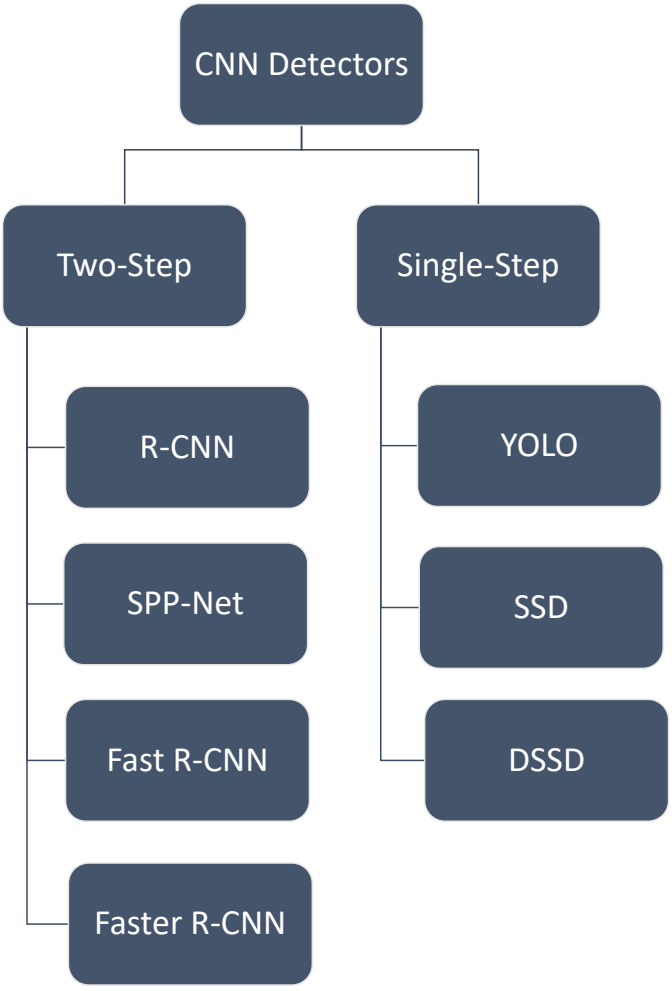


Figure 1.3: Classification chart of CNN Detectors [12]

step methods [12]. Furthermore, more features could be extracted using two-step methods including instance segmentation, semantic segmentation and keypoints.

1.3.2 Pose Estimation

As mentioned earlier, pose estimation is an essential and necessary tool for achieving autonomous robotics. The applications of pose estimation are huge and so are the research and papers conducted on this area.

Object grasping is an essential task in industrial robotic which is reliant on pose

estimation of the target objects. Wang et al. have developed a method to obtain pose estimation through RGB-D frames. RGB-D frames have four layers of colour data in addition to the depth layer. By fusing the visual data with that of depth, the algorithm first detects and segments the object through CNN and then fuses the point cloud to obtain the pose and depth of the object [15].

Human pose estimation is also one of the areas of deep pose estimation which has numerous applications profoundly in medical robotics. Srivastav et al. discuss the utilization of R-CNN and ResNet50 to obtain the human pose in surgical environment by detecting humans, obtaining keypoints and translating the 2D keypoints them into 3D coordinates [16].

Object tracking and pursuit is another highly in-demand task in robotics and mechatronics. This task has been widely investigated and researched to be performed by deep image processing algorithms. Lai et al. have discussed the absolute distance estimation of flying UAVs using deep VGG-16 model [17]. Jin et al. have presented a method of obtaining 6D pose estimation of target UAV with relational graphs and perspective-n-point (PnP) algorithms [18].

Yavarabadi et al. have designed FastUAV-Net, an architecture which detects and tracks UAVs using YOLO-v3 with onboard Nvidia Jetson TX2 with the speed of 29 FPS. However, this paper only detects and tracks the UAV without obtaining the relative distance and pose [19].

Furthermore, Rezaei et al. have designed Traffic-Net, a deep learning architecture which detects, monitors and obtain the relative distance of vehicles and humans using a monocular camera. This method too obtains the 2D bounding box of objects and then translates the 2D bounding box into a 3D bounding box for distance estimation [20].

1.3.3 Flight Control and Pursuit

The proper utilization of earlier discussed deep learning algorithms is reliant on a proper flight control system to properly control and fly the UAV into desired states.

With the development and vast production of drones, digital instruments and controllers have experienced a thorough consideration by the researchers and developers.

Flight control systems were previously reliant on analog implementation using either pneumatic valves, hydraulic valves or analog electrical boards with resistors and capacitors. These flight control systems were later implemented digitally using microcontrollers, PLCs or any other digital processing units [21, 22].

Therefore, with the entwined advancement and popularity of drones and digital controllers, flight control of quadcopters and drones has experienced a huge progress in the previous decades[23].

Axel Reizenstein has developed a algorithm to control the trajectory and position of a quadcopter using PID and LQR controllers. This work incorporates Kalman filter to estimate the position, velocity and acceleration along z axis for height control and utilizes IMU and GPS data for controlling position along x and y axes[24].

Surma and Barczyk have utilized PID and LMPC (linear model-predictive control) control and pursue the position and trajectory of Parrot[1] AR and Bebop drones. This algorithm employs YOLO-v2 to detect the target drone and then obtains the 2D bounding box in order to estimate the pose and relative distance. C++ implementation of PID and LMPC of this work ensures the real-time positioning of the drones due to the static typing discipline of C++ language [25].

1.4 Thesis Outline

The current chapter has discussed the motivation and challenge behind the undertaking of this research and thesis in addition to the review and summary of the recent

[1]https://www.parrot.com/us/drones

publications in the area of visual artificial intelligence, deep object detection, deep pose estimation and modern flight control systems.

In the chapter 2, summarizes the hardware and software components which have been utilized for this research. We shall investigate the properties of Parrot Anafi and Parrot Bebop drones in addition to the Vicon motion capture system in hardware components part. Furthermore, we shall review the software components including PyTorch, Detectron2, Olympe, Sphinx and Robot Operating Systems (ROS) which have been used to implement the deep learning algorithms in addition to the digital flight control systems for automating and controlling the drones.

Chapter 3 discusses the design and implementation of feedback flight control systems to enable the Anafi drone to track and pursue the target Bebop drone in real-time horizon on ROS. We shall first identify the dynamics of Anafi drone and then design and tune a digital 4-axis PID controller which will be implement on ROS using C++.

Chapter 4 starts with an introduction on camera model and intrinsics for understanding the principals of object projection on image plane and camera calibration. This introduction continues with detailed analysis on 3D bounding box geometry and projection formulations on image plane by defining vertices and utilizing projection matrix. Subsequently and most importantly, keypoint-RCNN sections discusses the backbone of object and keypoint detection by analyzing the architecture and data structure which is followed by training and validation results. Next, the basics of perspective-n-point algorithm is discussed followed by its applications for pose estimation.

In chapter 5, we shall evaluate and investigate the accuracy of the three proposed deep learning methods alongside the flight control system. We shall replace Vicon position data with the deep pose estimator estimations in order to assess the vision-based pose estimation and pursuit.

Eventually, chapter 6 provides a summary of the contributions and findings of this thesis in addition to an insight for future directions and undertakings in the area.

1.5 Thesis Contributions

The scientific and engineering contributions of this research are listed as follows:

- Development of `sphinx_ros` robotic package which provides a the necessary tools for integrating ROS and Parrot Sphinx simulation environment. This package enables the manual control of the drone in addition to PID tuning of the simulated drone.

- Development of `anafi_ros` robotic package which includes a variety of tools for remote manual drone control, synchronized data acquisition, real-time PID tuning and most importantly the Vicon-based and AI-based pursuit algorithms and setups. This package incorporates different machine learning libraries like PyTorch with Olympe library to provide a framework for integrating flight controllers with deep neural networks.

- Design and implementation of 4-axis digital PID flight controller for x, y, z and yaw axes on ROS Noetic through `anafi_ros` and `sphinx_ros` packages.

- Novel 3D bounding box estimation using keypoint-RCNN with 96.6% average precision for keypoint estimation instead of conventional 2D to 3D conversion methods.

- Novel mobile real-time pose estimation of target drones by integrating keypoint-RCNN, 3D bounding box and perspective-n-point resulting in the state-of-the-art mean average error of 0.05 [m] over x-axis, 0.12 [m] over y-axis and 0.06 [m] over z-axis.

- Novel sensor-free pursuit algorithm for pursuing targets by eliminating the use of LiDAR , radar and ultrasonic sensors and using mobile commercial monocular cameras on drones integrated with deep keypoint-RCNN and perspective-n-point instead.

Chapter 2

Hardware and Software

In this chapter we shall have an overview of the utilized and Incorporated hardware and software components throughout this research.

2.1 Hardware Components

2.1.1 Parrot Anafi

Figure 2.1: Parrot Anafi quadcopter drone

Founded in 1994, Parrot® is one of the main leading manufacturers in drone industry [28]. Parrot introduced Anafi in 2018 and its Python controlling interface,

Olympe, two years later in 2020 which will be discussed thoroughly in Section 2.2.1.

Anafi is a light drone with the weight of 320 g and can fly as fast as 54 km/h or 15 m/s for the maximum flight travel of 14 km for full battery charge [29]. The battery has 2700 mAh capacity with an 8 V output which requires 90 minutes to be fully charged to fulfill a flight-time of 25 minutes.

The onboard Sony IMX230 camera has a high video resolution of 4096x2160 and a frame rate of 24 fps. Furthermore, the resolution is adjustable to 3840x2160 and D 1920x1080 pixels as well. However, the video streaming is only available on 720x1280 pixels with the frame rates of 24, 25 or 30 fps. The stream feed is encoded into H264 format for minimizing the impact of packet losses and to dilute errors by encoding the frame 45 slices of 16 pixels height which is then refreshed by a batch of 5, every 3 images. [29]

Figure 2.2: Parrot Bebop 2 quadcopter drone

The main platform for connecting the software components to Anafi drone is

Olympe which is a Python library that provides the packages and functions for flying Anafi in both a simulated or real environment with is discussed in Section 2.2.1 in details.

2.1.2 Parrot Bebop

Parrot Bebop 2 is another product of Parrot® which was introduced in November 2015. This drone has only been used as the target drone for the pursuer Anafi.

2.1.3 Vicon Motion Capture System

Motion capture is a process of recording the movement and poses of the objects, robots or people using specific techniques. Academy Award® -winning Vicon is a brand which manufactures and provides the required hardware and software components for obtaining motion capture of the desired items.

Figure 2.3: Vicon Vero camera

There are several different methods for performing motion capture including optical-passive method which is based on using retroreflective markers that are tracked by

infrared cameras, optical-active method that uses LED-emitting markers that are tracked by special cameras, video markerless method which is based on relying software for tracking the objects' movement and lastly the inertial method which is based on utilizing inertial sensors without the use of cameras.

Among the aforementioned methods, optical-passive has proven to be the most accurate, flexible and functional type of motion capture and is the most used method in the industry. Vicon too utilizes the optical-passive method by providing infrared cameras and passive markers.

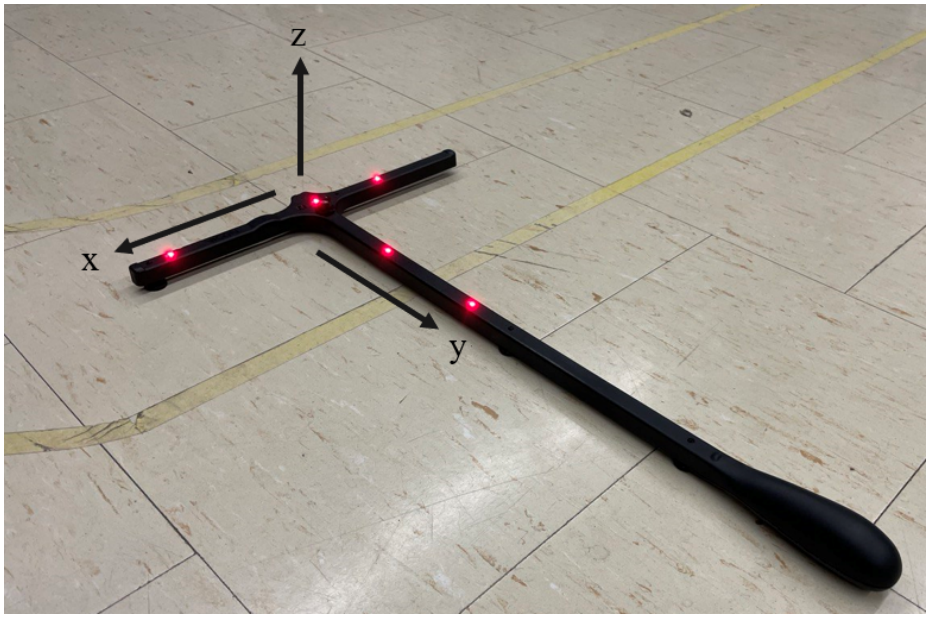


Figure 2.4: Vicon calibration wand

The Vicon motion capture system install in the Mechatronic Systems Lab of the University of Alberta has 10 Vero Vicon cameras in addition to the calibration wand and the Vicon Tracker software installed on a specific computer dedicated for Vicon only. Vicon has played a key-role in this research for obtaining the required training data and precised pose and relative distance of the drones respective to the defined origin.

2.1.4 Graphics Processing Unit

Unlike the Central Processing Units (CPUs) that fulfill the processing thought serial manner, the Graphics Processing Units (GPUs) are built to perform the processing in parallel manner which distinguishes them as the primary tools for video processing [30].

GPUs were initially designed to boost the performance of computers for gaming industry, to better render and visualize the graphics of the games. However, with the advances in the area of machine learning and the release of versatile libraries, GPUs were later utilized to perform the training and evaluating the machine learning and deep learning models and algorithms. Compute Unified Device Architecture (CUDA) is the platform that is used for connecting the deep learning libraries to GPU and is thoroughly discussed in Section 2.2.3.

Nvidia RTX 3090 is used in this research for training, evaluating and real-time image processing of the streaming feed of the Anafi drone using PyTorch and ROS which is discussed in Section 2.2.7. RTX 3090 has 10496 CUDA cores, boost clock of 1.70 GHz and a memory size of 24 GB which has made it the second top-performing manufactured GPU in the industry.

2.2 Software Components

2.2.1 Olympe

Olympe is an open-source Python controller programming interface for Parrot Drone including Anafi and Anafi thermal. Olympe is part of the Parrot Ground SDK which allows the developers to develop and implement Olympe on different platforms including mobile or desktop applications [31].

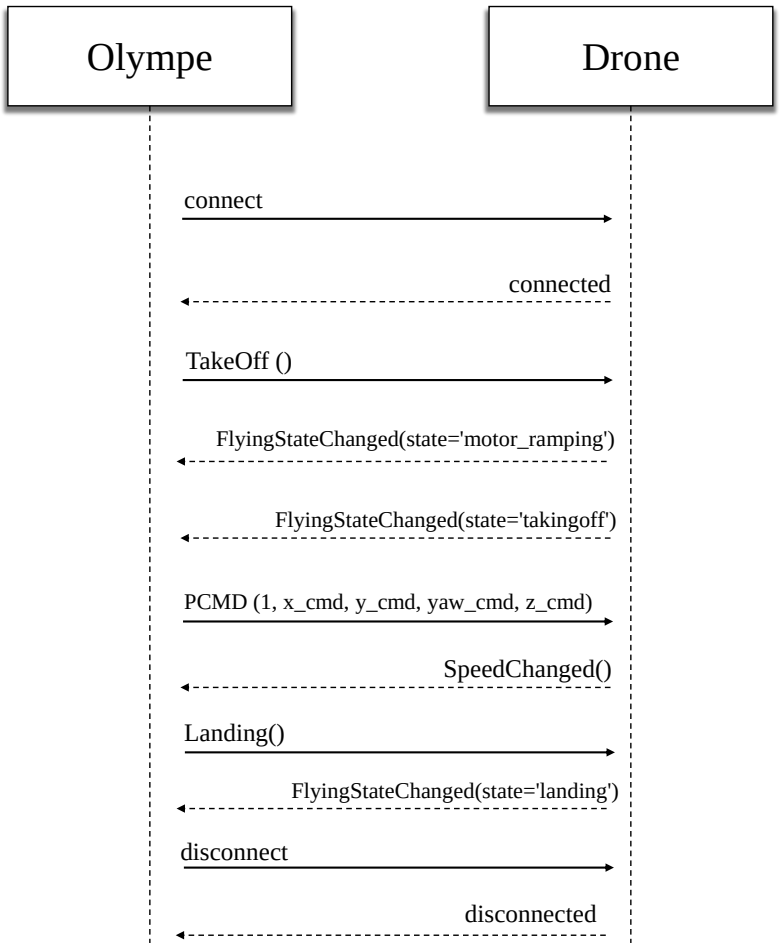


Figure 2.5: Olympe PCMD command sequence [32]

Olympe utilizes ARSDK messages at its core to communicate with the drone. The ARSDK is a protocol of sending flight commands as scripts to control the drone. The command protocols could be summarized as follows [33]:

- Ardrone3 Animations
- Ardrone3 GPS
- Ardrone3 Piloting and Speed Settings
- Ardrone3 Settings State
- Ardrone3 PRO State
- Ardrone3 Sound

PCMD (piloting command) is a function in Ardrone3 piloting library which takes four input to control the roll, pitch, yaw and thrust of the drone as a signed percentage from -100% to 100%. The change in Euler angles of the drone leads to the movement and speed change which would enable the position and speed control of the drone in a closed-loop manner.

2.2.2 Sphinx

Parrot® Sphinx is a software package developed by Parrot to simulate Anafi, Anafi AI and Anafi thermal on a simulated environment. Sphinx is founded on two components, Gazebo and Unreal Engine [34].

Gazebo is a simulation toolbox for robotic applications which has been designed to provide a graphical user interface for ROS users which shall be discussed thoroughly in Section 2.2.7. Unreal Engine (UE) is a software framework for visualizing 3D graphical environment. UE has been widely used in gaming industry as the core game engine for simulating and rendering the virtual environment.

Sphinx utilizes both Gazebo (v.11) and Unreal Engine (v.4) to provide a versatile simulation environment for Anafi drones which could be connected and monitored via Olympe and ROS through the implemented pipelines.

2.2.3 CUDA

Compute Unified Device Architecture (CUDA®) is an application programming interface for parallel computing which enables the program to utilize the GPU for processing. CUDA has been widely used in machine learning applications for deploying GPU as the core unit for training and validating the neural networks instead of CPU which would lead to staggering difference in processing time [35].

2.2.4 PyTorch

Designed by Meta AI lab, PyTorch is a machine learning library which is written in C, C++, Objective C, Python and CUDA. This library provides numerous functions and packages for developing and training neural networks. PyTorch is able to utilize CUDA in order to communicate with the GPU for processing enhancement, whereas it is also able to do use CPU as the processing unit as well [36].

2.2.5 Detectron 2

As stated, PyTorch has numerous amount of functions for machine learning, however, it is a general-purpose library. To better utilize machine learning for object detection and regional neural networks, Meta AI lab introduced Detectron in 2018 and Detectron 2 in 2019. Detectron is a PyTorch-based machine learning which deploys RCNN for instance segmentation, semantic segmentation and keypoint detection[37, 38].

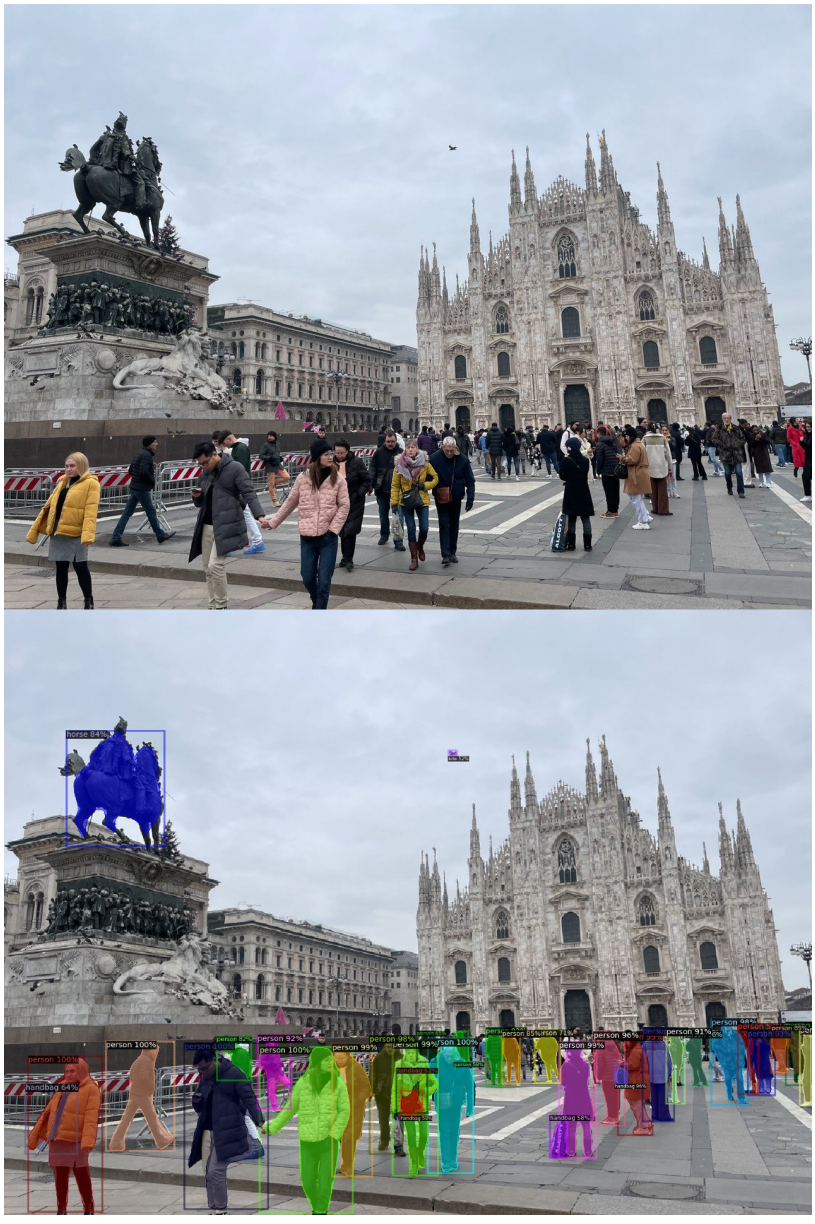


Figure 2.6: Instance segmentation using Detectron [38]

Detectron has prebuilt and pretrained RCNN models which could be used for custom datasets as well. Detectron supports COCO format labeled data which is a protocol for annotating RCNN datasets as JSON files.

2.2.6 Vicon Tracker

Vicon tracker an engineering motion capture software which is utilized for robotic tracker, virtual engineering, human factors engineering, design method optimization and pre-visualization of the drones in addition to facial and body capture of actors for gaming and cinematic industry [39].

Vicon tracker could be linked to ROS using `vicon_bdridge` and Datastream SDK which is discussed thoroughly in 2.2.7.3. Tracker has the built-in tool for calibrating the cameras and configuring the objects.

2.2.7 ROS

Robot operating system (ROS) is ironically not an operating system (OS) but a set of software libraries and tools that enables the users to develop and build robot applications. Developed by Stanford Artificial Intelligence Laboratory (SAIL), ROS is an open-source suite which supports codes in C++, Python and Lisp [40].

Nodes and topics are the backbones of every ROS application or package. A node is basically a piece of code written in C++ or Python which which could be executed once or repeatedly to perform a task. Nodes could subscribe from or publish to other nodes through topics. Topics are in fact messages of data which are sent and received by nodes. For instance, a topic could be a frame, position data, speed set points or any other data formats. Custom topics formats could be developed in ROS as messages. Basically, messages define the format of the desired topics which are to be received or sent by the node [41].

The ROS was initially released in March 2010 as ROS Box Turtle. There has have another twelve distributions by ROS ever since once or twice in every two years.

ROS Noetic Ninjemys is the latest distribution of ROS which has been chosen for this project due to its exclusive compatibility to Ubuntu 20.04 [42].

ROS Noetic exclusively targets Python3 and has enabled the migration of the previous Python2 packages and libraries into the Python3 environment. In the following sections, we shall summarize the ROS Noetic libraries and packages used throughout this research.

2.2.7.1 `tf`

Transform (`tf`) library is a powerful tool for tracking and multiple coordinate frames of one or more robots at the same time. The relationship between coordinates are maintained on `tf` as a stamped data tree structure. Furthermore, tf provides the users with the function to transform points, vectors or pose data into different systems of coordinates.

There are two key features on library `tf` called `tf_broadcaster` and `tf_listener`. A `tf_broadcaster` basically broadcasts the coordinates of the objects and the broadcasted frames which could be received or listened by other ROS nodes. This would eliminates the need for defining publishers and subscribers individually for communication between nodes.

2.2.7.2 `cv_bridge`

Based on ROS protocols, an image is transported as `sensor_msgs.mgs.Image` which is a class that transports the image frames as `bgr8` format. On the other hand, image data are usually processed and handled with OpenCV (Open Computer Vision) which is a comprehensive library equipped with numerous amount of functions for computer vision applications. OpenCV handles image data as numpy arrays with one channel for black and white images and three channels for colour images. Therefore, `cv_bridge` provides a platform to transform the data from OpenCV format to that of ROS and vice versa using `cv_bridge.CvBridge.cv2_to_imgmsg` and `cv_-`

`bridge.CvBridge.imgmsg_to_cv2` functions.

2.2.7.3 `vicon_bridge`

Similar to `cv_bridge`, `vicon_bridge` is tool for connecting the Vicon Tracker applications to ROS as `tf` messages and frames. `vicon_bridge` was originally developed by ETH Autonomous Systems Lab (ETHASL) as an open-source ROS driver. This driver publishes the coordinates of the Vicon objects as a quaternion coordinates as `tf` and also as separate topics with `vicon/<subject_name>/<segment_name>` format.

2.2.7.4 `sphinx_ros`

`sphinx_ros` is a ROS driver for Parrot Anafi and Anafi Ai flight simulation on Sphinx and Gazebo using ROS Noetic as the middleware platform. This driver is based on official Parrot arsdk-ng and arsdk-xlm[1]. This driver is developed in Mechatronic Systems Lab of the University of Alberta by Amir Hossein Ebrahimnezhad and is maintained by Dr. Martin Barczyk.

`sphinx_ros` has several nodes for flying, PID controlling and data acquisition which could be summarized as follow:

- `sp_ctrl_streaming.py`

  This Python node is used for manual keyboard control of the Anafi drone and publishing the streaming feed as `/anafi/frames` topic.

- `sp_pursuer.py`

  This Python node controls the Anafi drone automatically by subscribing to `sp_pid` in addition to publishing the frames as `/anafi/frames`.

- `sp_pid.cpp`

  This C++ node is responsible of generating the required command velocities for controlling the Anafi drone by listening to Vicon data from the virtual

[1]https://github.com/Parrot-Developers/olympe

`tf_broadcaster` and then publishing the `/cmd_vel` topic which contains the command velocities for x, y, z and yaw.

- `sp_opencv_sync.cpp`

  This c++ node subscribes to `/anafi/frames` and `tf_broadcaster` to synchronize the frames and the corresponding relative position of the target drone in order to generate the precise and synchronized training data for deep learning models.

2.2.7.5 `anafi_ros`

`anafi_ros` is also a ROS driver for Parrot Anafi physical drone flight, control and maneuvering. Similar to `sphinx_ros`, `anafi_ros` too is based on official Parrot arsdk-ng and arsdk-xlm. This driver is as well developed in Mechatronic Systems Lab of the University of Alberta by Amir Hossein Ebrahimnezhad and is maintained by Dr. Martin Barczyk.

`anafi_ros` has versatile nodes for data acquisition, flying, controlling, implementing and deploying deep learning models in a real-time horizon. Listed below, is a summary of functional nodes of `anafi_ros`:

- `af_ctrl_streaming.py`

  This Python node is used for manual keyboard control of the Anafi drone and publishing the streaming feed as `/anafi/frames` topic.

- `af_pursuer.py`

  This Python node controls the Anafi drone automatically by subscribing to `af_pid` in addition to publishing the frames as `/anafi/frames`.

- `af_pid.cpp`

  This C++ node is responsible of generating the required command velocities for controlling the Anafi drone by listening to Vicon data from the `tf_broadcaster`

and then publishing the `/cmd_vel` topic which contains the command velocities for x, y, z and yaw.

- `af_opencv_sync.cpp`

  This c++ node subscribes to `/anafi/frames` and `tf_broadcaster` to synchronize the frames and the corresponding relative position of the target drone in order to generate the precise and synchronized training data for deep learning models.

- `af_mask_detector.py`

  Deep Python node for detecting and extracting the target drone mask by subscribing to `/anafi/frames` and feeding it into trained mask-RCNN model. Subsequently, this node publishes the predicted relative position of the target drone as `/anafi/relpos`.

- `af_keypoint_detector.py`

  This Python node utilizes the trained deep keypoint-RCNN model for obtaining the keypoints of the target drone. Subsequently, this node feeds the extracted keypoints into an MLP for obtaining the relative pose and publishing it as `/anafi/relpos`.

- `af_vertix_detector.py`

  This node utilizes the same keypoint-RCNN model to first detect the target drone along with its keypoints. Then, by feeding the keypoints into an MLP, it obtains the vertices of the 3D bounding box of the target drone and subsequently, it extracts the relative pose from the 3D bounding box. This node simultaneously visualizes the 3D bounding box of the target drone as well.

2.2.7.6 `rqt`

`rqt` (ROS Qt) is a framework for providing GUI development on ROS which is based on Qt cross-platform GUI creating software. `rqt` has several plugins including `rqt_-`

`graph` for visualizing ROS computation graph, `rqt_dep` for visualizing ROS dependency graph, `rqt_plot` for visualizing numeric values in 2D plot and `rqt_topic` for displaying debugging information about ROS topics [43].

2.2.7.7 `rviz`

`rviz` is also a GUI tool for visualizing on ROS as alike to `rqt`. `rviz` has many display types including odometry, pointcloud, depthcloud, tf, etc. This GUI has been used in this research for visualizing the pursuer and target drones with respect to the world frame.

2.2.8 MATLAB

Originally developed by Cleve Moler as a library for linear algebra, MATLAB® (Matrix Laboratory) has become one of the leading applications for numerical analysis in difference branches of science and engineering [44].

Most notably, MATLAB has a variety of specialized toolboxes including control system toolbox™and system identification toolbox™which are used in this research for designing and evaluating digital position and speed control systems for the Anafi drone [45].

Despite the existence of a myriad of functions and libraries in MATLAB, it is not recommended to use MATLAB as the real-time control backbone due to the fact that MATLAB is an interpreter and not a compiler. Therefore, executing codes on MATLAB takes considerably more time compared to that of C++. As a result, MATLAB is only used for design purposes throughout this research and the designed controller are implemented on C++ for guaranteeing real-time position and speed control.

Chapter 3

Control and Pursuit

In this chapter we shall design and implement the proper control and pursuit algorithms for positioning of the Anafi and Bebop drones using digital Proportional Integral Derivative (PID) controller. However, before getting into the controller design, we shall first review the mathematical preliminaries of mobile robotics through linear algebra and signal processing topics in Section 3.1.1 and 3.1.2. Consequently, discrete-time model of the drones and parameters are obtained and identified in Section 3.2 in addition to the control philosophy and ROS implementation of the controllers. Lastly, the pursuit framework and implementation are discussed and assessed in Section 3.3.

3.1 Mathematical Foundation

This section outlines and reviews the concepts and definitions of linear algebra and signal processing which are essential for design and implementation of digital flight control systems. Section 3.1.1 is majorly based on Chapter 2 of the cited PhD thesis [46].

3.1.1 Linear Algebra

Linear algebra is in fact the mathematical foundation of modern robotics which provides the necessary tools for modelling and analyzing the behaviour of dynamical systems. Only by understanding the principals of linear algebra can we control and

monitor the aerial robots, quad-copters and drones properly.

3.1.1.1 Vectors and Matrices

Vectors and matrices are the alphabets of linear algebra. A vector is a column representation of a set of numbers which could refer to coordinates of an object or other parameters. A Matrix is a multi-dimensional numerical or parametric representation of different variables which could act as a function to perform mathematical operations on an input vector. A time vector with n elements could be represented as follows:

$$\vec{v}_t = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_n \end{pmatrix} \tag{3.1}$$

Similarly, a matrix with m rows and n columns is defined as follows:

$$M_{mn} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix} \tag{3.2}$$

3.1.1.2 Space and Subspace

A vector space is a set of vectors elements that could be added or multiplied by scalars thought that space. A vector space could be consisted of real or complex vectors which each has different applications in mobile robotics. A three-dimensional real vector space could be defined as follows:

$$\exists\, V \;:\; (v_1, v_2, v_3) \;\in\; \mathbb{R}^3 \tag{3.3}$$

Therefore for any vector in V we shall have:

$$^{V}\vec{v} = a_1\vec{v_1} + a_2\vec{v_2} + a_3\vec{v_3}\,; (a_1, a_2, a_3) \;\in\; \mathbf{R} \tag{3.4}$$

A subspace is in fact a vector space which exists under the inherited operations. A three-dimensional real spherical subspace could be defined as follows:

$$S \;=\; \{ \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \mid v_1^3 + v_2^2 + v_3^3 = r^3 \;,\; r \in \mathbb{R}\} \tag{3.5}$$

3.1.1.3 Rotation Matrix

A rotation matrix is in fact transformation matrix and is both orthogonal and unitary which maps coordinates with respect to different frames through Euclidean space by using Euler angles of ϕ, θ and ψ. In mobile robotics, objects are described in three-dimensional space and could rotate around three axes. Therefore, there are three rotation matrices for each x, y and z axes as follows [46]:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{bmatrix} \tag{3.6}$$

$$R_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{3.7}$$

$$R_z(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

Where ϕ is rotation on x or roll-axis angle, θ is rotation on y-axis or pitch and ψ is rotation on z-axis or yaw angle. For obtaining the corresponding rotation matrix to transform the world frame S_w to the vector space of the robot S_r, with rotation angles of ϕ_1 on x-axis, θ_1 on y-axis and ψ_1 on z-axis we can multiply the rotation matrices of each axis commutatively as follows:

$$\begin{matrix} {}^{S_r}_{S_w}T \end{matrix} = R_x(\phi_1)\ R_y(\theta_1)\ R_z(\psi_1) \tag{3.9}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi_1) & sin(\phi_1) \\ 0 & -sin(\phi_1) & cos(\phi_1) \end{bmatrix} \begin{bmatrix} cos(\theta_1) & 0 & sin(\theta_1) \\ 0 & 1 & 0 \\ -sin(\theta_1) & 0 & cos(\theta_1) \end{bmatrix} \begin{bmatrix} cos(\psi_1) & -sin(\psi_1) & 0 \\ sin(\psi_1) & cos(\psi_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

Therefore, we have:

$${}^{S_r}_{S_w}T = \begin{bmatrix} c_{\theta_1}c_{\psi_1} & s_{\phi_1}s_{\theta 1}c_{\psi_1} - c_{\phi_1}s_{\psi 1} & s_{\phi_1}s_{\theta 1}c_{\psi_1} + s_{\phi_1}s_{\psi 1} \\ c_{\theta 1}c_{\psi 1} & s_{\phi_1}s_{\theta 1}s_{\psi_1} + c_{\phi_1}c_{\psi 1} & c_{\phi_1}s_{\theta 1}s_{\psi_1} - s_{\phi_1}c_{\psi 1} \\ -s_{\theta 1} & s_{\phi 1}s_{\theta 1} & c_{\phi 1}c_{\theta 1} \end{bmatrix} \tag{3.11}$$

3.1.1.4 Quaternion System

As it was mentioned in the previous section, orientation of a mobile robot could be represented by Euler angles and transformed using the rotation matrices. However,

the rotation matrix of Euler angles require a lot of memory and are difficult to interpolate. Therefore, to solve this issue, quaternions are dominantly preferred to Euler angles in robotics applications. A quaternion vector space of $r \in \mathbb{H}$ is defined as follows:

$$r = r_0 + r_1\mathbf{i} + r_2\mathbf{j} + r_3\mathbf{k} \tag{3.12}$$

Where:

$$(r_0, r_1, r_2, r_3) \in \mathbb{R}^4 \, , \, \{1, \mathbf{i}, \mathbf{j}, \mathbf{k}\} \in \mathbb{H} \tag{3.13}$$

$\mathbb{R}^4$ is the four-dimensional space of real numbers and $\mathbb{H}$ is in fact the four-dimensional space of

$$R = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{3.14}$$

3.1.2 Signal Processing

A signal is basically a function that represent and transmit information. A signal could be continuous or discrete, quantized or unquantized, periodic or apreiodic [47].

An analog signal is a continuous-time unquantized signal, an instance is represented below with its corresponding graph:

$$x(t) = cos(t) \, , \, t \in \mathbb{R} \tag{3.15}$$

On the contrary, a digital signal discrete-time signal with quantized values which means that the output does not contain any real value, but it contains sampled quantized values. Digital signal are represented with n as the their argument with belongs to integer number set. A digital signal with its corresponding figure it represented and plotted as follows:

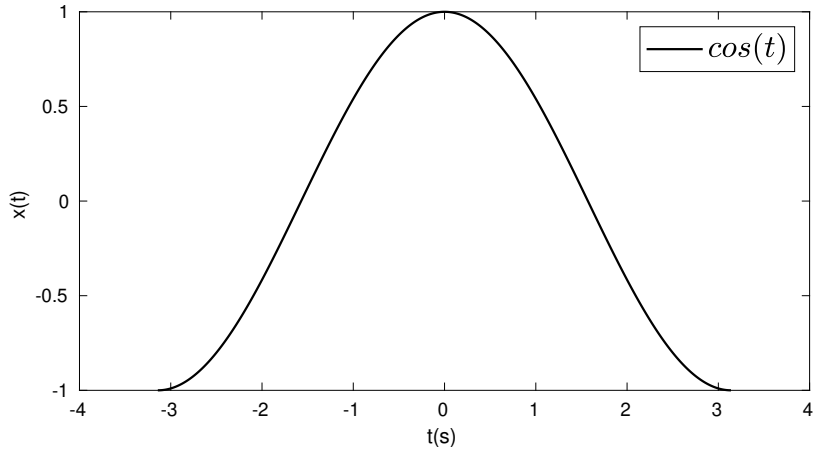


Figure 3.1: An Analog Signal

$$x[n] \ = \ cos[\frac{2n}{\pi}] \ , \ n \ \in \mathbb{Z} \tag{3.16}$$

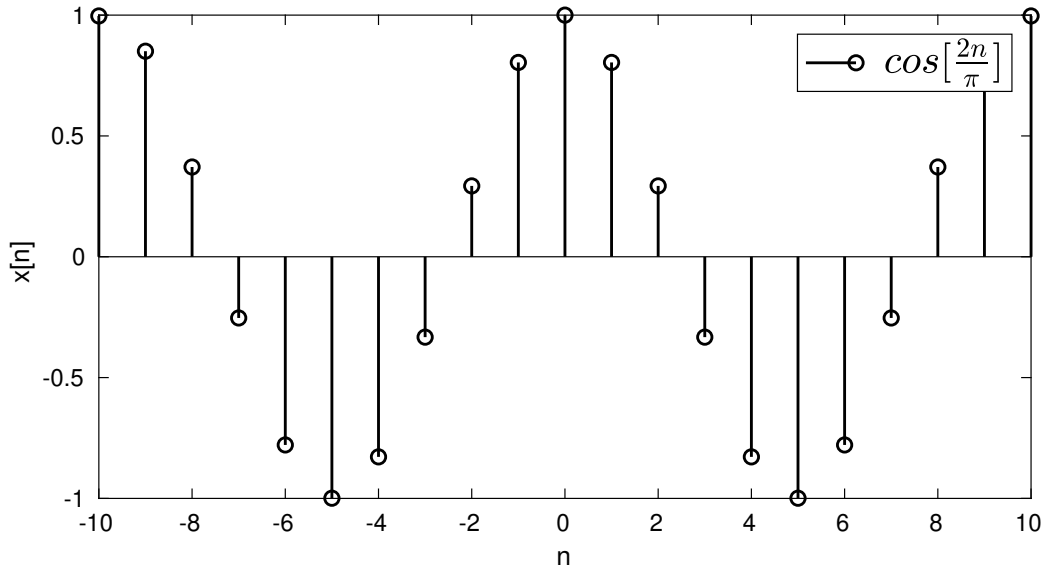


Figure 3.2: A Digital Signal

With the advances in digital processors, the use of digital signals over the analogs has become a trend due to their higher accuracy and less noise, more security and flexibility.

3.1.2.1 LTI Systems

A system is in fact a process in which input signals are converted into output signals uniquely. Systems could be as well analog or digital. A linear time-invariant (LTI) system is a system that has the three following properties [48, 49]:

1. Homogeneity:

$$\text{if: } x[n] \longrightarrow y[n] \Rightarrow a\,x[n] \longrightarrow a\,y[n] \tag{3.17}$$

2. Additivity :

$$\text{if: } x_1[n] \longrightarrow y_1[n]\ ,\ x_2[n] \longrightarrow y_2[n] \Rightarrow x_1[n] + x_2[n] \longrightarrow y_1[n] + y_2[n] \tag{3.18}$$

3. Time-Invariance:

$$\text{if: } x[n] \longrightarrow y[n] \Rightarrow x[n-k] \longrightarrow y[n-k] \tag{3.19}$$

LTI systems play a key-role in control systems design due to their simplicity and comprehensibility. In fact, we try to convert the non-linear systems into LTI systems to better understand and analyze them. Of course, this has its cons as well, and in some applications it is needed to design non-linear controllers to control the non-linear dynamics accordingly. However, in this research the dynamics of the drone are considered as LTI although they are in fact non-linear in nature.

3.1.2.2 Fourier Transform

Introduced by Jean Baptiste Joseph Fourier in 1822 [50], Fourier transform is technically a mathematical tool for analyzing time-domain signals in frequency domain and vice versa using inverse Fourier transform.

The formulations for continuous Fourier transform and its inverse are as follows:

$$\text{Analysis Equation: } X(j\omega) = \int_{-\infty}^{+\infty} x(t)\, e^{-j\omega t}\, dt \tag{3.20}$$

$$\text{Synthesis Equation: } x(t) \;=\; \frac{1}{2\pi}\int_{-\infty}^{+\infty} X(j\omega)\, e^{j\omega\, t}\, d\omega \tag{3.21}$$

For obtaining the Fourier transform of discrete-time signals we can use a modified version of Fourier transform called Discrete-Time Fourier Transform (DTFT) which is formulated as follows [51–53]:

$$\text{DTFT: } X(e^{j\omega}) \;=\; \sum_{n=-\infty}^{+\infty} x[n]\, e^{-j\omega\, n} \tag{3.22}$$

$$\text{IDTFT: } x[n] \;=\; \frac{1}{2\pi}\int_{\langle 2\pi\rangle} X(e^{j\omega})\, e^{j\omega\, n}\, d\omega \tag{3.23}$$

Although it is possible to obtain the frequency components of digital signals using DTFT, it is challenging to store the $X(e^{j\omega})$ due to its continuous nature. Therefore, it needed obtain a sampled form of DTFT to have finite amount of frequency components to store. To fulfil this need, Discrete Fourier Transform (DFT) was invented and formulated as below [51]:

$$\text{DFT: } X[k] \;=\; \sum_{n=0}^{N-1} x[n]\, e^{-jk\frac{2\pi}{N}n} \;;\; k \;=\; 0, 1, ..., N-1 \tag{3.24}$$

$$\text{IDFT: } x[n] \;=\; \frac{1}{N}\sum_{k=0}^{N-1} X[k]\, e^{jk\frac{2\pi}{N}n} \;;\; n \;=\; 0, 1, ..., N-1 \tag{3.25}$$

3.1.2.3 Z-Transform

DTFT and DFT are powerful tool for analyzing digital signals in frequency domain, however the summation is only consistent for stable systems. For analyzing unstable systems in frequency domain, it is needed to acquire Z-transform which is defined as below [49]:

$$X(z) \;=\sum_{-\infty}^{+\infty} x[n]\; z^{-n} \tag{3.26}$$

And the inverse Z-transform as:

$$x[n] = \frac{1}{2\pi j} \oint X(z) z^{n-1} dz \tag{3.27}$$

We shall acquire Z-transform for investigating the stability and performance of the digital model and controllers.

3.1.2.4 Digital Filters

Filters are essential tools for handling signals. They are used for noise cancellation and demodulation of both analog and digital signals.

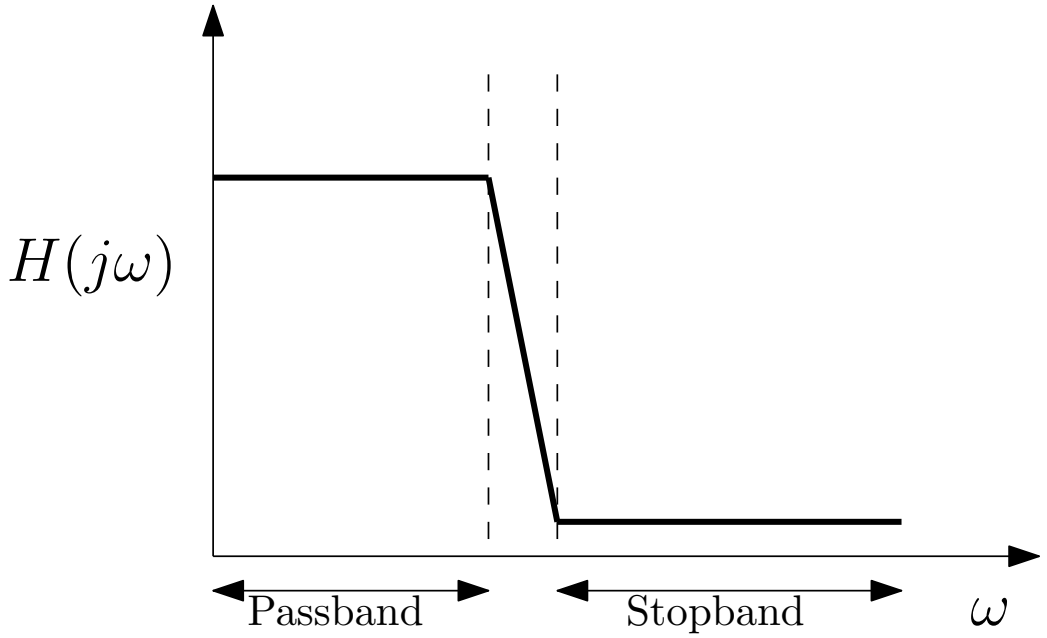


Figure 3.3: Schematic representation of Magnitude Component of Low-Pass Filters

Digital filters are usually classified into different categories based on their transfer function and functionality including low-pass, high-pass, band-pass and band-stop filters.

In this research, we shall acquire low-pass digital filters for noise-cancellation and smoothening velocity signals. Digital filters are categorized into two categories including FIR (Finite Impulse Response) and IIR (Infinite Impulse Response) filters. FIR filters are easier to design, however they require more memory and are more

difficult to implement. On the other hand, IIR filters require less memory and have comparably better performance. There are different methods for designing IIR filters including Butterworth, Chebyshev, and Elliptic methods [49].

In the following sections, we shall Butterworth filters to enhance the velocity signals of different axes.

