Multiple Drone Tracking and Pursuit

by

Ferran Pampols Termens

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

Master's degree in Industrial Engineering (Polytechnic University of Catalonia, UPC)
Specialization in Automatic Control

Department of Mechanical Engineering
University of Alberta

Abstract

In recent years, drones have gained increasing importance in various fields such as agriculture, surveillance, and the military, revolutionizing how these sectors operate. In agriculture, drones are used for crop monitoring, spraying pesticides, and assessing field conditions. In surveillance, they provide real-time monitoring and data collection for security purposes. In the military, drones play a crucial role in reconnaissance, target tracking, and tactical operations. This is evident in the ongoing Ukraine conflict, where anti-drone technologies are becoming a critical research area. The versatility and efficiency of drones in these diverse applications have spurred significant advancements in drone technology and algorithms.

In this context, this thesis proposes a methodology to detect and track multiple drones and then pursue one of them. To achieve this, a combination of hardware and software components is used in a controlled, closed environment. For the detection and tracking of the drones, AI methods are employed, including a Mask R-CNN for object segmentation and a Keypoint R-CNN to extract specific keypoints of the drones captured in the frames, enabling the drawing of a 3D bounding box around the detected drones. These keypoints are also used to determine the pose of the desired drone using the Perspective-n-Point (PnP) method. The flight trajectory control system for the pursuing drone is designed using a PID strategy due to its tuning flexibility. All these tasks, implemented as ROS 2 nodes communicating with each other, enable the successful hardware demonstration of an autonomous drone pursuit system.

Resum

En els darrers anys, els drons han adquirit una importància creixent en diversos camps com l'agricultura, la vigilància i l'àmbit militar, revolucionant el funcionament d'aquests sectors. En l'agricultura, els drons s'utilitzen per al monitoratge de cultius, la fumigació de pesticides i l'avaluació de les condicions del camp. En la vigilància, proporcionen monitoratge en temps real i recopilació de dades per a finalitats de seguretat. En l'àmbit militar, els drons juguen un paper crucial en el reconeixement, el seguiment d'objectius i les operacions tàctiques. Això és evident en el conflicte actual a Ucraïna, on les tecnologies anti-drons s'estan convertint en una àrea crítica d'investigació. La versatilitat i eficiència dels drons en aquestes diverses aplicacions han impulsat avenços significatius en la tecnologia i els algoritmes de drons.

En aquest context, aquesta tesi proposa una metodologia per detectar i rastrejar múltiples drons i després perseguir-ne un. Per aconseguir això, s'utilitza una combinació de components de maquinari i programari en un entorn controlat i tancat. Per a la detecció i el rastreig dels drons, s'empren mètodes d'intel·ligència artificial, incloent-hi un Mask R-CNN per a la segmentació d'objectes i un Keypoint R-CNN per extreure punts clau específics dels drons capturats a les imatges, cosa que permet dibuixar una caixa delimitadora 3D al voltant dels drons detectats. Aquests punts clau també s'utilitzen per determinar la posició del dron desitjat utilitzant el mètode de perspectiva-n-punts (PnP). El sistema de control de la trajectòria de vol per al dron perseguidor es dissenya utilitzant una estratègia PID a causa de la seva flexibilitat d'ajust. Totes aquestes tasques, implementades com a nodes ROS 2 que es

comuniquen entre si, permeten la demostració exitosa del maquinari d'un sistema autònom de persecució de drons.

Resumen

En los últimos años, los drones han cobrado una importancia creciente en diversos campos como la agricultura, la vigilancia y el ámbito militar, revolucionando el funcionamiento de estos sectores. En la agricultura, los drones se utilizan para el monitoreo de cultivos, la fumigación de pesticidas y la evaluación de las condiciones del campo. En la vigilancia, proporcionan monitoreo en tiempo real y recolección de datos para fines de seguridad. En el ámbito militar, los drones desempeñan un papel crucial en el reconocimiento, el seguimiento de objetivos y las operaciones tácticas. Esto es evidente en el conflicto actual en Ucrania, donde las tecnologías anti-drones se están convirtiendo en un área crítica de investigación. La versatilidad y eficiencia de los drones en estas diversas aplicaciones han impulsado avances significativos en la tecnología y algoritmos de drones.