3.2 Control

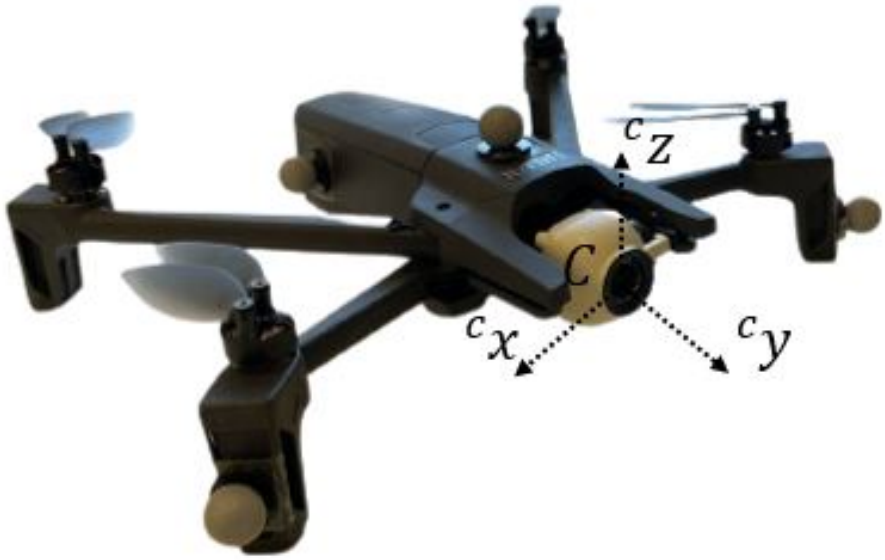


Figure 3.4: Anafi Coordinates Cartesian Representation

3.2.1 Generalized Coordinates

The state of the Anafi drone with respect to the world frame could be represented by the following generalized coordinates as follows:

$$ {}^{\mathcal{W}}\vec{q} = \begin{pmatrix} {}^{\mathcal{W}}q_1 \\ {}^{\mathcal{W}}q_2 \\ {}^{\mathcal{W}}q_3 \\ {}^{\mathcal{W}}q_4 \end{pmatrix} = \begin{pmatrix} {}^{\mathcal{W}}x_a \\ {}^{\mathcal{W}}y_a \\ {}^{\mathcal{W}}z_a \\ {}^{\mathcal{W}}\psi_a \end{pmatrix} \tag{3.28} $$

As it is represented in Figure 3.4, ${}^{\mathcal{W}}x_a$, ${}^{\mathcal{W}}y_a$, ${}^{\mathcal{W}}z_a$ mark the Cartesian position of the centre of the mass of the Anafi drone with respect to the world frame of $\mathcal{W}$ and ${}^{\mathcal{W}}\psi_a$ is the angular rotation of the drone on z-axis. The reason of exclusion of Euler angles of ϕ_a and θ_a is the that in fact, these two angles act as control parameters for moving the drone on y and z axes respectively.

3.2.2 Modelling

Obtaining a proper model for the drone is an essential preliminary for designing the proper controllers. State-space representation is a way to demonstrate the dynamics of an LTI system and transform n second order ordinary differential equations into $2n$ first order ordinary differential equations. A continuous-time state-space of an LTI system has the following general form [21]:

$$ \dot{x}(t) = Ax(t) + Bu(t) \tag{3.29} $$

$$ y(t) = Cx(t) + Du(t) \tag{3.30} $$

The continuous-time state-space of the Anafi drone with respect to the defined set of generalized coordinates is as follows:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \\ \dot{x}_5(t) \\ \dot{x}_6(t) \\ \dot{x}_7(t) \\ \dot{x}_8(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & a_x & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a_y & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_z & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_\psi \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \\ x_6(t) \\ x_7(t) \\ x_8(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ b_x & 0 & 0 & 0 \\ 0 & b_y & 0 & 0 \\ 0 & 0 & b_z & 0 \\ 0 & 0 & 0 & b_\psi \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \\ u_4(t) \end{bmatrix} \tag{3.31}$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \\ x_6(t) \\ x_7(t) \\ x_8(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \\ u_4(t) \end{bmatrix} \tag{3.32}$$

Where:

$$\begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \\ x_6(t) \\ x_7(t) \\ x_8(t) \end{bmatrix} = \begin{bmatrix} r_x^{(a)}(t) \\ r_y^{(a)}(t) \\ r_z^{(a)}(t) \\ r_\psi^{(a)}(t) \\ v_x^{(a)}(t) \\ v_y^{(a)}(t) \\ v_z^{(a)}(t) \\ v_\psi^{(a)}(t) \end{bmatrix} \tag{3.33}$$

And:

$$\begin{bmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \\ u_4(t) \end{bmatrix} = \begin{bmatrix} u_x^{(a)}(t) \\ u_y^{(a)}(t) \\ u_z^{(a)}(t) \\ u_\psi^{(a)}(t) \end{bmatrix} \tag{3.34}$$

The superscript denotes the drone type, (a) for Anafi and (b) for Bebop, and r and v are position and speed vectors which each has four components of x, y, z and ψ as represented.

As it is shown in Equation 3.31, the state space is represented with position and velocities of generalized coordinates which results in having eight states of $x(t)$. Matrix A has four unknown parameters of a_x, a_y, a_z, a_ψ which will be identified in the next section. The input vector of $u(t)$ consists of four inputs which affect the accelerations directly weighted by the unknown coefficients of b_x, b_y, b_z, b_ψ. As it was stated earlier, $\dot{x}(t)$ consists of the time-derivative of the continuous states which are related to the states and inputs by A and B matrices. The output vector of $y(t)$ consists of eight outputs which are identical to the states and therefore matrix C is an identify matrix of size eight and matrix D is an eight by four zero matrix as it is shown in Equation 3.32.

As it was mentioned earlier in the previous section, the controllers are to be designed and implemented digitally. Therefore, it is needed to transform the continuous-time state-space into a discrete-time state space accordingly.

The discrete state-space has the following general form [54]:

$$x[k+1] = A_d\, x[k] + B_d\, u[k] \tag{3.35}$$

$$y[k] = C_d\, x[k] + D_d\, u[k] \tag{3.36}$$

Using Euler's forward difference method we have [54, 55]:

$$\dot{x} \approx \frac{x(t+T) - x(t)}{T} \tag{3.37}$$

Where T is the sampling period of the discrete signal. By applying Equation 3.37 into Equation 3.31 we shall have:

$$\frac{x(t+T) - x(t)}{T} = Ax(t) + Bu(t) \tag{3.38}$$

Therefore:

$$x(t+T) = (I + AT)\, x(t) + BT\, u(t) \tag{3.39}$$

By rewriting Equation 3.31 into discrete form we shall have:

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \\ x_3[k+1] \\ x_4[k+1] \\ x_5[k+1] \\ x_6[k+1] \\ x_7[k+1] \\ x_8[k+1] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & T & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & T & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & T & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & T \\ 0 & 0 & 0 & 0 & T\,a_x + 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & T\,a_y + 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & T\,a_z + 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & T\,a_\psi + 1 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \\ x_5[k] \\ x_6[k] \\ x_7[k] \\ x_8[k] \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ T\,b_x & 0 & 0 & 0 \\ 0 & T\,b_y & 0 & 0 \\ 0 & 0 & T\,b_z & 0 \\ 0 & 0 & 0 & T\,b_\psi \end{bmatrix} \begin{bmatrix} u_1[k] \\ u_2[k] \\ u_3[k] \\ u_4[k] \end{bmatrix} \tag{3.40}$$

Where:

$$\begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \\ x_5[k] \\ x_6[k] \\ x_7[k] \\ x_8[k] \end{bmatrix} = \begin{bmatrix} r_x^{(a)}[k] \\ r_y^{(a)}[k] \\ r_z^{(a)}[k] \\ r_\psi^{(a)}[k] \\ v_x^{(a)}[k] \\ v_y^{(a)}[k] \\ v_z^{(a)}[k] \\ v_\psi^{(a)}[k] \end{bmatrix} \tag{3.41}$$

And:

$$\begin{bmatrix} u_1[k] \\ u_2[k] \\ u_3[k] \\ u_4[k] \end{bmatrix} = \begin{bmatrix} u_x^{(a)}[k] \\ u_y^{(a)}[k] \\ u_z^{(a)}[k] \\ u_\psi^{(a)}[k] \end{bmatrix} \tag{3.42}$$

Subsequently, for the digital $y[k] = Cx[k] + Du[k]$ we have:

$$\begin{bmatrix} y_1[k] \\ y_2[k] \\ y_3[k] \\ y_4[k] \\ y_5[k] \\ y_6[k] \\ y_7[k] \\ y_8[k] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \\ x_5[k] \\ x_6[k] \\ x_7[k] \\ x_8[k] \end{bmatrix}$$

$$+ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \\ u_\psi \end{bmatrix} \tag{3.43}$$

Where:

$$\begin{bmatrix} y_1[k] \\ y_2[k] \\ y_3[k] \\ y_4[k] \\ y_5[k] \\ y_6[k] \\ y_7[k] \\ y_8[k] \end{bmatrix} = \begin{bmatrix} r_x^{(a)}[k] \\ r_y^{(a)}[k] \\ r_z^{(a)}[k] \\ r_\psi^{(a)}[k] \\ v_x^{(a)}[k] \\ v_y^{(a)}[k] \\ v_z^{(a)}[k] \\ v_\psi^{(a)}[k] \end{bmatrix} \tag{3.44}$$

By substituting $T\,a_x+1$, $T\,a_y+1$, $T\,a_z+1$, $T\,a_\psi+1$ with a'_x, a'_y, a'_z, a'_ψ respectively and $T\,b_x$, $T\,b_y$, $T\,b_z$, $T\,b_\psi$ with b'_x, b'_y, b'_z, b'_ψ respectively as well we shall have:

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \\ x_3[k+1] \\ x_4[k+1] \\ x_5[k+1] \\ x_6[k+1] \\ x_7[k+1] \\ x_8[k+1] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & T & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & T & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & T & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & T \\ 0 & 0 & 0 & 0 & a'_x & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a'_y & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a'_z & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a'_\psi \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \\ x_5[k] \\ x_6[k] \\ x_7[k] \\ x_8[k] \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ b'_x & 0 & 0 & 0 \\ 0 & b'_y & 0 & 0 \\ 0 & 0 & b'_z & 0 \\ 0 & 0 & 0 & b'_\psi \end{bmatrix} \begin{bmatrix} u_1[k] \\ u_2[k] \\ u_3[k] \\ u_4[k] \end{bmatrix} \quad (3.45)$$

Therefore, it is only needed to identify a'_x, a'_y, a'_z, a'_ψ parameters of A_d matrix and b'_x, b'_y, b'_z, b'_ψ of B_d as well. The discrete position data is provided by Vicon system, however , the velocity data should be retrieved by applying backward difference on positions signals. Using this method for obtaining velocity would cause noises on velocity signals. Therefore, it is needed to enhance the velocity signals before identifying parameters.

3.2.3 Filter Design

Pose and orientation of the objects could be retrieved from the Vicon system, however for obtaining the axial velocities, it is needed to obtain the speed through numerical

methods. Using backward difference we have:

$$v_x^{(k)} = \frac{x^{(k)} - x^{(k-1)}}{t^{(k)} - t^{(k-1)}} \tag{3.46}$$

Due to the noises on position signals, estimated velocity signals would contain noises with higher amplitudes as well. Let's take a look at the v_x signal as below:

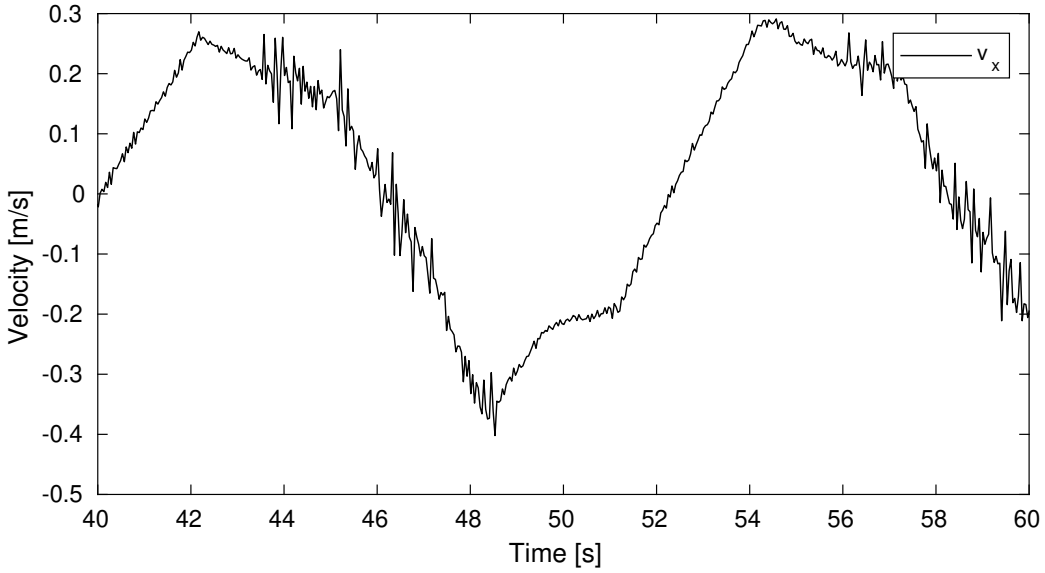


Figure 3.5: Raw x axis Velocity Signal

As it is represented in Figure 3.5, huge oscillations are visible in the digital signal of translational velocity of the drone which overshadow the real data.

3.2.3.1 Butterworth Filter

Butterworth is an approach for designing filters, either analog or digital. The idea is to have a low-pass filter with the following analog magnitude-squared frequency response [49]:

$$|H(j\omega)|^2 = \frac{1}{1 + (\frac{\omega}{\omega_c})^{2N}} \tag{3.47}$$

Where ω_C is the analog cut-off frequency and N is in fact the order of the filter. To fulfill the desired transfer function stated in Equation 3.47, the poles must be places on a circle with the radius of w_c around the origin. Consequently, the transfer will have the following form:

$$H(s) = \frac{1}{1 + a_1 \left(\frac{s}{\omega_c}\right) + a_2 \left(\frac{s}{\omega_c}\right)^2 + \cdots + a_N \left(\frac{s}{\omega_c}\right)^N} \tag{3.48}$$

To transform the analog filter to digital, we shall use the bilinear transform as follows:

$$s = \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}} \tag{3.49}$$

As a result, the discrete filter transfer function will have the following form:

$$H(z) = \frac{1}{1 + a_1 \left(\frac{2}{T\omega_c} \frac{1-z^{-1}}{1+z^{-1}}\right) + a_2 \left(\frac{2}{T\omega_c} \frac{1-z^{-1}}{1+z^{-1}}\right)^2 + \cdots + a_N \left(\frac{2}{T\omega_c} \frac{1-z^{-1}}{1+z^{-1}}\right)^N} \tag{3.50}$$

Which could be simplified into the following generalized form:

$$H(z) = \frac{b'_0 + b'_1 \, z^{-1} + b'_2 \, z^{-2} + \cdots + b'_N \, z^{-N}}{1 + a'_1 \, z^{-1} + a'_2 \, z^{-2} + \cdots + a'_N \, z^{-N}} \tag{3.51}$$

It should be also noted that the bilinear transform results in the following relationships between the angles of analog and digital horizons:

$$\Omega = \tan(\frac{\omega}{2}) \tag{3.52}$$

$$\omega = 2\tan^{-1}(\Omega) \tag{3.53}$$

We aim to design a digital IIR (infinite impulse response) Butterworth signal, but first we need to analyze the noise spectrum in frequency domain in order to determine the cut-off frequency. For obtaining the frequency spectrum of v_x we shall use FFT (Fast Fourier Transform) using MATLAB `fft()` command [56].

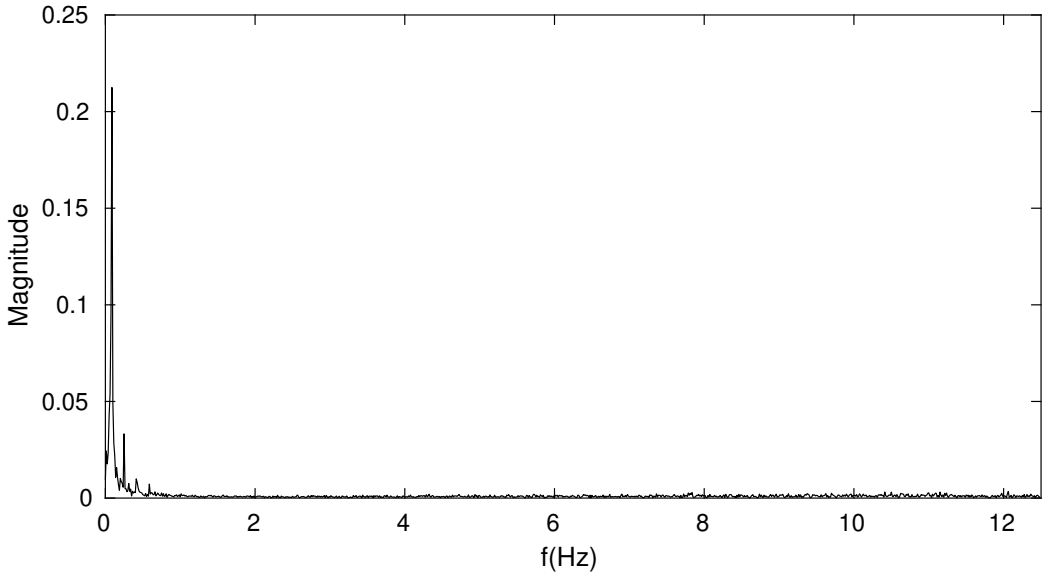


Figure 3.6: Frequency Spectrum of v_x

As it is shown in Figure3.6 can be inferred that the noise covers frequencies higher than 0.7 hz, and therefore the components below 0.7 hz belong to the actual data. Therefore, by designing a low-pass IIR filter, we can properly apply noise cancellation and obtain proper and smooth filtered speed data. Similarly, the cut-off frequencies for y, z and ψ are respectively 0.5, 1.2 and 1.2 hz based on the spectrum signals attached in Appendix B.

A second-order Butterworth filter has the following general form:

$$H(z) = \frac{b_0 + b_1\, z^{-1} + b_2\, z^{-2}}{1 + a_1\, z^{-1} + a_2\, z^{-2}} \tag{3.54}$$

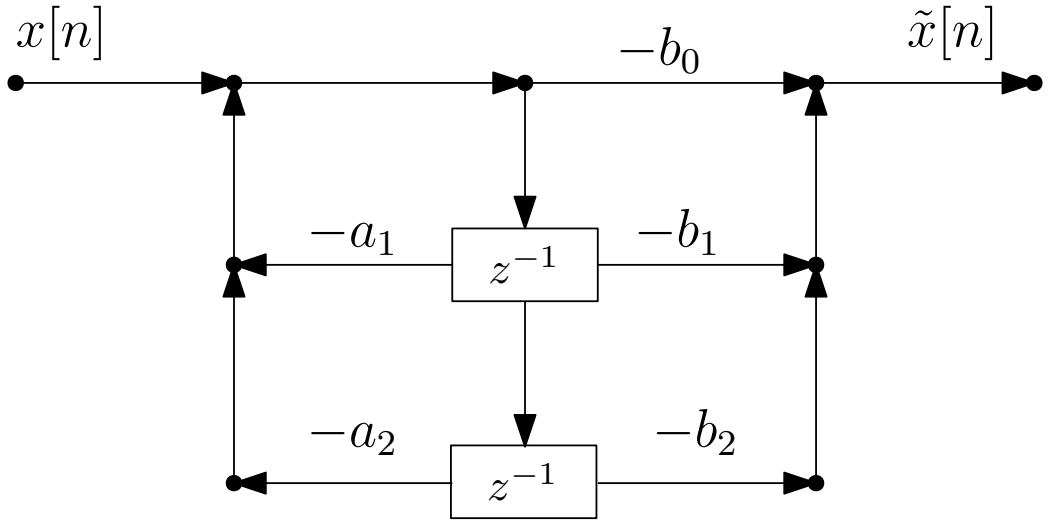


Figure 3.7: Direct Form Realization of Second Order Butterworth Filter

Using MATLAB `butter()` function and the determined cut-off frequencies, we shall have [57]:

- For x axis:

$$H_x(z) = \frac{0.0069 + 0.0137\, z^{-1} + 0.0069\, z^{-2}}{1 - 1.7523\, z^{-1} + 0.7797\, z^{-2}} \tag{3.55}$$

- For y axis:

$$H_y(z) = \frac{0.0036 + 0.0072\, z^{-1} + 0.0036\, z^{-2}}{1 - 1.8227\, z^{-1} + 0.8372\, z^{-2}} \tag{3.56}$$

- For z axis:

$$H_z(z) = \frac{0.0187 + 0.0373\, z^{-1} + 0.0187\, z^{-2}}{1 - 1.5782\, z^{-1} + 0.6528\, z^{-2}} \tag{3.57}$$

- For Yaw axis:

$$H_{\psi}(z) = \frac{0.0187 + 0.0373\, z^{-1} + 0.0187\, z^{-2}}{1 - 1.5782\, z^{-1} + 0.6528\, z^{-2}} \tag{3.58}$$

Consequently, the relationship between the raw and filtered signals in time-time is as follows:

- For x axis:

$$\tilde{v}_x[n] = 0.0069\, v_x[n] + 0.0137\, v_x[n-1] + 0.0069\, v_x[n-2] \tag{3.59}$$

$$+1.7523\, \tilde{v}_x[n-1] - 0.7797\, \tilde{v}_x[n-2] \tag{3.60}$$

- For y axis:

$$\tilde{v}_y[n] = 0.0036\, v_y[n] + 0.0072\, v_y[n-1] + 0.0036\, v_y[n-2] \tag{3.61}$$

$$+1.8227\, \tilde{v}_y[n-1] - 0.8372\, \tilde{v}_y[n-2] \tag{3.62}$$

- For z axis:

$$\tilde{v}_z[n] = 0.0187\, v_z[n] + 0.0373\, v_z[n-1] + 0.0187\, v_z[n-2] \tag{3.63}$$

$$+1.5782\, \tilde{v}_z[n-1] - 0.6528\, \tilde{v}_z[n-2] \tag{3.64}$$

- For Yaw axis:

$$\tilde{v}_{\psi}[n] = 0.0187\, v_{\psi}[n] + 0.0373\, v_{\psi}[n-1] + 0.0187\, v_{\psi} \tag{3.65}$$

$$+1.5782\, \tilde{v}_{\psi}[n-1] - 0.6528\, \tilde{v}_{\psi}[n-2] \tag{3.66}$$

The proper filters have been implemented as a class on C++ within anafi_ros package.

3.2.4 System Identification

After obtaining the velocity signals and removing the noise, we can use the enhanced data for obtaining the parameters of the discrete-time state space mentioned in Section 3.2.2.

By rewriting the lower half of the Equation 3.40 based on time-samples as follows we have:

$$\begin{bmatrix} x_5^{(1)} & x_6^{(1)} & x_7^{(1)} & x_8^{(1)} \\ x_5^{(2)} & x_6^{(2)} & x_7^{(2)} & x_8^{(2)} \\ x_5^{(3)} & x_6^{(3)} & x_7^{(3)} & x_8^{(3)} \\ \vdots & \vdots & \vdots & \vdots \\ x_5^{(N)} & x_6^{(N)} & x_7^{(N)} & x_8^{(N)} \end{bmatrix} =$$

$$\begin{bmatrix} x_5^{(0)} & x_6^{(0)} & x_7^{(0)} & x_8^{(0)} & u_1^{(0)} & u_2^{(0)} & u_3^{(0)} & u_4^{(0)} \\ x_5^{(1)} & x_6^{(1)} & x_7^{(1)} & x_8^{(1)} & u_1^{(1)} & u_2^{(1)} & u_3^{(1)} & u_4^{(1)} \\ x_5^{(2)} & x_6^{(2)} & x_7^{(2)} & x_8^{(2)} & u_1^{(2)} & u_2^{(2)} & u_3^{(2)} & u_4^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_5^{(N-1)} & x_6^{(N-1)} & x_7^{(N-1)} & x_8^{(N-1)} & u_1^{(N-1)} & u_2^{(N-1)} & u_3^{(N-1)} & u_4^{(N)} \end{bmatrix} \begin{bmatrix} a_x & 0 & 0 & 0 \\ 0 & a_y & 0 & 0 \\ 0 & 0 & a_Z & 0 \\ 0 & 0 & 0 & a_\psi \\ b_x & 0 & 0 & 0 \\ 0 & b_y & 0 & 0 \\ 0 & 0 & b_z & 0 \\ 0 & 0 & 0 & b_\psi \end{bmatrix} \tag{3.67}$$

Now, we need rewrite the matrices for each axis separately so we could have:

$$\begin{bmatrix} v_x^{(1)} \\ v_x^{(2)} \\ v_x^{(3)} \\ \vdots \\ v_x^{(N)} \end{bmatrix} = \begin{bmatrix} v_x^{(0)} & u_x^{(0)} \\ v_x^{(1)} & u_x^{(1)} \\ v_x^{(2)} & u_x^{(2)} \\ \vdots & \vdots \\ v_x^{(N-1)} & u_x^{(N-1)} \end{bmatrix} \begin{bmatrix} a'_x \\ b'_x \end{bmatrix} \tag{3.68}$$

$$\begin{bmatrix} v_y^{(1)} \\ v_y^{(2)} \\ v_y^{(3)} \\ \vdots \\ v_y^{(N)} \end{bmatrix} = \begin{bmatrix} v_y^{(0)} & u_y^{(0)} \\ v_y^{(1)} & u_y^{(1)} \\ v_y^{(2)} & u_y^{(2)} \\ \vdots & \vdots \\ v_y^{(N-1)} & u_y^{(N-1)} \end{bmatrix} \begin{bmatrix} a'_y \\ b'_y \end{bmatrix} \tag{3.69}$$

$$\begin{bmatrix} v_z^{(1)} \\ v_z^{(2)} \\ v_z^{(3)} \\ \vdots \\ v_z^{(N)} \end{bmatrix} = \begin{bmatrix} v_z^{(0)} & u_z^{(0)} \\ v_z^{(1)} & u_z^{(1)} \\ v_z^{(2)} & u_z^{(2)} \\ \vdots & \vdots \\ v_z^{(N-1)} & u_z^{(N-1)} \end{bmatrix} \begin{bmatrix} a'_z \\ b'_z \end{bmatrix} \tag{3.70}$$

$$\begin{bmatrix} v_\psi^{(1)} \\ v_\psi^{(2)} \\ v_\psi^{(3)} \\ \vdots \\ v_\psi^{(N)} \end{bmatrix} = \begin{bmatrix} v_\psi^{(0)} & u_\psi^{(0)} \\ v_\psi^{(1)} & u_\psi^{(1)} \\ v_\psi^{(2)} & u_\psi^{(2)} \\ \vdots & \vdots \\ v_\psi^{(N-1)} & u_\psi^{(N-1)} \end{bmatrix} \begin{bmatrix} a'_\psi \\ b'_\psi \end{bmatrix} \tag{3.71}$$

The N by 1 matrix on left is called the output matrix (y) and the N by 2 matrix on the right side of the equation is the matrix of regressors (X) which is multiplied by the matrix of the unknown parameters (β). To obtain the system parameters of the $\hat{\beta}$ matrix, we shall take the least square approach for obtaining the estimated parameters as $\hat{\beta}$ matrix which is formulated as follows [58, 59]:

$$\hat{\beta} = (X^T X)^{-1} X^T y \tag{3.72}$$

Therefore, by obtaining the proper data, $\hat{\beta}$ could be identified using Equation 3.72. For obtaining the proper identification dataset, a pulse input with time period of $10(s)$ is applied to the drone separately on each four axes.

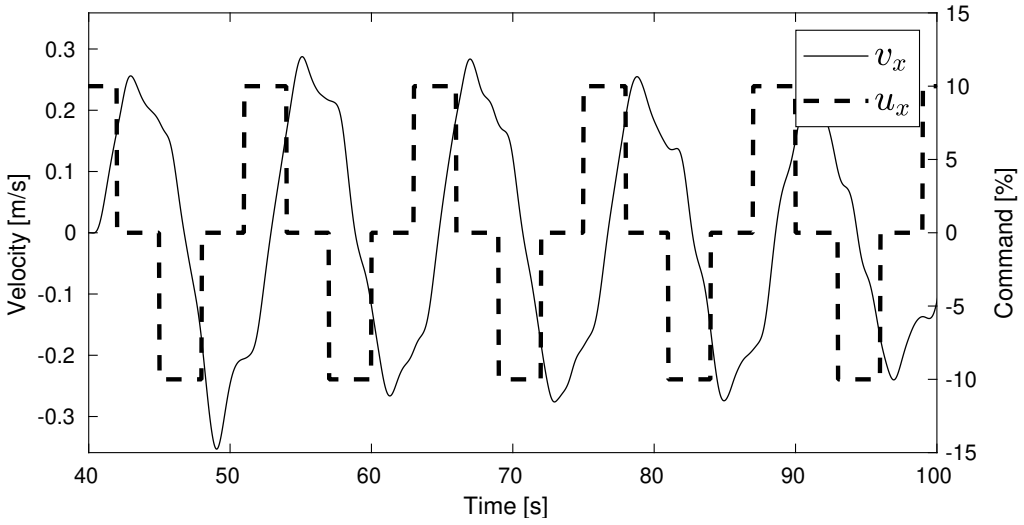


Figure 3.8: Pulse Train and Corresponding Response on x-axis

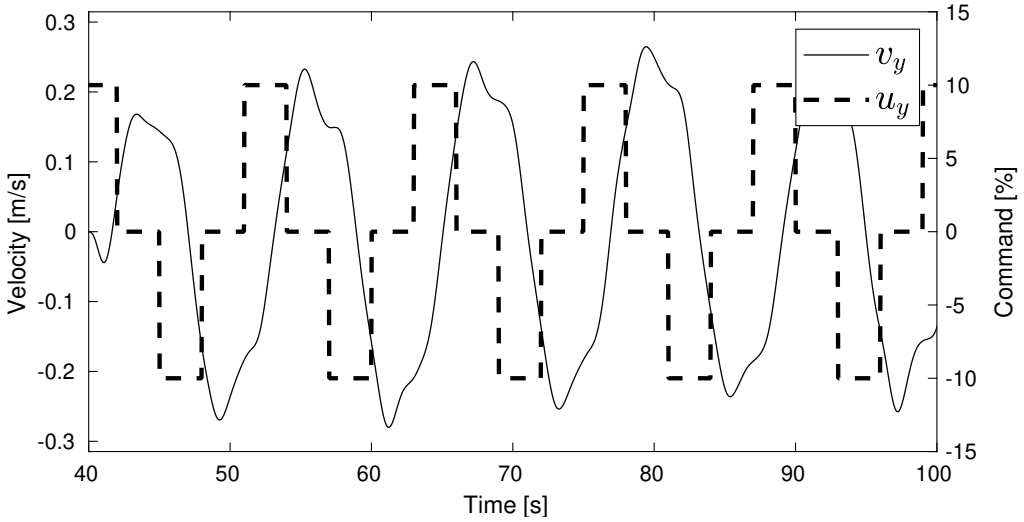


Figure 3.9: Pulse Train and Corresponding Response on y-axis

As it is represented in Figure 3.8, a periodic pulse input (u_x) with the amplitude of 10% and time period of 10 seconds is applied to the drone and and the velocity response (v_x) of the system is a periodic signal as well with an amplitude of 0.25 m/s.

Similarly, for y axis the same periodic pulse input has been applied and the output has the average amplitude of 0.22 m/s as it is represented in Figure 3.9.

For the z axis as well, the input periodic signal has the same frequency and the magnitude of 10% as the x and y axes, however as it is represented in Figure 3.10, the dynamical nature of z is different from the previous two ones. In fact, z axis has

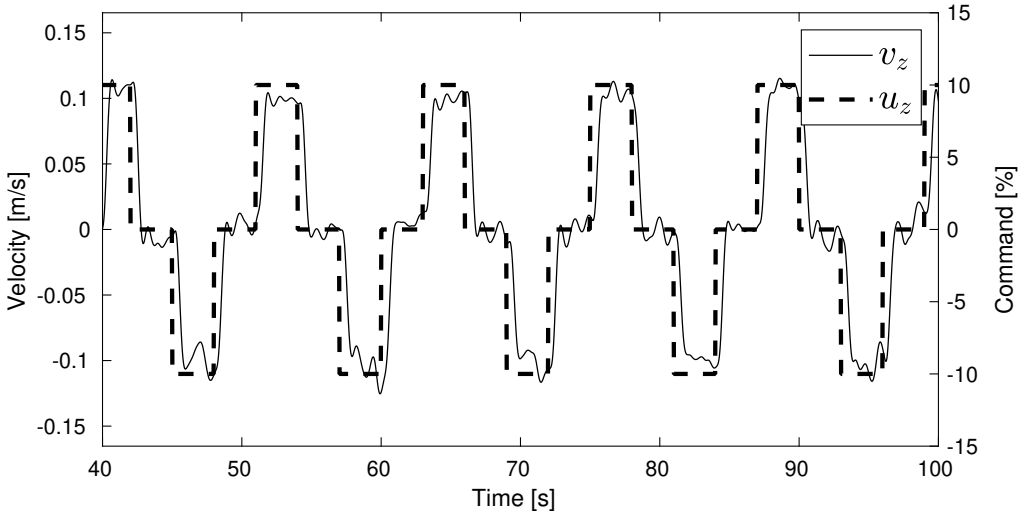


Figure 3.10: Pulse Train and Corresponding Response on z-axis

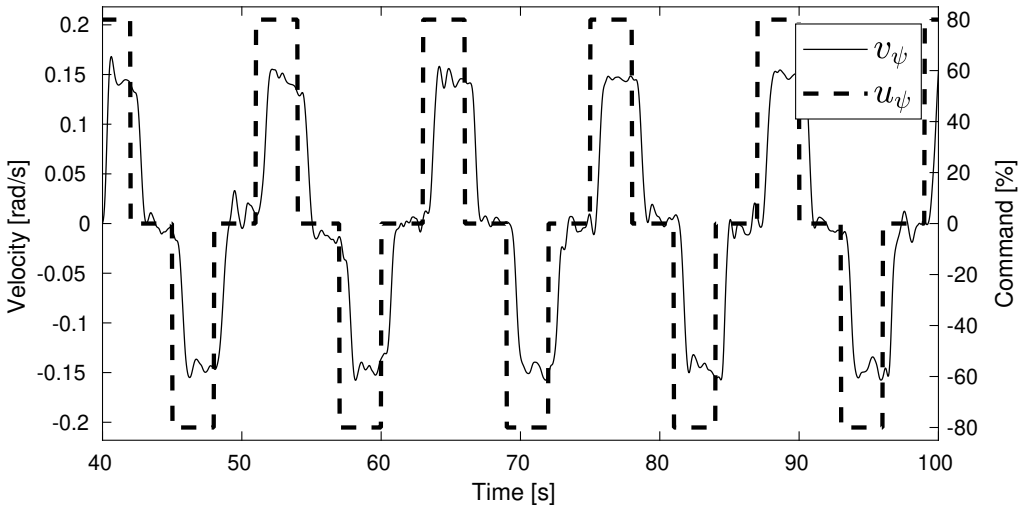


Figure 3.11: Pulse Train and Corresponding Response on ψ-axis

a faster dynamic than the previous two axes.

For identifying the ψ axis, a periodic pulse input with an amplitude of 20% has been applied to the system with the same time period of 10 seconds. The system responds to the input with an output periodic velocity signal with the same frequency and magnitude of 0.15 rad/s as plotted in Figure 3.11.

Now, after obtaining the proper identification dataset, we can now acquire the least square formulation to estimate the system parameters.

Using MATLAB `lsqr()` function we shall have [60]:

$$\hat{\beta}_x = \begin{bmatrix} 0.9969 \\ 0.0005 \end{bmatrix},\ \hat{\beta}_y = \begin{bmatrix} 0.9985 \\ 0.0005 \end{bmatrix},\ \hat{\beta}_z = \begin{bmatrix} 0.5377 \\ 0.0049 \end{bmatrix},\ \hat{\beta}_\psi = \begin{bmatrix} 0.9486 \\ 0.0001 \end{bmatrix} \tag{3.73}$$

Therefore, now we can rewrite Equation 3.40 and 3.32 as follows:

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \\ x_3[k+1] \\ x_4[k+1] \\ x_5[k+1] \\ x_6[k+1] \\ x_7[k+1] \\ x_8[k+1] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0.04 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0.04 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0.04 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0.04 \\ 0 & 0 & 0 & 0 & 0.9969 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.9985 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5377 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9486 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \\ x_5[k] \\ x_6[k] \\ x_7[k] \\ x_8[k] \end{bmatrix}$$
$$+ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.0005 & 0 & 0 & 0 \\ 0 & 0.0005 & 0 & 0 \\ 0 & 0 & 0.0049 & 0 \\ 0 & 0 & 0 & 0.0001 \end{bmatrix} \begin{bmatrix} u_1[k] \\ u_2[k] \\ u_3[k] \\ u_4[k] \end{bmatrix} \tag{3.74}$$

$$\begin{bmatrix} y_1[k] \\ y_2[k] \\ y_3[k] \\ y_4[k] \\ y_5[k] \\ y_6[k] \\ y_7[k] \\ y_8[k] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \\ x_5[k] \\ x_6[k] \\ x_7[k] \\ x_8[k] \end{bmatrix}$$

$$+ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_3 \end{bmatrix} \quad (3.75)$$

Now that the system is identified, it is time to evaluate the identified model and compare the estimated and actual response of the system. By defining the discrete state space model on MATLAB using `ss()` function and simulating the response using `lsim()` we would have the following results [61, 62].

As it is shown in Figures 3.12, 3.13, 3.14 and 3.15, the originals signals r_x, r_y, r_z and r_ψ and model estimated position signals $\hat{r}_x$, $\hat{r}_y$, $\hat{r}_z$ and $\hat{r}_\psi$ almost fit and thus the model is able to properly represent the dynamics of the system.

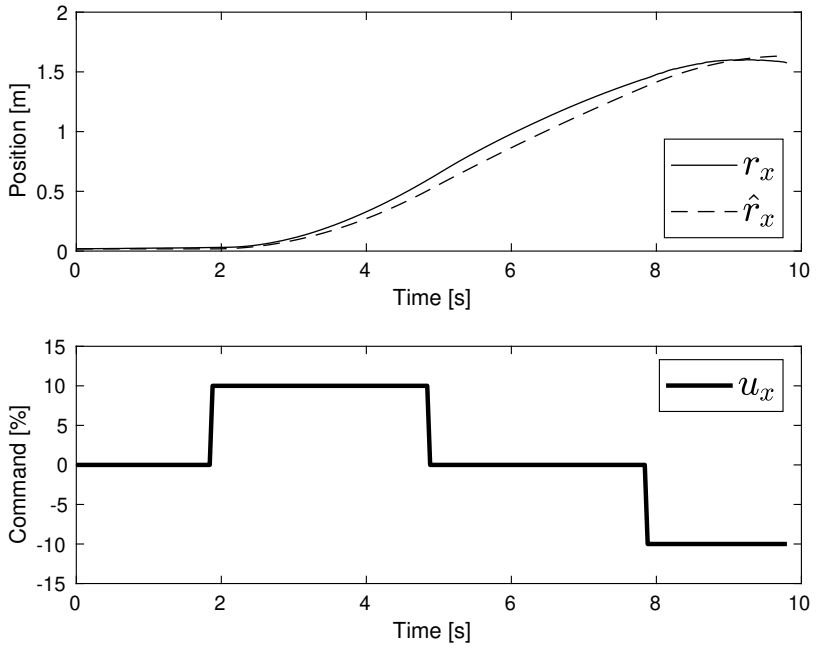


Figure 3.12: Original signal and the model output on x-axis

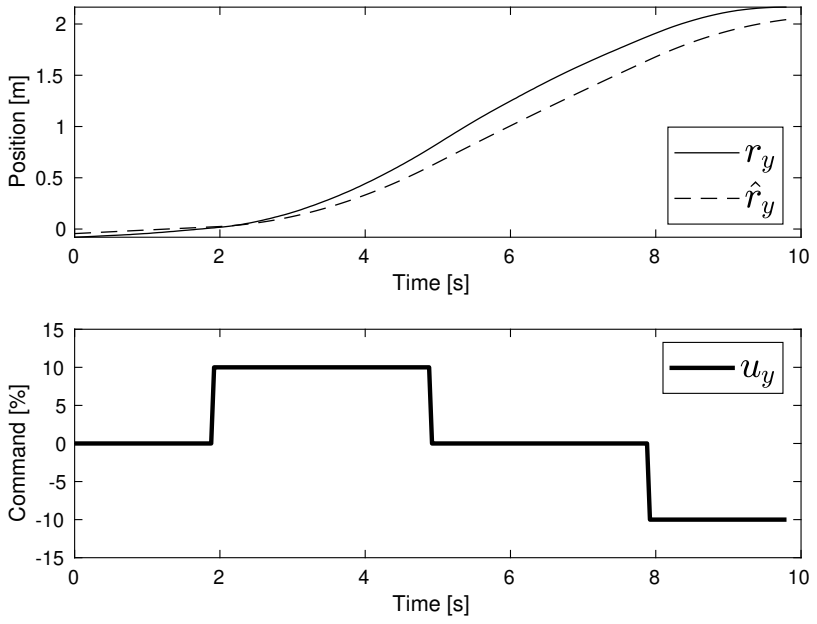


Figure 3.13: Original signal and the model output on y-axis

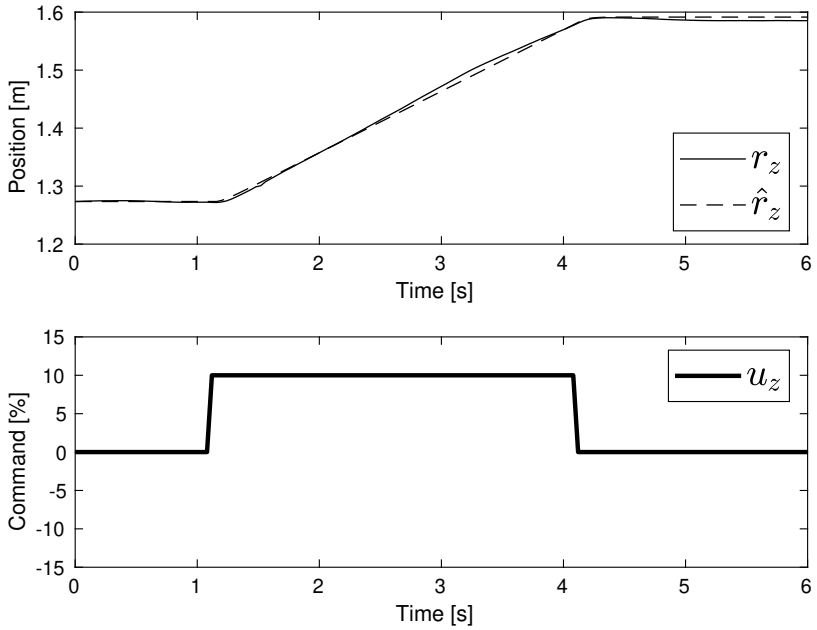


Figure 3.14: Original signal and the model output on z-axis

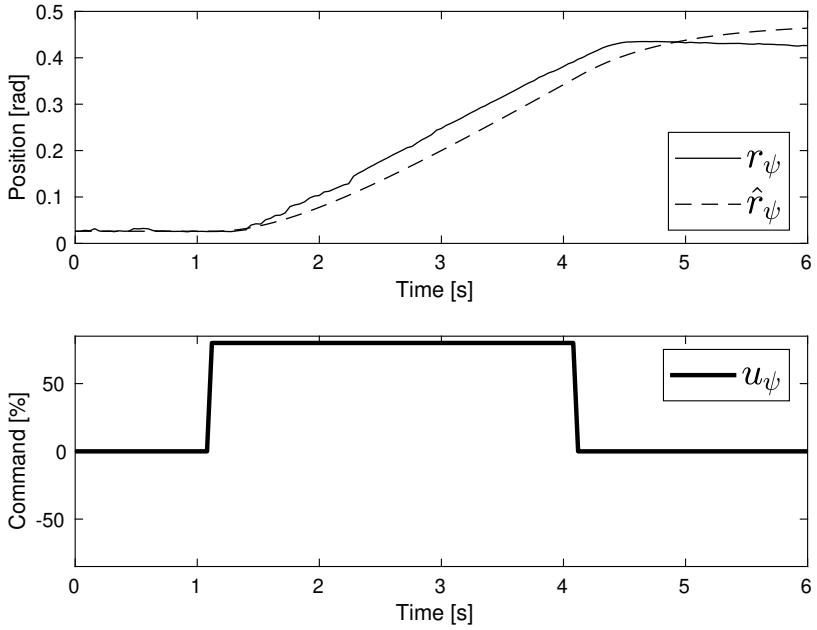


Figure 3.15: Original signal and the model output on ψ-axis

3.2.5 Digital PID Control

Developed in early 20^{th} century by Nicolas Minorsky [63], Proportional Integral Derivative (PID) control has proven to be the most practical, versatile and popular controller in the industry. The ingenuity of PID is that it can control most systems with only three parameters. The only challenge then is to find the proper three parameters for tuning the controller [21, 64].

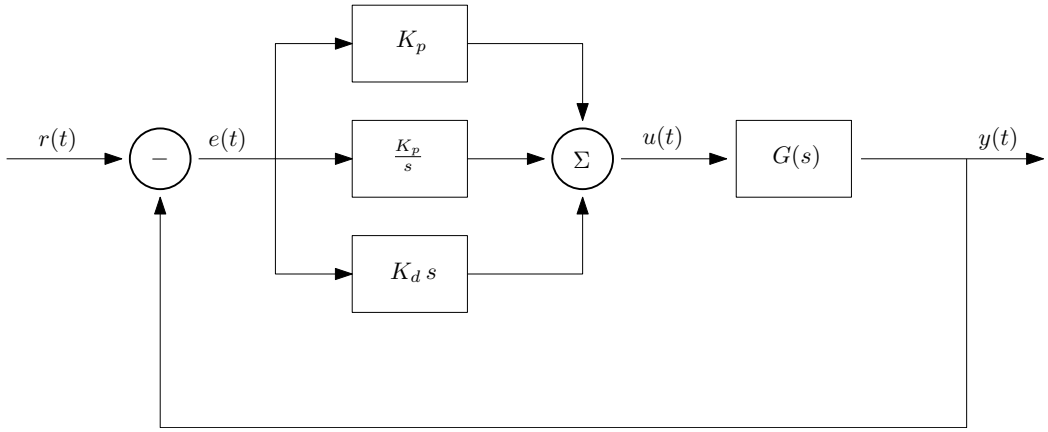


Figure 3.16: Block Diagram of Continuous PID Controller

Figure 3.16, represents the schematic block diagram of a continuous system with PID controller. As it is shown, the PID block contains three components of gain (K_p), integrator ($\frac{K_i}{s}$) and derivative ($K_d\, s$). The relationship between the command signal $u(t)$ and error $e(t)$ is as follows [21]:

$$u(t) = K_p\, e(t) + K_i \int_0^t e(\tau) d\tau + K_d\, \dot{e}(t) \tag{3.76}$$

However, as it was mentioned earlier, due to the digital nature of the processing unit of ROS, it is needed to analyze PID controller in discrete-time horizon to fulfill the performance requirements. By using the Z-transform and utilizing the same architecture, we shall have the following block diagram of Figure 3.17.

As it is represented, the digital PID controller is composed of three components as the analog one, however, the placement of the poles are dependant of the sampling

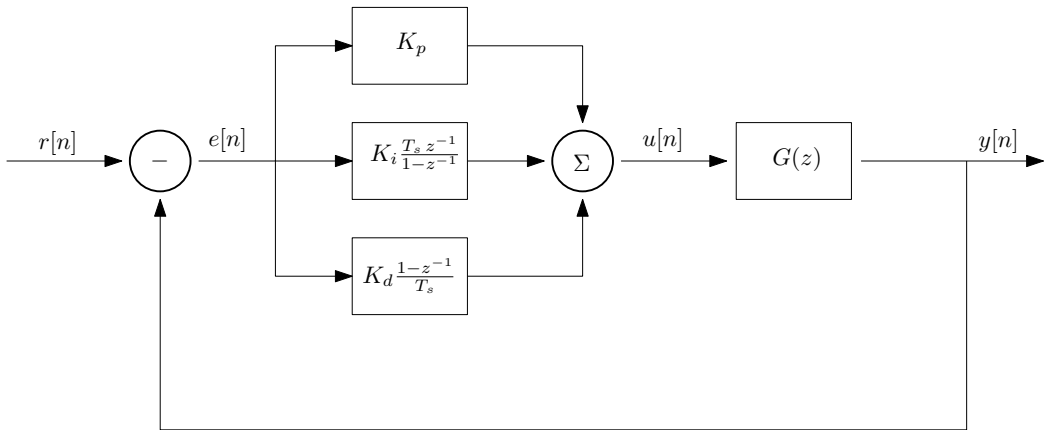


Figure 3.17: Block Diagram of Digital PID Controller

time T_s as well as the coefficients. The following equations expresses the relationship between the control signal and error [55]:

$$u[n] = K_p\, e[n] + K_i\, T_s \sum_{i=0}^{n} e[n] + \frac{K_d}{T_s}\left(e[n] - e[n-1]\right) \tag{3.77}$$

The advantage of the digital controllers to the analog counterparts is the easy and accessible implementation on microcontrollers or any other digital processing units using programming languages like C++ or Python. Before the invention of digital computers, the PID controllers were implemented using electronic devices including op-amps, resistors and capacitors. Therefore, making changes to the controller would require hardware manipulation of the controller whereas for digital controllers, it would only a few seconds to edit the codes and recompile it.

Flight Control

For designing a flight controller for the Anafi drone, it is needed to utilize for PID loop to control each four axes independently. Therefore, as it is represented in Figure 3.18, the controller consists of four PID loops tasked with controlling x, y, z and ψ. The Vicon acts as the feedback sensor, feeding the pose data of the Anafi into the controller through ROS topics. At the same time, the feedback data are quantized and digitalized by Vicon and `vicon_bridge` as well. Afterwards, the error is

calculated by subtracting the feedback data (`/anafi/fbData`) from the setpoint data (`/anafi/spData`) through the PID block. Consequently, the digital control signals are transformed and transmitted to the drone by Olympe library and wi-fi routers.

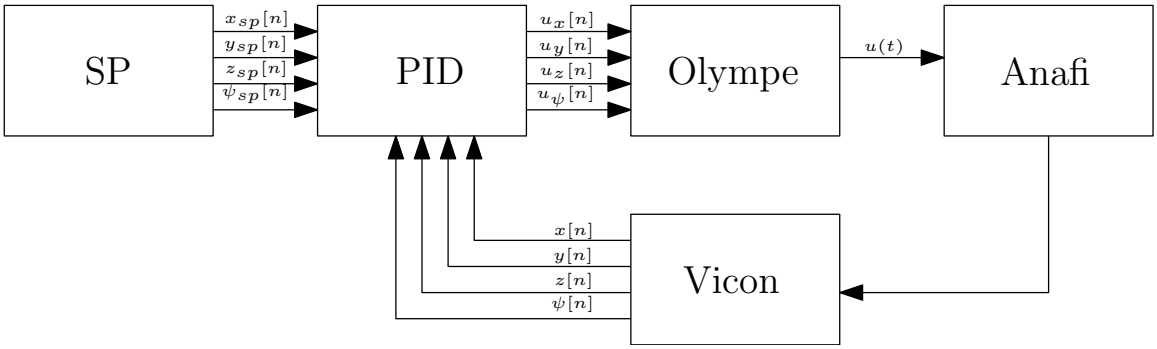


Figure 3.18: 4-Axis Flight Control PID Framework

The framework of the flight controller is now set up and now it is only needed to tune the PID parameters accordingly.

Tuning

There are a myriad of methods for tuning PID controllers, which could be categorized into theoretical and experimental or continuous and discrete methods. Analytical methods are solely based on the identified models whereas the experimental ones are based on engaging the controller in real-time horizon with the system and tuning the parameters by observing the corresponding responses.

As it was discussed earlier, systems could be expressed in continuous or discrete time manners. Continuous systems are represented by Laplace transform whereas discrete systems should be represented using Z-transform. Thus, the nature of the system must be taken into consideration for tuning the controllers.

For tuning the flight controller, we shall use a combination of theoretical and analytical methods. First, by using pole-placement method [65] the parameters are tuned to ensure the stability of the system, then by returning the parameters experimentally, the output response is going to be enhanced and re-tuned.

As it is shown in Figure 3.19, the open loop poles are in fact the eigenvalues of A matrix and the eigenvalues are placed on the main diagonal of A as it is a triangular matrix. Therefore, each axis has a stable real pole inside the unit circle and another real pole on $z = 1$ which acts as an integrator. As a matter of fact, we need to either move this pole into the centre of unit circle or cancel them by a zero to stabilize and increase the speed of the system.

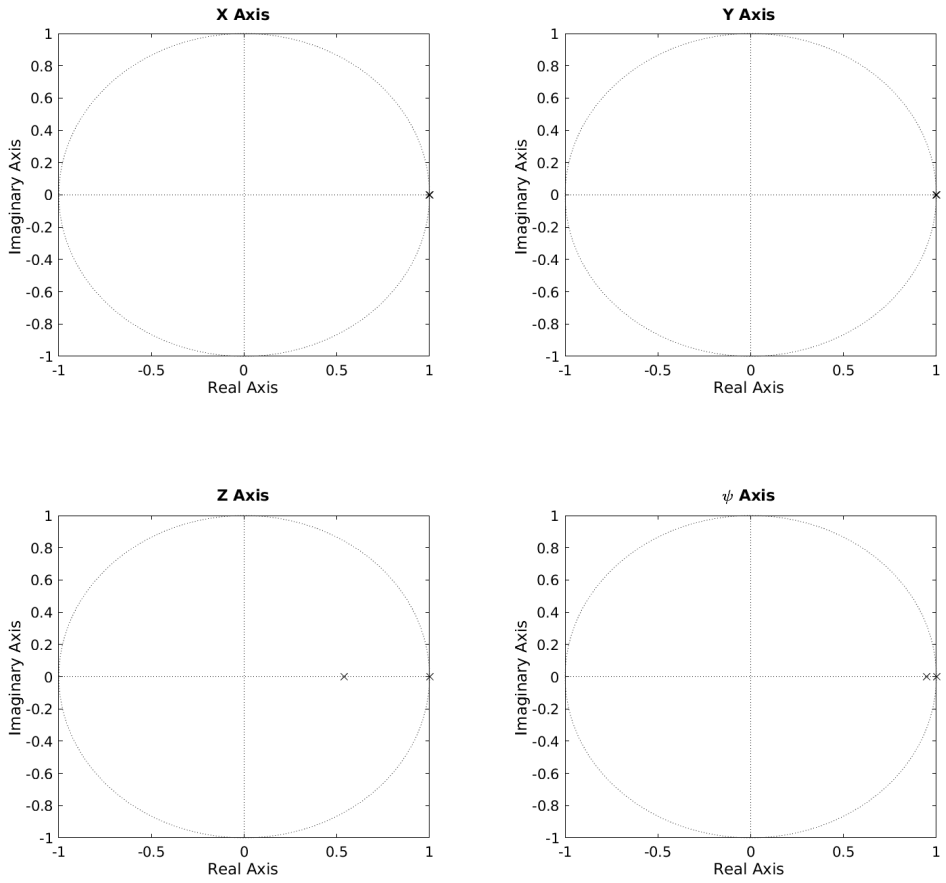


Figure 3.19: PID Control Implementation on ROS

The derived pole-placement parameters for PID flight controllers are summarized in Table 3.1. Utilizing these coefficients into PID controllers in a close loop manner would modify the open-loop poles and zeros into the new placement as represented

Table 3.1: PID Coefficients

Axis	K_p	K_i	K_d
Anafi World Frame X Axis ($^{\mathcal{W}}x$)	4.603	0.901	7.604
Anafi World Frame Y Axis ($^{\mathcal{W}}y$)	4.603	0.901	7.604
Anafi World Frame Z Axis ($^{\mathcal{W}}z$)	7.5	5.094	0.12
Anafi World Frame Yaw Axis ($^{\mathcal{W}}\psi$)	7.538	0.98	0.452

in Figure 3.20.

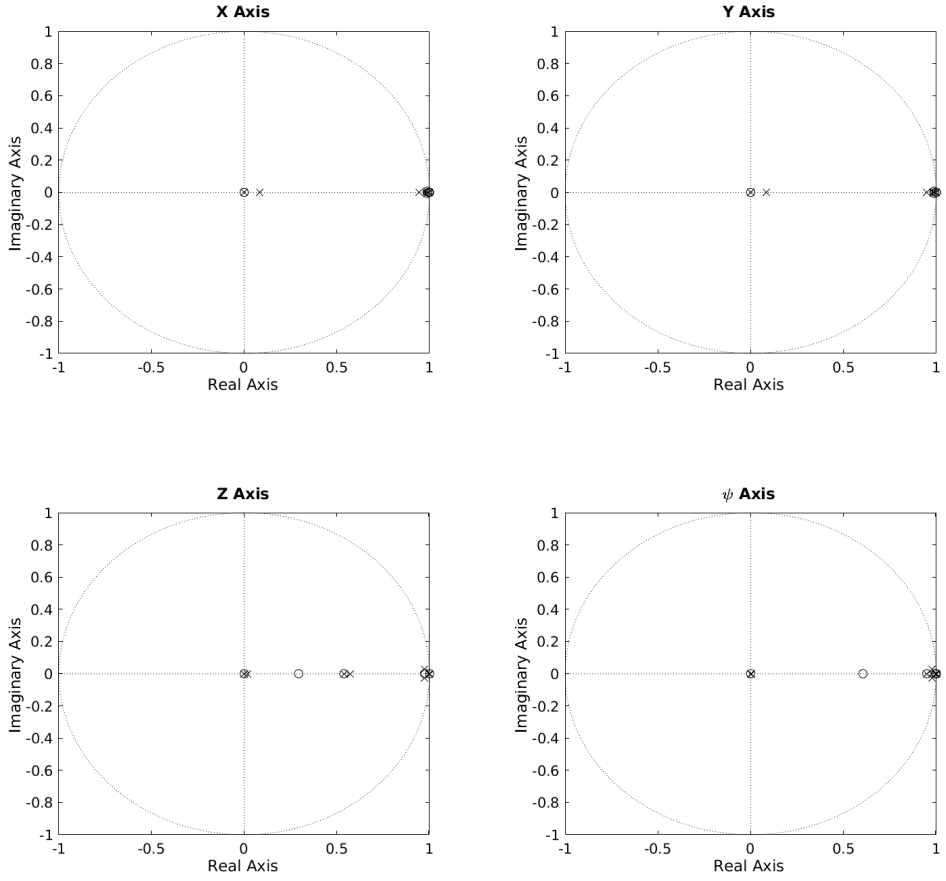


Figure 3.20: PID Control Implementation on ROS

As it is shown in Figure 3.20, the open-loop poles near the unit circle are cancelled

by two zeros and new poles are placed near the origin which results in stable and fast response in the output. The evaluation and results of the designed controller shall be discussed in Chapter 5 thoroughly.

3.2.6 ROS Implementation

As it was mentioned in the introduction of this chapter, the PID controllers and flight control systems of the Anafi drones are to be implemented on ROS for integrating different modules and processing units into a single entity.

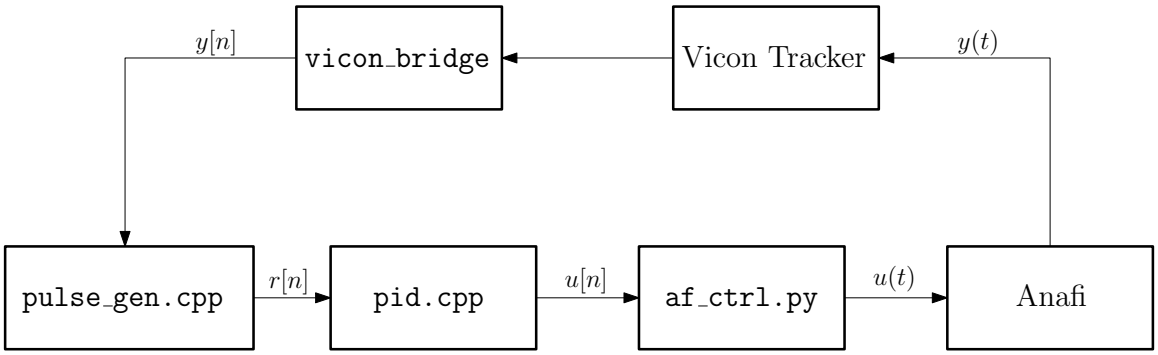


Figure 3.21: PID Control Implementation on ROS

Figure 3.21 illustrates the block diagram of the PID control implementation on ROS. As it is shown, the pose of Anafi is captured by Vicon system and is then transmitted to `vicon_bridge` ROS package which then broadcasts the pose as a `/tf` topic which could be listened by any ROS node. At the same time, the desired setpoints (`/anafi/spData`) are published by the `pulse_gen.cpp` C++ node [66] which could be connected to `rqt` GUI as well to change the setpoints manually by user as well. Then, the current state of the drone along with the desired setpoints are fed into the PID block which then publishes the control as `/cmd_vel` topic. Eventually, the `af_ctrl.py` subscribes to the `/cmd_vel` topic and transmits the command velocities to the drone using Olympe library through wireless communications.

Algorithm 1 details the steps for obtaining the control signals and command velocities by implementing the PID block on a C++ node on ROS. The algorithms is quite

straightforward, the current state of the done is received as a stamped transform message in additions to the setpoint data which is received through a custom ROS topic called spData. Consequently, through the while loop the current states and setpoints are stored and then in the for loop, the error is calculated and is then passed to PID blocks to obtain the control signal. Anti-windup option has been implemented too to ensure the proper performance of integrator. Subsequently, after the for loop, the obtained control signals are pass to chopper if block to prevent malfunction of the transmitter.

Algorithm 1 PID ROS Algorithm

Input: Anafi Stamped Transform, Setpoint Data
Output: Velocity Commands

```
nh ← ros::NodeHandle
tranfsorm ← tf::StampedTransform
while nh.ok() do
    x, y, z, ψ ← transform
    for i = 1 to 4 do
        Calculate the Error (e)
        Multiply the Error by Gain (e_p)
        Integrate the Error (e_i)
        Differentiate the Error (e_d)
        if e_i > 50 then                          ▷ Integrator Anti-windup
            e_3 ← 50
        else if e_i < −50 then
            e_i ← −50
        end if
        u_i ← e_i + e_p + e_i                     ▷ Control Signal Calculation
    end for
    if u_i > 100 then                             ▷ Signal Chopper
        u_i ← 100
    else if u_i < −100 then
        u_i ← −100
    end if
end while
```

3.3 Pursuit

As it was discussed in section 3.2.6, the PID block is tasked with tracking the desired setpoints by sending the required flight control signals to the drone. For enabling the drone to pursue a target, we need to obtain the pose of the target and transmit the pose data as the setpoint into the PID block. In this research, the target is in fact a Bebop drone.

The Bebop is going to be tracked by two approaches, Vicon and AI. As it was mentioned earlier, Vicon tracker can captures the frame pose of Anafi and Bebop drones and sending them to ROS using `vicon_bridge`. On the other hand, we aim to utilize deep learning methods to obtain the relative distance and pose of Bebop with respect to Anafi. Therefore, Vicon is going to be employed for obtaining training data, tuning data and evaluating pursuit algorithms in absence of deep learning nodes.

As a result, once the competency of the pursuit algorithms are evaluated with Vicon, then the deep learning methods could be deployed to perform the pursuit based on the estimated pose.

3.3.1 Formulation

For formulating the pursuit, we need to have an understanding of the geometric topology of the drones in addition to their spatial representation.

Figure 3.22 illustrates the schematic representation of the frame coordinates of the Anafi ($\mathcal{A}$) and Bebop ($\mathcal{B}$) drones with respect to the world frame ($\mathcal{W}$). The goal here is to find the coordinates of the desired setpoint, but first we need to go through some definition. The displacement vectors of Anafi and Bebop are defined as follows:

$$\vec{d}_a = \begin{pmatrix} x_a \\ y_a \\ z_a \end{pmatrix} \; ; \; \vec{d}_b = \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} \tag{3.78}$$

For performing the pursuit without annihilating the target, it is needed to fly the

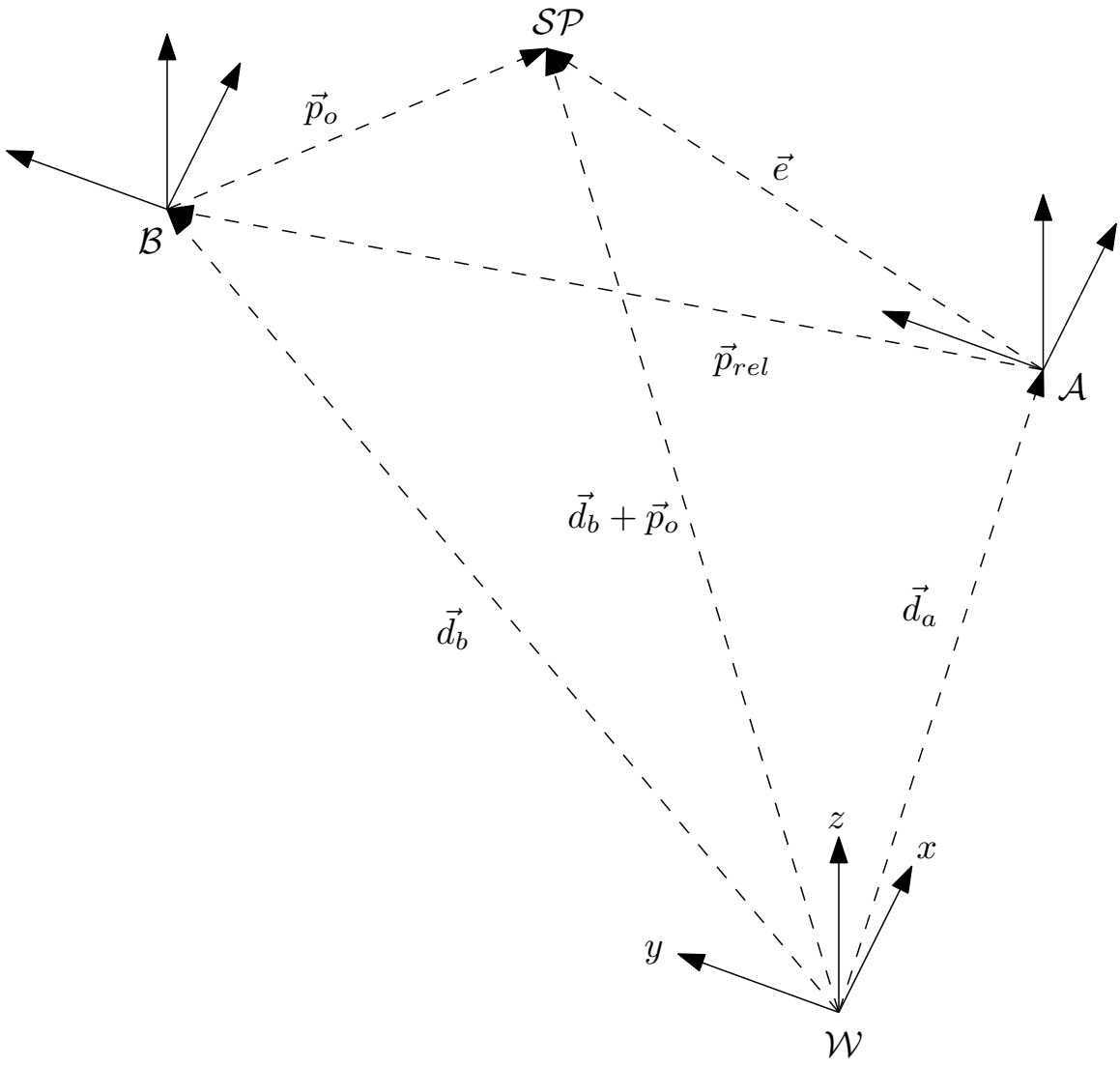


Figure 3.22: Geometric Schematics of Aerial Pursuit

Anafi over an offset zone with respect to Bebop meaning that a minimum distance would be maintained for preventing the crash of the drones. For this research, the offset is 1.5 meters over x axis as follows:

$$\vec{p}_o = \begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix} = \begin{pmatrix} -1.5 \\ 0 \\ 0 \end{pmatrix} \tag{3.79}$$

Consequently, the setpoint coordinates could be calculated through the following formulation:

$$\vec{p}_{sp} = \vec{d}_b + R_b \vec{p}_o \tag{3.80}$$

Where R_b is the rotation matrix of Bebop drone. As a result, the error vector of desired setpoint and current state of Anafi is as follows:

$$\vec{e} = \vec{p}_{sp} - \vec{d}_a = \vec{d}_b + R_b \vec{p}_o - \vec{d}_a \tag{3.81}$$

Equation 3.81 is used for obtaining the setpoint error when the pose and coordinates of Bebop are being tracked by the vicon. For obtaining the setpoint coordinates using relative distance which is to be obtained from the AI algorithms we have:

$$\vec{p}_{sp} = \vec{d}_a + R_a \vec{p}_{rel} + R_b \vec{p}_o \tag{3.82}$$

Therefore:

$$\vec{e} = \vec{d}_a + R_a \vec{p}_{rel} + R_b \vec{p}_o - \vec{d}_a = R_a \vec{p}_{rel} + R_b \vec{p}_o \tag{3.83}$$

Where R_a is the rotation matrix of Anafi drone.

3.3.2 Framework

As it was discussed earlier, we shall set two distinguished frameworks for pursuit algorithms, one for Vicon and and one for deep learning methods.

The Vicon-based pursuit framework is represented in Figure 3.23. This frameworks resembles the PID framework in some sense. The difference here though, is that the fact that the setpoints are generated by `vicon_pursuit.cpp` node which receives the pose of Anafi and Bebop from the `vicon_bridge` [67] and publishes the `/anafi/spData` for the PID node which then sends the command to Python node communicating with the drone.

On the other hand, for the AI-based pursuit it is aimed to obtain the relative pose of the Bebop drone through vision data. Figure 3.24 demonstrates the proposed

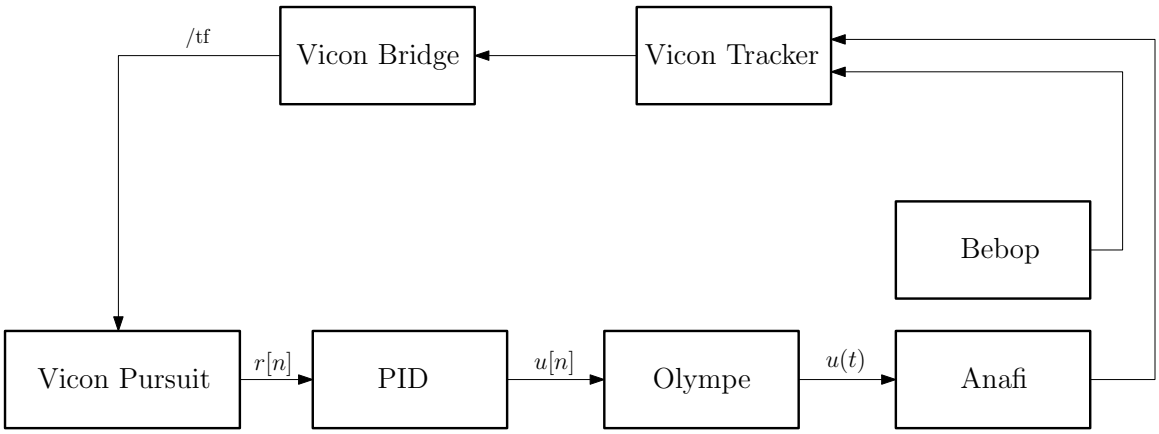


Figure 3.23: Vicon-Based Pursuit Framework on ROS

framework for processing and utilizing the vision data into pursuit mechanism.

As it is shown, Anafi drone captures the frames of Bebop and sends them as `/anafi/frames` topic into RCNN node. This node utilizes PyTorch library and Keypoint-RCNN architecture to obtain the keypoints (frame projection of the vertices of the 3D bounding box circumscribing the Bebop), and then are published as `/anafi/kpData` topic into PnP node.

PnP node is a C++ node which includes OpenCV library containing the `solvePnP()` function which takes the object points, image points (keypoints), camera matrix and distortion coefficients as input arguments and returns `rvec`, `tvec` or relative rotation and translation of the target with respect to the camera. The relative pose data are then published as `/anafi/pnp` topic into the Vicon Pursuit node.

As the Vicon-based pursuit method, Vicon Pursuit is tasked with generating the reference signal or the setpoints for the PID block. As opposed to the previous method, here Vicon Pursuit block subscribes to Vicon and PnP at the same time to obtain the current world frame pose of the Anafi and then it calculates the world frame pose of Bebop by using the relative distance and current pose of the Anafi. The reference signal ($r[n]$) or setpoints are then published as `/anafi/spData` topic into the PID block.

The rest of the process is the same as the PID block is independent of the method

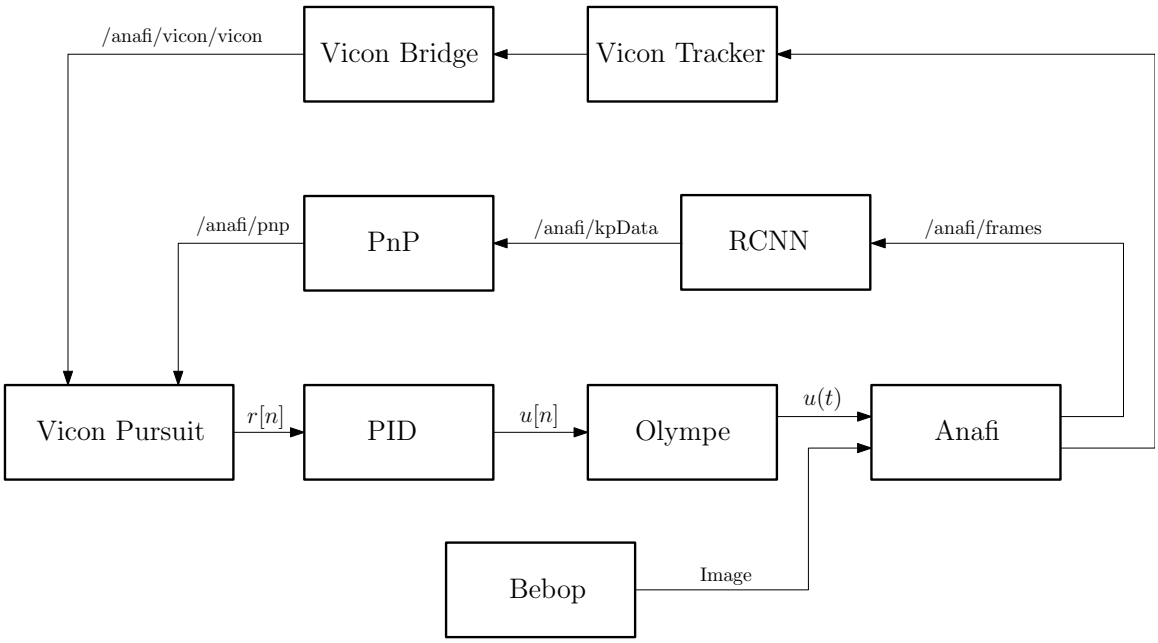


Figure 3.24: AI-Based Pursuit Framework on ROS

and only works by taking the references signal and then publishing and feeding the control signal ($u[n]$) into the Olympe as `/cmd_vel`. Consequently, as mentioned in the previous method, the Olympe sends the control command velocities to the drone via PCMD.

Chapter 4

Pose Estimation

This chapter describes the basis of computer vision concepts by first going through the camera model and projection matrix. These concepts are necessary for understanding the role played by the camera in transforming the spatial coordinates into camera coordinates. Then, we shall review the concept of 3D bounding box and its representation in world coordinate systems and in the one of the camera. In the next section, the framework of the AI algorithms shall be discussed. This framework includes region-based convolutional neural networks which are utilized into a processing entity to detect and drone and its circumscribing 3D bounding as keypoints. Eventually, the last section shall describe the procedure of obtaining the relative pose from the detected 3D bounding box keypoints using Perspective-n-Point method.

4.1 Camera Model

Camera is a device which in fact maps the 3D spatial points into a 2D frame and stores the corresponding visual data of each point as a one-channel black and white image or a three-channel color one. Camera could store the visual as an analog signal on negatives or as digital signal on memories. We discussed the nature of analog and signals in Section 3.1.2 thoroughly.

Figure 4.1 represents the schematics of perspective projection from world frame to image plane. For the know point of P could be represented by vector $\vec{x}_w$ in the

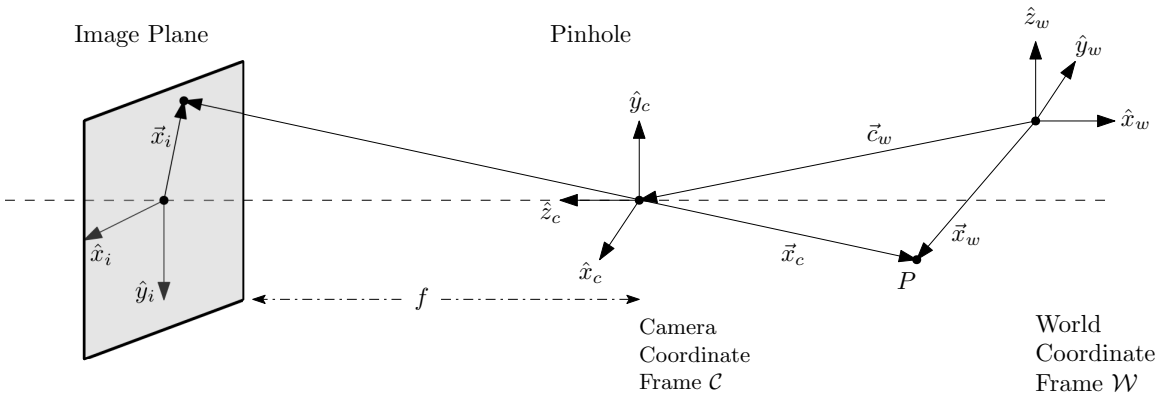


Figure 4.1: Camera Perspective Projection Schematics

world plane. By using a matrix transformation which will later will be discussed as extrinsic matrix, $\vec{x}_w$ would be transformed into $\vec{x}_c$ which is the spatial representation of P with respect to Camera plane [68, 69].

$$\mathtt{x}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \quad \Longrightarrow \quad \mathtt{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \quad \Longrightarrow \quad \mathtt{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{4.1}$$

By using trigonometry over pinhole we shall have:

$$\frac{x_i}{f} = \frac{x_c}{z_c} \;\; ; \;\; \frac{y_i}{f} = \frac{y_c}{z_c} \tag{4.2}$$

Therefore:

$$x_i = f\,\frac{x_c}{z_c} \;\; ; \;\; y_i = f\,\frac{y_c}{z_c} \tag{4.3}$$

As it was stated earlier, digital cameras digitize the image into pixels to store

them as arrays. Thus, the image plane is transformed into pixel plane which as it is represented in Figure 4.2. It is common to assign the left top corner of the image as the origin for pixel coordinates which is called principal point represented as (o_x, o_y) [70–72].

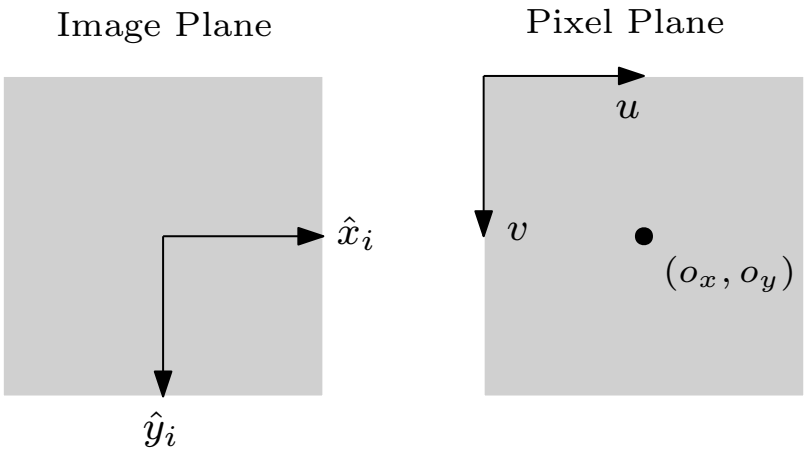


Figure 4.2: Image and Pixel Planes

Rewriting Equation 4.3 into digital form yields the following formulations:

$$u = m_x f \frac{x_c}{z_c} + o_x \tag{4.4}$$

$$v = m_y f \frac{y_c}{z_c} + o_y \tag{4.5}$$

Hence:

$$u = f_x \frac{x_c}{z_c} + o_x \tag{4.6}$$

$$v = f_y \frac{y_c}{z_c} + o_y \tag{4.7}$$

A 2D point on image plane represents infinite number of points in 3D spatial plane. Therefore, to transform the 2D vector (u, v) into 3D plane homogeneously, it is required to consider the $\tilde{w}$ coefficients into the vector as it is formulated below:

$$\mathtt{u} \equiv \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{w}\, u \\ \tilde{w}\, v \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \tag{4.8}$$

Hence, the transformed homogeneous 3D vector $\tilde{u}$ is:

$$\tilde{\mathtt{u}} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \tag{4.9}$$

Similarly, to represent the 3D spatial vector of (x, y, z) homogeneously, it is needed to transform it into a 4D vector by adding $\tilde{w}$ coefficient as follows:

$$\mathtt{x} \equiv \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{w}\, x \\ \tilde{w}\, y \\ \tilde{w}\, z \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{bmatrix} \tag{4.10}$$

Thus:

$$\tilde{\mathtt{x}} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{bmatrix} \tag{4.11}$$

4.1.1 Intrinsic Matrix

By rewriting Equation 4.9 and substituting $\tilde{w}$ with z_c we shall have:

$$\tilde{\mathtt{u}} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} z_c\, u \\ z_c\, v \\ z_c \end{bmatrix} \begin{bmatrix} f_x x_c + z_c o_x \\ f_y y_c + z_c o_y \\ z_c \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{4.12}$$

Consequently:

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{4.13}$$

$$\tilde{\mathtt{u}} = M_{int}\, \tilde{\mathtt{x}}_c \tag{4.14}$$

The 3x4 matrix which transforms the $\mathtt{x}_c$ into $\mathtt{u}$ is called the intrinsic matrix. The 3x3 left half of the intrinsic matrix which is composed of camera parameters is called camera matrix consisting of focal and principle parameters.

$$M_c = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

4.1.2 Extrinsic Matrix

Intrinsic matrix transforms the 3D object coordinate $\mathtt{x}_c$ (with respect to camera frame) to $\tilde{\mathtt{u}}$ on image plane. However, in most cases the 3D object point is represented with respect to the world coordinate frame. To transform the world coordinates

into camera coordinates, it is needed to obtain the following matrix to perform the transformation.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{4.16}$$

$$\tilde{\mathtt{x}}_c = M_{ext}\, \tilde{\mathtt{x}}_w \tag{4.17}$$

The 4x4 matrix transforming the $\mathtt{x}_w$ into $\mathtt{x}_c$ is called the extrinsic matrix. This matrix consists of two sections of rotation matrix and translation matrix. Rotation matrix is in fact the combination of 3 axial rotations explained in Section 3.1.1.3. Additionally, the translation vector (t_x, t_y, t_z) expresses the translational displacement of camera frame with respect to the world frame.

4.1.3 Projection Matrix

Combining the intrinsic and extrinsic matrices leads to the following calculations:

$$\tilde{\mathtt{u}} = M_{int}\, M_{ext}\, \tilde{\mathtt{x}}_w = P\, \tilde{\mathtt{x}}_w \tag{4.18}$$

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{4.19}$$

Therefore, the derived 3x4 matrix which maps the world frame coordinates directly to the image plane, is called the projection matrix.

4.1.4 Camera Calibration

The extrinsic matrix could be retrieved form the Vicon system by obtaining the orientation and pose of the Anafi camera with respect to the world frame. However,

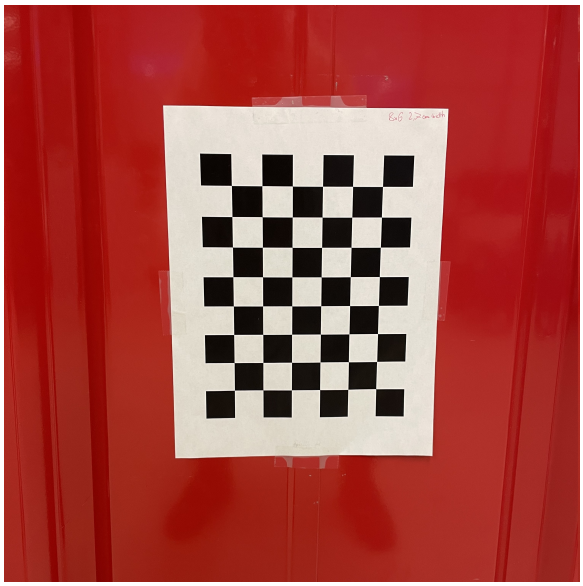


Figure 4.3: Calibration Checkerboard

for obtaining the intrinsic matrix and obtaining the camera parameters, it is needed to perform a process which is called camera calibration.

`camera_calibration` is a ROS package which automatically obtain the camera matrix by utilizing OpenCV library. The idea is to detect the corners of the squares in an identified checkerboard, as shown in Figure 4.3, and find the best intrinsic matrix to map the spatial coordinates of the checkerboard corners into the detected edges as shown in Figure 4.4.

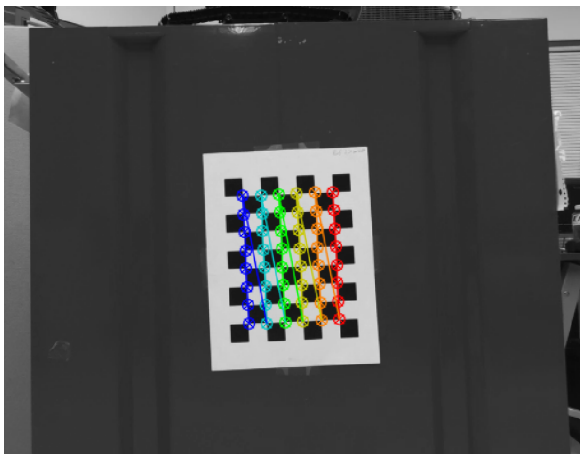


Figure 4.4: Calibration Process

Using `camera_calibration` package, the intrinsic parameters of Anafi camera are as follow:

$$M_{int} = \begin{bmatrix} 941.35 & 0 & 638.96 & 0 \\ 0 & 948.55 & 375.83 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.20}$$

4.2 3D Bounding Box

In a sense, a bounding box is a visual and geometric entity which circumscribes an object. A 2D bounding box is in fact a rectangle which determines the 2D pose of the detected object through image plane. Therefore, a 2D bounding box could be identified using four parameters of corner coordinates (u_0, v_0) of the box in addition to the length and width of the box (h, w) as it is represented in Figure 4.5.

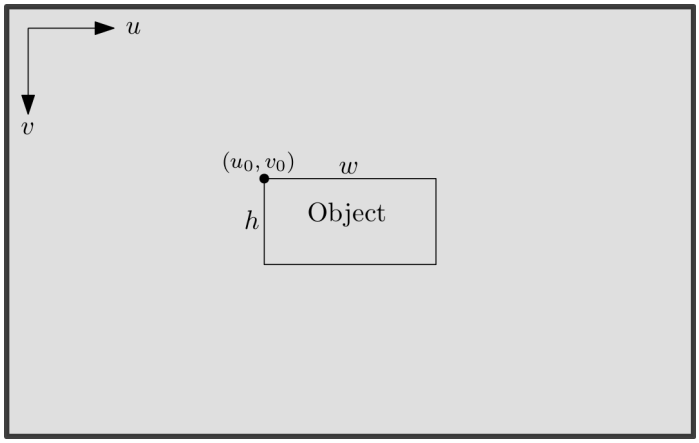


Figure 4.5: 2D Bounding Box on Image Plane

As it was earlier mentioned in Section 1.3, it has been attempted in the literature to extract the 2D bounding box information for pose estimation. Experimental trials have proved the inefficiency and impracticality of 2D bounding box for depth and pose estimation. On the other hand, a 3D bounding box provides us with the necessary information for estimating the pose with PnP method with eight corresponding vertices of the box.

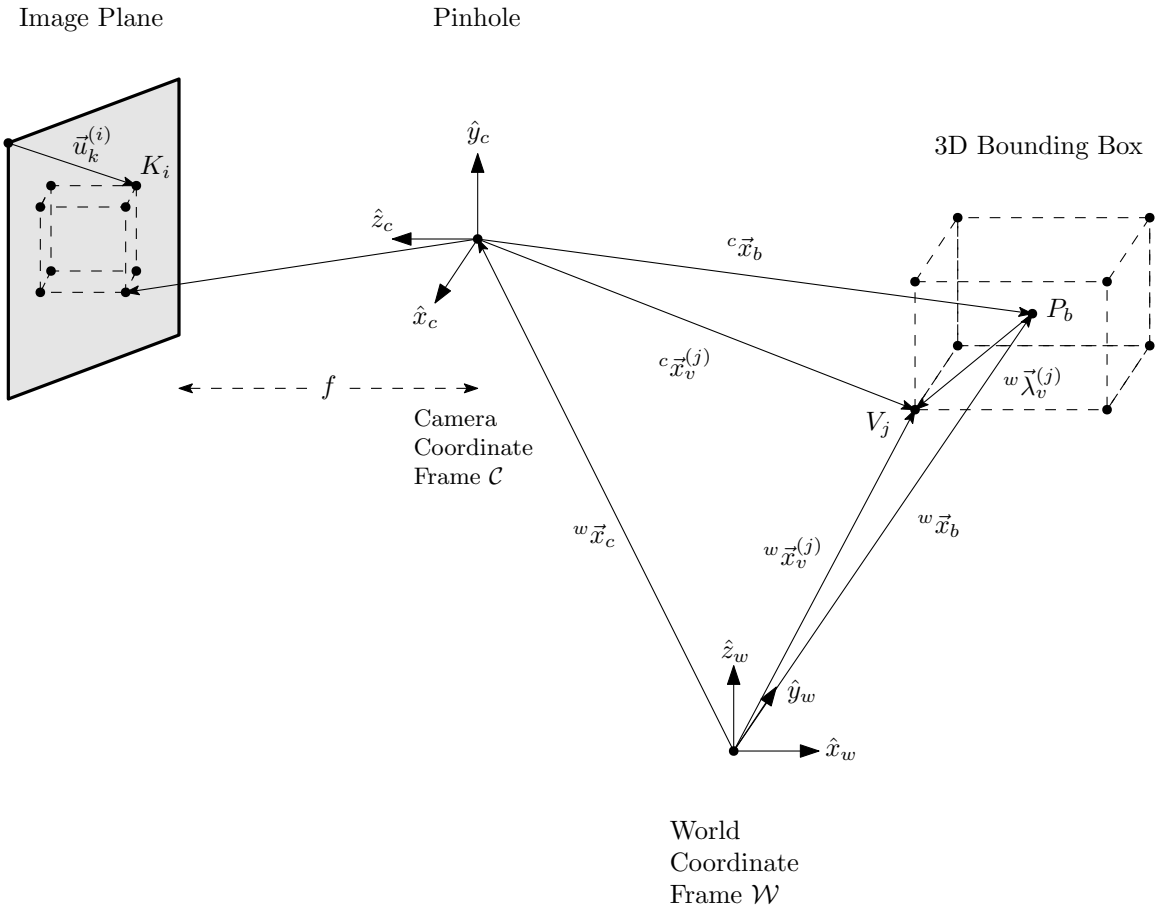


Figure 4.6: 3D Bounding Box Projection Schematics

Figure 4.6 represents the schematics of the spatial 3D bounding box and projected on the image plane in details. The box has eight vertices ($V_0 \dots V_7$) along with the Bebop centre of mass denoted as P_b. The Cartesian coordinates of P_b could be expressed in two ways with respect to world coordinate frame $\mathcal{W}$ as ${}^w\vec{x}_b$ or with respect to camera coordinate frame $\mathcal{C}$ as ${}^c\vec{x}_b$. Based on the set definitions we have:

$$^c\vec{x}_b = {}^w_c R \left[{}^w\vec{x}_b - {}^w\vec{x}_c \right] \tag{4.21}$$

Where ${}^w_c R$ is the rotation transform matrix from $\mathcal{W}$ to $\mathcal{C}$.

4.2.1 Vertices

Based on the equations obtained in the previous section, we have obtained the formulation for obtaining the position vector of P_b with respect to world and camera frames. However, for obtaining the keypoints, we need to calculate the Cartesian coordinates of vertices with respect to the camera. To do so, first we need to obtain the vertices coordinates with respect to world frame defined as follows:

$$
{}^{w}\mathbf{x}_v = \begin{bmatrix} {}^{w}x_v^{(0)} & {}^{w}x_v^{(1)} & {}^{w}x_v^{(2)} & {}^{w}x_v^{(3)} & {}^{w}x_v^{(4)} & {}^{w}x_v^{(5)} & {}^{w}x_v^{(6)} & {}^{w}x_v^{(7)} \\ {}^{w}y_v^{(0)} & {}^{w}y_v^{(1)} & {}^{w}y_v^{(2)} & {}^{w}y_v^{(3)} & {}^{w}y_v^{(4)} & {}^{w}y_v^{(5)} & {}^{w}y_v^{(6)} & {}^{w}y_v^{(7)} \\ {}^{w}z_v^{(0)} & {}^{w}z_v^{(1)} & {}^{w}z_v^{(2)} & {}^{w}z_v^{(3)} & {}^{w}z_v^{(4)} & {}^{w}z_v^{(5)} & {}^{w}z_v^{(6)} & {}^{w}z_v^{(7)} \end{bmatrix} \tag{4.22}
$$

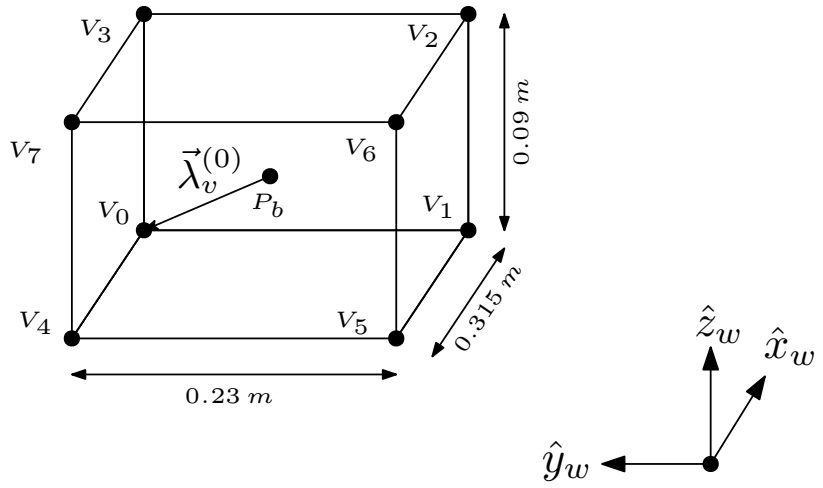


Figure 4.7: 3D Bounding Box Dimensions

Since the dimension of the box is known, we could augment the centroid coordinates P_b to eight vertices using auxiliary vectors ${}^{w}\vec{\lambda}_v^{(j)}$ defined as follows:

$$
{}^{w}\Lambda_v = \begin{bmatrix} {}^{w}\vec{\lambda}_v^{(0)} & {}^{w}\vec{\lambda}_v^{(1)} & {}^{w}\vec{\lambda}_v^{(2)} & {}^{w}\vec{\lambda}_v^{(3)} & {}^{w}\vec{\lambda}_v^{(4)} & {}^{w}\vec{\lambda}_v^{(5)} & {}^{w}\vec{\lambda}_v^{(6)} & {}^{w}\vec{\lambda}_v^{(7)} \end{bmatrix} \tag{4.23}
$$

Based on the experimental measurements of the Bebop drone as shown in Figure 4.7, the circumscribing box has a dimension of 0.23 by 0.315 by 0.09 meters. Considering the centroid at $(0.125, 0.1575, -0.04)$, the custom auxiliary matrix ${}^{w}\Lambda_{v}$ is as follows:

$$
{}^{w}\Lambda_{v} = \begin{bmatrix}
0.1250 & 0.1250 & 0.1250 & 0.1250 & -0.1050 & -0.1050 & -0.1050 & -0.1050 \\
0.1575 & -0.1575 & 0.1575 & -0.1575 & 0.1575 & -0.1575 & 0.1575 & -0.1575 \\
-0.04 & -0.04 & 0.05 & 0.05 & -0.04 & -0.04 & 0.05 & 0.05
\end{bmatrix} \tag{4.24}
$$

Subsequently, for obtaining the ${}^{w}\mathtt{x}_{v}$ we can augment ${}^{w}\vec{x}_{b}$ and add it to the ${}^{w}\Lambda_{v}$ as follows:

$$
{}^{w}\mathtt{x}_{v} = {}^{w}\vec{x}_{b}\, M_{a} + {}^{w}R_{b}\, {}^{w}\Lambda_{v} \tag{4.25}
$$

Where ${}^{w}_{c}R_{b}$ is the rotation matrix of Bebop drone with respect to world frame. Applying this matrix to auxiliary matrix would take the drone rotation into consideration for calculating the vertices. Consequently we have:

$$
{}^{w}\mathtt{x}_{v} = \begin{bmatrix} {}^{w}x_{b} \\ {}^{w}y_{b} \\ {}^{w}z_{b} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} + {}^{w}R_{b} \begin{bmatrix} {}^{w}\vec{\lambda}_{v}^{(0)} & {}^{w}\vec{\lambda}_{v}^{(1)} & {}^{w}\vec{\lambda}_{v}^{(2)} & {}^{w}\vec{\lambda}_{v}^{(3)} & {}^{w}\vec{\lambda}_{v}^{(4)} & {}^{w}\vec{\lambda}_{v}^{(5)} & {}^{w}\vec{\lambda}_{v}^{(6)} & {}^{w}\vec{\lambda}_{v}^{(7)} \end{bmatrix} \tag{4.26}
$$

$$= \begin{bmatrix} {}^{w}x_b & {}^{w}x_b & {}^{w}x_b & {}^{w}x_b & {}^{w}x_b & {}^{w}x_b & {}^{w}x_b & {}^{w}x_b \\ {}^{w}y_b & {}^{w}y_b & {}^{w}y_b & {}^{w}y_b & {}^{w}y_b & {}^{w}y_b & {}^{w}y_b & {}^{w}y_b \\ {}^{w}z_b & {}^{w}z_b & {}^{w}z_b & {}^{w}z_b & {}^{w}z_b & {}^{w}z_b & {}^{w}z_b & {}^{w}z_b \end{bmatrix}$$

$$+ {}^{w}R_b \begin{bmatrix} 0.1250 & 0.1250 & 0.1250 & 0.1250 & -0.1050 & -0.1050 & -0.1050 & -0.1050 \\ 0.1575 & -0.1575 & 0.1575 & -0.1575 & 0.1575 & -0.1575 & 0.1575 & -0.1575 \\ -0.04 & -0.04 & 0.05 & 0.05 & -0.04 & -0.04 & 0.05 & 0.05 \end{bmatrix}$$

$$= \begin{bmatrix} {}^{w}x_v^{(0)} & {}^{w}x_v^{(1)} & {}^{w}x_v^{(2)} & {}^{w}x_v^{(3)} & {}^{w}x_v^{(4)} & {}^{w}x_v^{(5)} & {}^{w}x_v^{(6)} & {}^{w}x_v^{(7)} \\ {}^{w}y_v^{(0)} & {}^{w}y_v^{(1)} & {}^{w}y_v^{(2)} & {}^{w}y_v^{(3)} & {}^{w}y_v^{(4)} & {}^{w}y_v^{(5)} & {}^{w}y_v^{(6)} & {}^{w}y_v^{(7)} \\ {}^{w}z_v^{(0)} & {}^{w}z_v^{(1)} & {}^{w}z_v^{(2)} & {}^{w}z_v^{(3)} & {}^{w}z_v^{(4)} & {}^{w}z_v^{(5)} & {}^{w}z_v^{(6)} & {}^{w}z_v^{(7)} \end{bmatrix} \quad (4.27)$$

For projection calculation which will be discussed in following sections, it is required to transform into ${}^{w}\mathbf{x}_v$ homogeneously into ${}^{w}\tilde{\mathbf{x}}_v$ as follows:

$${}^{w}\tilde{\mathbf{x}}_v = \begin{bmatrix} {}^{w}x_v^{(0)} & {}^{w}x_v^{(1)} & {}^{w}x_v^{(2)} & {}^{w}x_v^{(3)} & {}^{w}x_v^{(4)} & {}^{w}x_v^{(5)} & {}^{w}x_v^{(6)} & {}^{w}x_v^{(7)} \\ {}^{w}y_v^{(0)} & {}^{w}y_v^{(1)} & {}^{w}y_v^{(2)} & {}^{w}y_v^{(3)} & {}^{w}y_v^{(4)} & {}^{w}y_v^{(5)} & {}^{w}y_v^{(6)} & {}^{w}y_v^{(7)} \\ {}^{w}z_v^{(0)} & {}^{w}z_v^{(1)} & {}^{w}z_v^{(2)} & {}^{w}z_v^{(3)} & {}^{w}z_v^{(4)} & {}^{w}z_v^{(5)} & {}^{w}z_v^{(6)} & {}^{w}z_v^{(7)} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (4.28)$$

The next objective is calculating ${}^{c}\mathbf{x}_v$. By taking another look into Figure 4.6, it can be inferred that:

$${}^{c}\vec{x}_v^{(j)} = {}^{w}_{c}R\left[{}^{w}\vec{x}_v^{(j)} - {}^{w}\vec{x}_c\right] = {}^{w}_{c}R\,{}^{w}\vec{x}_v^{(j)} - {}^{w}_{c}R\,{}^{w}\vec{x}_c \quad (4.29)$$

Equation 4.29 would lead us into calculating extrinsic matrix discussed in Section 4.1. Let's take:

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = -{}^{w}_{c}R \begin{bmatrix} {}^{w}x_c \\ {}^{w}y_c \\ {}^{w}z_c \end{bmatrix} \quad (4.30)$$

And:

$$ {}^{w}_{c}R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.31} $$

Subsequently:

$$ M_{ext} = \begin{bmatrix} {}^{w}_{c}R & -{}^{w}_{c}R\,{}^{w}\vec{x}_{c} \\ 0 & 1 \end{bmatrix} \tag{4.32} $$

$$ = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.33} $$

At the same time we have:

$$ {}^{c}\tilde{\mathtt{x}}_{v} = M_{ext}\,{}^{w}\tilde{\mathtt{x}}_{v} \tag{4.34} $$

Therefore:

$$
{}^{c}\tilde{\mathtt{x}}_v = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} {}^{w}x_v^{(0)} & {}^{w}x_v^{(1)} & {}^{w}x_v^{(2)} & {}^{w}x_v^{(3)} & {}^{w}x_v^{(4)} & {}^{w}x_v^{(5)} & {}^{w}x_v^{(6)} & {}^{w}x_v^{(7)} \\ {}^{w}y_v^{(0)} & {}^{w}y_v^{(1)} & {}^{w}y_v^{(2)} & {}^{w}y_v^{(3)} & {}^{w}y_v^{(4)} & {}^{w}y_v^{(5)} & {}^{w}y_v^{(6)} & {}^{w}y_v^{(7)} \\ {}^{w}z_v^{(0)} & {}^{w}z_v^{(1)} & {}^{w}z_v^{(2)} & {}^{w}z_v^{(3)} & {}^{w}z_v^{(4)} & {}^{w}z_v^{(5)} & {}^{w}z_v^{(6)} & {}^{w}z_v^{(7)} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}
$$

$$
= \begin{bmatrix} {}^{c}x_v^{(0)} & {}^{c}x_v^{(1)} & {}^{c}x_v^{(2)} & {}^{c}x_v^{(3)} & {}^{c}x_v^{(4)} & {}^{c}x_v^{(5)} & {}^{c}x_v^{(6)} & {}^{c}x_v^{(7)} \\ {}^{c}y_v^{(0)} & {}^{c}y_v^{(1)} & {}^{c}y_v^{(2)} & {}^{c}y_v^{(3)} & {}^{c}y_v^{(4)} & {}^{c}y_v^{(5)} & {}^{c}y_v^{(6)} & {}^{c}y_v^{(7)} \\ {}^{c}z_v^{(0)} & {}^{c}z_v^{(1)} & {}^{c}z_v^{(2)} & {}^{c}z_v^{(3)} & {}^{c}z_v^{(4)} & {}^{c}z_v^{(5)} & {}^{c}z_v^{(6)} & {}^{c}z_v^{(7)} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{4.35}
$$

4.2.2 Keypoints

Keypoints are in fact the projected vertices on image plane. For a 3D bounding box with eight vertices, there would be eight corresponding keypoints respectively. Figure 4.8 illustrates the projected 3D bounding box along with the eight keypoints on the image plane. Each keypoint could be represented with a 2D vector $\vec{u}_k^{(i)}$ on the image plane consisting of two parameters of u_i and v_i.

The following matrix entails the keypoint coordinates as a matrix consisting of eight vectors.

$$
\mathtt{u}_k = \begin{bmatrix} u_0 & u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 \\ v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{bmatrix} \tag{4.36}
$$

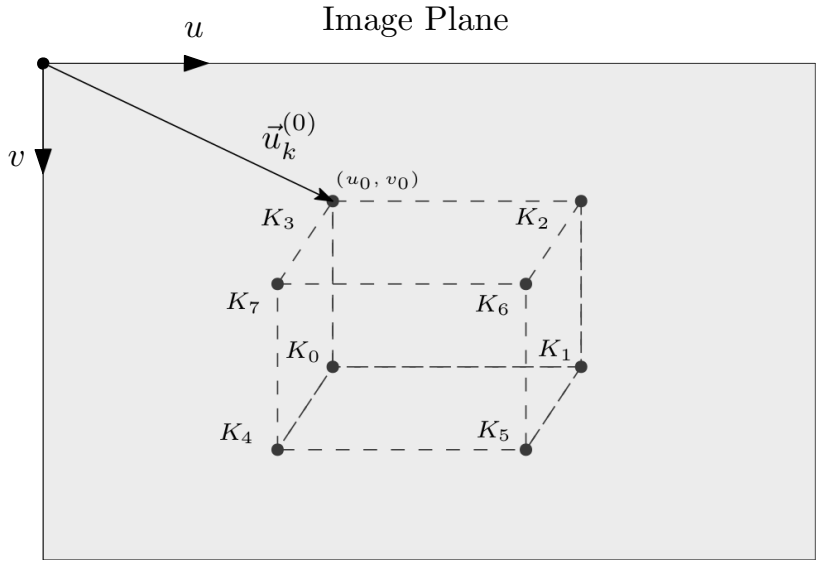


Figure 4.8: Projected 3D Bounding Box

However, as stated before, while projecting the vertices into keypoints, a third axis appears into the vectors which then needs to be normalized. The un-normalized keypoint matrix could be represented as follows:

$$\tilde{\mathtt{u}}_k = \begin{bmatrix} \tilde{u}_0 & \tilde{u}_1 & \tilde{u}_2 & \tilde{u}_3 & \tilde{u}_4 & \tilde{u}_5 & \tilde{u}_6 & \tilde{u}_7 \\ \tilde{v}_0 & \tilde{v}_1 & \tilde{v}_2 & \tilde{v}_3 & \tilde{v}_4 & \tilde{v}_5 & \tilde{v}_6 & \tilde{v}_7 \\ \tilde{w}_0 & \tilde{w}_1 & \tilde{w}_2 & \tilde{w}_3 & \tilde{w}_4 & \tilde{w}_5 & \tilde{w}_6 & \tilde{w}_7 \end{bmatrix} \tag{4.37}$$

Where:

$$\tilde{\mathtt{u}}_k = M_{int}\, {}^c\tilde{\mathtt{x}}_v \tag{4.38}$$

$$= \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} {}^cx_v^{(0)} & {}^cx_v^{(1)} & {}^cx_v^{(2)} & {}^cx_v^{(3)} & {}^cx_v^{(4)} & {}^cx_v^{(5)} & {}^cx_v^{(6)} & {}^cx_v^{(7)} \\ {}^cy_v^{(0)} & {}^cy_v^{(1)} & {}^cy_v^{(2)} & {}^cy_v^{(3)} & {}^cy_v^{(4)} & {}^cy_v^{(5)} & {}^cy_v^{(6)} & {}^cy_v^{(7)} \\ {}^cz_v^{(0)} & {}^cz_v^{(1)} & {}^cz_v^{(2)} & {}^cz_v^{(3)} & {}^cz_v^{(4)} & {}^cz_v^{(5)} & {}^cz_v^{(6)} & {}^cz_v^{(7)} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{4.39}$$

4.2.3 Projection

In this section, we going to review the whole process of creating the box in 3D manner and then projecting it and normalizing the keypoints on the image plane.

$$ {}^{w}\vec{x}_b \xrightarrow{{}^{w}\Lambda_v} {}^{w}\mathtt{x}_v \rightarrow {}^{w}\tilde{\mathtt{x}}_v \xrightarrow{M_{ext}} {}^{c}\tilde{\mathtt{x}}_v \xrightarrow{M_{int}} \tilde{\mathtt{u}}_k \xrightarrow{\text{Norm}} \mathtt{u}_k \tag{4.40} $$

In brief, first the spatial coordinates ${}^{w}\vec{x}_b$ of Bebop drone with respect to world frame is obtained via Vicon. Then the derived vector is augmented into a matrix consisting of eight position vectors of eight vertices. The vertices matrix is then augmented as well into ${}^{w}\tilde{\mathtt{x}}_v$. Thus, using extrinsic matrix,${}^{w}\tilde{\mathtt{x}}_v$ is transformed into ${}^{c}\tilde{\mathtt{x}}_v$ which entails the vertices coordinates with respect to camera frame. Consequently, applying intrinsic matrix M_{int} would map ${}^{c}\tilde{\mathtt{x}}_v$ into image plane coordinates represented as $\tilde{\mathtt{u}}_k$. Eventually, $\tilde{\mathtt{u}}_k$ is normalized over the last element to substitute $\tilde{w}_i$ with one and eliminate the last row for transforming $\tilde{\mathtt{u}}_k$ into a 2D matrix of $\mathtt{u}_k$.

4.3 Keypoint RCNN

This section entails the procedure of utilizing deep learning for detecting target and corresponding keypoints. As it was discussed earlier in Chapter 1, recent advances in computer vision and hardware engineering have enabled the design and utilization of deep convolutional networks on commercial processors and computers.

The aim of this section is to train and utilize a proper neural network to first detect the target drone on a 2D image and then extract the keypoints or bounding box vertices out of the detected target. By doing so, we guarantee the provision of necessary data for pose estimation by PnP algorithm in the next section.

4.3.1 Dataset

Before getting into the deep learning architecture and design, it is necessary to understand the structure of the data and what we feed as the input and what we expect

as the output from the network.

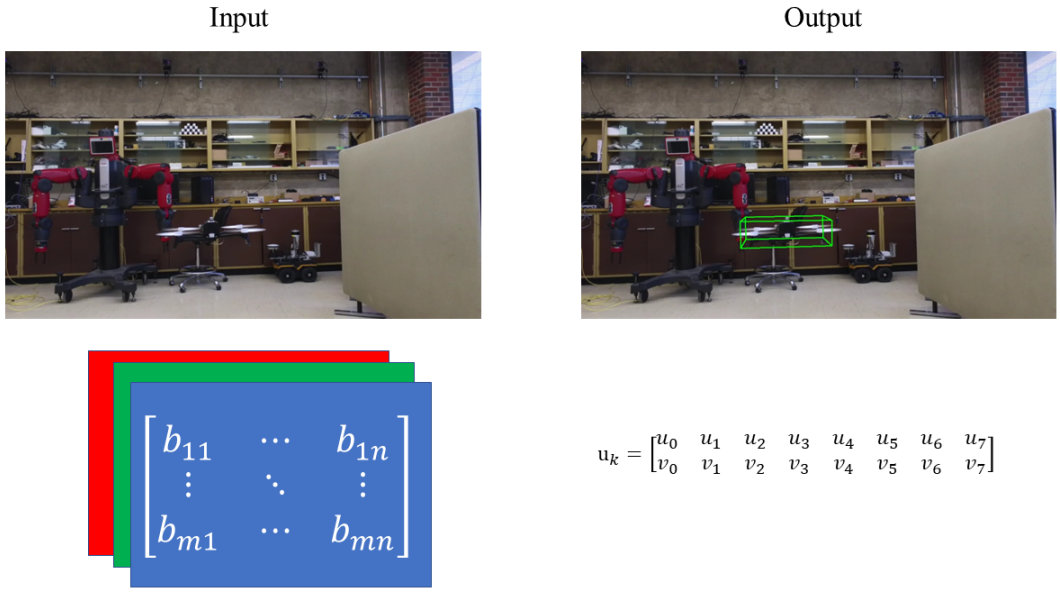


Figure 4.9: Dataset structure

Figure 4.9 represents the input and output data structure. The input is a 3-channel RGB color image which is in fact a 720x1280x3 matrix and the output is a 2x8 matrix representing eight keypoints. Therefore, it is aimed to design a deep learning architecture to process the input image and map the input matrix into the proper output keypoint matrix.

Deep learning networks could be trained via supervised learning algorithms which are based on feeding the right input and output data and applying gradient descent on the network. Therefore, for the next section, we are going to discuss the data acquisition and annotation algorithms to generate the proper training and validation datasets.

4.3.1.1 Acquisition

It is aimed to obtain 20,000 frames with their corresponding keypoints of each frame. However, this huge amount of number of frames require an automatic procedure for capturing and annotating the frames on a csv file for later training and validation.

For this undertaking, we shall use ROS as previous chapter to set up a data acquisition platform for storing images and required data.

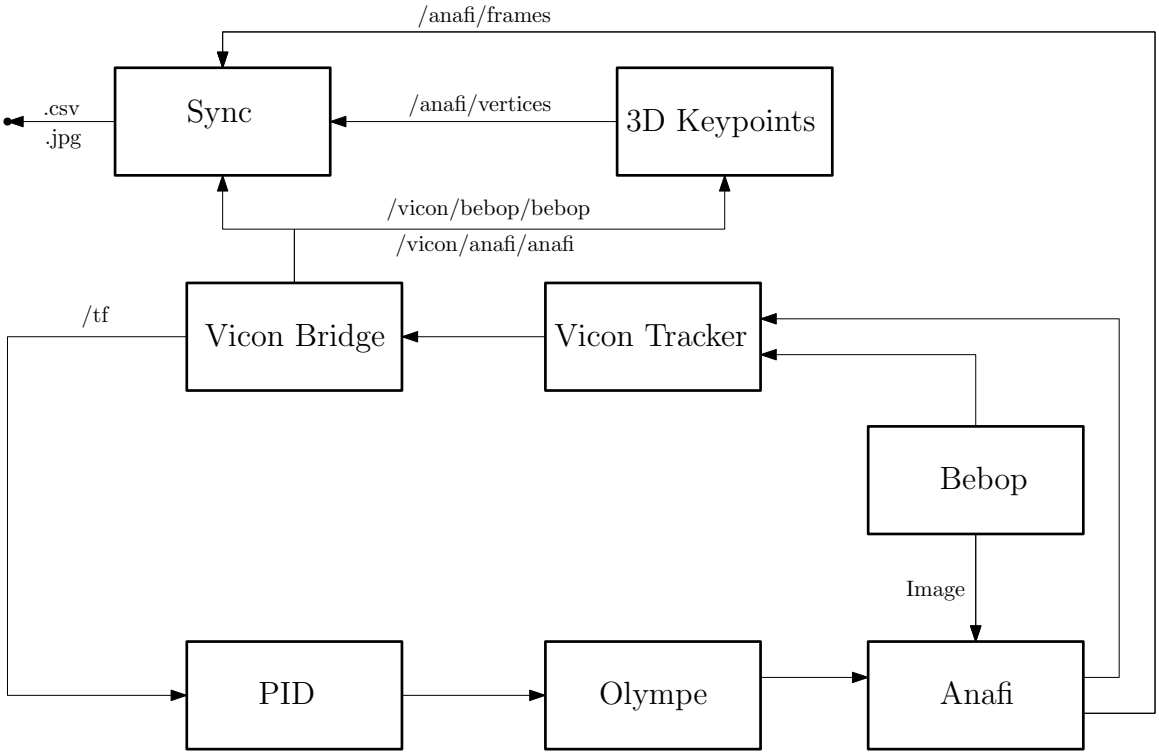


Figure 4.10: Data Acquisition Framework on ROS

Data acquisition framework has been represented in Figure 4.10. It resembles the frameworks discussed in Chapter 3. It all starts from Anafi where takes the image of Bebop and transmits it through wi-fi and ROS. Simultaneously, the pose and coordinates of Anafi and Bebop are recorded and transmitted ro ROS via `vicon_tracker` and Vicon Tracker. The transmitted poses are published into 3D Keypoints node which then publishes the vertices, ${}^{c}\tilde{\mathbf{x}}_v$ discussed in Section 4.2.1, as /anafi/vertices topic. Then published vertices are then received by Sync node. This node subscribes

to three topics: /anafi/frames, /anafi/vertices and /tf. By doing so, it is able to synchronize the images and corresponding vertices and pose information and save them as jpg and csv files. Sync node saves the following information on csv file with order:

1. Frame ID
2. Time
3. Relative Pose $(x_{rel}, y_{rel}, z_{rel})$
4. Vertices $(u_0, v_0, ..., u_7, v_7)$
5. Anafi Pose $(x_a, y_a, z_a, \phi_a, \theta_a, \psi_a)$
6. Bebop Pose $(x_b, y_b, z_b, \phi_b, \theta_b, \psi_b)$

As the images are saved separately as .jpg files, it is necessary to record the ID of each frame to retrieve its corresponding pose and vertices data accordingly. Secondly, time is recorded which indicates the relative time of each frame with respect to the simulation commencement. Relative pose $(x_{rel}, y_{rel}, z_{rel})$ is in fact the relative pose of Bebop to the Anafi drone with respect to the world frame. This data should be later transformed by rotation matrices to camera frame.

Most importantly, the eight vertices and sixteen respective coordinates are recorded. This data will be later used as training and validation data for the deep neural networks. Anafi and Bebop pose data recorded as well for later analysis of the simulation and pose estimation performance.

4.3.1.2 Offset Removal

In practice, capturing frames and bounding box vertices via Vicon would cause an offset between the box and the drone as it is shown in Figure 4.11.

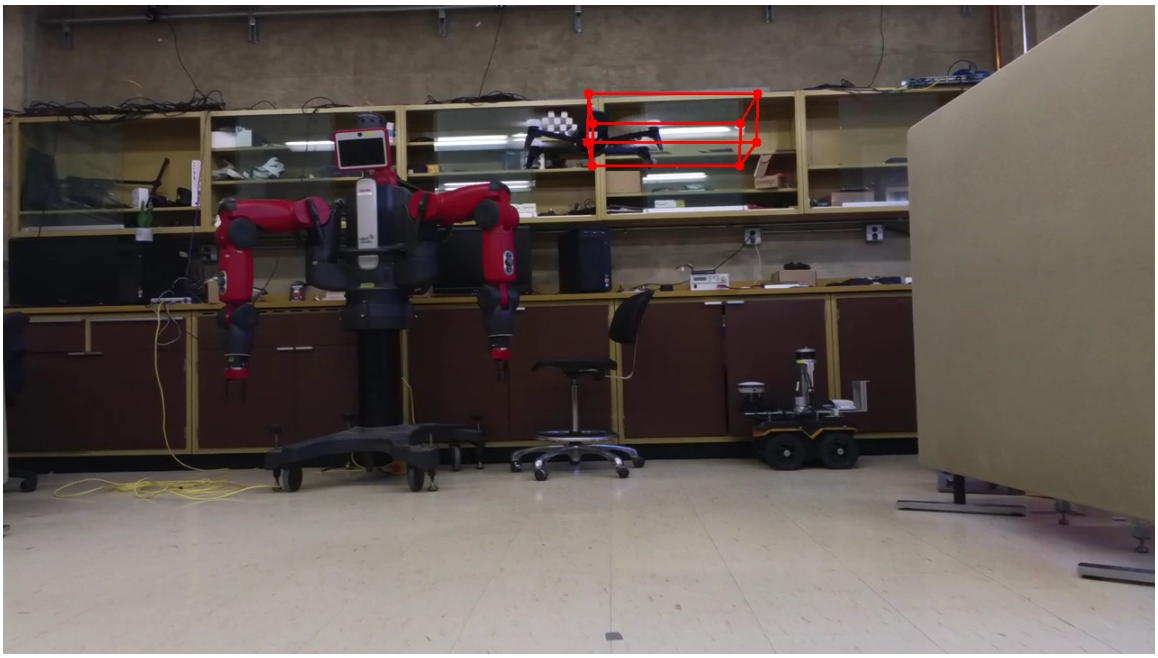


Figure 4.11: Bounding Box Offset

Recalibrating the Vicon and Camera Matrix wouldn't make any improvement. Therefore it is needed to remove the offset through another methods. The solution for removing the offset lies in 2D bounding box detection. For removing the offset, we need to have a reference for the actual position of the drone on the image plane. Therefore, by detecting the drone and its corresponding 2D bounding box, a support vector could be sketched for moving the 3D bounding to the intended position.

Figure 4.12 illustrates the schematics of offset removal using 2D bounding box. Let's define the 2D bounding box centroid as C_2 where:

$$C_2 = \begin{bmatrix} u_c^{(2)} \\ v_c^{(2)} \end{bmatrix} \tag{4.41}$$

And the 3D bounding box centroid as C_3 where:

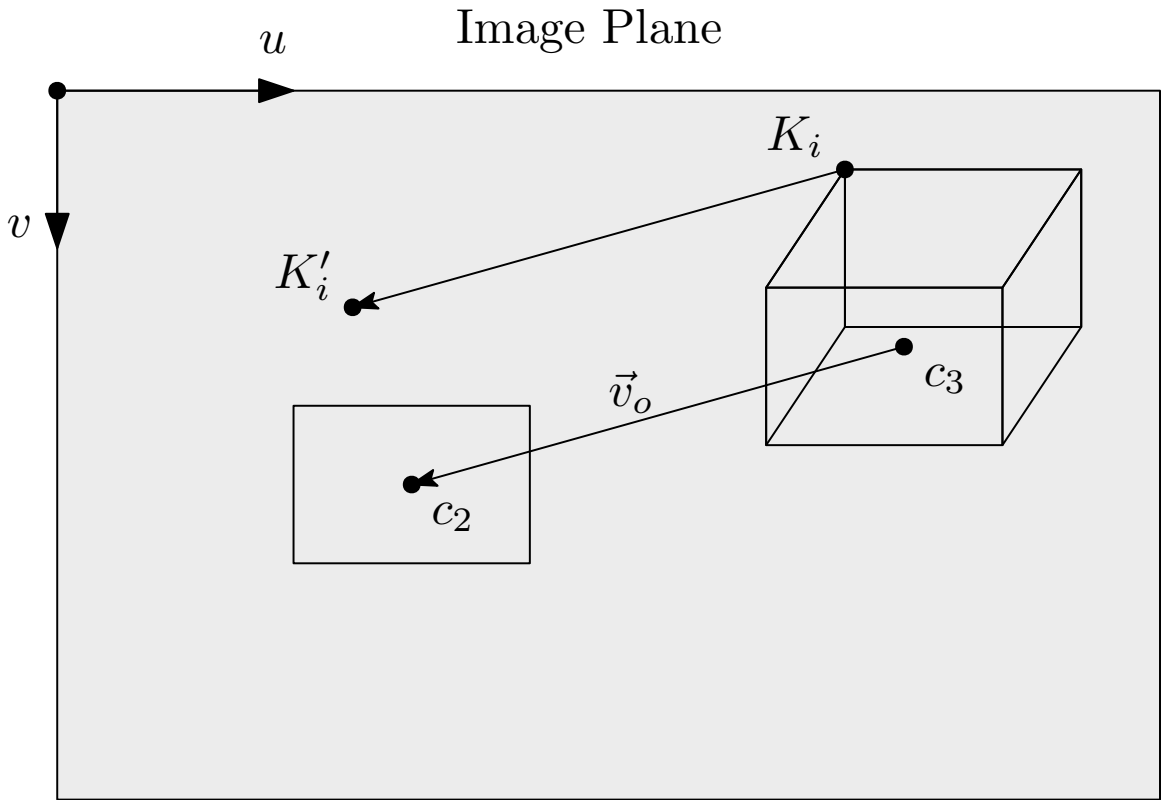


Figure 4.12: Offset Removal Schematics

$$C_3 = \begin{bmatrix} u_c^{(3)} \\ v_c^{(3)} \end{bmatrix} \tag{4.42}$$

For calculating C_3 we have:

$$C_3 = \frac{\sum_{i=0}^{8} K_i}{8} = \frac{1}{8} \begin{bmatrix} \sum_{i=0}^{8} u_k^{(i)} \\ \sum_{i=0}^{8} v_k^{(i)} \end{bmatrix} \tag{4.43}$$

Therefore:

$$\vec{v}_o = C_2 - C_3 \tag{4.44}$$

Thus, by having the $\vec{v}_o$ we can obtain the rectified keypoints by adding the offset vector to the original keypoints as follows:

$$K_i' = K_i + \vec{v}_o \tag{4.45}$$

Now that the offset removal procedure has be laid out, the only remaining challenge is to detect the 2D bounding box circumscribing the drone. For doing so, we shall acquire Detectron2 deep learning package for detecting the 2D bounding box. Before training the network, it is needed to obtain a proper dataset of images and corresponding 2D bounding box information. Using Labelme annotation package [73], 300 frames have been labeled and annotated for Detectron2.

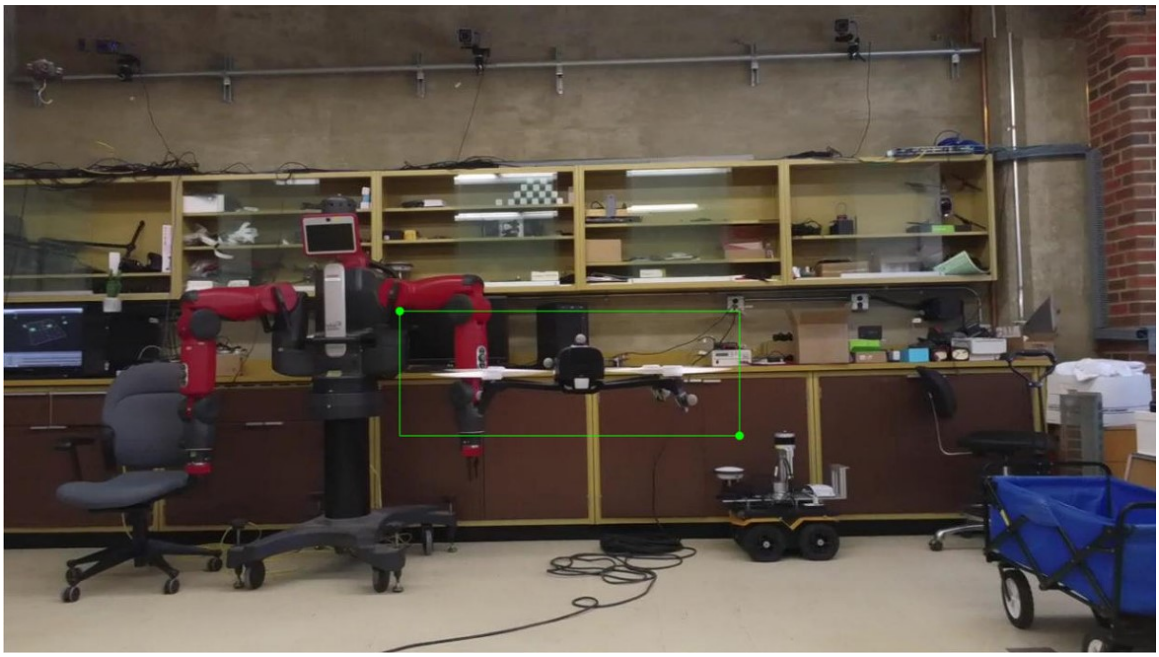


Figure 4.13: 2D Bounding Box Manual Annotation

Using manually labeled dataset, Detectron2 RCNN network is trained for detecting Bebop drone. The robustness of RCNN architecture is in that the network detects the drone without the need of any curtains or any kind of manipulator.

Figure 4.14 represents the rectification process of bounding box. The red box is the original uncalibrated box, whereas the green box is the rectified calibrated box which perfectly fits the drone. This process is performed on all dataset frames in order to adjust the whole dataset for later training purposes.

Figure 4.14: Rectified Bounding Box in Green and Original One in Red

4.3.1.3 Annotation

The 3D bounding box information has been recorded as a .csv file. However, PyTorch training engines required the annotation data or the box information in our case, to be recorded as a .json file with the following format:

Listing 4.1: Annotation Protocol as JSON File

```
{
  "bboxes": [[628, 42, 825, 129]],
  "keypoints": [[[ [623, 135, 1],
        [808, 136, 1],
        [808, 82, 1],
        [623, 82, 1],
        [630, 93, 1],
        [845, 94, 1],
        [845, 31, 1],
        [630, 31, 1] ]]]
}
```

The .json file contains the information for the 2D bounding box as well as the keypoints or 3D bounding box vertices information according to COCO format. COCO or common objects in context is a large dataset which contains raw and annotated data for mask or keypoint detection. COCO's protocols for annotation have become the standard model for annotating datasets [74]. According to COCO each keypoint

should be annotated as $[x_i, y_i, v_i]$ where x_i and y_i are the image plane coordinates of the keypoint and v_i is a binary value which represents the visibility of the keypoint.

4.3.2 Networks

As it was earlier discussed in Section 1.3.1, a convolutional neural network is a potent and powerful tool for processing image frames to detect objects. However, detecting a target drone is one of the essential requirements needed. Most needed, we are looking for detecting the target and extracting the keypoints (or bounding box vertices) by further processing.

Before making efforts into extracting keypoints, researchers were aimed into obtaining semantic segmentation of the target classes using fully convolutional U-net architecture [75]. However, this network was only able to segment the image based on classes rather than instances. To enhance U-net even further, region-based convolutional networks (RCNN) were introduced which were based on integrating FCNs with region proposal networks (RPN) to better detect the targets by proposing target regions and optimizing them on classes. RCNNs were then later enhanced into Fast-RCNN and Mask-RCNN networks capable of instance segmentation of class members instead of generalized semantic segmentation. Subsequently, Mask-RCNN paved the way for target class keypoint extraction which has numerous and invaluable practicalities for pose estimation in robotic applications.

4.3.2.1 Architecture

The current model utilized for this project is `KEYPOINT_RCNN_RESENT50_FPN` developed by Torch [76]. In fact, the Keypoint-RCNN is an augmented version of Mask-RCNN which incorporates RPN with FPN and ResNet to extract the keypoints from the instance region.

Generalized schematics of the network is represented in Figure 4.15. The input raw image is fed into ROI and Deep ConvNet layers for extracting the features and

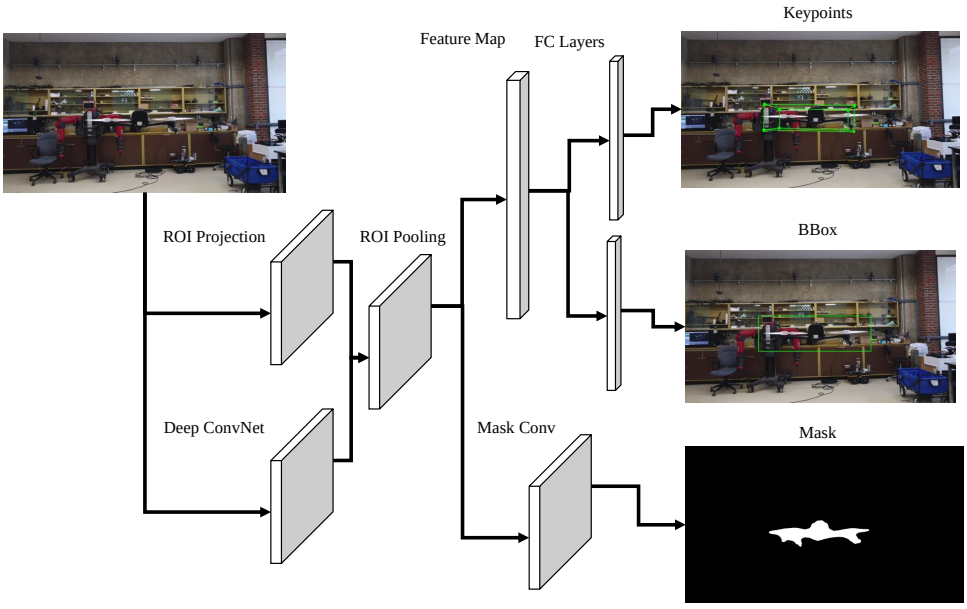


Figure 4.15: Keypoint RCNN Schematic Architecture

target classes. The results are then processed by ROI pooling for fine-tuning the targets. At this point, the output could be either processed by Mask Conv for instance segmentation or by feature map and fully connected regressor layers for keypoint and bounding box extraction. Fundamentally, Keypoint-RCNN is a special case of Mask-RCNN as both anchor the region of interest (RoI) for feature extraction. Torch `KEYPOINT_RCNN_RESENT50_FPN` is an optimized network for keypoint extraction by removing the Mask Conv blocks.

4.3.2.2 Training and Validation

As earlier stated in Section 4.3.1.1, 20,000 frames were collected which were divided into 18,000 (90%) training samples and 2,000 (10%) validation samples. There datasets were then calibrated and annotated accordingly to Torch and COCO protocols. For training, PyTorch vision detection[1] default engine has been utilized for training and validating the dataset. However, Vision engine imports Pycocotool[2]

[1]https://github.com/pytorch/vision/tree/main/references/detection
[2]https://github.com/cocodataset/cocoapi/tree/master/PythonAPI/pycocotools

library and cocoeval.py for evaluating the average accuracy of the model which is developed for 17 keypoints by default and it required to modify the source code and number keypoint OKS validation coefficients to eight. Before understanding OKS, we need to go through an earlier developed validation criterion for bounding box detection called intersection over union or IoU defined as below:

$$\texttt{IoU} = \frac{A_i}{A_u} \tag{4.46}$$

Where A_i is the intersection area of the predicted bounding box and the ground truth and A_u is the union of the two. Thus, IoU is one if the predicting fully overlaps the truth and IoU is zero if there's no overlap or intersection.

Similar to IoU. object keypoint similarity or OKS is a criterion for evaluating the similarity between ground truth keypoints and predicted ones formulated as follows:

$$\texttt{OKS} = \sum_{i=1}^{n} e^{-\frac{d_i^2}{2s^2k_i^2}} \tag{4.47}$$

Where n is the number of keypoints, d_i is the Euclidean distance between ground truth keypoint K_i and predicted one $\hat{K}_i$, s is the scale retrieved by dividing the area of the object bounding box by the total image area and k is the keypoint fall-off constant.

So, the earlier mentioned Pycocotools library incorporates IoU and OKS for obtaining the average accuracy and recall of bounding box and keypoint detection network. Table 4.1 and 4.2 summarize the average accuracy and recall of bounding box and keypoint detection for different IoUs and areas after 20 Epochs. It's common to take IoU of 0.5 as reference and based on this criterion, our method has achieved 99.0% average accuracy for bounding box detection and 97.8% average accuracy for keypoint detection and thus improving the state-of-the-art of drone keypoint detection. It should be stated as well that the current state-of-the-art [18] relies on stationary camera whereas our method has utilized mobile camera on drone for the keypoint

detection appending another novelty to this research and corresponding results.

Figures 4.16a to 4.16f represent the loss trends over 20 epochs for overall, classifier, box regressor, keypoints, objectness and RPN losses. It could be observed that the overall trend of the loss functions is descending. The overall training of the network took 15 hours for 20 epochs.

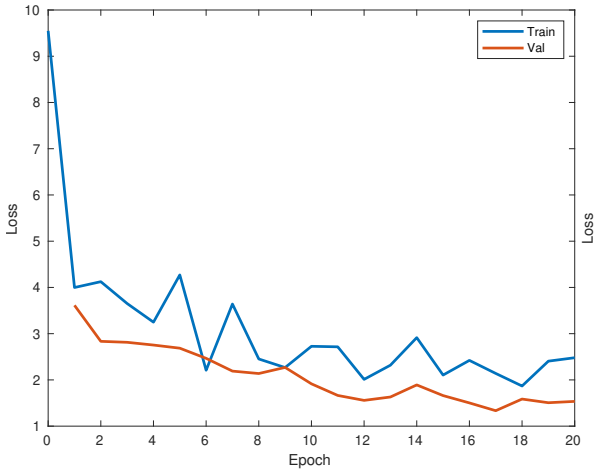
The accuracy and loss results ensures the robustness of keypoint-RCNN in predicting custom keypoints for aerial robots. The stated results validate the overall performance of the network's keypoint detection. However, a more detailed validation on box and individual vertex detection will be discussed in Section 5.2 of Chapter 5.

Table 4.1: Bounding Box Average Precision (AP) and Average Recall (AR)

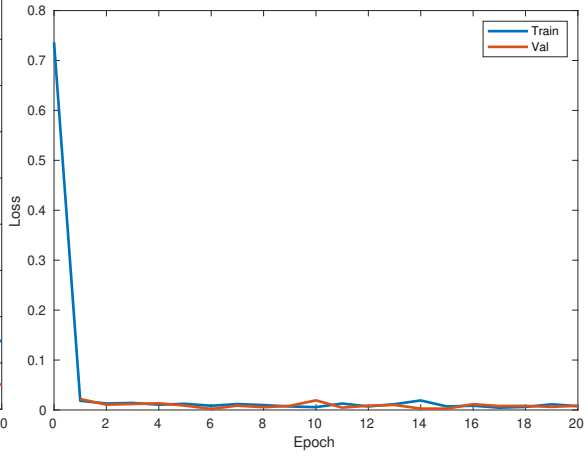
Criterion	IoU	Area	Value
AP	0.50:0.95	All	0.844
AP	0.50	All	0.990
AP	0.75	All	0.978
AP	0.50:0.95	Small	-1.000
AP	0.50:0.95	Medium	0.834
AP	0.50:0.95	Large	0.848
AR	0.50:0.95	All	0.877
AR	0.50	All	0.877
AR	0.75	All	0..
AR	0.50:0.95	Small	-1.000
AR	0.50:0.95	Medium	0.858
AR	0.50:0.95	Large	0.880

Table 4.2: Keypoint Average Precision (AP) and Average Recall (AR)

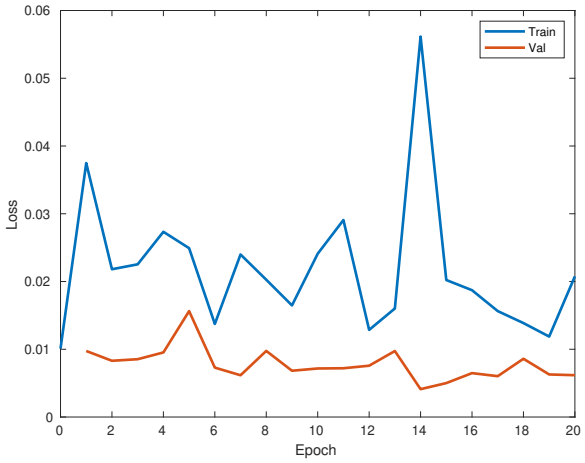
Criterion	IoU	Area	Value
AP	0.50:0.95	All	0.937
AP	0.50	All	0.978
AP	0.75	All	0.962
AP	0.50:0.95	Medium	0.934
AP	0.50:0.95	Large	0.934
AR	0.50:0.95	All	0.954
AR	0.50	All	0.984
AR	0.75	All	0.971
AR	0.50:0.95	Medium	0.961
AR	0.50:0.95	Large	0.953



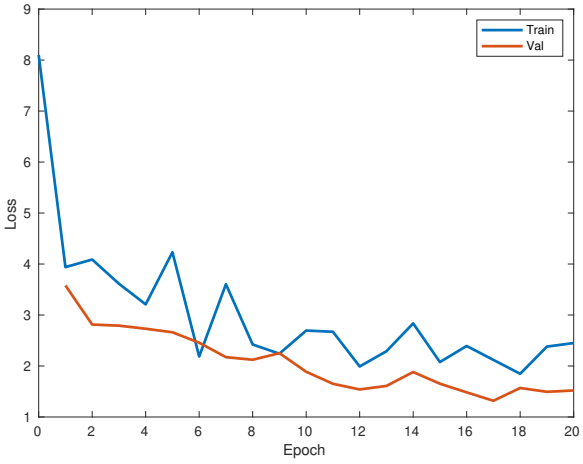
(a) Overall Loss



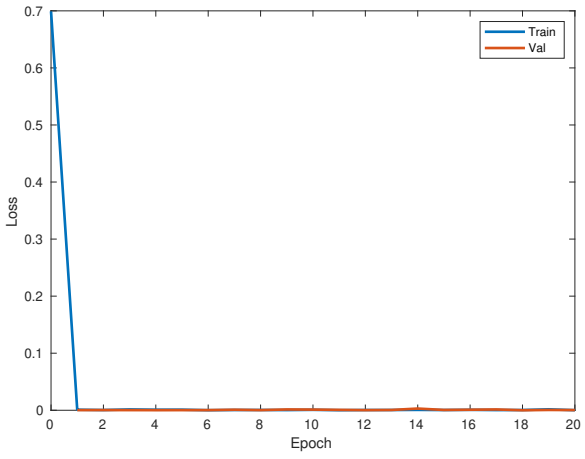
(b) Loss Classifier



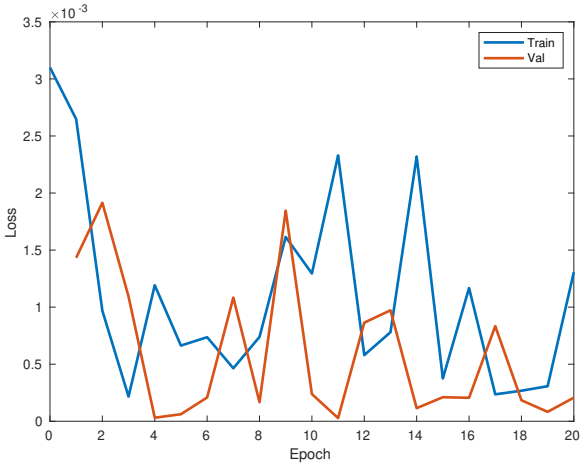
(c) Loss Box Regressor



(d) Loss Keypoints



(e) Loss Objectness



(f) Loss RPN

Figure 4.16: Keypoint RCNN Train and Validation Datasets Loss Over Epochs

4.4 Perspective-n-Point

To understand the Perspective-n-Point algorithm, we need to go through a concept called the computer vision trinity illustrated in Figure 4.17. In computer vision, there are three main sets of parameters including camera matrix, world coordinates and image coordinates. In fact, camera matrix acts as a transform which maps the world coordinate into the image plane as image coordinates.

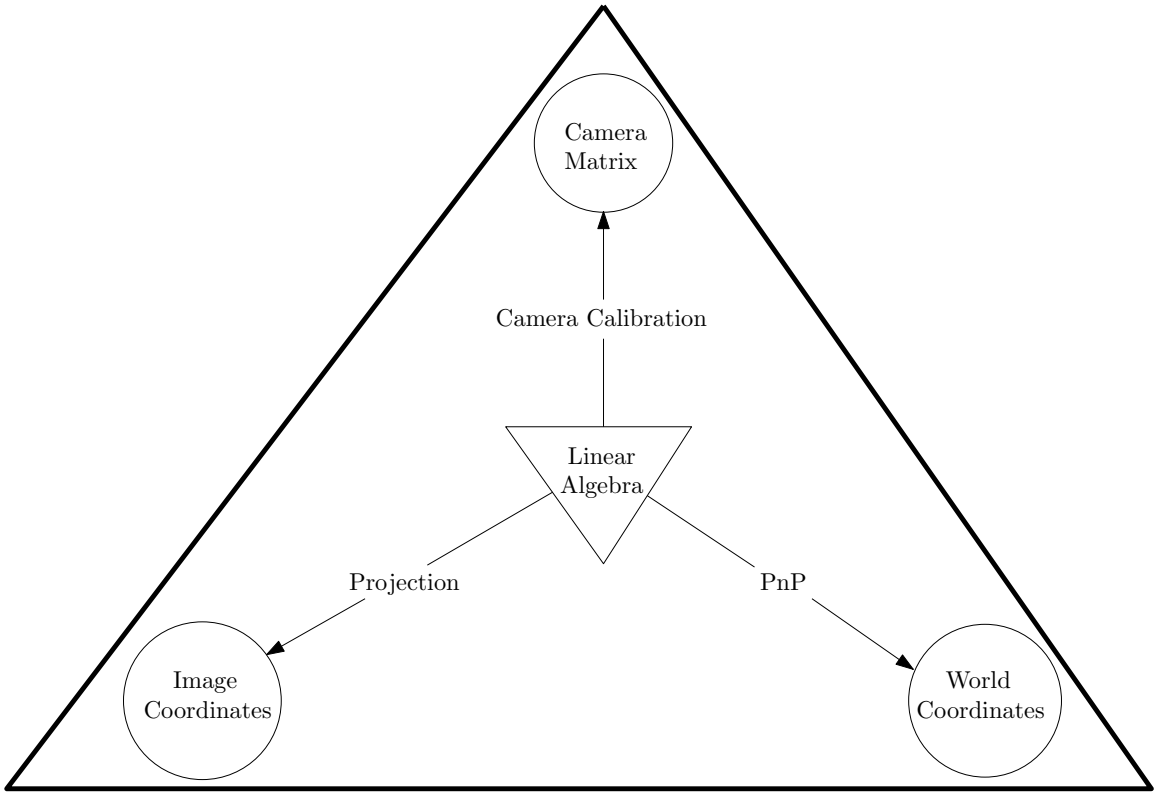


Figure 4.17: Computer Vision Trinity

Although, in most cases two of these parameters are known and the third needs to be identified. If the world and image coordinates are known and camera matrix needs to be identified, camera calibration could be utilized using linear algebra tools. In other case, if the camera matrix and world coordinates are provided, image coordinates could be retrieved using projection formulations. In our case, where the camera matrix is obtained through calibration and image coordinates are detected using R-CNN, the world coordinates are to be calculated through a method called

Perspective-n-Point or commonly known as PnP.

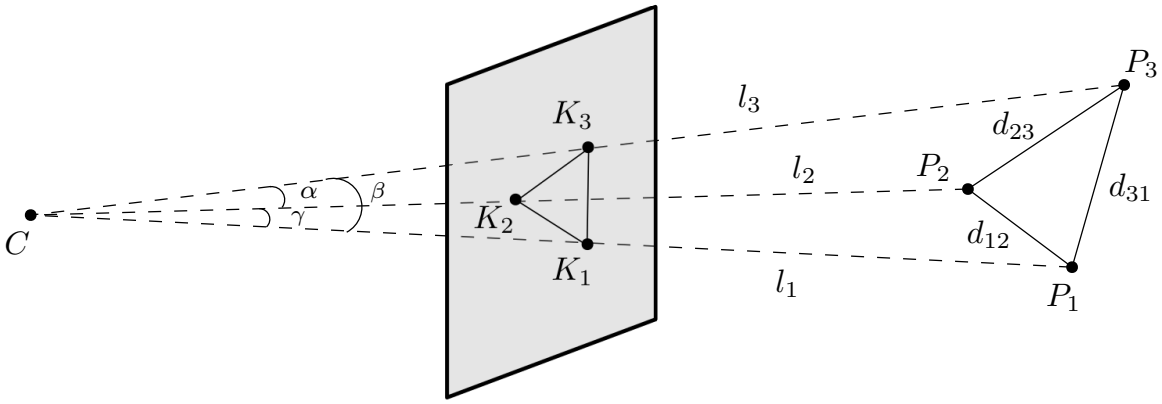


Figure 4.18: Perspective Three Point Projection

Consider Figure 4.18 where it illustrates a three-point object projection on camera plane which is known as P3P problem. The objective of this problem is to retrieve the distance of spatial points relative to camera. The dimension of the box is provided meaning d_{12}, d_{23} and d_{32} are known. T obtain the l parameters, first it is needed to obtain the ray angles as follows [77, 78]:

$$\cos\gamma = \frac{\vec{k}_1 \,.\, \vec{k}_2}{||\vec{k}_1||\,||\vec{k}_2||} \tag{4.48}$$

Where $\vec{k}_i$ is the corresponding vector from camera centre to the keypoint. Similarly:

$$\cos\alpha = \frac{\vec{k}_2 \,.\, \vec{k}_3}{||\vec{k}_2||\,||\vec{k}_3||} \tag{4.49}$$

$$\cos\beta = \frac{\vec{k}_3 \,.\, \vec{k}_1}{||\vec{k}_3||\,||\vec{k}_1||} \tag{4.50}$$

Now, by having the ray angles, we can write [79]:

$$l_1^2 + l_2^2 - 2\,l_1\,l_2\,\cos\gamma = d_{12}^2 \tag{4.51}$$

$$l_2^2 + l_3^2 - 2\,l_2\,l_3\,\cos\alpha = d_{23}^2 \tag{4.52}$$

$$l_3^2 + l_1^2 - 2\,l_3\,l_1\,\cos\beta = d_{31}^2 \tag{4.53}$$

By defining the following substitution:

$$u = \frac{l_2}{l_1}\ ,\ v = \frac{l_3}{l_1} \tag{4.54}$$

We have:

$$l_1^2 = \frac{d_{12}^2}{u^2 + v^2 - 2\,u\,v\,\cos\alpha} \tag{4.55}$$

$$l_1^2 = \frac{d_{23}^2}{1 + v^2 - 2\,v\,\cos\beta} \tag{4.56}$$

$$l_1^2 = \frac{d_{31}^2}{1 + u^2 - 2\,u\,\cos\gamma} \tag{4.57}$$

By solving one equation for u and substituting it into another equation we have the following quartic equations [80]:

$$a_4\,v^4 + a_3\,v^3 + a_2\,v^2 + a_1\,v + a_0 = 0 \tag{4.58}$$

By solving Equation 4.58 for v going through the obtained calculations, we can obtain l_1, l_2 and l_3. However, the problem with P3P problem is that due to the nature of quatric equations, there are four solutions for v and thus four sets of solutions for the (l)s. To resolve this issue, there need to be more than three points which would lead to PnP problem where ($n > 3$) leading to ($n - 2$) quatric equations. In this project, we are dealing with eight points or vertices of the bounding box entailing ten quatric equations. OpenCV library has a variety of functions for solving PnP problem on C++ [81] including EPnP [82] or RANSAC [83]. For this project, the

default `cv::solvePnP()` has been utilized on ROS for obtaining relative target pose based on subscribed /anafi/keypoints topic.

4.4.1 Filtering

PnP output tends to have noises and spikes quite often which arises the need to resort to designing low-pass digital filter as earlier discussed in Section 3.2.3.

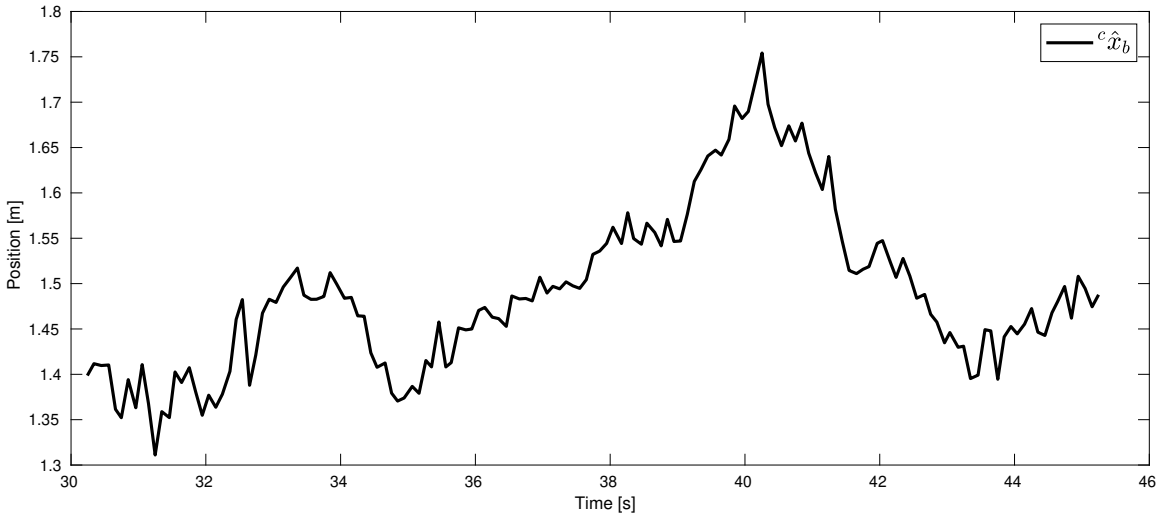


Figure 4.19: Sample Raw PnP Estimation for x-Axis

As it is illustrated in Figure 4.19, it is obvious that the signal needs filtering and enhancement. Therefore, low-pass Butterworth filter are to be designed according to the formulations discussed Section 3.2.3, however, one big difference in this section is that the cut-off frequencies are set higher than the target ones in order to decrease the delay.

Table 4.3: Cut-off Frequencies for PnP Low-Pass Filters

Axis	Frequency [Hz]
X	2.0
Y	2.0
Z	2.5

Considering the specified cut-off frequencies summarized in Table 4.3 we have:

- For x axis:

$$H_x(z) = \frac{0.0675 + 0.1349\,z^{-1} + 0.0675\,z^{-2}}{1 - 1.1430\,z^{-1} + 0.4128\,z^{-2}} \tag{4.59}$$

- For y axis:

$$H_y(z) = \frac{0.0675 + 0.1349\,z^{-1} + 0.0675\,z^{-2}}{1 - 1.1430\,z^{-1} + 0.4128\,z^{-2}} \tag{4.60}$$

- For z axis:

$$H_z(z) = \frac{0.0976 + 0.1953\,z^{-1} + 0.0976\,z^{-2}}{1 - 0.9428\,z^{-1} + 0.3333\,z^{-2}} \tag{4.61}$$

Figure 4.20 includes the raw and filtered signals for PnP estimation over x-axis. It could be observed that the low-pass filter has successfully eliminated the noises and enhanced the signal. All axes will be discussed thoroughly in the next Chapter.

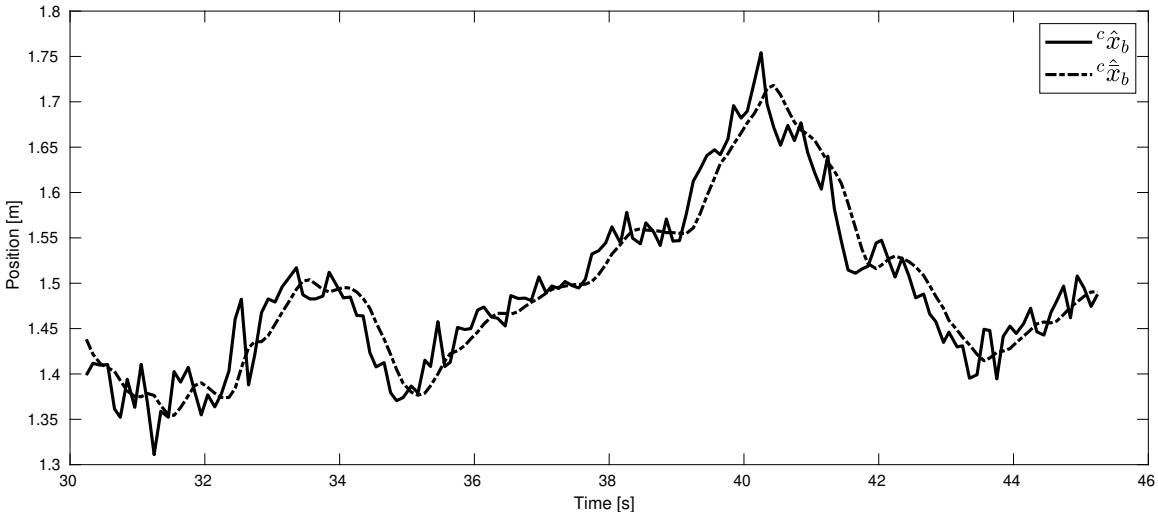


Figure 4.20: Sample Raw and Filtered PnP Signals for x-Axis

Chapter 5

Experiments and Results

After a long journey of going through the preliminaries and concepts of control, pursuit and deep learning in the previous chapters, we now aim to evaluate and assess the performed experiments and verify the results in accordance with the desired objectives.

The first step to achieve autonomous pursuit is a reliable control framework. Therefore, we shall first evaluate the tracking performance of the designed PID controller over four different axes of x, y, z and yaw in section 5.1. Subsequently, we need to validate the 3D bounding box estimation of the Keypoint-RCNN network by comparing the ground truth vertices coordinates with the predicted ones one by one. In Section 5.3, pose estimation performance shall be evaluated in two offline and real-time manners to demonstrate the delay factor in real-time pose estimation.

Eventually, Section 5.4 validates the autonomous pursuit of the target drone in two scenarios of Vicon-based and PnP-based AI pursuits.

5.1 PID Performance Evaluation

PID design procedures were earlier discussed in Chapter 3. Now, in this section, it is aimed to assess the controller tracking performance on different axes to ensure the agility and dexterity of the controller in a real-time manner for autonomous flights.

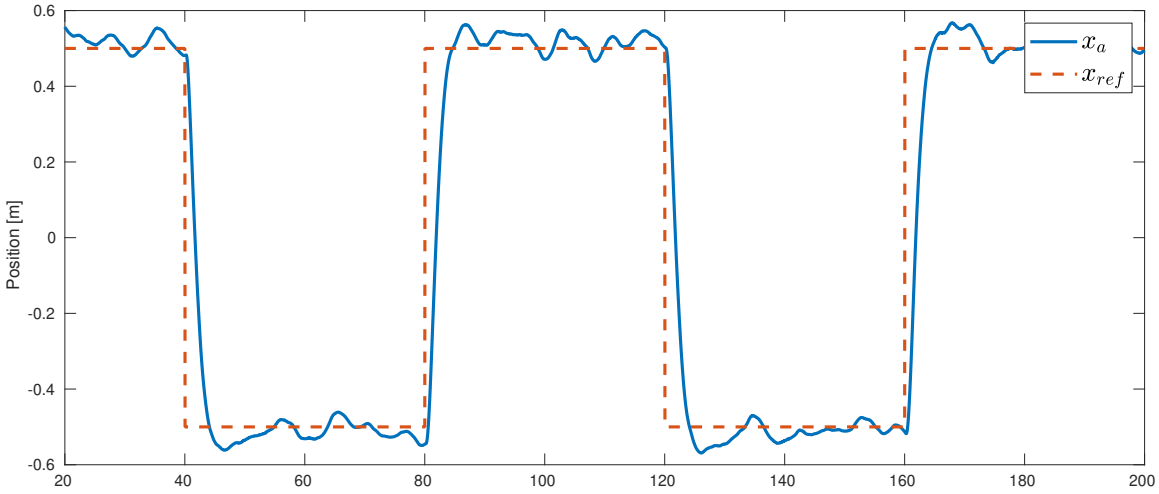


Figure 5.1: X-axis PID Tracking Assessment

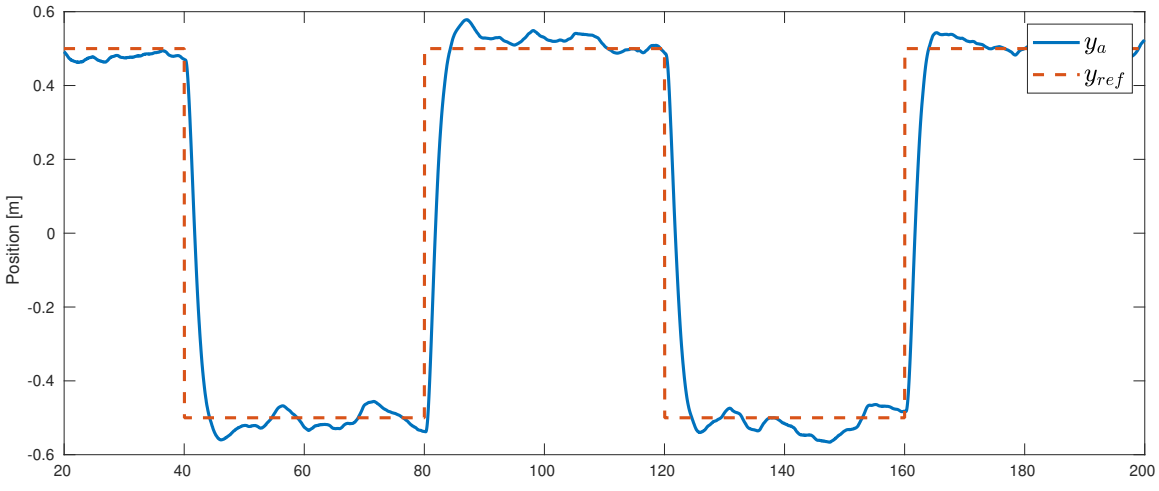


Figure 5.2: Y-axis PID Tracking Assessment

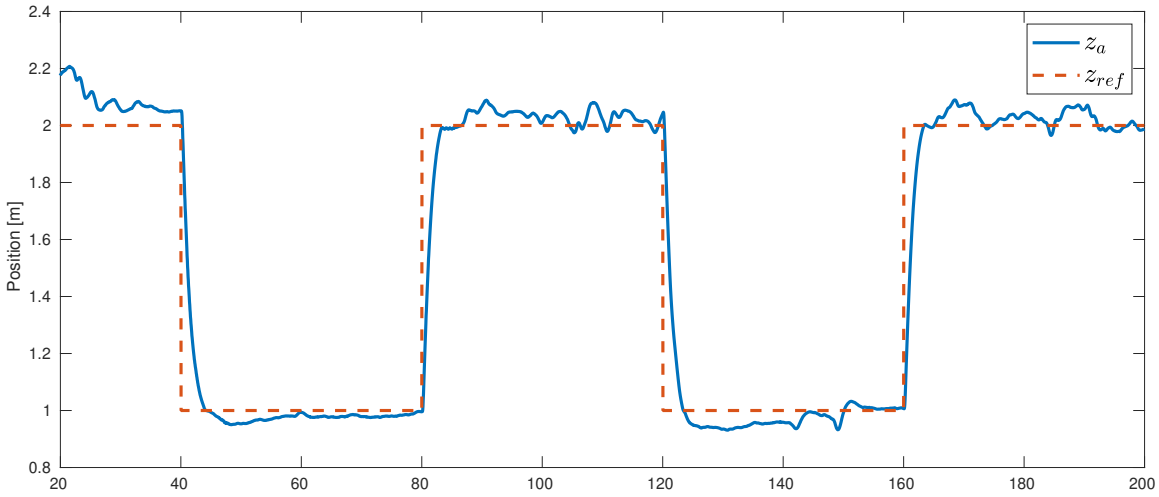


Figure 5.3: Z-axis PID Tracking Assessment

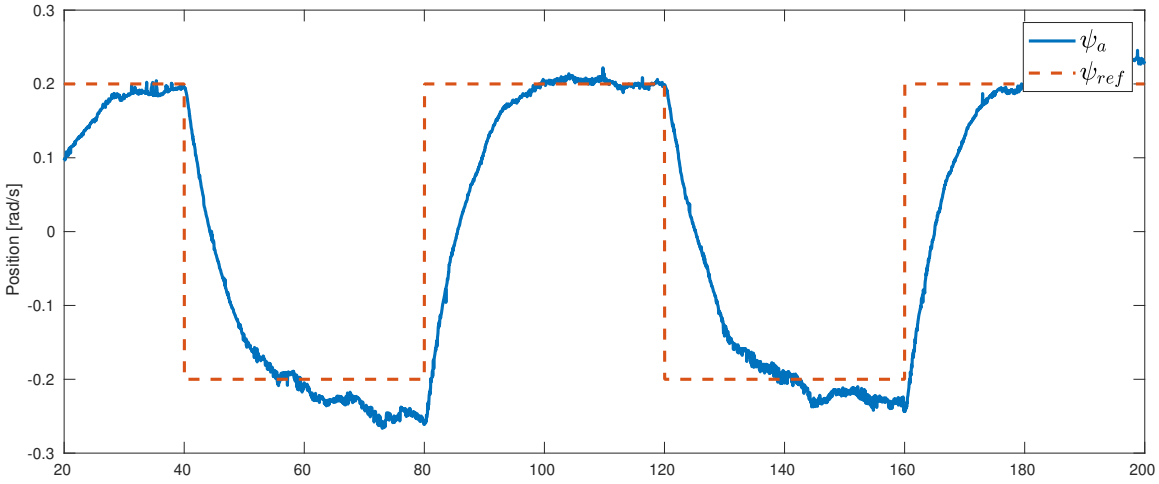


Figure 5.4: Ψ-axis PID Tracking Assessment

Figures 5.1 to 5.4 represent the input axial reference signals and the corresponding output response of the system. The input is in fact a pulse train and is utilized to evaluate the tracking performance of the controller in transient and steady state zones. It could be inferred from the results that the controller has been able to track the input properly and reliably with some overshoot. To have a more objective sense of the controller performance, Table 5.1 entails the details of overshoot, rise time, settling time and mean average error of the axial responses.

As it is summarized in Table 5.1, the controller has fast tracking rise time for X, Y

Table 5.1: PID Axial Assessment

Axis	Rise Time [s]	Settling Time [s]	Overshoot [%]	MAE
X	2.76	8.78	12.52	0.0646 [m]
Y	2.61	6.69	15.65	0.0635 [m]
Z	2.41	5.094	NA	0.0637 [m]
Ψ	11.41	0.98	NA	0.0663 [rad]

and Z axes with a rise time less than three seconds. Although, due to the slow dynamic of Ψ axis, the rise time takes over 11.41 seconds over this axis. Settling times are too desirable with a maximum settling time of 8.78 seconds over S axis. No considerable overshoot has been observed over Z and Ψ, however, X and Y have experienced an overshoot of 12.52% and 15.65% which are acceptable. Most importantly, the mean absolute error criterion expresses how accurate has the controller tracked the reference signals over the experiment run-time. The MAE for X, Y and Z are all lest than 0.065 meters or 6.5 cm which means the average error for each sample output signals on these axes has an average error of only 6.5 cm. The MAE for Ψ is almost 0.67 rad or 3.8 degrees. Overall, the axial MAE results indicate the robust tracking performance of the designed flight controller.

5.2 Box Estimation Assessment

Estimating the right and proper 3D bounding box is essential for later pose estimation. Any error in the box data would lead to amplified error in pose data. Therefore, it is essential to verify the estimated box data in order to prevent any malfunctioning in pose estimation and flight control. To assess the 3D bounding box, we need to assess each predicted keypoint (or vertex) individually. As stated before in Chapter 4, each keypoint is represented by two parameters (u_i and v_i) on image plane. Therefore, it is needed to evaluate each axis data individually as well.

To validate the data, 2,000 validation frames in addition to corresponding rectified bounding box vertices data have been collected. The frames were then inputted to the R-CNN network and the predicted data were recorded in the same csv file along with the ground truth data. The results are plotted in Figures 5.7 to 5.14.The results show robust and perfect performance of the R-CNN network and the predicted data thoroughly match the Vicon data. It should be noted again that this dataset is captured in a busy and noisy background without using any kind of curtain or background manipulator.

Error and Accuracy

Before getting into the quantitative evaluation of the keypoint estimation, it is required to set up the preliminaries for error and accuracy calculation of vertices and keypoints. As stated before, each keypoint is represented on image plane using two axes of u and v. Hence, each axial error could be defined as follows:

$$e_u^{(i)} = u^{(i)} - \hat{u}^{(i)} \tag{5.1}$$

$$e_v^{(i)} = v^{(i)} - \hat{v}^{(i)} \tag{5.2}$$

What is meant by accuracy, is a metric which maps the error into a one-dimensional parameter ranging from 100% for zero error, and 0% for infinite error. Exponential function could satisfy this need accordingly. Thus, the axial accuracy could be formulated as follows:

$$\alpha_u^{(i)} = 100\, e^{-\zeta\, |u^{(i)} - \hat{u}^{(i)}|} \tag{5.3}$$

$$\alpha_v^{(i)} = 100\, e^{-\zeta\, |v^{(i)} - \hat{v}^{(i)}|} \tag{5.4}$$

Where α is the accuracy metric in percentage and ζ is the accuracy scaling parameter. For one dimensional accuracy metric with $\zeta = 0.01$, we have:

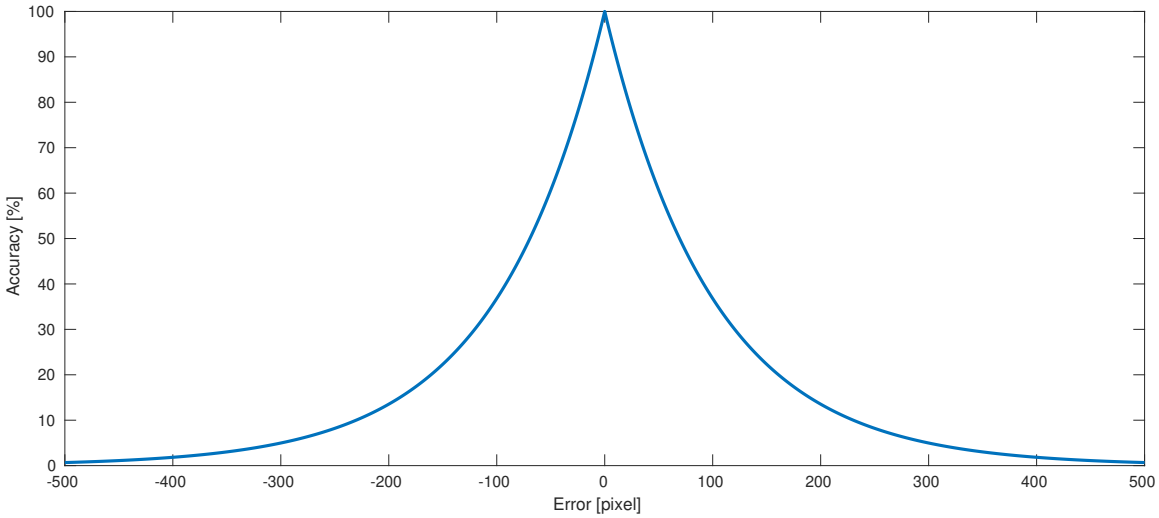


Figure 5.5: One-Dimensional Accuracy Plot

Based on Figure 5.5, a twenty-pixel error would lead to 80% accuracy and a hundred-pixel error would lead to 36% accuracy. Accuracy would converge to zero by increasing the error to infinity.

For obtaining an accuracy metric for the keypoints, it is needed to incorporate both u and v parameters at the same time. To do so, Euclidean distance of the estimated keypoint and the original one is chosen as the basis for calculating error as follows:

$$\vec{e}_k^{(i)} = \begin{bmatrix} u^{(i)} \\ v^{(i)} \end{bmatrix} - \begin{bmatrix} \hat{u}^{(i)} \\ \hat{v}^{(i)} \end{bmatrix} = \begin{bmatrix} u^{(i)} - \hat{u}^{(i)} \\ v^{(i)} - \hat{v}^{(i)} \end{bmatrix} \tag{5.5}$$

Thus:

$$\alpha_k^{(i)} = 100\, e^{-\zeta\, ||\vec{e}_k^{(i)}||_2} \tag{5.6}$$

Where:

$$||\vec{e}_k^{(i)}||_2 = \sqrt{(u^{(i)} - \hat{u}^{(i)})^2 + (v^{(i)} - \hat{v}^{(i)})^2} \tag{5.7}$$

Figure 5.6 illustrates the two-dimensional accuracy plot of keypoint with respect to u and v axes and $\zeta = 0.01$. Based on this figure, the further we distance from

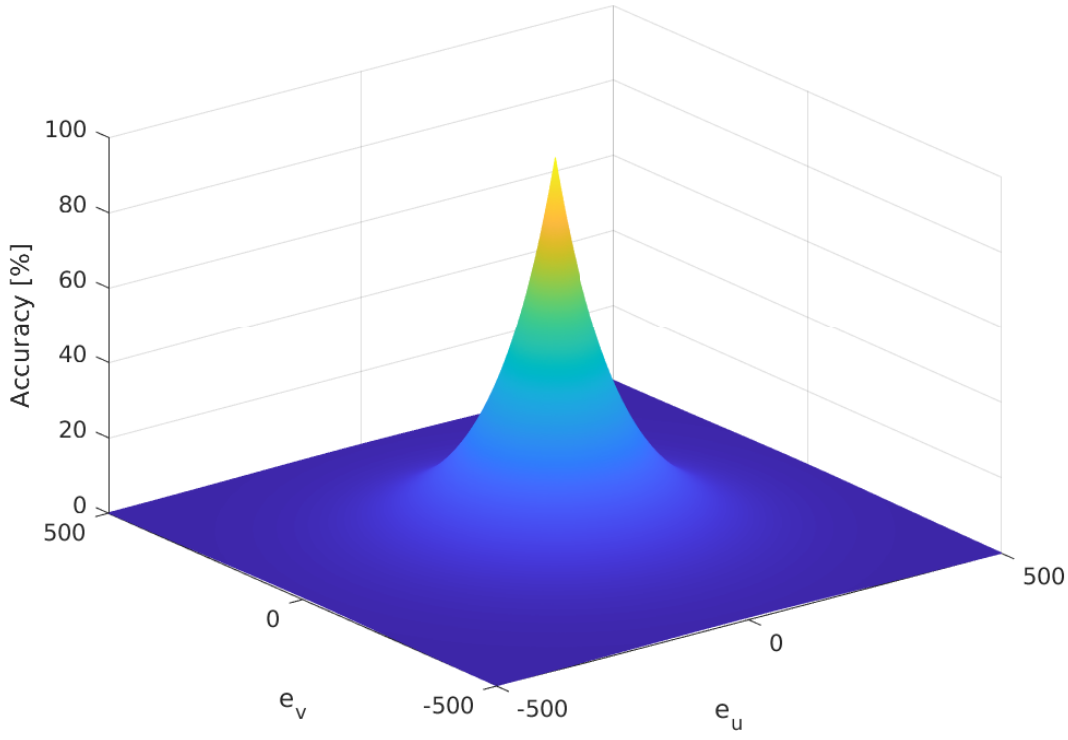


Figure 5.6: Two-Dimensional Accuracy Plot

the target on concentric circles, the less accuracy we get. Estimated keypoints on the same circle with same radius woud lead to same accuracy value.

The mean average error and accuracy data of each keypoint is summarized in Table 5.2. As it is represented, all keypoints have a MAE of less than 10 pixels ranging from 6.63 to maximum of 8.86 pixels for K_7 on u axis and ranging from 2.96 to 5.06 pixels on v axis. The accuracy results are based on Equation 5.3 and all axes have obtained +90% accuracy for axial estimation. Furthermore, overall accuracy is higher on v compared to u axis and this could be a results of wider domain of u axis with 1280 pixels compared to 720 pixels of v.

Table 5.3 entails the overall MAE and accuracy for keypoint assessment in a two-dimensional manner instead of assessing the keypoints in axial manner. Based on Equation 5.6, the accuracy column indicates how close the estimated keypoints are to

the target ones. All eight estimated keypoints are more than 90% accurate. It should be noted again that theses numbers are retrieved over 2,000 raw validation frames. The MAE and accuracy results are the state-of-the-art results for this novel method of 3D bounding box estimation.

Table 5.2: Bounding Box Keypoints' Validation MAE and Accuracy

Keypoint	MAE [u]	MAE [v]	Accuracy [u]	Accuracy [v]
K_0	6.92	3.04	93.88	97.09
K_1	6.63	3.20	93.80	96.89
K_2	6.87	3.57	93.60	96.62
K_3	7.06	3.11	93.73	96.98
K_4	8.84	2.96	91.86	97.24
K_5	8.40	4.38	92.71	95.81
K_6	8.53	5.06	92.62	95.22
K_7	8.86	3.75	91.81	96.38

Table 5.3: Bounding Box Overall Validation MAE and Accuracy

Keypoint	MAE	Accuracy [%]
K_0	8.16	92.74
K_1	8.03	92.49
K_2	8.43	92.23
K_3	8.35	92.53
K_4	9.90	90.98
K_5	10.53	90.76
K_6	10.83	90.56
K_7	10.54	90.27

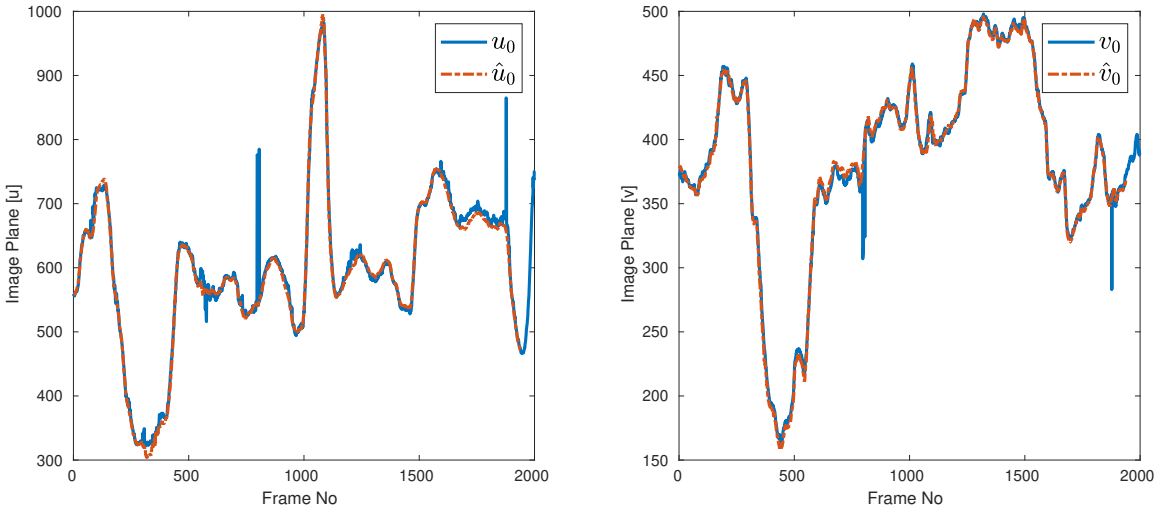


Figure 5.7: K_0 Estimation Assessment

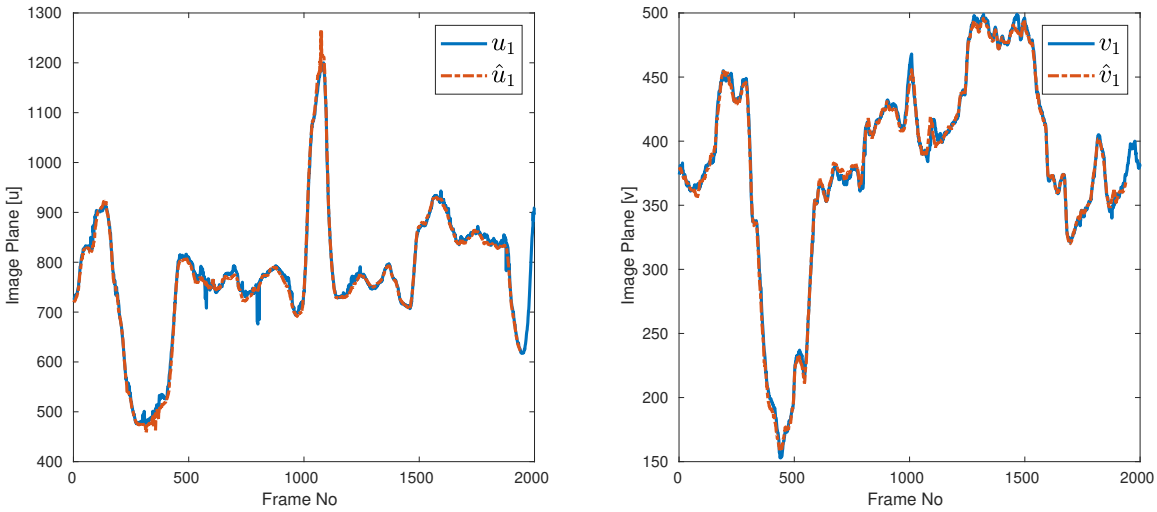


Figure 5.8: K_1 Estimation Assessment

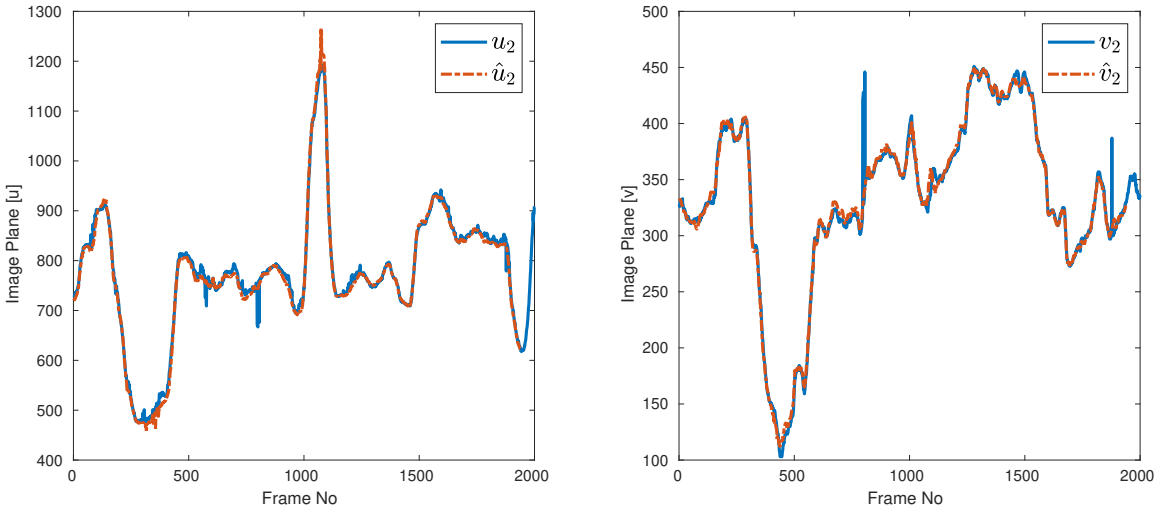


Figure 5.9: K_2 Estimation Assessment

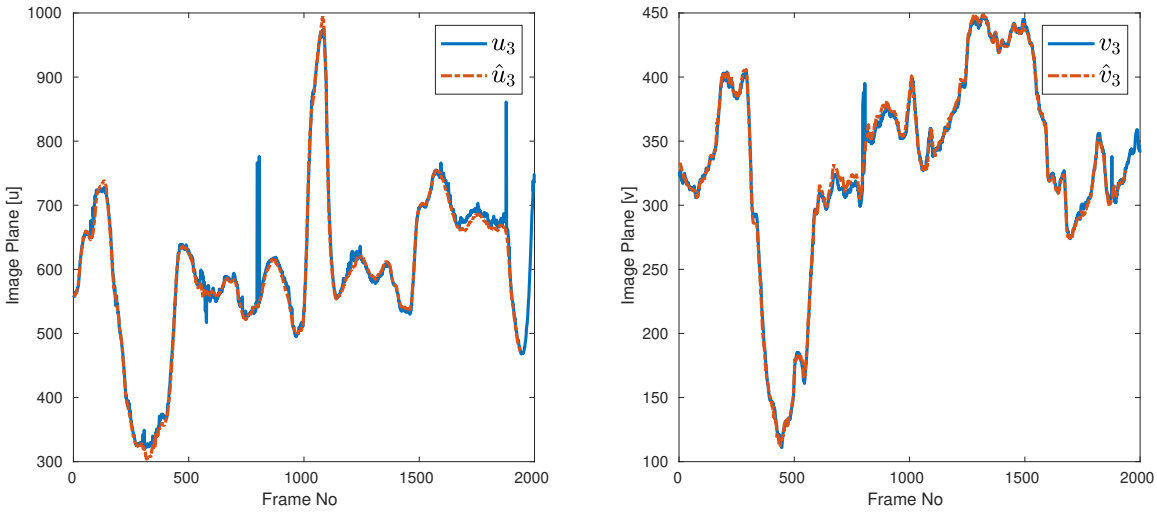


Figure 5.10: K_3 Estimation Assessment

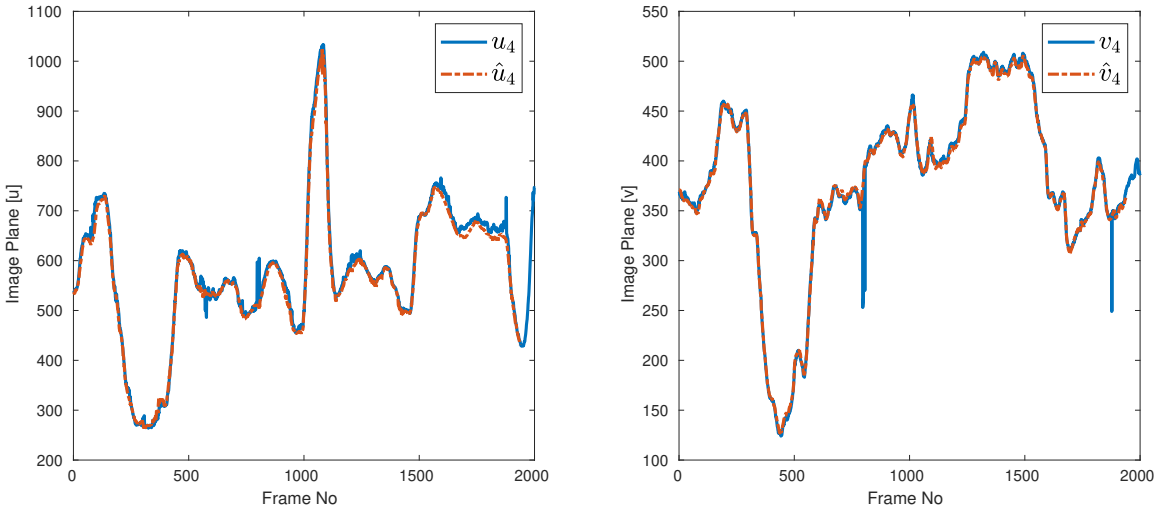


Figure 5.11: K_4 Estimation Assessment

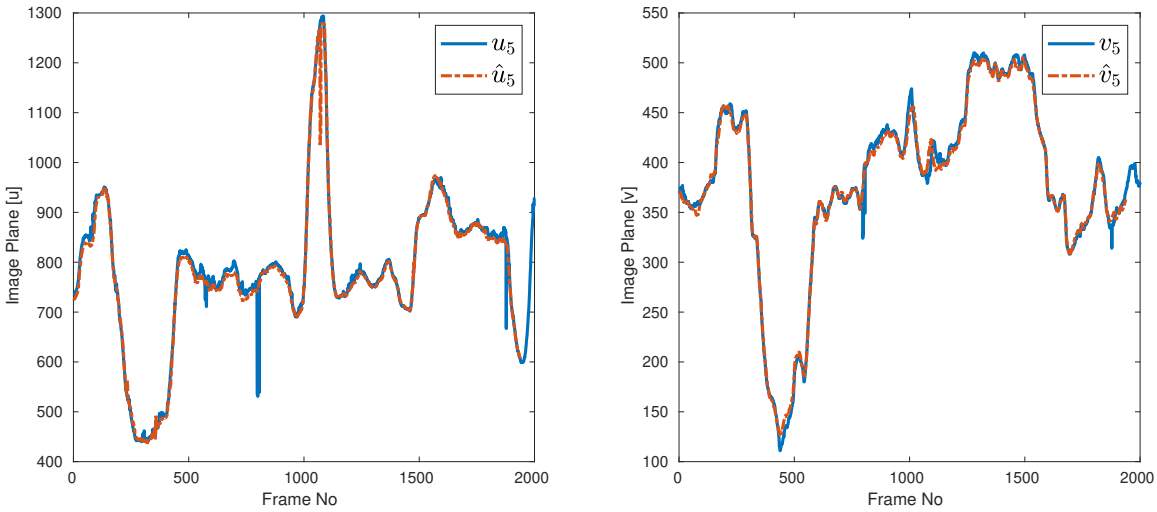


Figure 5.12: K_5 Estimation Assessment

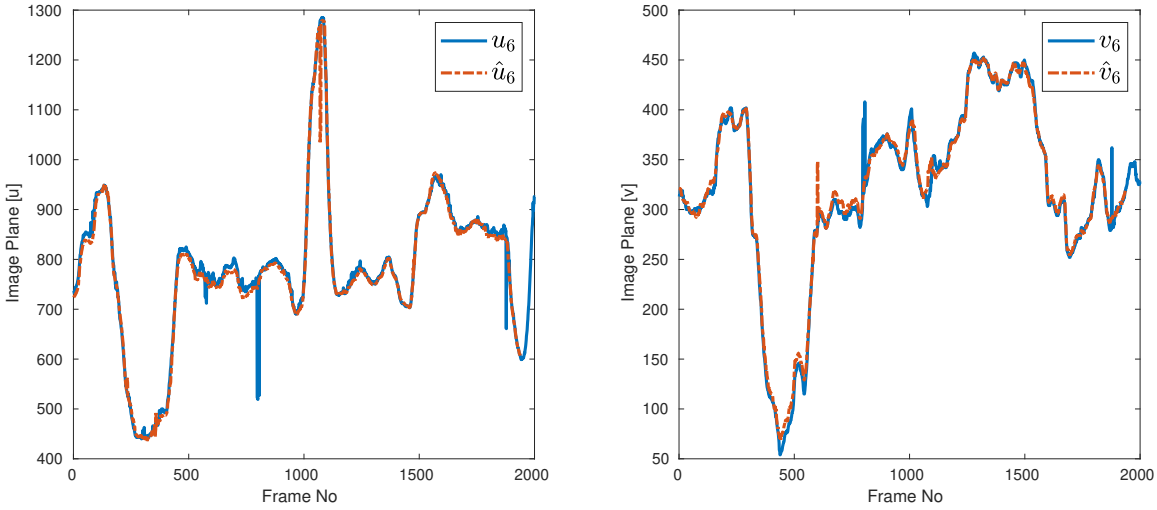


Figure 5.13: K_6 Estimation Assessment

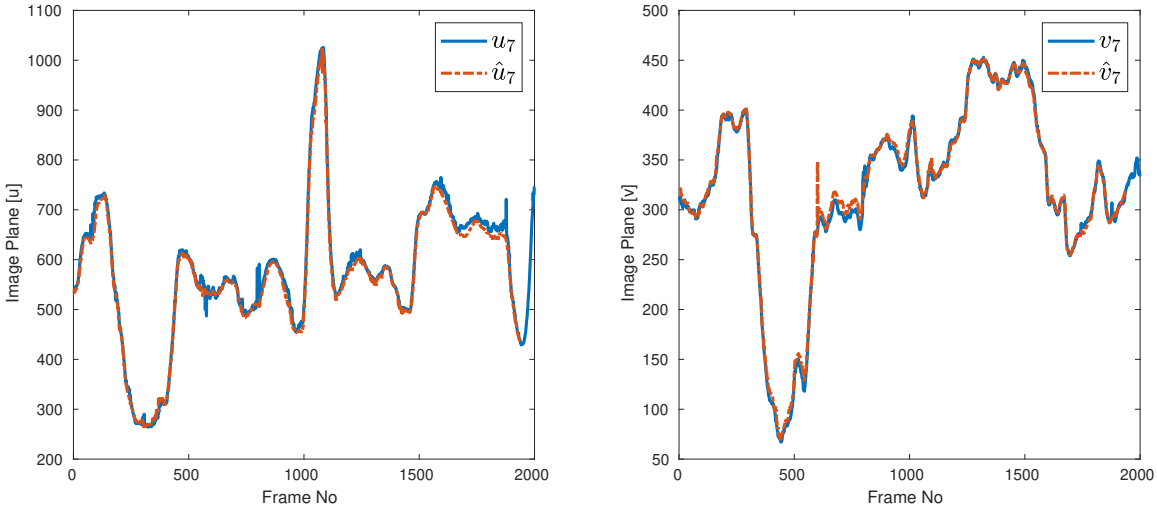


Figure 5.14: K_7 Estimation Assessment

5.3 Pose Estimation Assessment

Being ensured by the performance of the keypoint detection and box estimation algorithms paves the way for the next assessment section of pose estimation. It was earlier discussed in Chapter 4 that by obtaining the proper projected vertices coordinates on the image plane and feeding them into PnP algorithm, relative pose of the target object could be retrieved accordingly.

However, as mentioned earlier, the PnP output contains noises caused by the delays and errors in box estimation which are amplified while be processes through PnP. To overcome the noises and removing the noises from the output pose data, low-pass Butterworth filter were designed and introduced in Section 4.4.1. Although the designed filters cancel the noise desirably, they cause delays in the output signals which are added to the already existing delays caused by Wi-Fi transmission and image processing.

As a result, to process the PnP performance, it is needed to assess it in two manners of offline and real-time processing. In case of offline processing, pose estimation is solely assessed through PnP without any delays as the images are processed one-by-one in a queue manner which eliminates the delay. On the other hand, in real-time pose estimation, delays are present and some images might even be missed which in that case, the pose data is retrieved from the previously estimated data.

5.3.1 Offline Pose Estimation

For offline pose estimation, a flight test was run and the frames and pose were collected in the same manner the deep learning training data were collected earlier described in Section 4.3.1.1. Both drones were flown and the target Bebop was flown arbitrarily to cover different poses and configurations. The frames and csv data were stored and saved simultaneously while the test was being run.

For post processing the frames and obtaining offline pose data, first the stored raw

frames were inputted into the R-CNN network one by one to obtain the keypoints. The retrieved keypoints were then saved on the same csv file according to the corresponding frame IDs. After processing the frames and retrieving the keypoints, a C++ node is run to post-process the keypoints and calculate the pose information. The obtained PnP pose data are subsequently saved in the same csv file with respect to frame IDs.

Table 5.4: Offline Pose Estimation MAE and Accuracy

Signal	MAE [m]	Accuracy [%]
${}^{c}x_b$	0.0381	99.96
${}^{c}x_b^f$	0.0363	99.96
${}^{c}y_b$	0.1199	99.88
${}^{c}y_b^f$	0.1191	99.88
${}^{c}z_b$	0.0562	99.94
${}^{c}z_b^f$	0.0569	99.94
${}^{c}d_b$	0.0366	99.96
${}^{c}d_b^f$	0.0359	99.96

Figures 5.15 to 5.17 illustrate the offline-pose estimation of Bebop drone with respect to camera frame. Each plot contains three variables including:

- ${}^{c}x_b$ Ground truth position based on Vicon
- ${}^{c}\breve{x}_b$ Estimated offline PnP position parameter
- ${}^{c}\breve{x}_b^f$ Estimated filtered offline PnP position parameter

So, the ground truth is the Vicon position data retrieved from `vicon_bridge`. Estimated offline PnP position parameter is in fact raw output of `cv:solvePnP` required to be filtered due to unflattering and spikes on the output signals. Subsequently, the filtered signals are included as well denoted by the (f) superscript.

Moreover, Figure 5.18 represents the Euclidean distance between the Anafi and the Bebop drones to bring more insight into overall distance estimation of the target object.

To have a more numerical bedrock for analyzing the offline pose estimation we should take a look at Table 5.4 which includes the MAE and accuracy for x, y, z axes and Euclidean distance as well. The MAE and accuracy parameters are obtained for both raw and filtered PnP signals with respect to ground truth Vicon signals.

As it is summarized in Table 5.4, the MAE and accuracy results demonstrate the precise and accurate performance of our algorithm with 0.0381[m] mean absolute error for x axis, 0.1199[m] for y, 0.0562[m] for z and 0.0366[m] for overall distance estimation. By ensuring the offline performance of the RCNN-PnP algorithm, we can now set sail to evaluate the more important real-time analysis of pose estimation algorithm in next section.

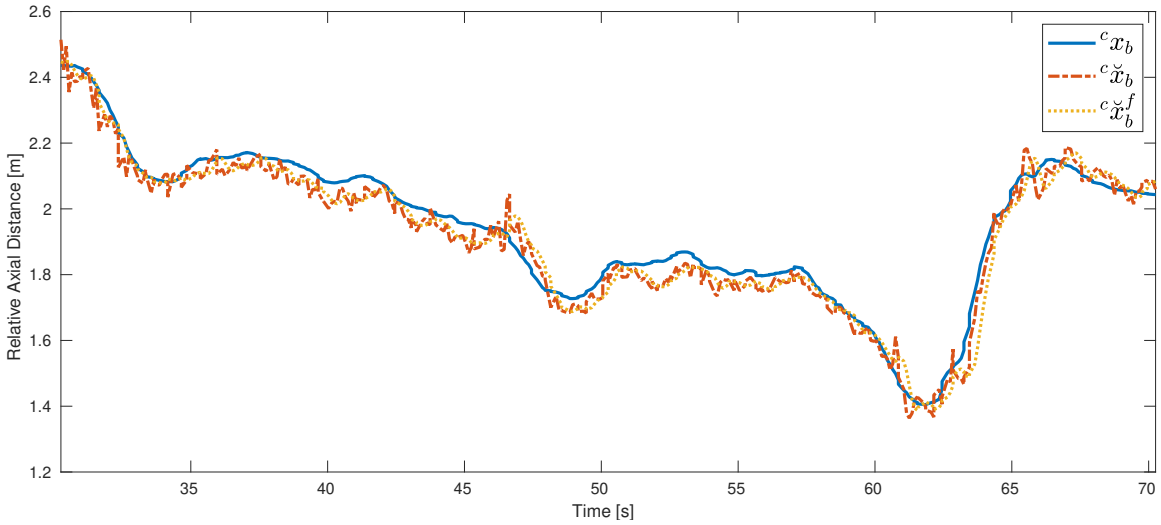


Figure 5.15: Original and Estimated Offline x Axis Signal

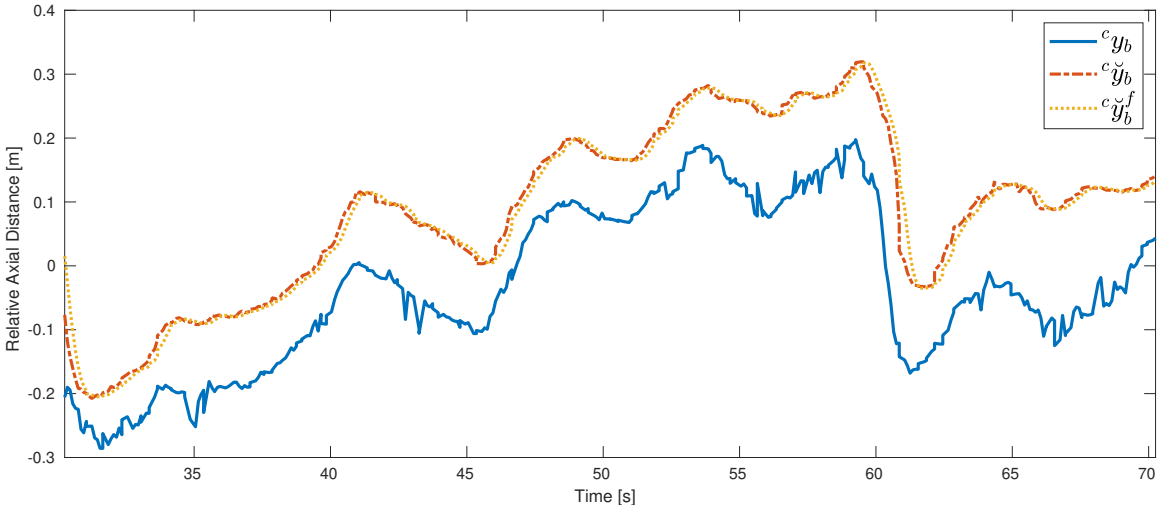


Figure 5.16: Original and Estimated Offline y Axis Signal

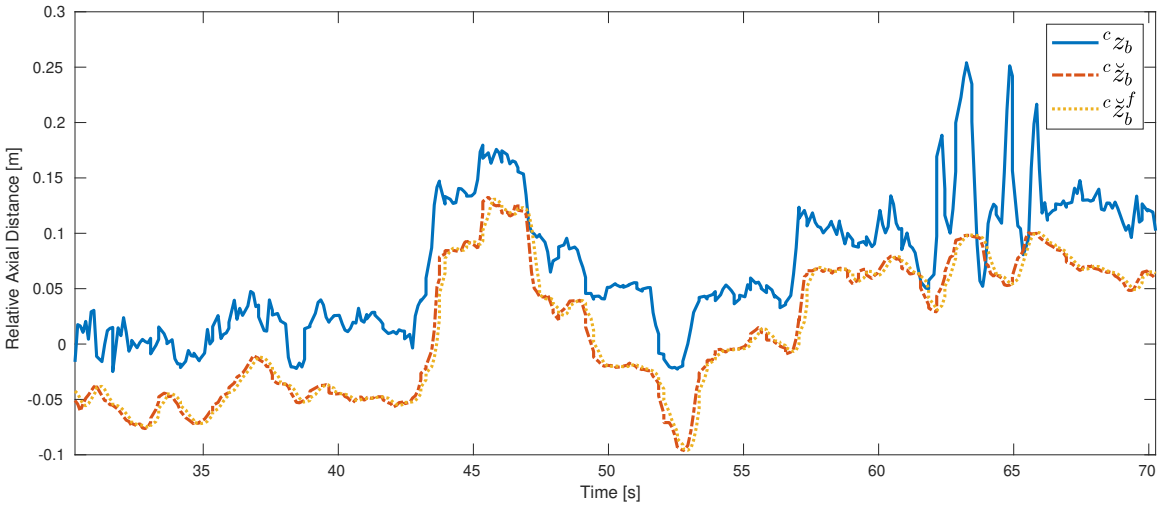


Figure 5.17: Original and Estimated Offline z Axis Signal

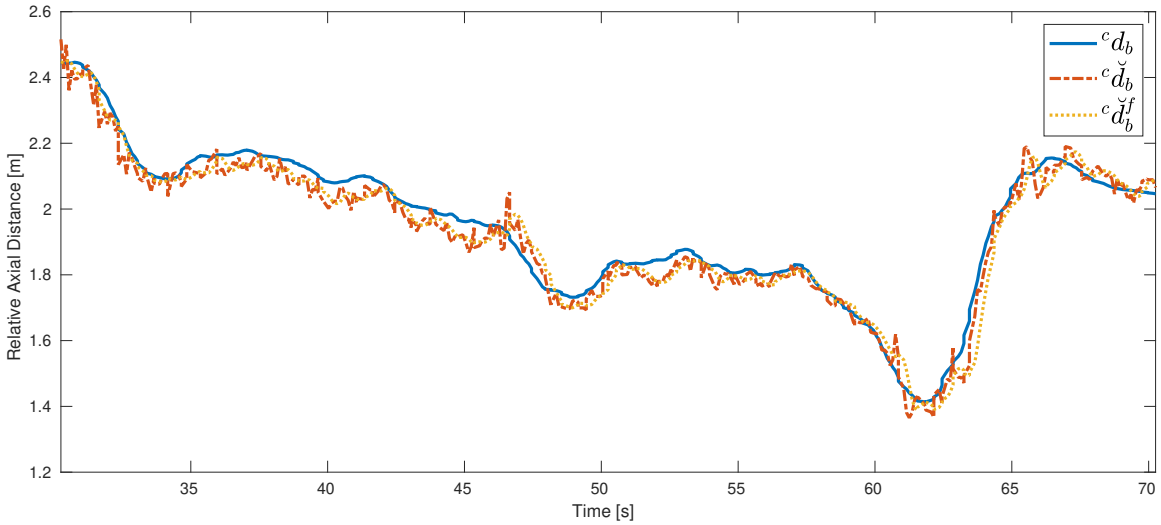


Figure 5.18: Original and Estimated Offline Distance Signal

5.3.2 Real-Time Pose Estimation

Real-time processing is indeed more demanding and desirable in robotics applications as the due to the real-time nature of the tasks and applications and this project is no exception. To fulfill this need, it is necessary to evaluate the pose estimation in a real-time manner.

Meanwhile, it is necessary to have a bedrock for comparing our RCNN-PnP based method with the real-time results with the sate-of-the-art algorithms to have a better understating and insight on the robustness and dexterity of our method for real-time pose estimation. We already discussed RGN (Relational Graph Networks) by Jin et al. [18] in Chapter 1. This method utilizes relational graph networks with PnP for obtaining the target drone pose. It should be noted that this method is based on using a stationary camera, on the contrary, our method is reliant on mobile mounted camera on the pursuer drone. Nevertheless, this method is a proper set-point for comparing our method as purpose is to estimate the target pose of a Bebop drone which is the same drone used in this research as well.

Table 5.5: Real-Time Pose Estimation MAE Comparison

	Parameter			
Method	${}^{c}x_{b}^{f}$	${}^{c}y_{b}^{f}$	${}^{c}z_{b}^{f}$	d^{f}
RGN	0.0757	N/A	N/A	N/A
Ours	0.0485	0.1178	0.0582	0.0485

Table 5.5 summarizes the MAE results for filtered poses estimation for x, y, z and Euclidean distance. Our methods provide the MAE results for all parameters; however, RGN method only provides the MAE for x axis. Considering the provided data, our method has improved the state-of-the-art pose estimation by decreasing the MAE by almost 0.0272[m].

Our state-of-the-art results over y and z axes are as robust and precise as x axis

as well by having 0.1178[m] and 0.0582[m] of MAE respectively. Euclidean distance estimation proves robustness and agility as well by having only 0.0485[m] of MAE over 400 sample frames and pose data.

Table 5.6: Real-Time Pose Estimation Accuracy

d Axis	Accuracy [%]
${}^{c}x_{b}$	99.95
${}^{c}y_{b}$	99.88
${}^{c}z_{b}$	99.94
${}^{c}d_{b}$	99.95

Table 5.6 summarized the accuracy results of the axial pose and Euclidean distance estimation using Euler function. High accuracy results are also evident in Figure 5.19 to 5.22. The scale of precision and accuracy of our results are unprecedented in aerial and mobile Robotics, paving the way for real-time monocular vision-based pursuit in the next section.

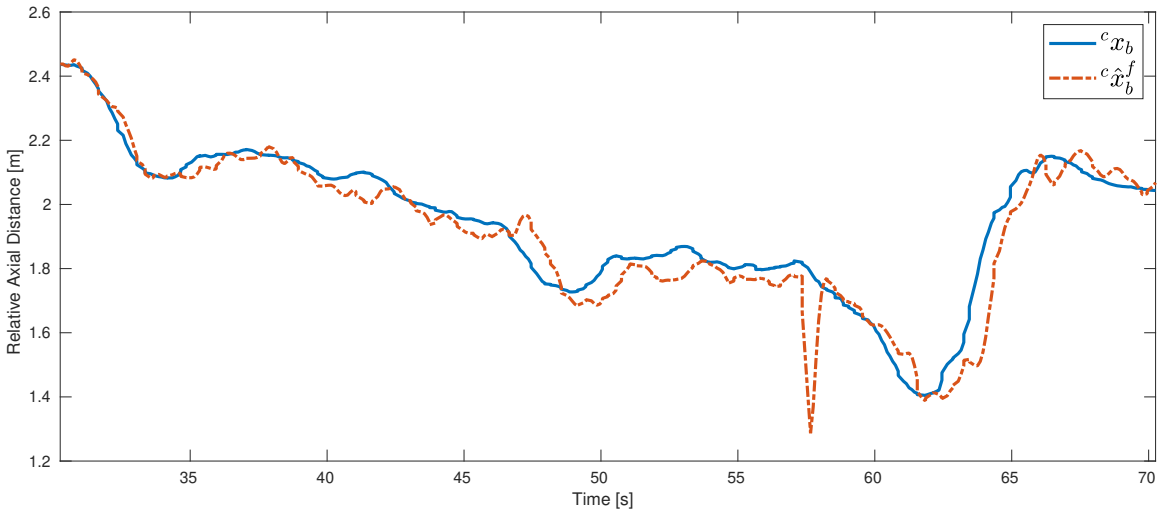


Figure 5.19: Original and Estimated Real-Time x Axis Signal

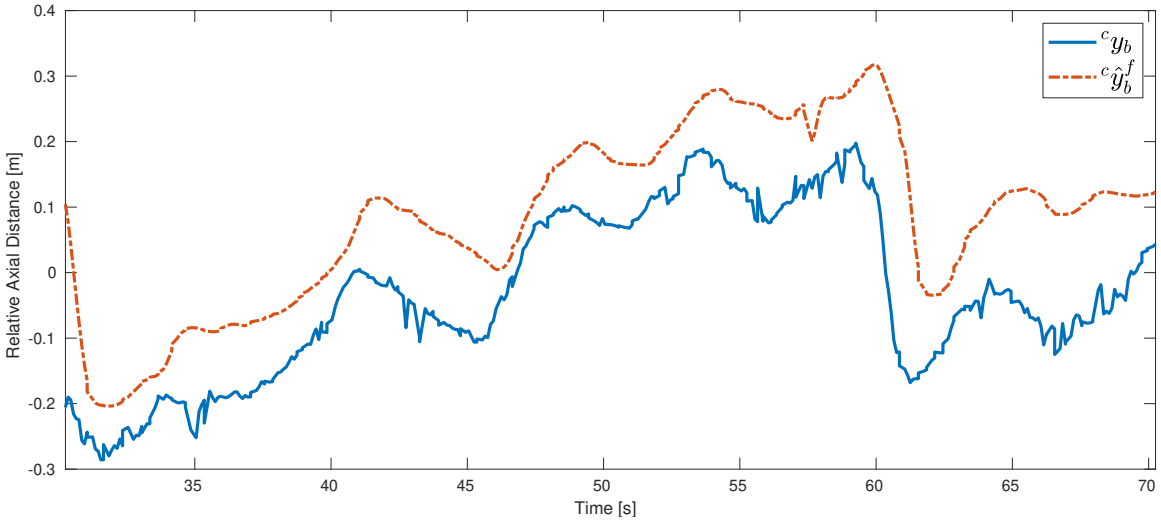


Figure 5.20: Original and Estimated Real-Time y Axis Signal

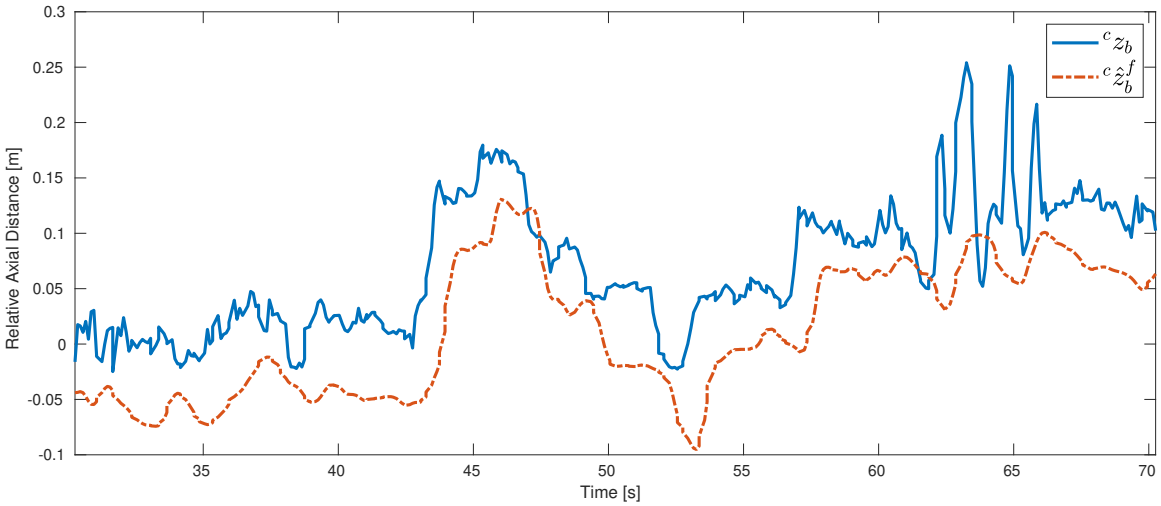


Figure 5.21: Original and Estimated Real-Time z Axis Signal

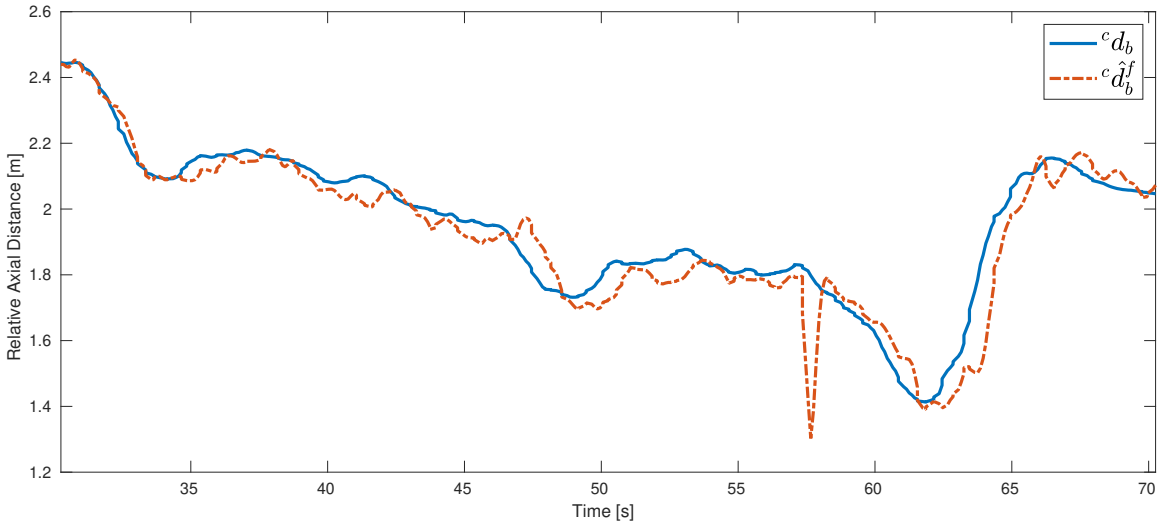


Figure 5.22: Original and Estimated Real-Time Distance Signal

5.4 Pursuit Assessment

All previous Sections including flight controller, 3D bounding box detection and pose estimation were designed and assessed in order to set up a platform for implementing deep autonomous pursuit algorithm to fulfill autonomous UAV pursuit needs. Now that we are assured of the performance of the controllers and deep pose estimation algorithms, we can set sail for performing tests to evaluate and assess pursuit.

For better understating over UAV pursuit, the assessment has been divided into two sections including Vicon-based UAV pursuit (VUP) and PnP-based UAV pursuit (PUP) in the following sections.

5.4.1 VUP

Vicon-based UAV pursuit algorithm is based on utilizing Vicon for retrieving position data of Anafi and Bebop at the same time and obtaining the relative pose of the target in order to generate the setpoint commands which was earlier discussed in Section 3.3. It's good to take a look at Figure 3.23 again to get another glimpse of the Vicon-based pursuit framework. As it is shown, the Vicon pursuit block subscribes to `/tf` topic in order to the retrieved the pose data, and subsequently publishes the reference signals to PID block for tracking purposes.

For evaluating Vicon-based pursuit, a test with a runtime of 70[s] has been performed and is visualized through Figures 5.23 to 5.26. The Figures demonstrate agile and robust tracking of the dynamic reference signals. It should be noted that the reference yaw angle has been set to zero and the controller has subsequently tried to stabilize yaw angles which is evident in Figure 5.26.

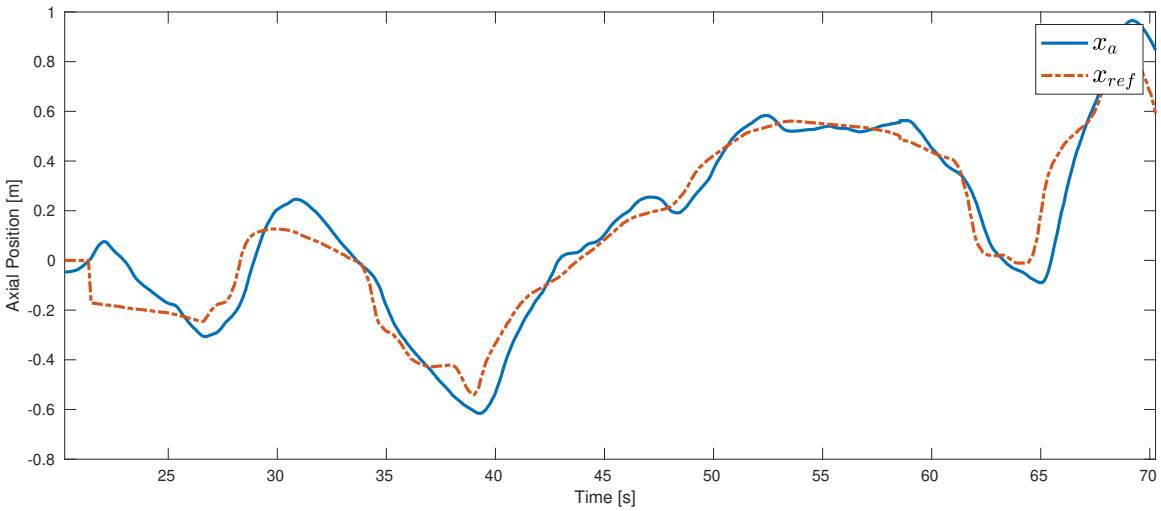


Figure 5.23: Pursuit Assess

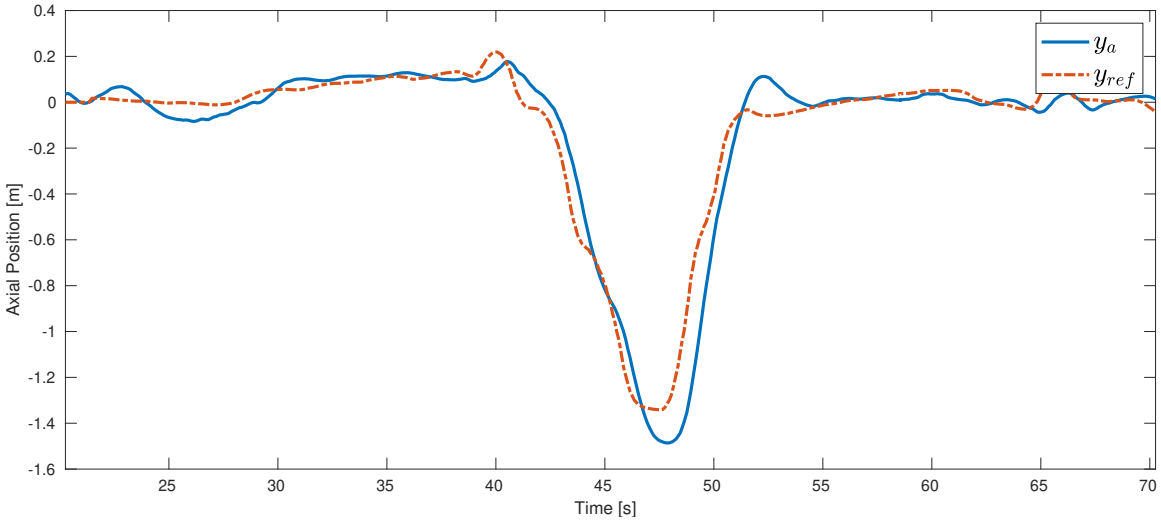


Figure 5.24: Pursuit Assess

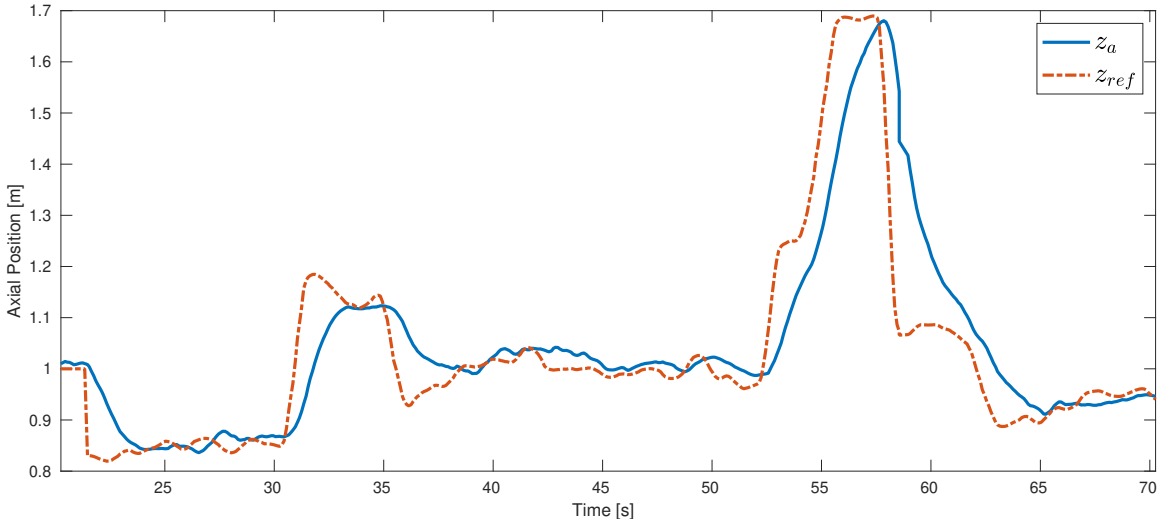


Figure 5.25: Pursuit Assess

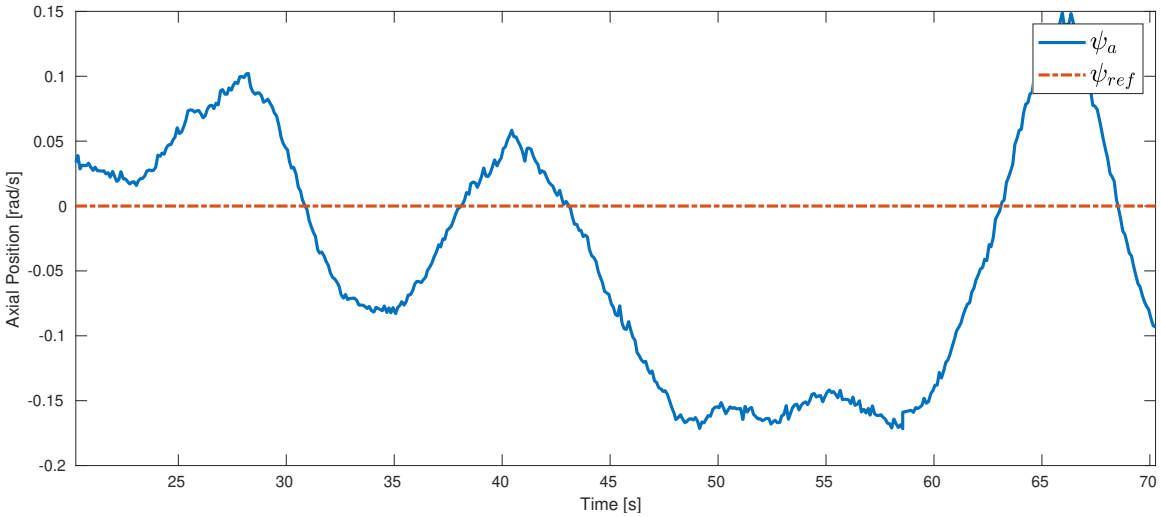


Figure 5.26: Pursuit Assess

Table 5.7 summarizes the tracking MAE for four target axes. As demonstrated, Anafi has pursued Bebop accurately and robustly with an overall MAE of less than 0.09[m] for x, y and z and 0.8[rad] for Ψ axis. The demonstrated results ensure the proper tracking performance of the controller.

Table 5.7: VUP Assessment

Axis	MAE
X	0.0744 [m]
Y	0.0592 [m]
Z	0.0598 [m]
Ψ	0.0834 [rad]

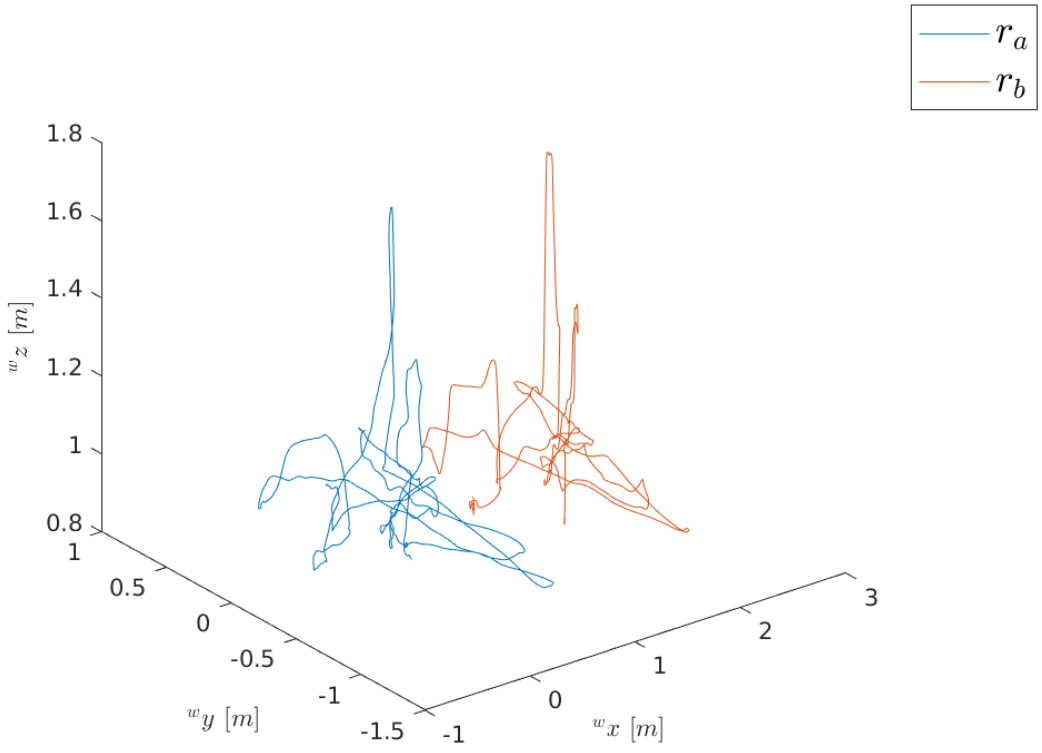


Figure 5.27: Vicon-Based Pursuit Trajectory

Overall Vicon-based pursuit trajectory has also been drawn in Figure 5.27. As

shown, the Anafi (in blue) has successfully tracked the Bebop (in red) and maintained the 1.5[m] distance to prevent collision.

5.4.2 PUP

As it was mentioned times and times in the previous sections, the whole idea of integrating keypoint-RCNN, PnP and pursuit algorithms was to create a platform for pursuing target drones using only vision without Vicon, LiDAR , radar or any other sensor or measurement tools which is called for short PUP or PnP-RCNN based UAV Pursuit.

PUP ROS implementation was earlier discussed in Chapter 3 Figure 3.24. As it was demonstrated, the target Vicon pose data (`/bebop/vicon/vicon`) was replaced by (`/anafi/pnp`) containing the relative pose of the target with respect to Anafi camera frame.

For having better analysis on axial pursuit, PUP has been performed and tested in an axis-based manner. To do so, while each axis is being tested, the other axes are set to a fixed setpoint for restricting the pursuit to the desired axis.

X-Axis Pursuit

For x-axis pursuit, the ROS launch files have been modified in a way to generate a static setpiont for y, z, and yaw axes according to the values recorded in Table 5.8 as 0.0 [m] for y, 1.0 [m] for z and 0.0 [rad] for yaw. The tracking performance for static setpoints should demonstrate the same results as the Vicon-based pursuit which is confirmed by the values summarized in Table 5.8. The only difference, is the MAE of x-axis which is based on utilizing RCNN-PnP for estimating the pose and tracking the estimated reference signal and target.

Figure 5.28 includes the graphs of real time depth estimation and pursuit over x-axis depicted as 5.28a and 5.28b subfigures. As it is represented, the online-pose estimation shows consistent and agile performance which only comes with the issue

Table 5.8: X-axis PnP-based Pursuit Results

Axis	Pursuit Type	Setpoint	Tracking MAE
X	RCNN-PnP	Dynamic	0.1694 [m]
Y	Vicon	0.0 [m]	0.0258 [m]
Z	Vicon	1.0 [m]	0.0167 [m]
Ψ	Vicon	0.0 [rad]	0.0908 [rad]

of a 0.52 [s] delay caused by wireless signal transmission and filtering.

Even with the presence of delay, the drone has been able to track the estimated reference signal (x_{ref}) and the target drone over x-axis as demonstrated in Figure 5.28b.

Y-Axis Pursuit

Similarly to x-axis, three axes are set to fixed setpoints here except y-axis set to dynamic PnP setpoints. The experimental results for the RCNN-PnP pursuit are summarized in table 5.9.

The huge difference in y-axis pursuit is the oscillatory tracking response depicted evidently in Figure 5.29b. A detailed look at the respective pose estimation plot in Figure 5.29a reveals a delay of 0.81 [s] which is relatively more than the corresponding delay of x-axis pursuit. According to control theory, delays destabilize systems quite extensively.

Table 5.9: Y-axis PnP-based Pursuit Results

Axis	Pursuit Type	Setpoint	Tracking MAE
X	Vicon	0.0 [m]	0.0478 [m]
Y	RCNN-PnP	Dynamic	0.1278 [m]
Z	Vicon	1.0 [m]	0.0112 [m]
Ψ	Vicon	0.0 [rad]	0.0334 [rad]

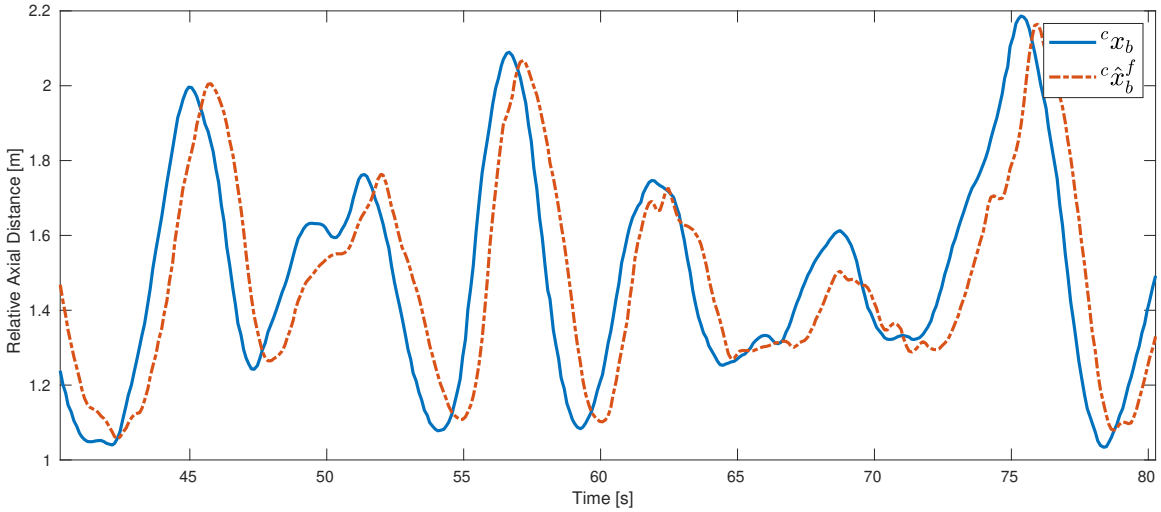
Z-Axis Pursuit

For the last part, z-axis pursuit is in fact tracking the elevation of target drone with respect to camera frame. Similarly, all axes except z-axis are set to zero as detailed in Table 5.10.

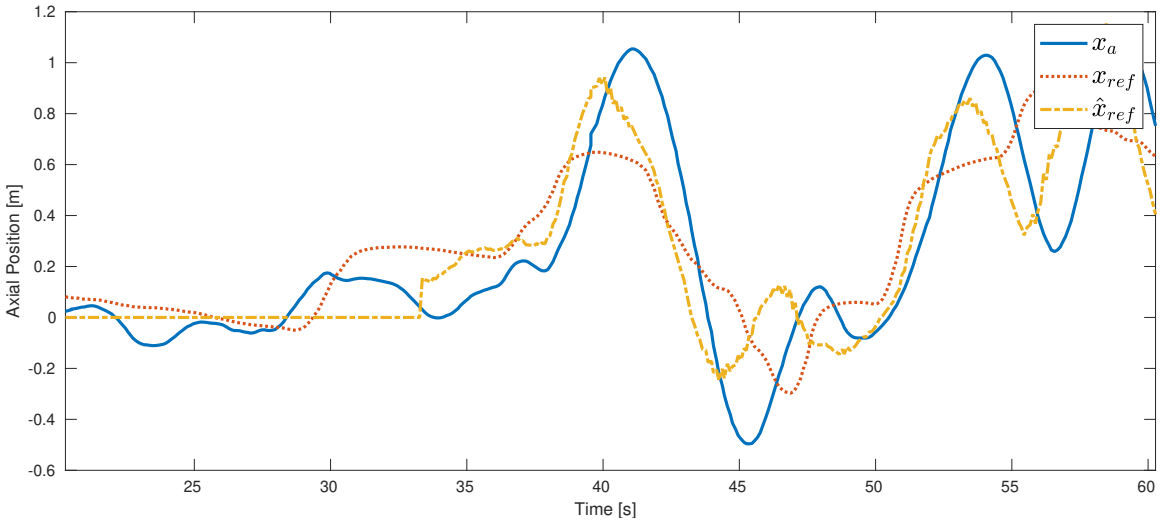
Table 5.10: Z-axis PnP-based Pursuit Results

Axis	Pursuit Type	Setpoint	Tracking MAE
X	Vicon	0.0 [m]	0.0599 [m]
Y	Vicon	0.0 [m]	0.0355 [m]
Z	RCNN-PnP	Dynamic	0.0768 [m]
Ψ	Vicon	0.0 [rad]	0.0454 [rad]

0.0768 [m] of MAE tracking error for z-axis is quite extraordinary and better than two other axes. This could be contributed to the slow dynamic of vertical thrust and relatively less delay of 0.7 [s] compared to y-axis. Consequently, the corresponding pursuit response of z-axis comes with insignificant oscillations over the input.

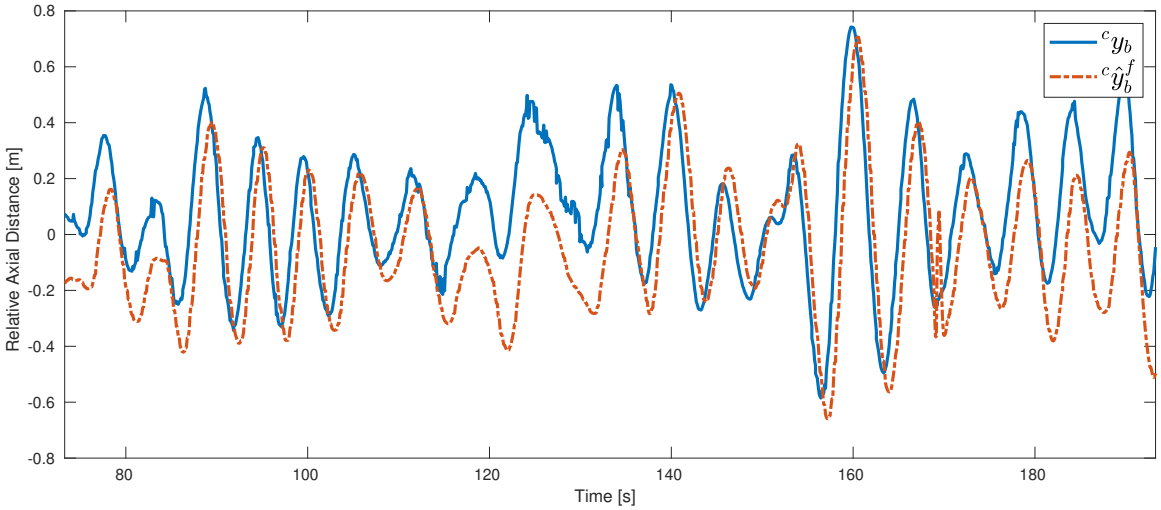


(a) Pose Estimation

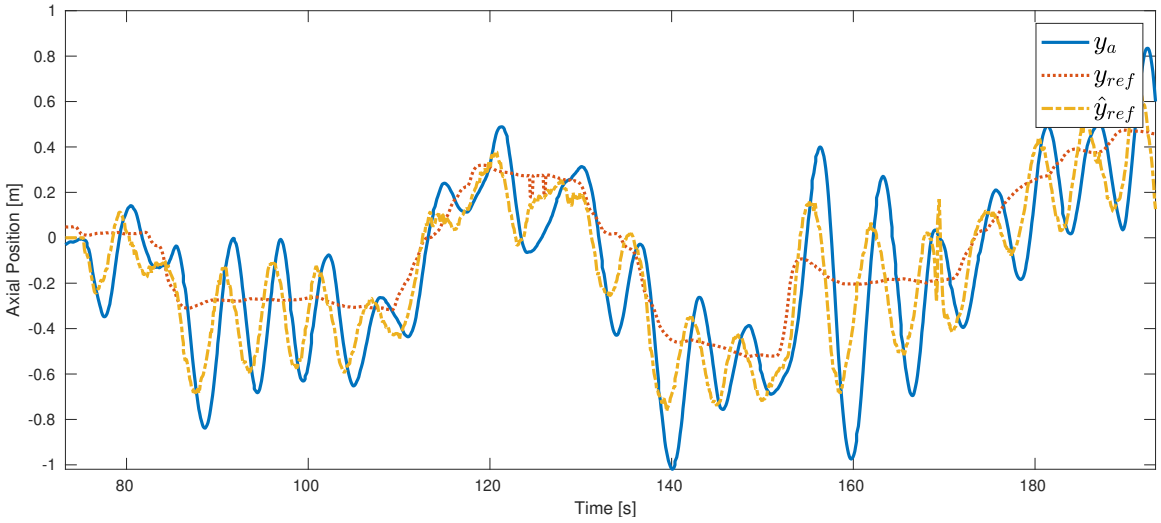


(b) Pursuit and Tracking

Figure 5.28: X-axis PnP-based Pursuit Assessment

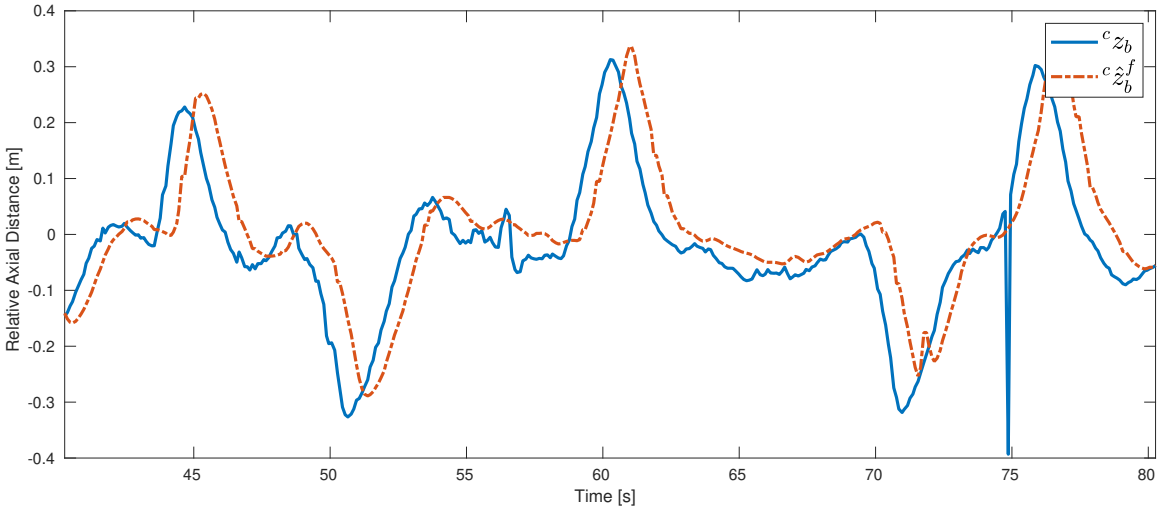


(a) Pose Estimation

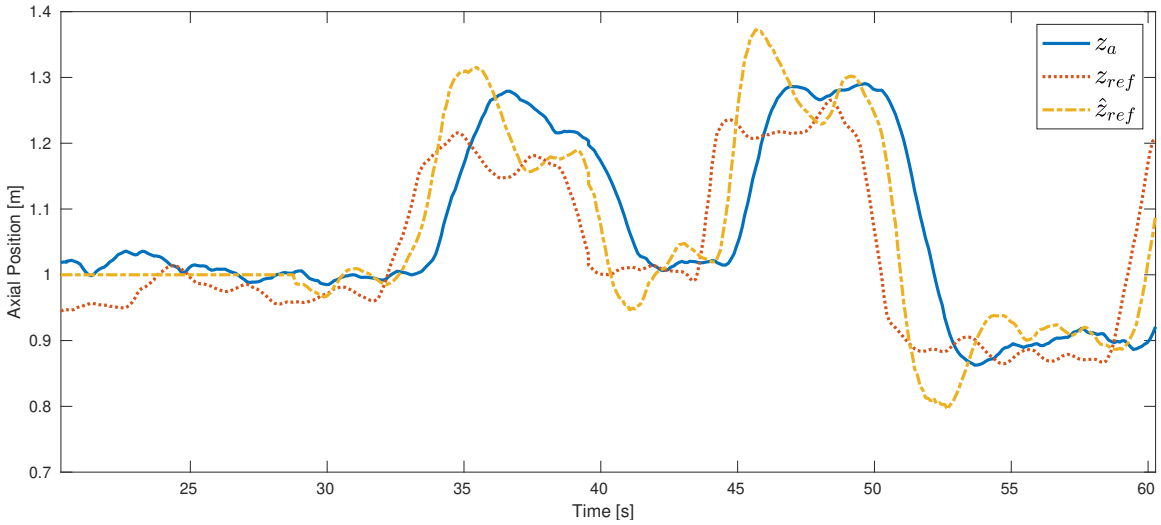


(b) Pursuit and Tracking

Figure 5.29: Y-axis PnP-based Pursuit Assessment



(a) Pose Estimation



(b) Pursuit and Tracking

Figure 5.30: Z-axis PnP-based Pursuit Assessment

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This study demonstrates the feasible implementation of real-time autonomous pursuit algorithms based on utilizing state-of-the-art networks and architectures. This implementation is based on thorough investigations and designs on control theory on one side and computer vision principals on the other hand. In accordance with this, the mathematics of 3D bounding box were investigated from scratch and utilized in ROS for pose estimation and pursuit.

All data acquisition, flight control, deep learning and PnP algorithms were implemented into two developed ROS packages for simulated drone and the real one as `sphinx_ros` and `anafi_ros`. Development of the algorithms into ROS packages creates a framework for knowledge transfer and future improvements and enhancements on the codes and algorithms.

Design and implementation were then followed by through and comprehensive assessments and validations on keypoint detection, bounding box estimation, flight controller, offline and real-time mobile pose estimation in addition to Vicon based and PnP-based target pursuit. The results proved robust keypoint detection of the RCNN, precise mobile pose estimation of the target drone and vigorous performance of the controller and pursuit algorithms.

6.2 Future Directions

Despite the novel and promising results of this thesis, there are some certain features which could be improved and enhanced for further research into the area of autonomous flights and pursuits.

We used Parrot Sphinx for simulation purposes of Anafi along with the flight controllers. However, Sphinx could be also used for testing and evaluating deep learning and pose estimation algorithms. The challenge with the current Sphinx version is its incapability of populating new simulated drone. In other words, only one drone could be populated at the same time. Thus, developing an algorithm to populate more drones at the same time would create a framework for testing pose estimation and pursuit algorithms in a simulated environment which would result in a decrease in the number of crashes and malfunctions.

As it was mentioned in Chapter 5, the pose estimation delay leads to instability and more oscillatory repose and error in tracking. This processing delay could be overcome by utilizing on-board processing units like NVIDIA Jetson which is an advanced AI embedded system. Incorporating Jetson would eliminate the transmission delay between the drone and the off-board GPU, leading to the attenuation of delay.

Another way to eliminate the delay it to substitute the Olympe library with MAVROS MAVLink. As it was mentioned earlier, Olympe is a Python library for communicating with the drone which was incorporated into ROS for this research. However, Python is less faster than C++ due to the static typing discipline of C++ compared to the dynamic one of Python. Therefore, translating Olympe flight protocols into MAVLink C++ commands would relatively speed up the communications.

Bibliography

[1] G. J. McCall, "The vendian (ediacaran) in the geological record: Enigmas in geology's prelude to the cambrian explosion," *Earth-Science Reviews*, vol. 77, no. 1-3, pp. 1–229, 2006.

[2] *Mankind Rising - Where do Humans Come From.* YouTube, accessed on : 07-26-2022. [Online]. Available: www.youtube.com/watch?v=StqZI9pMq0U.

[3] J. O. Alan Chodos, "December 1958: Invention of the laser," *American Physical Society*, [Online]. Available: https://www.aps.org/publications/apsnews/200312/history.cfm.

[4] R. Collis, "Lidar," *Applied optics*, vol. 9, no. 8, pp. 1782–1788, 1970.

[5] W. Zhen, Y. Hu, J. Liu, and S. Scherer, "A joint optimization approach of lidar-camera fusion for accurate dense 3-d reconstructions," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3585–3592, 2019.

[6] *Tesla Autonomy Day.* YouTube, 2019, accessed on : 07-26-2022. [Online]. Available: https://www.youtube.com/watch?v=Ucp0TTmvqOE.

[7] C. M. Bishop, "Neural networks and their applications," *Review of scientific instruments*, vol. 65, no. 6, pp. 1803–1832, 1994.

[8] A. Lenail, "Publication-ready nn-architecture schematics," [Online]. Available: http://alexlenail.me/NN-SVG/LeNet.html.

[9] B. Krose and P. v. d. Smagt, *An introduction to neural networks.* 2011.

[10] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[12] J. Fayyad, M. A. Jaradat, D. Gruyer, and H. Najjaran, "Deep learning sensor fusion for autonomous vehicle perception and localization: A review," *Sensors*, vol. 20, no. 15, p. 4220, 2020.

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[14] K. S. Chahal and K. Dey, "A survey of modern object detection literature using deep learning," *arXiv preprint arXiv:1808.07256*, 2018.

[15] C. Wang *et al.*, "Densefusion: 6d object pose estimation by iterative dense fusion," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 3343–3352.

[16] V. Srivastav, A. Gangi, and N. Padoy, "Self-supervision on unlabelled or data for multi-person 2d/3d human pose estimation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2020, pp. 761–771.

[17] Y.-C. Lai and Z.-Y. Huang, "Detection of a moving uav based on deep learning-based distance estimation," *Remote Sensing*, vol. 12, no. 18, p. 3035, 2020.

[18] R. Jin, J. Jiang, Y. Qi, D. Lin, and T. Song, "Drone detection and pose estimation using relational graph networks," *Sensors*, vol. 19, no. 6, p. 1479, 2019.

[19] A. Yavariabdi, H. Kusetogullari, T. Celik, and H. Cicek, "Fastuav-net: A multi-uav detection algorithm for embedded platforms," *Electronics*, vol. 10, no. 6, p. 724, 2021.

[20] M. Rezaei, M. Azarmi, and F. M. P. Mir, "Traffic-net: 3d traffic monitoring using a single camera," *arXiv preprint arXiv:2109.09165*, 2021.

[21] R. C. Dorf and R. H. Bishop, *Modern control systems*. Pearson Prentice Hall, 2008.

[22] S. Bennett, *A history of control engineering, 1930-1955*. IET, 1993.

[23] D. Ibrahim, *Microcontroller based applied digital control*. John Wiley & Sons, 2006.

[24] A. Reizenstein, *Position and trajectory control of a quadcopter using pid and lq controllers*, 2017.

[25] C. Surma and M. Barczyk, "Linear model predictive control for vision-based uav pursuit," *Journal of Unmanned Vehicle Systems*, vol. 8, no. 4, pp. 334–363, 2020.

[26] C. L. Phillips, H. T. Nagle, and A. Chakrabortty, *Digital control system analysis and design*. Prentice Hall Englewood Cliffs, NJ, 1990, vol. 2.

[27] K. J. Åström, T. Hägglund, and K. J. Astrom, *Advanced PID control*. ISA-The Instrumentation, Systems, and Automation Society Research Triangle Park, 2006, vol. 461.

[28] [Online]. Available: https://fr.linkedin.com/company/parrot.

[29] *Anafi White Paper v1.4*. Parrot®, accessed on : 09-01-2022. [Online]. Available: https://www.parrot.com/assets/s3fs-public/2020-07/white-paper_anafi-v1.4-en.pdf.

[30] *What's the Difference Between a CPU and a GPU?* Nvidia. [Online]. Available: https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/.

[31] J. Bouvard, N. Dessart, N. BRÛLEZ, and Y.-M. Morgan, *Olympe*, https://github.com/Parrot-Developers/olympe, 2022.

[32] *Olympe User Guide*. Parrot®, accessed on : 09-06-2022. [Online]. Available: https://developer.parrot.com/docs/olympe/userguide.html.

[33] *ARSDK Protocols*. Parrot® for Developers, accessed on : 09-05-2022. [Online]. Available: https://developer.parrot.com/docs/bebop/ARSDK_Protocols.pdf.

[34] *What is Parrot Sphinx*. Parrot®, accessed on : 09-07-2022. [Online]. Available: https://developer.parrot.com/docs/sphinx/index.html.

[35] *CUDA Zone*. Nvidia Developers, accessed on : 09-07-2022. [Online]. Available: https://developer.nvidia.com/cuda-zone.

[36] *PyTorch*. Meta AI, accessed on : 09-07-2022. [Online]. Available: https://github.com/pytorch/pytorch.

[37] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He, *Detectron*, https://github.com/facebookresearch/detectron, 2018.

[38] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, *Detectron2*, https://github.com/facebookresearch/detectron2, 2019.

[39] *Vicon Tracker User Guide*. Vicon. [Online]. Available: https://docs.vicon.com/download/attachments/133827160/Vicon%20Tracker%20User%20Guide.pdf?version=1&modificationDate=1601360389000&api=v2.

[40] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[41] A. Koubâa *et al.*, *Robot Operating System (ROS)*. Springer, 2017, vol. 1.

[42] *ROS Distributions*. ROS.org, accessed on : 09-09-2022. [Online]. Available: http://wiki.ros.org/Distributions.

[43] *ROS Qt Plugins*. ROS.org, accessed on : 09-10-2022. [Online]. Available: https://wiki.ros.org/rqt_common_plugins?distro=noetic.

[44] T. Haigh, "Cleve moler: Mathematical software pioneer and creator of matlab," *IEEE Annals of the History of Computing*, vol. 30, no. 1, pp. 87–91, 2008.

[45] *MATLAB®*. MathWorks®, accessed on : 09-08-2022. [Online]. Available: https://www.mathworks.com/products/matlab.htm.

[46] M. Barczyk, "Nonlinear state estimation and modeling of a helicopter uav," 2012.

[47] R. Priemer, *Introductory signal processing*. World Scientific, 1991, vol. 6.

[48] A. V. Oppenheim, J. Buck, M. Daniel, A. S. Willsky, S. H. Nawab, and A. Singer, *Signals & systems*. Pearson Educación, 1997.

[49] J. G. Proakis, *Digital signal processing: principles algorithms and applications*. Pearson Education India, 2001.

[50] J. B. J. Fourier, G. Darboux, *et al.*, *Théorie analytique de la chaleur*. Didot Paris, 1822, vol. 504.

[51] A. V. Oppenheim, "Applications of digital signal processing," *Englewood Cliffs*, 1978.

[52] A. V. Oppenheim, J. R. Buck, and R. W. Schafer, *Discrete-time signal processing. Vol. 2*. Upper Saddle River, NJ: Prentice Hall, 2001.

[53] S. Haykin and B. Van Veen, *Signals and systems*. John Wiley & Sons, 2007.

[54] K. Ogata, *Discrete-time control systems*. Prentice-Hall, Inc., 1995.

[55] C. R. Phillips and N. Nagle, "Digital control system analysis and design," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 452–453, 1985.

[56] *Fast Fourier transform*. MathWorks®, 2022. [Online]. Available: https://www.mathworks.com/help/matlab/ref/fft.html?searchHighlight=fft&s_tid=srchtitle_fft_1.

[57] *Butterworth filter design*. MathWorks®, 2022. [Online]. Available: https://www.mathworks.com/help/signal/ref/butter.html?searchHighlight=butter&s_tid=srchtitle_butter_1.

[58] O. Nelles, "Nonlinear dynamic system identification," in *Nonlinear System Identification*, Springer, 2001, pp. 547–577.

[59] D. Karnopp, D. L. Margolis, and R. C. Rosenberg, *System dynamics*. Wiley New York, 1990.

[60] *Solve system of linear equations*. MathWorks®, 2022. [Online]. Available: https://www.mathworks.com/help/matlab/ref/lsqr.html?searchHighlight=lsqr&s_tid=srchtitle_lsqr_1.

[61] *State-space model*. MathWorks®, 2022. [Online]. Available: https://www.mathworks.com/help/control/ref/ss.html.

[62] *Simulated response data*. MathWorks®, 2022. [Online]. Available: https://www.mathworks.com/help/control/ref/lti.lsim.html?searchHighlight=lsim&s_tid=srchtitle_lsim_1.

[63] S. Bennett, "A brief history of automatic control," *IEEE Control Systems Magazine*, vol. 16, no. 3, pp. 17–25, 1996.

[64] F. Golnaraghi and B. C. Kuo, *Automatic control systems*. McGraw-Hill Education, 2017.

[65] W. Tang, Q.-G. Wang, Z. Ye, and Z. Zhang, "Pid tuning for dominant poles and phase margin," *Asian Journal of Control*, vol. 9, no. 4, pp. 466–469, 2007.

[66] C. Surma, "Uav linear model predictive control using computer vision algorithms," 2019.

[67] *Vicon Bridge*. ROS Wiki, 2015. [Online]. Available: http://wiki.ros.org/vicon_bridge.

[68] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[69] S. Nayar, "Itroduction to Computer Vision," in *Monograph FPCV-0-1, First Principles of Computer Vision*, Columbia University, New York, Feb. 2022.

[70] S. Nayar, "Image Sensing," in *Monograph FPCV-1-2, First Principles of Computer Vision*, Columbia University, New York, Feb. 2022.

[71] *Linear Camera Model — Camera Calibration*. YouTube, 2021, accessed on : 12-18-2022. [Online]. Available: https : / / www . youtube . com / watch ? v = qByYk6JggQU&t=838s.

[72] S. Nayar, "Image Processing ii," in *Monograph FPCV-1-5, First Principles of Computer Vision*, Columbia University, New York, Mar. 2022.

[73] *Labelme*, https://github.com/wkentaro/labelme, 2020.

[74] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[75] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, Springer, 2015, pp. 234–241.

[76] *PyTorch Keypoint RCNN*. PyTorch, accessed on : 02-21-2023. [Online]. Available: https://pytorch.org/vision/main/models/generated/torchvision.models.detection.keypointrcnn_resnet50_fpn.html.

[77] C. Stachniss, "Projective 3-point (p3p) algorithm or spatial resection," Photogrammetry & Robotics Lab.

[78] *Projective 3-Point Algorithm using Grunert's Method (Cyrill Stachniss)*. YouTube, 2020, accessed on : 02-05-2023. [Online]. Available: https://www.youtube.com/watch?v=N1aCvzFll6Q.

[79] S. Li, C. Xu, and M. Xie, "A robust o (n) solution to the perspective-n-point problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 7, pp. 1444–1450, 2012.

[80] L. Quan and Z. Lan, "Linear n-point camera pose determination," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 21, no. 8, pp. 774–780, 1999.

[81] *Perspective-n-Point (PnP) pose computation*. Open Source Computer Vision, accessed on : 02-05-2023. [Online]. Available: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html.

[82] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Ep n p: An accurate o (n) solution to the p n p problem," *International journal of computer vision*, vol. 81, pp. 155–166, 2009.

[83] M. Zuliani, "Ransac for dummies with examples using the ransac toolbox for matlab™ & octave and more ...," 2014. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.475.1243&rep=rep1&type=pdf.

Appendix A: Sphinx Simulator Setup

Developing codes for drones in a simulated drones has numerous benefits by pretesting codes and algorithms to increase safety standards and reduce the chance of having crashes. Parrot Sphinx is a simulated tool for populating parrot drones including Anafi and Anafi Ai. Before installing Sphinx, it is required to install Olympe[1]. After installing Olympe, now we can start installing Sphinx based on the official website[2].

After installing both Olympe and Sphinx, the following commands need to be entered in separate terminal for launching Sphinx and unreal engine simultaneously.

```
sudo systemctl start firmwared.service
```

```
sphinx "/opt/parrot sphinx/usr/share/sphinx/drones/anafi_ai.drone"
::firmware="ftp://<login>:<password>@ftp2.parrot.biz/versions/anafi2/
pc/%23latest/images/anafi2-pc.ext2.zip"
```

```
parrot-ue4-empty
```

The Sphinx environment is represented in Figure A.1. This environment could be integrated with ROS for controller the drone and sending flight commands. The stream feed and pose data are also accessible and could be published on ROS via Olympe and Sphinx Python libraries.

[1]https://developer.parrot.com/docs/olympe/installation.html
[2]https://developer.parrot.com/docs/sphinx/installation.html

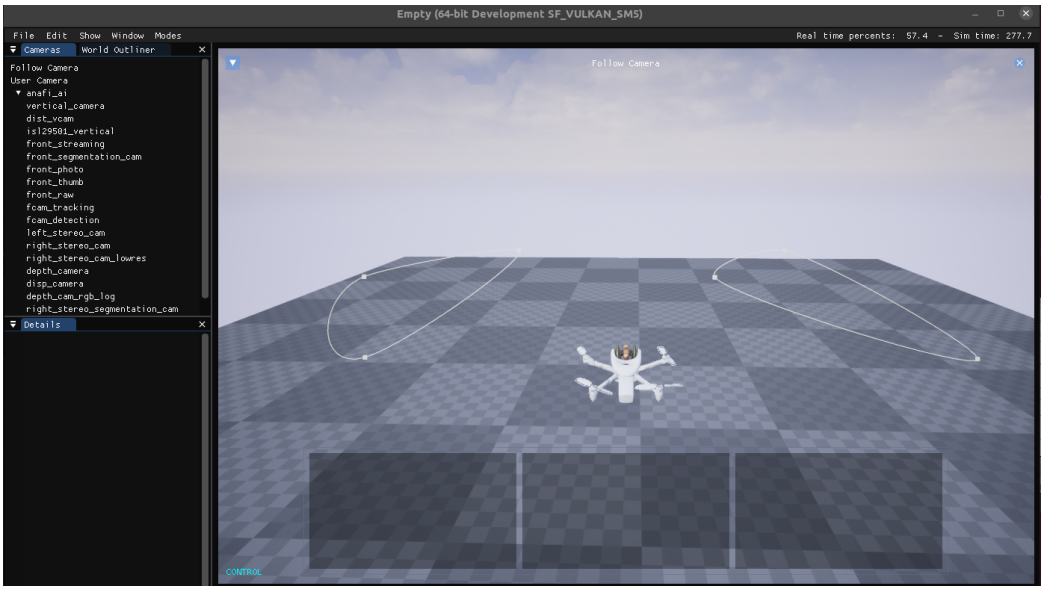


Figure A.1: Parrot Sphinx simulation environment

Sphinx also has a visual interface with Gazebo simulator and could be launched with the following command:

```
sphinx-gzclient
```

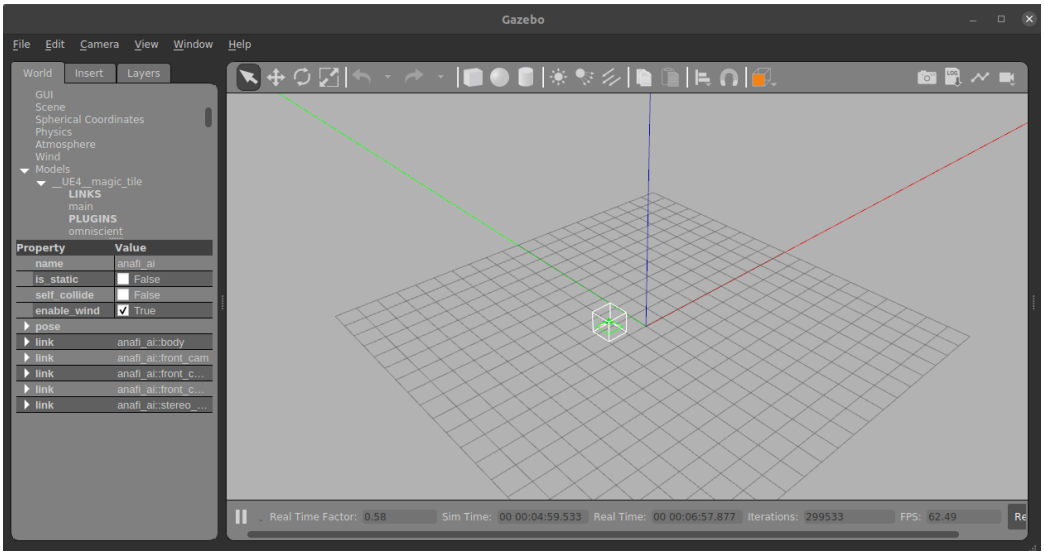


Figure A.2: Parrot Sphinx interface on Gazebo

Appendix B: Filter Design and Analysis

Table B.1: Anafi Velocity Filters' Specifications

Axis	Filter Type	Filter Order	Sampling Frequency	Cut-off Frequency
X	Low-pass	1^{st}	25 Hz	0.7 Hz
Y	Low-pass	1^{st}	25 Hz	0.5 Hz
Z	Low-pass	1^{st}	25 Hz	1.2 Hz
Ψ	Low-pass	1^{st}	25 Hz	1.2 Hz

Table B.2: Bebop Velocity Filters' Specifications

Axis	Filter Type	Filter Order	Sampling Frequency	Cut-off Frequency
X	Low-pass	1^{st}	25 Hz	0.7 Hz
Y	Low-pass	1^{st}	25 Hz	0.5 Hz
Z	Low-pass	1^{st}	25 Hz	1.2 Hz
Ψ	Low-pass	1^{st}	25 Hz	1.2 Hz

Table B.3: PnP Estimated Pose Filters' Specifications

Axis	Filter Type	Filter Order	Sampling Frequency	Cut-off Frequency
X	Low-pass	1^{st}	10 Hz	2.0 Hz
Y	Low-pass	1^{st}	10 Hz	2.0 Hz
Z	Low-pass	1^{st}	10 Hz	2.5 Hz

B.1 Anafi Velocity Filtering

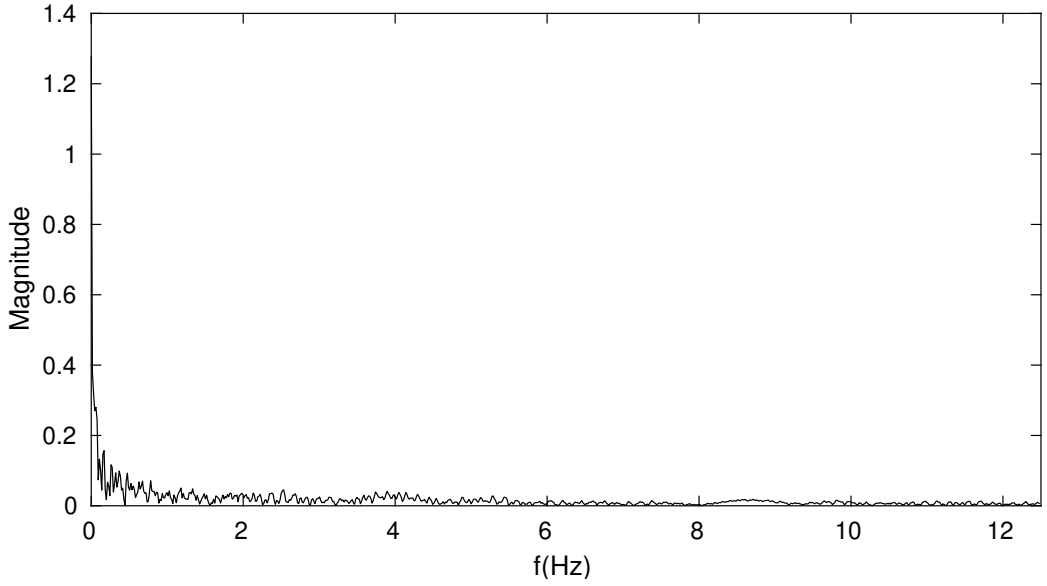


Figure B.1: Frequency Components of Estimated x_{pnp}

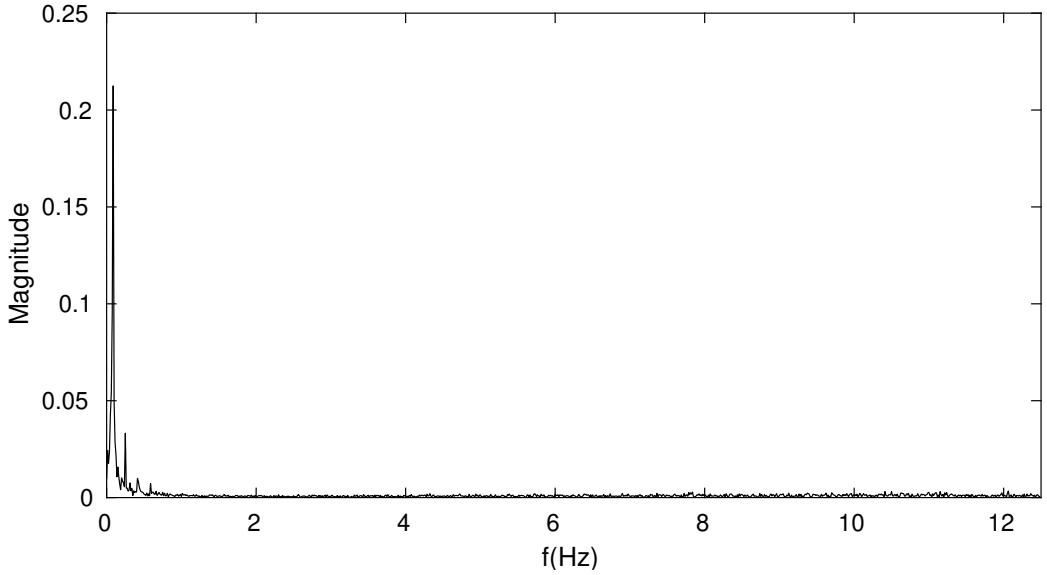


Figure B.2: Frequency Components of Anafi v_x

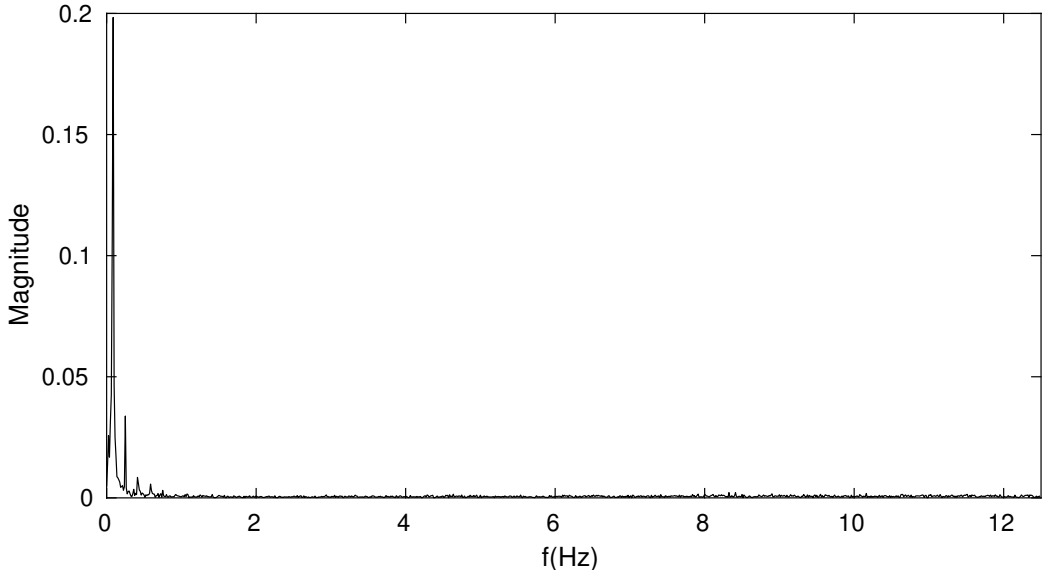


Figure B.3: Frequency Components of Anafi v_y

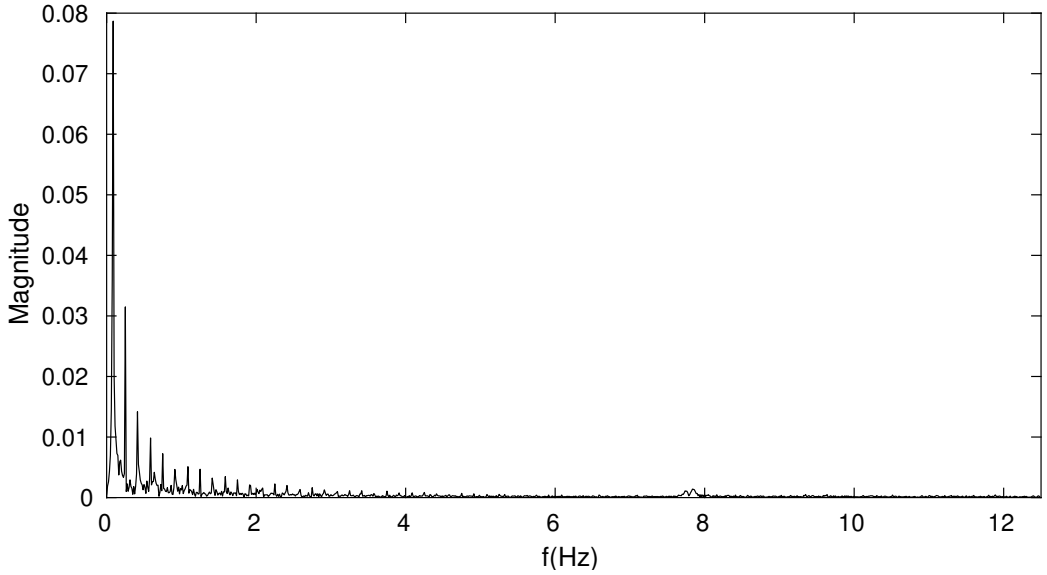


Figure B.4: Frequency Components of Anafi v_z

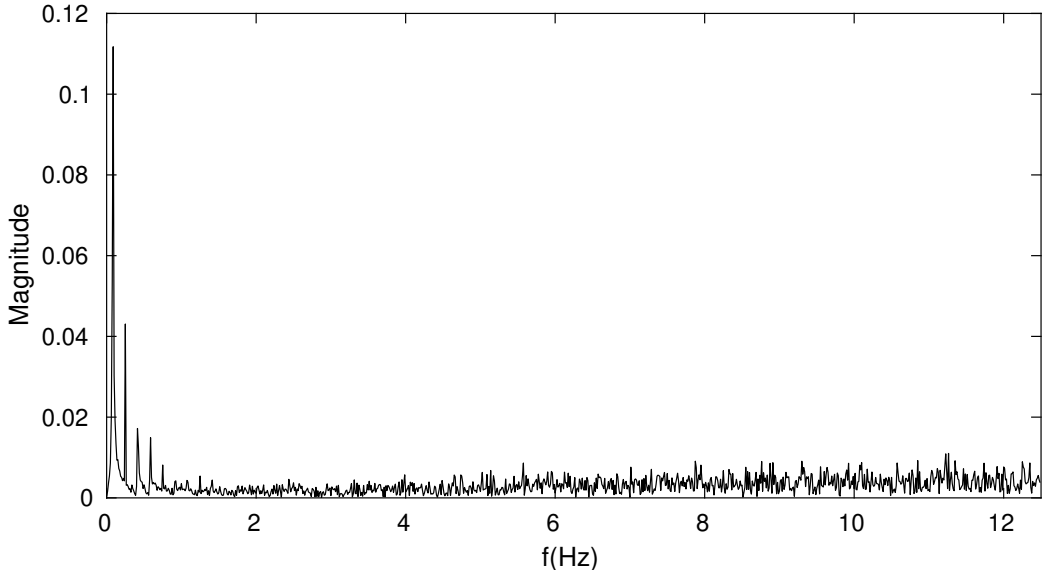


Figure B.5: Frequency Components of Anafi v_ψ

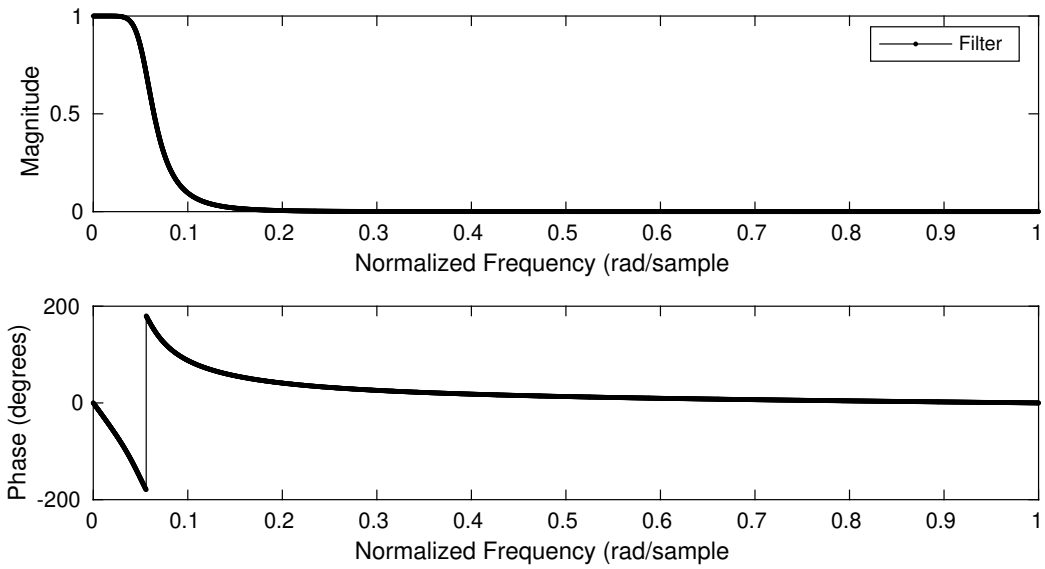


Figure B.6: Butterworth Magnitude and Phase Diagrams of Anafi x-axis

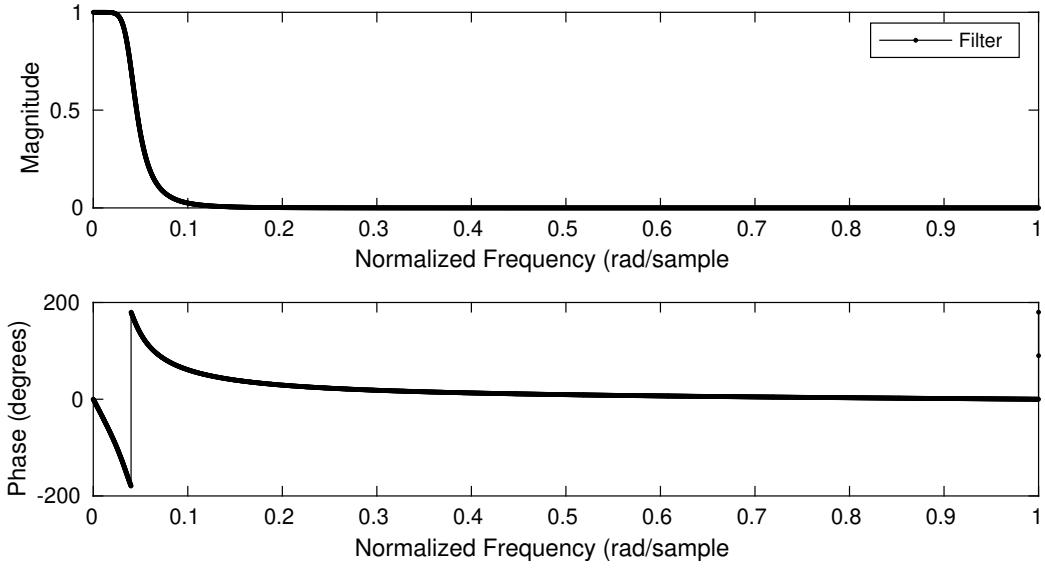


Figure B.7: Butterworth Magnitude and Phase Diagrams of Anafi y-axis

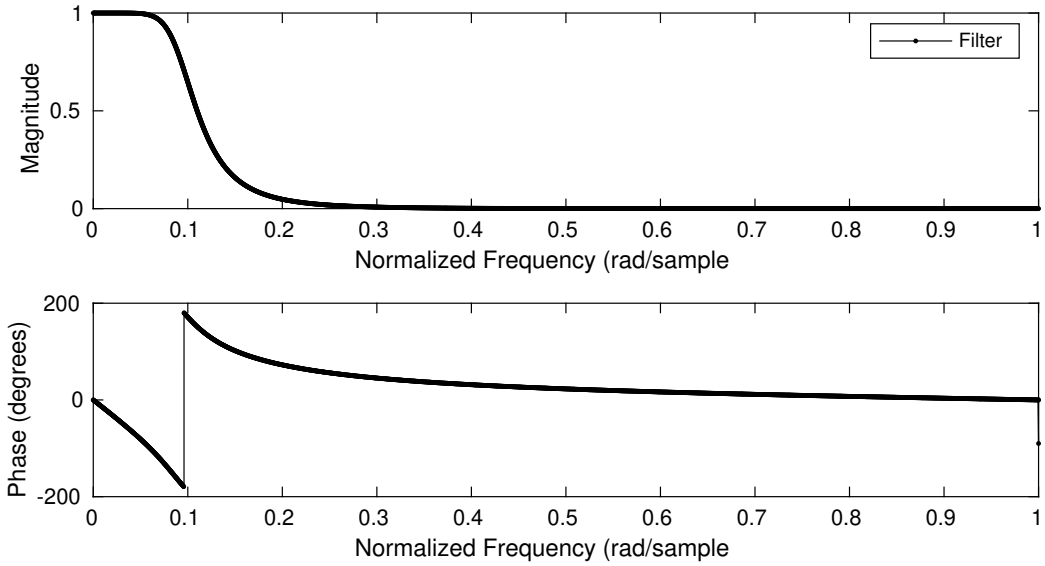


Figure B.8: Butterworth Magnitude and Phase Diagrams of Anafi z-axis

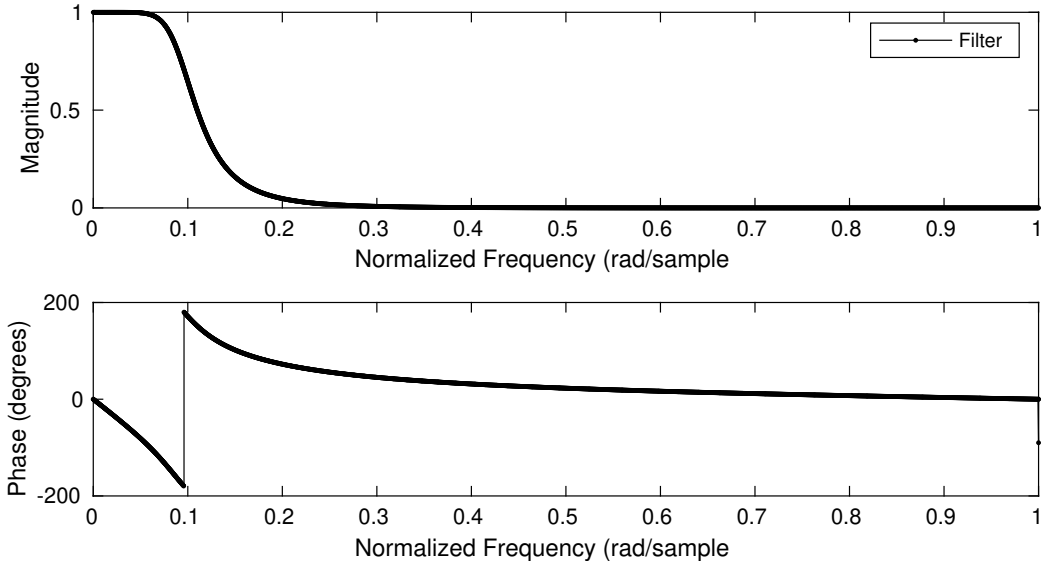


Figure B.9: Butterworth Magnitude and Phase Diagrams of Anafi ψ-axis

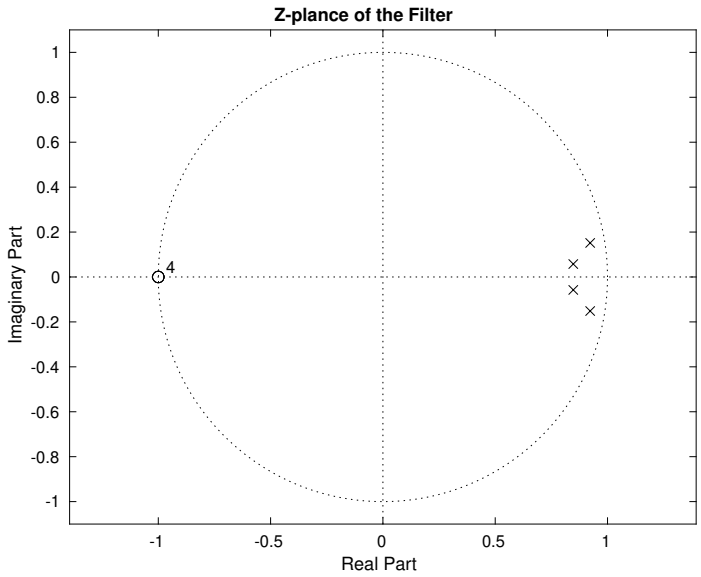


Figure B.10: Z-plane of Anafi Designed Butterworth Filter on x-axis

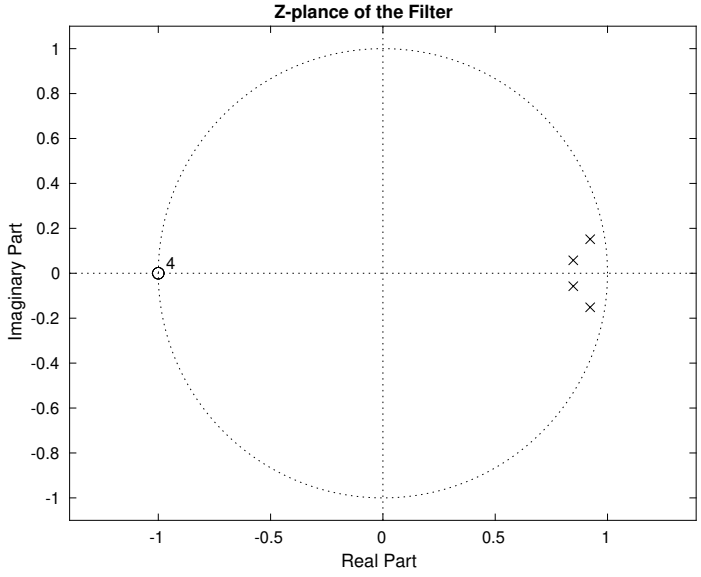


Figure B.11: Z-plane of Anafi Designed Butterworth Filter on y-axis

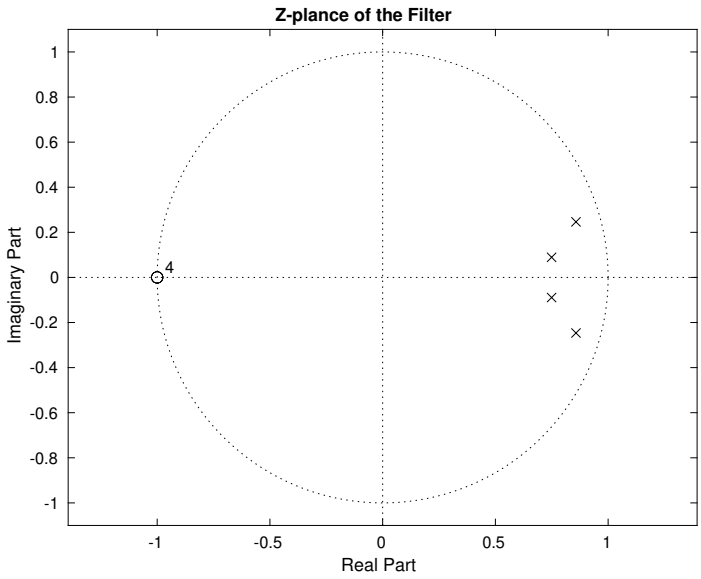


Figure B.12: Z-plane of Anafi Designed Butterworth Filter on z-axis

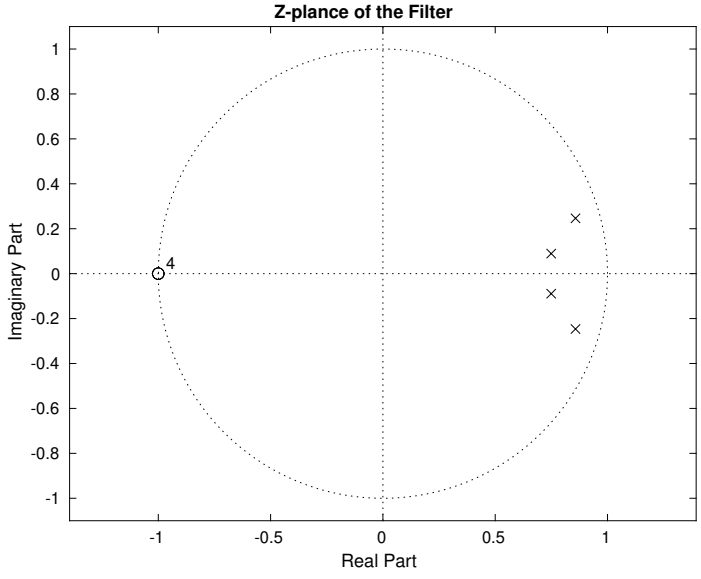


Figure B.13: Z-plane of Anafi Designed Butterworth Filter on ψ-axis

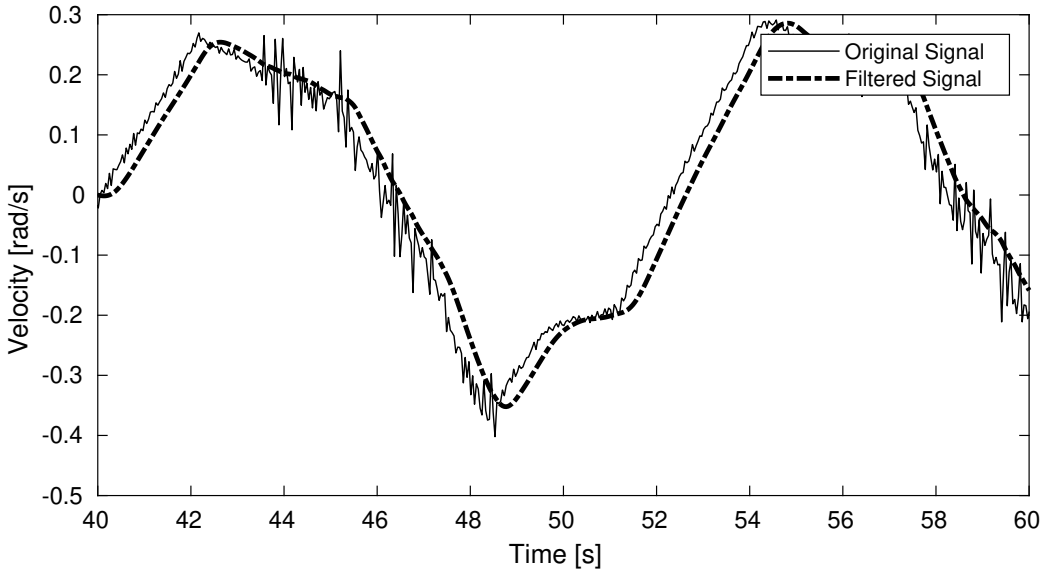


Figure B.14: Original and Filtered Signal of Anafi v_x

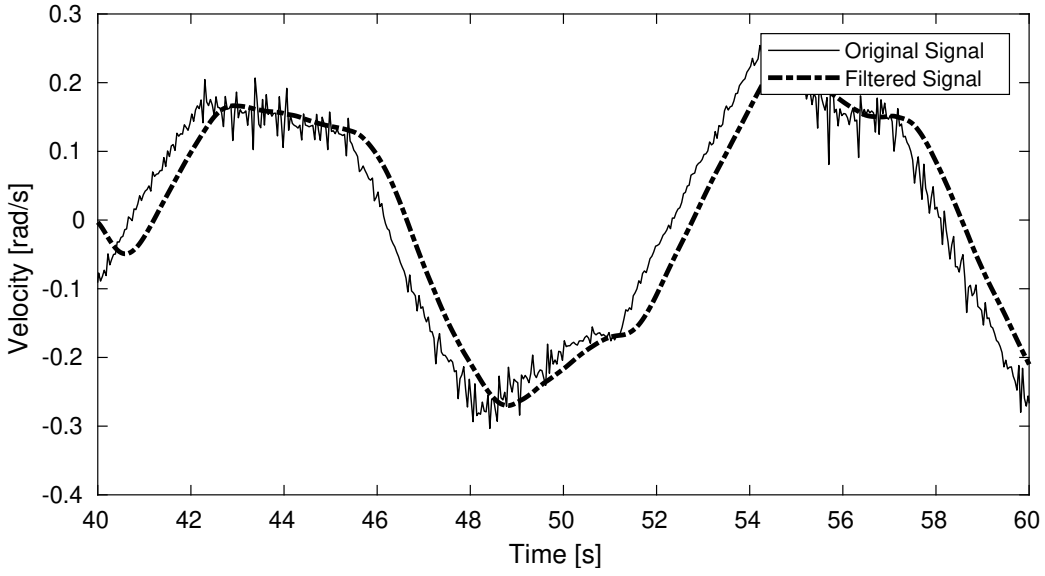


Figure B.15: Original and Filtered Signal of Anafi v_y

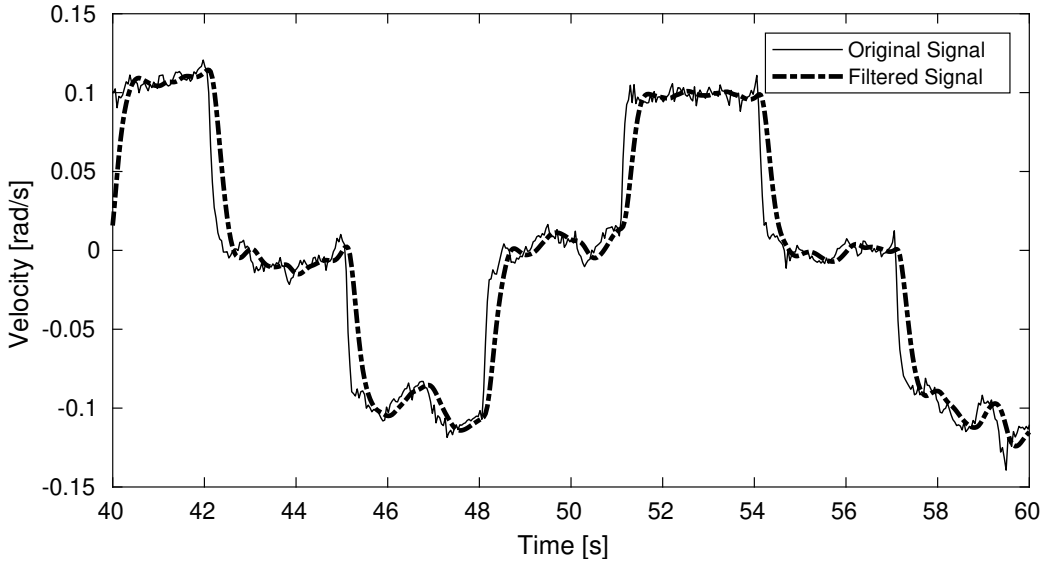


Figure B.16: Original and Filtered Signal of Anafi v_z

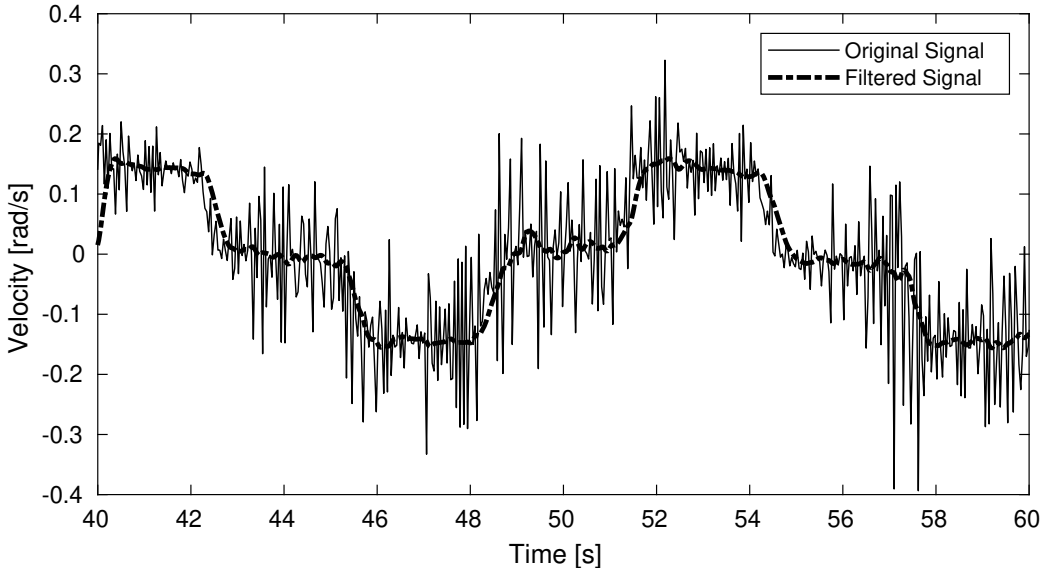


Figure B.17: Original and Filtered Signal of Anafi v_ψ

B.2 Bebop Velocity Filtering

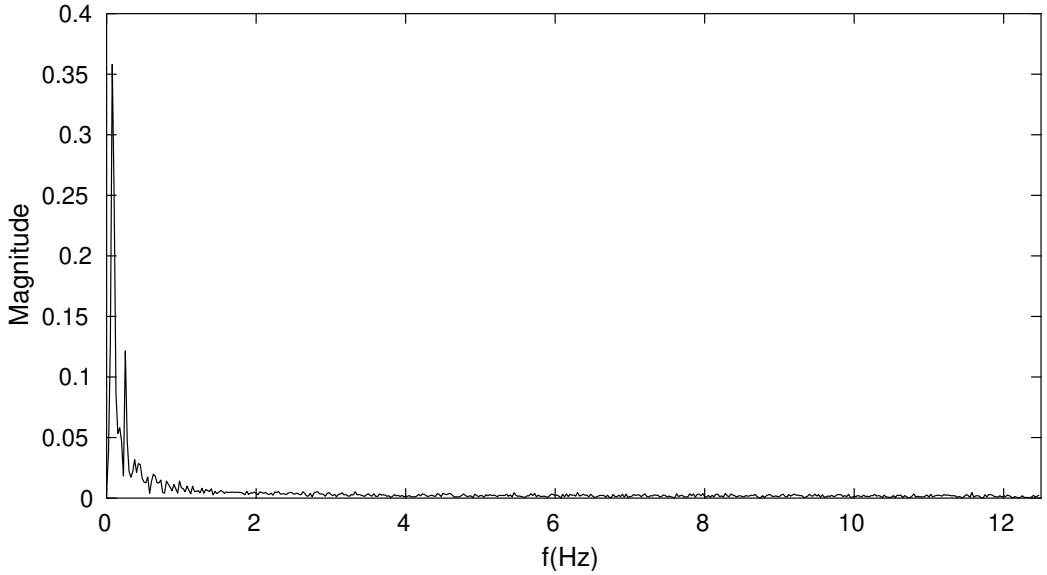


Figure B.18: Frequency Components of Bebop v_x

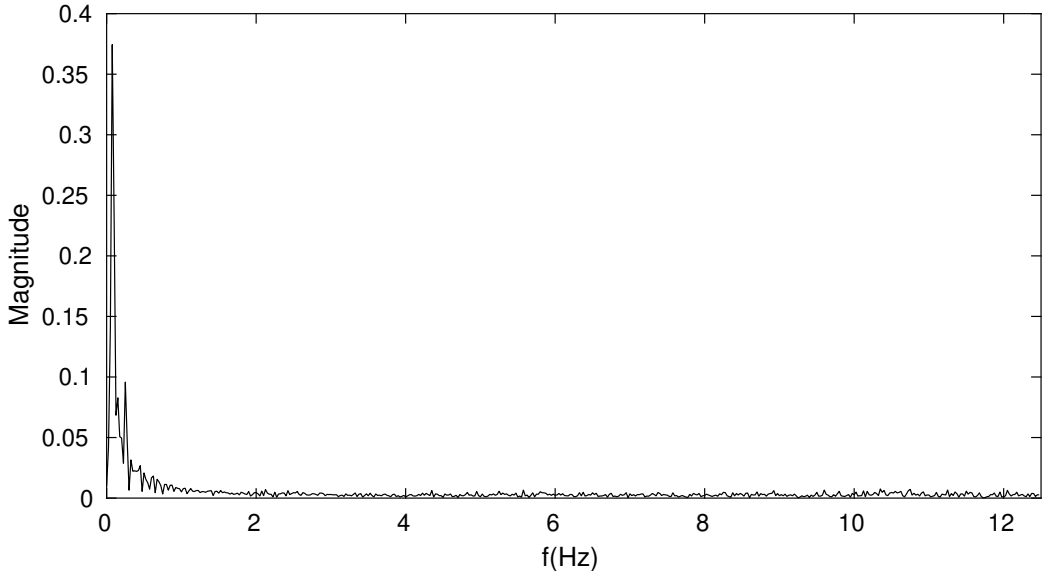


Figure B.19: Frequency Components of Bebop v_y

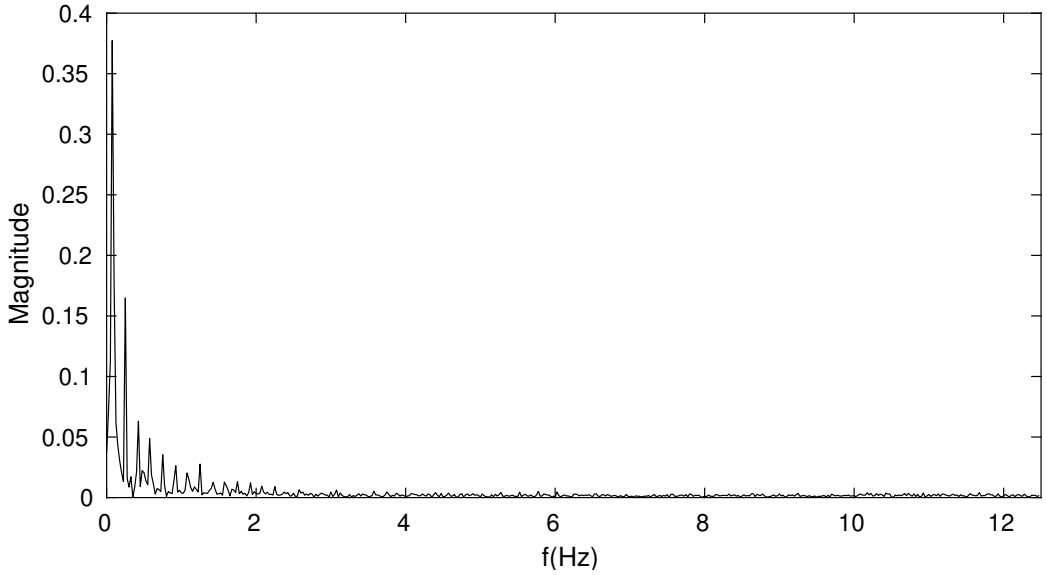


Figure B.20: Frequency Components of Bebop v_z

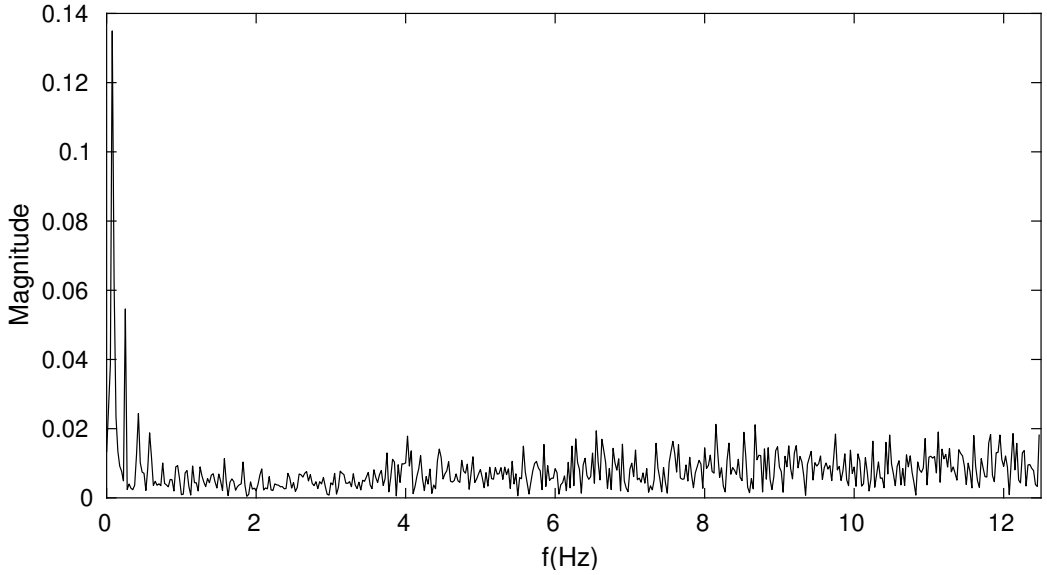


Figure B.21: Frequency Components of Bebop v_ψ

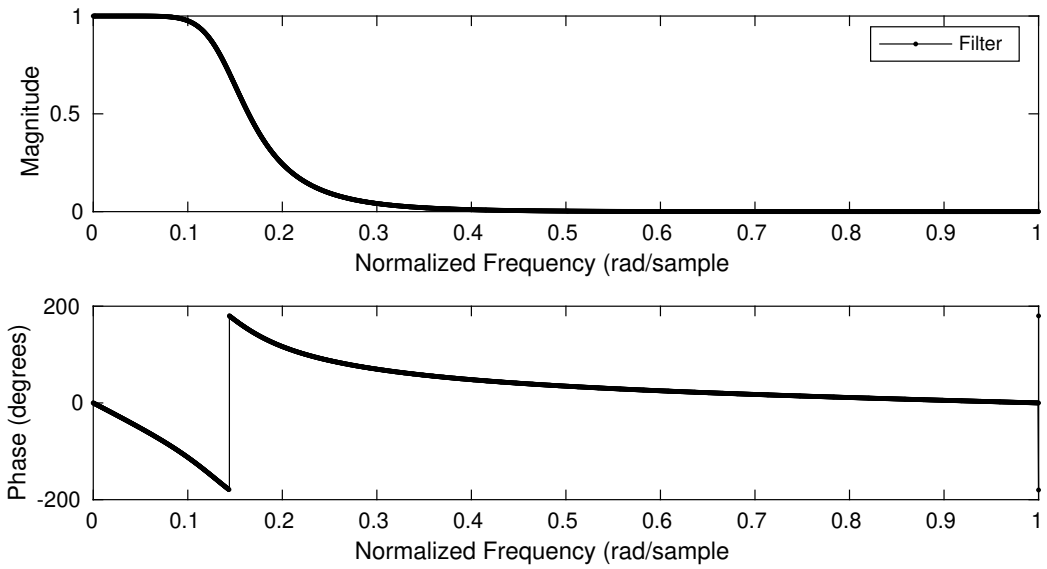


Figure B.22: Butterworth Magnitude and Phase Diagrams of Bebop x-axis

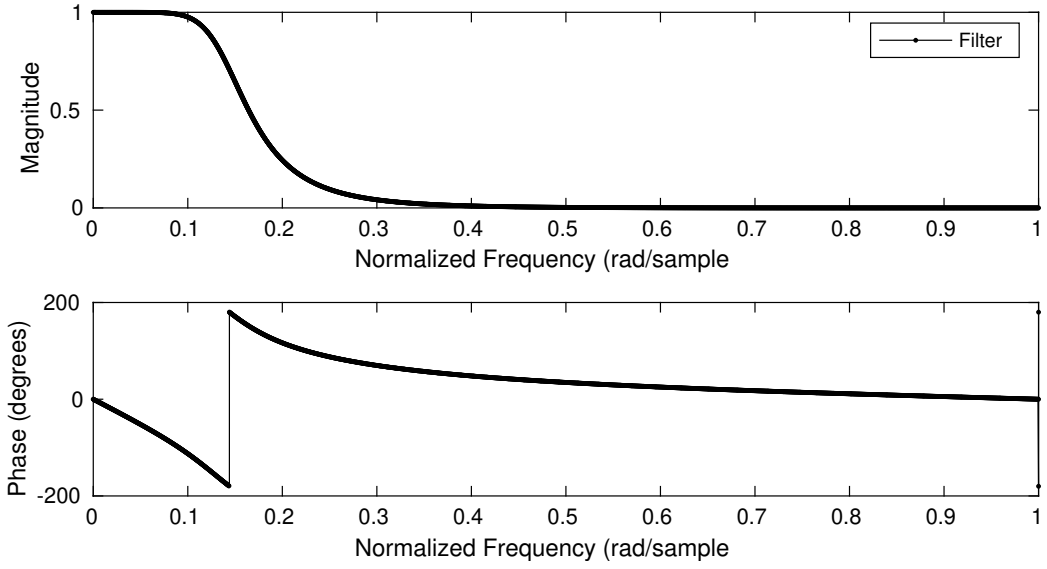


Figure B.23: Butterworth Magnitude and Phase Diagrams of Bebop y-axis

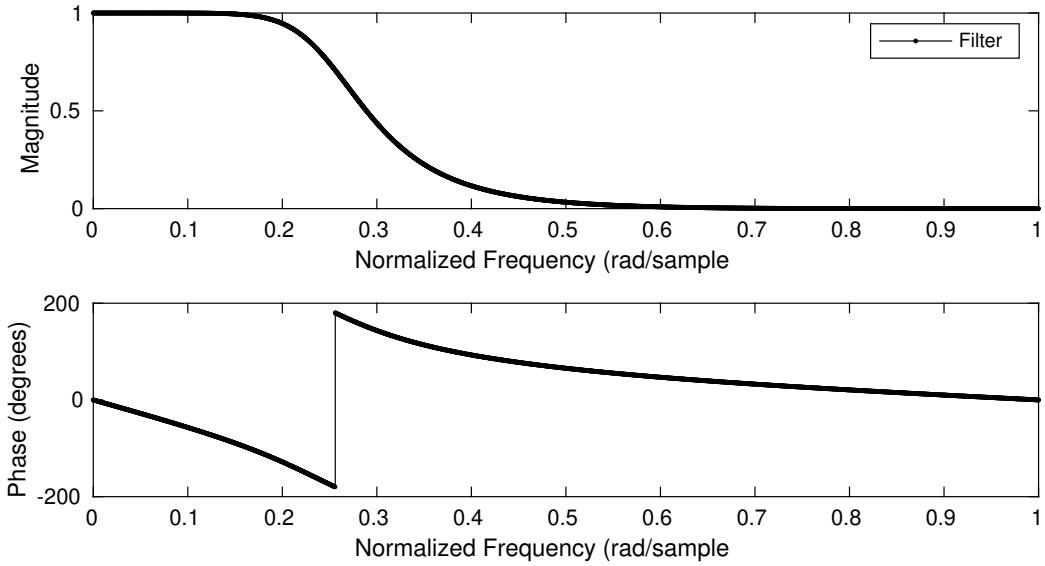


Figure B.24: Butterworth Magnitude and Phase Diagrams of Bebop z-axis

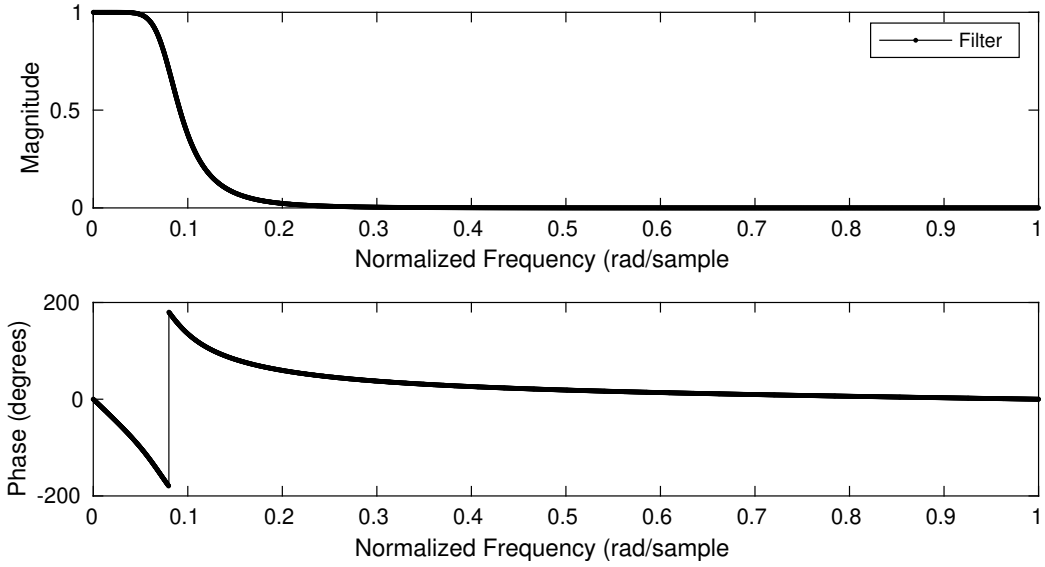


Figure B.25: Butterworth Magnitude and Phase Diagrams of Bebop ψ-axis

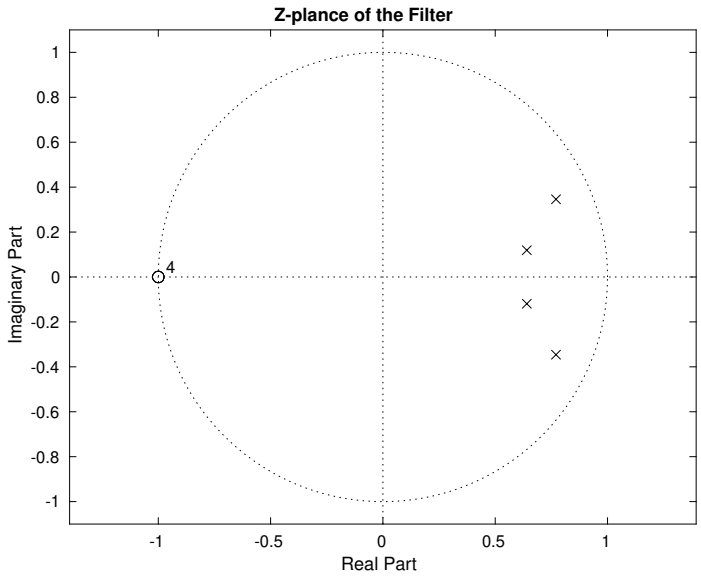


Figure B.26: Z-plane of Bebop Designed Butterworth Filter on x-axis

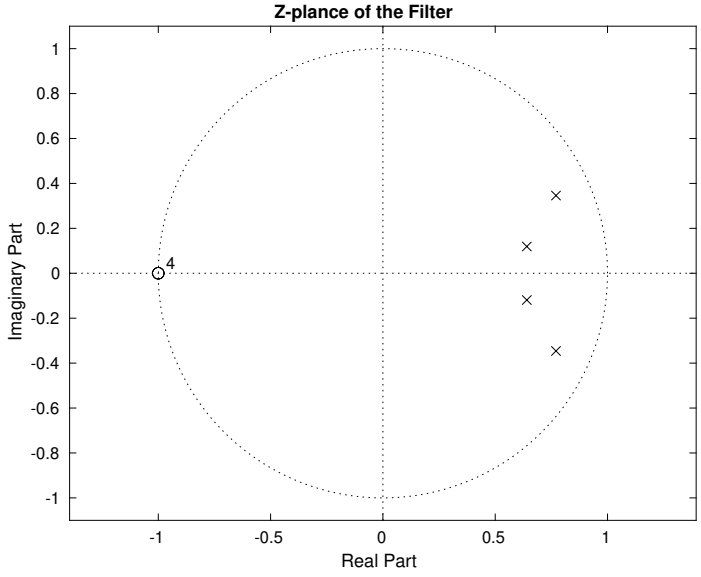


Figure B.27: Z-plane of Bebop Designed Butterworth Filter on y-axis

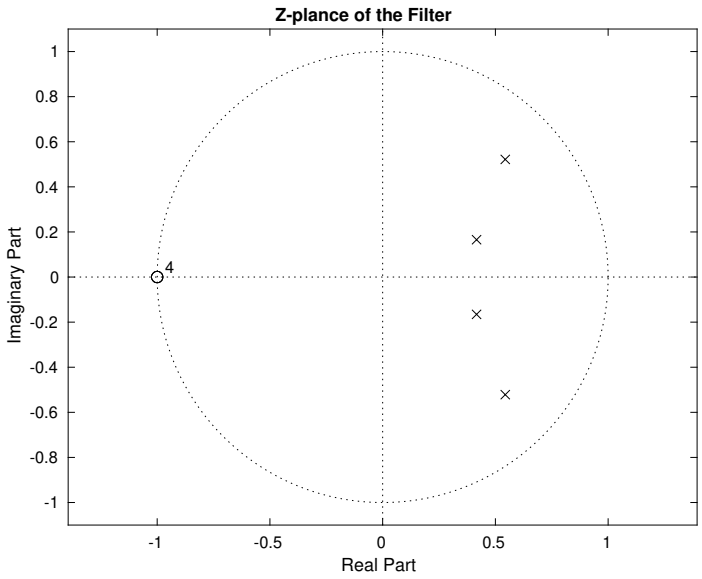


Figure B.28: Z-plane of Bebop Designed Butterworth Filter on z-axis

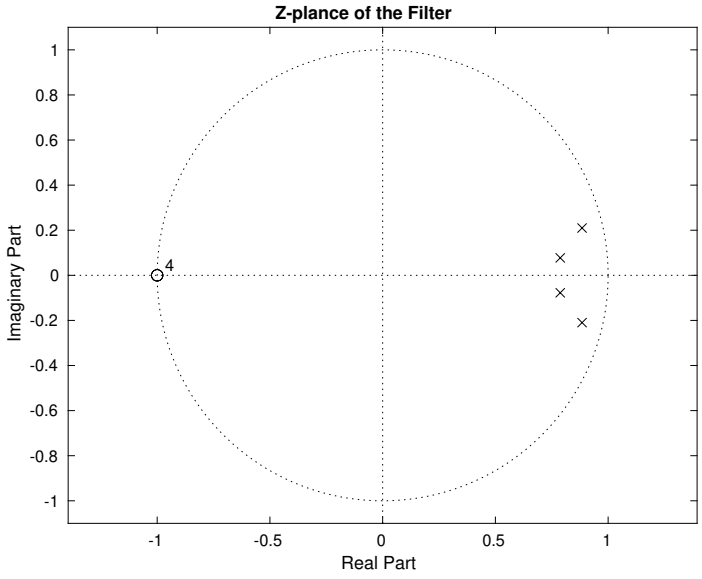


Figure B.29: Z-plane of Bebop Designed Butterworth Filter on ψ-axis

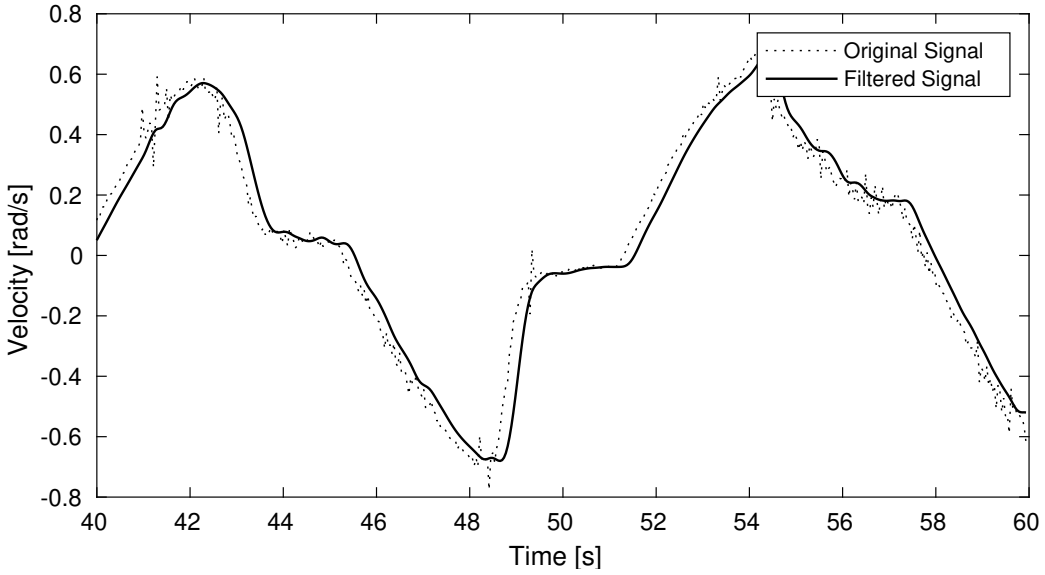


Figure B.30: Original and Filtered Signal of Bebop v_x

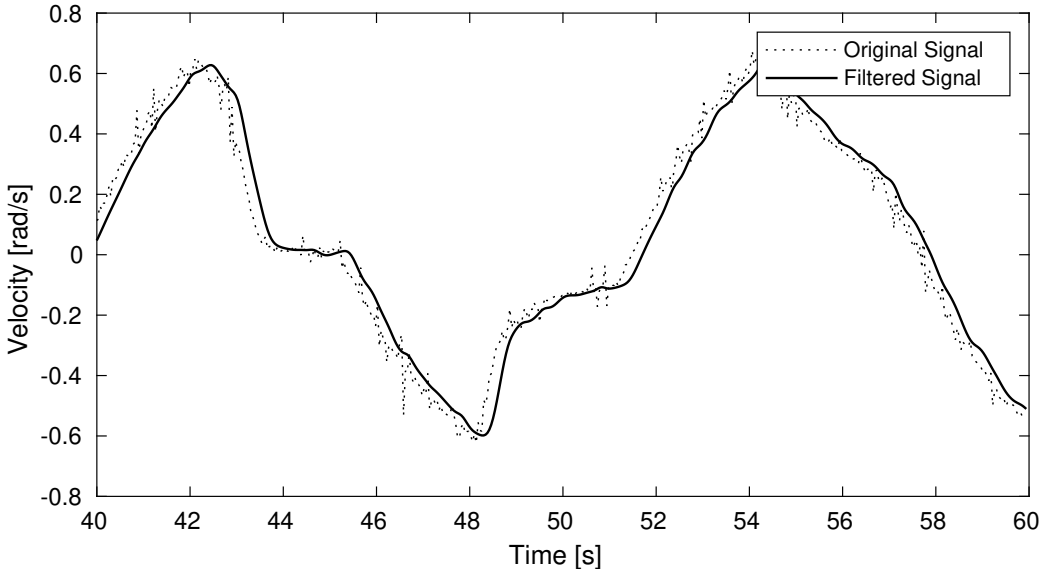


Figure B.31: Original and Filtered Signal of Bebop v_y

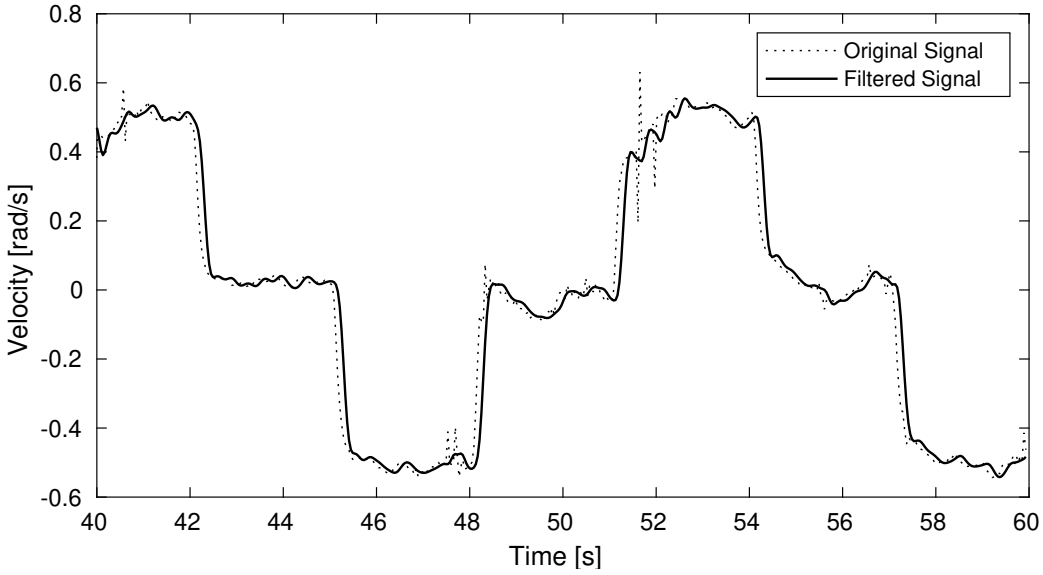


Figure B.32: Original and Filtered Signal of Bebop v_z

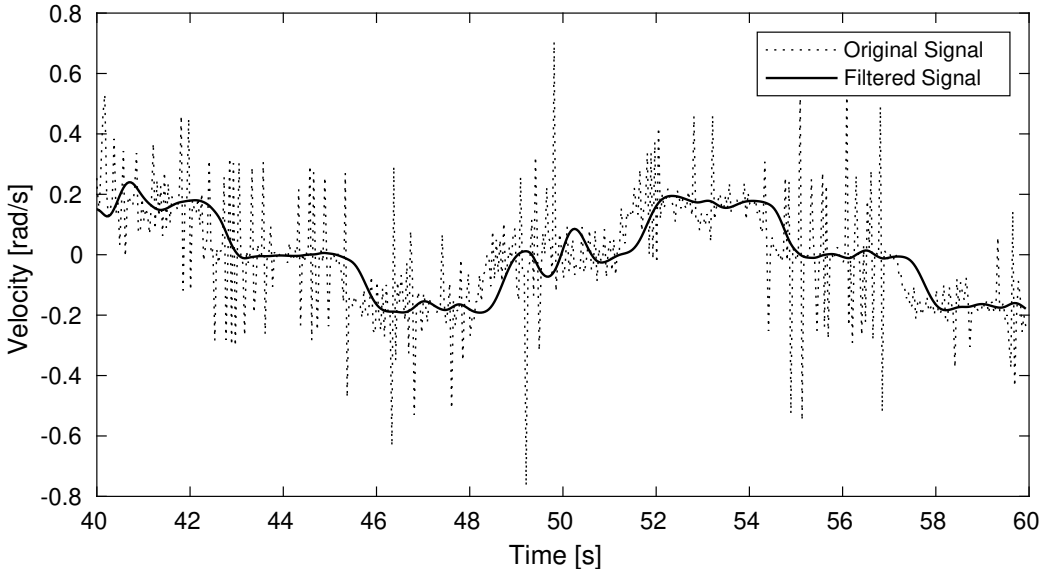


Figure B.33: Original and Filtered Signal of Bebop v_ψ

B.3 PnP Estimated Pose Filtering

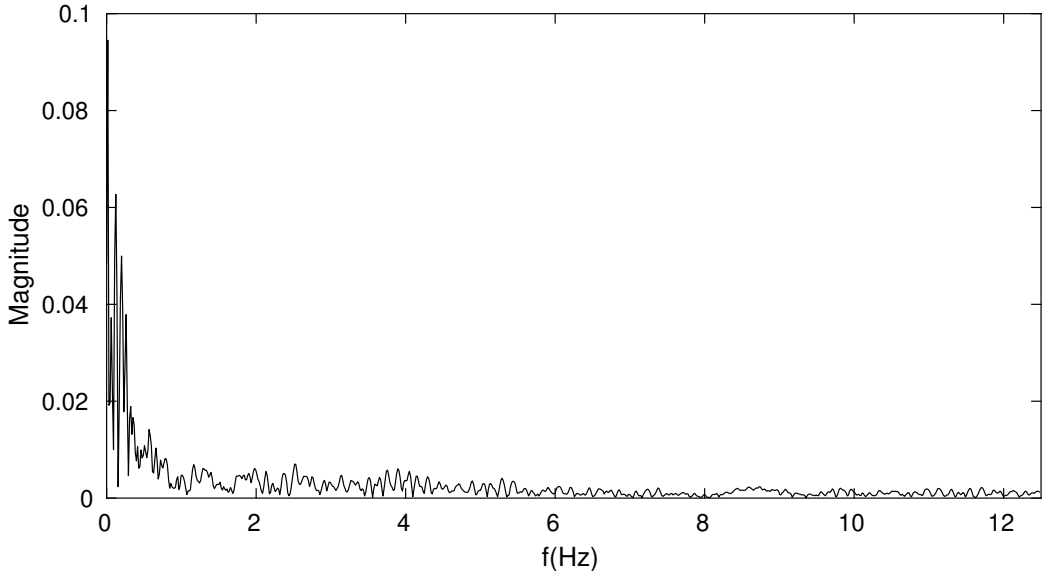


Figure B.34: Frequency Components of Estimated y_{pnp}

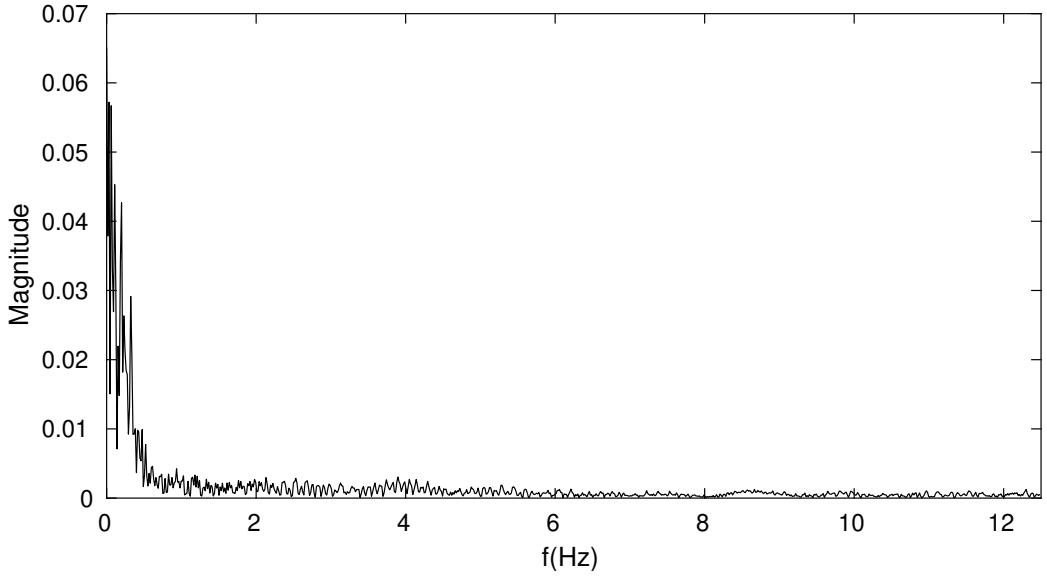


Figure B.35: Frequency Components of Estimated z_{pnp}

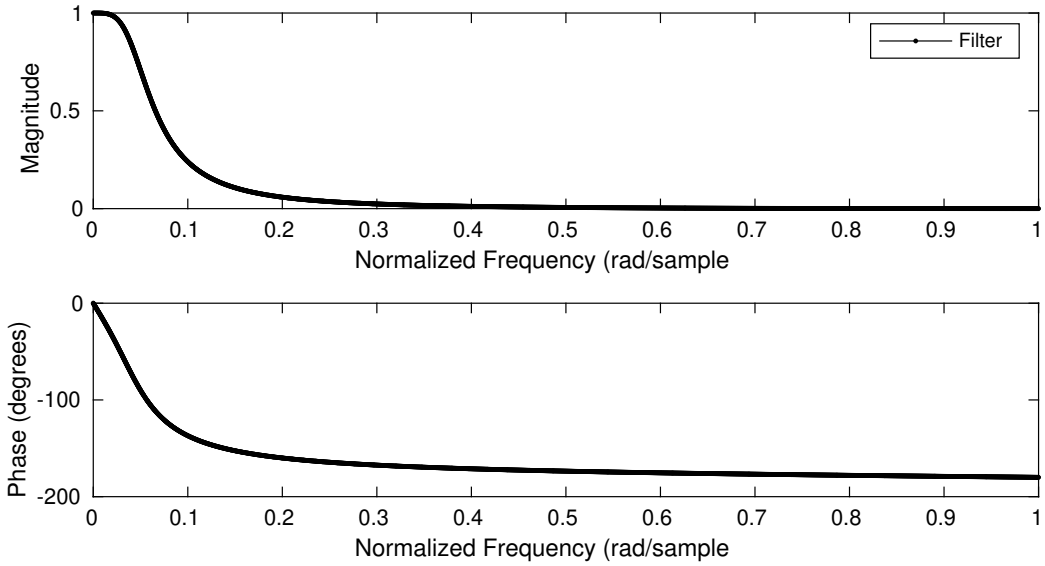


Figure B.36: Butterworth Magnitude and Phase Diagrams of x_{pnp}

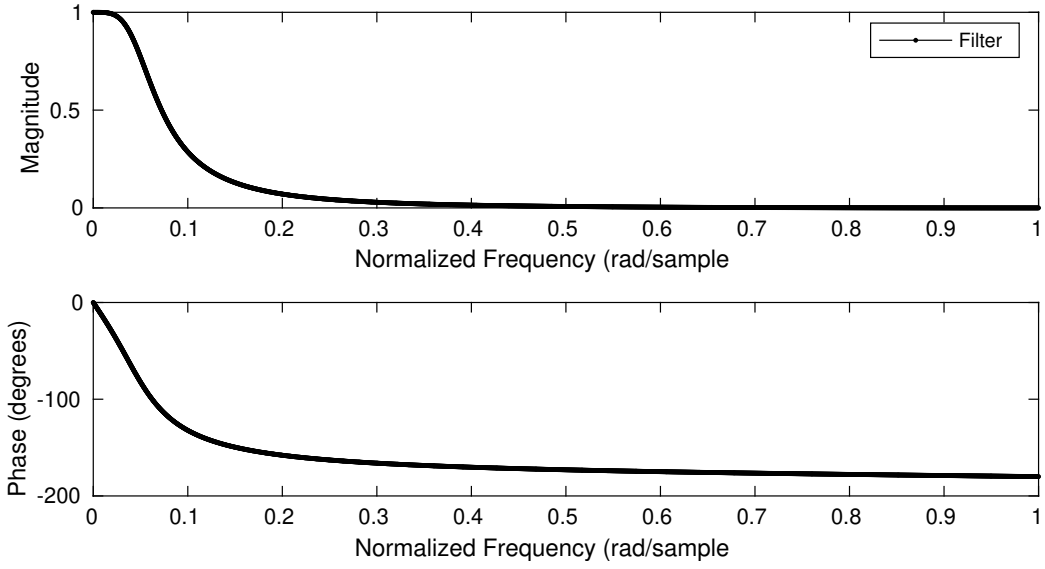


Figure B.37: Butterworth Magnitude and Phase Diagrams of y_{pnp}

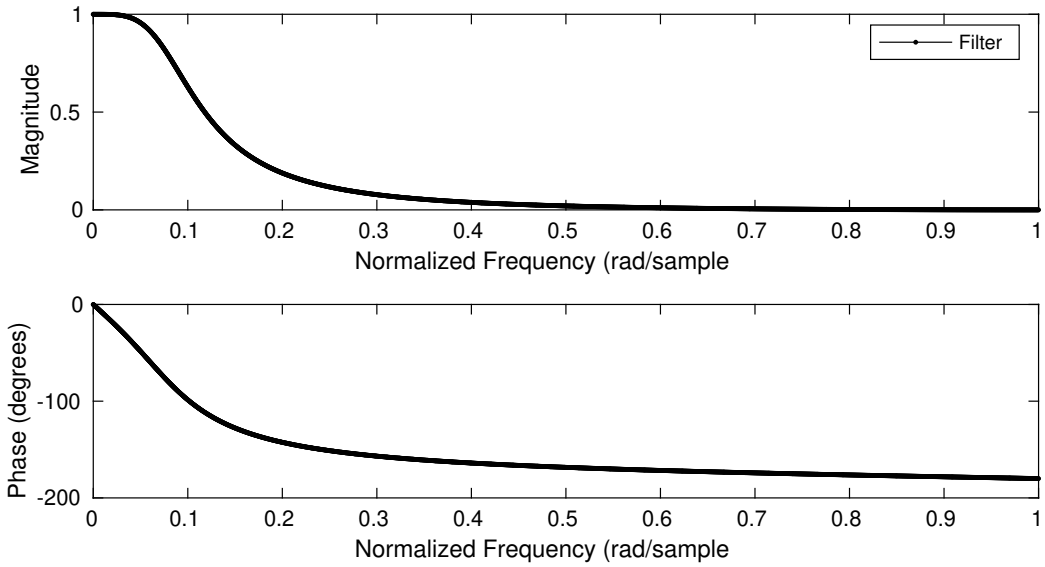


Figure B.38: Butterworth Magnitude and Phase Diagrams of z_{pnp}

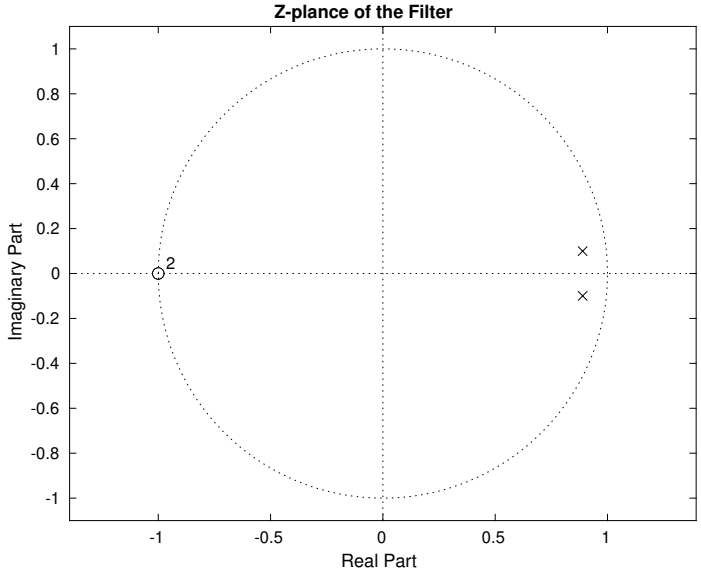


Figure B.39: Z-plane of Designed Butterworth Filter for x_{pnp}

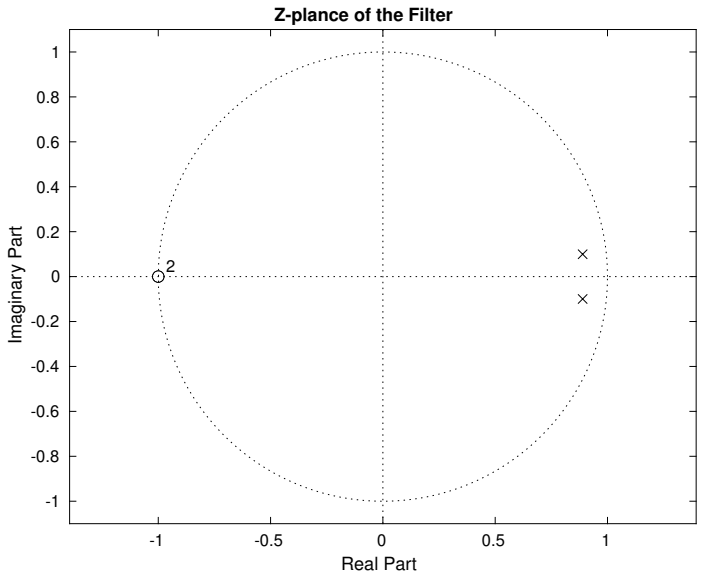


Figure B.40: Z-plane of Designed Butterworth Filter on y_{pnp}

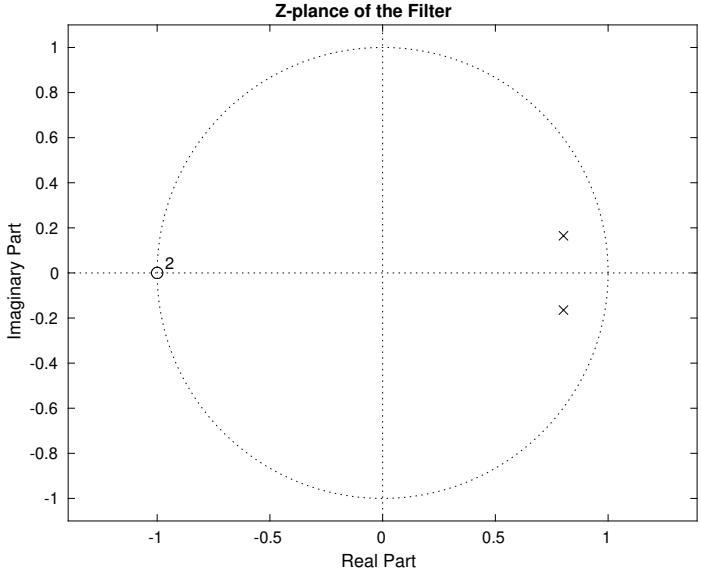


Figure B.41: Z-plane of Designed Butterworth Filter on z_{pnp}