En este contexto, esta tesis propone una metodología para detectar y rastrear múltiples drones y luego perseguir uno de ellos. Para lograr esto, se utiliza una combinación de componentes de hardware y software en un entorno controlado y cerrado. Para la detección y el rastreo de los drones, se emplean métodos de inteligencia artificial, incluyendo un Mask R-CNN para la segmentación de objetos y un Keypoint R-CNN para extraer puntos clave específicos de los drones capturados en las imágenes, lo que permite dibujar una caja delimitadora 3D alrededor de los drones detectados. Estos puntos clave también se utilizan para determinar la pose del dron deseado utilizando el método de perspectiva-n-puntos (PnP). El sistema de control de la trayectoria de vuelo para el dron perseguidor se diseña utilizando una estrategia PID debido a su flexibilidad de ajuste. Todas estas tareas, implementadas como nodos ROS

2 que se comunican entre sí, permiten la demostración exitosa del hardware de un sistema autónomo de persecución de drones.

.

Preface

This thesis is an original work by Ferran Pampols Termens. The research, design, and experiments were conducted in the Mechatronics Lab of the Mechanical Engineering building (MecE) at the University of Alberta, under the guidance of Dr. Martin Barczyk. Some of the Python codes written for the ROS 2 packages are based on the *anafi_ros* package developed by Amir Hossein Ebrahimnezhad.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to Dr. Martin Barczyk for his invaluable guidance and support throughout this project. It has been an absolute honor and an incredible experience working in the Mechatronic Systems Lab under his supervision. I also extend my thanks to Joshua Brown, Vit Chan Jeong, and Zijian Jiang for their assistance during the past months. Finally, I would like to specially thank my parents and my sister for their unwavering support and love, and for giving me the opportunity to pursue my passion.

Table of Contents

Abbreviations

CPU - Central Processing Unit

DOF - Degrees of Freedom

GPU - Graphic Processing Unit

IP - Internet Protocol

LIDAR - Light Detection and Ranging

MOT - Multiple Object Tracking

PID - Proportional Integral Derivative

RAM - Random-Access Memory

R-CNN - Region-Based Convolutional Neural Network

ROI - Region of Interest

ROS - Robot Operating System

SDK - Software Development Kit

SISO - Single Input Single Output

UAV - Unmanned Aerial Vehicle

1. Introduction

The rapid advancements in technology have led to new possibilities and challenges, particularly in areas such as automation and robotics. One significant area of development is the use of drones, which, beyond their commonly known military applications, can also be utilized for purposes such as passenger transport (1) and package delivery (2). As the capabilities of drones expand, so does the need for sophisticated algorithms to manage and control them effectively.

This thesis presents the design of an algorithm specifically developed to track multiple drones simultaneously and pursue a selected target. The algorithm utilizes cutting-edge techniques in deep learning and real-time data processing to ensure precise and efficient tracking. By addressing the challenges associated with dynamic environments and the need for accurate target selection, this research aims to contribute significantly to the field of drone technology. The proposed algorithm has potential applications in various sectors, including security, logistics, and disaster management, where effective drone coordination and target pursuit are crucial.

1.1. Motivation

The motivation for this research stems from the emerging functions of aerial drones over the past decade, with many more advancements anticipated in the coming years.

In agriculture, for instance, drones are employed to fumigate large areas and monitor vegetation growth. They are also used to detect yield-reducing pests, including insects,

weeds, and diseases, and to significantly enhance the ability to track crop growth and identify nutrient deficiencies (3). Another significant application is in logistics. For example, Amazon has implemented a drone delivery system to deliver packages to specific locations, recently approved by the Federal Aviation Administration (FAA) of the United States (4).

These drone applications have been made possible through AI technologies, such as computer vision and deep learning algorithms. These technologies are frequently associated with UAVs due to their capacity for information acquisition through sensors like LIDAR or cameras. In this research, such techniques will be employed to achieve accurate tracking of detected drones and their related pose estimation.

The recently mentioned applications are just a few examples compared to the vast number currently being implemented and anticipated in the coming years. This demonstrates the significant potential of UAV technology to effectively and flexibly solve a wide range of problems.

1.2. Thesis Objectives

The main objective of this thesis is to develop, implement and validate an algorithm for tracking multiple drones. This will involve drawing a 3D bounding box around each drone, and subsequently selecting and pursuing a single drone while it is in motion. The algorithm will not consider avoiding non-tracked drones or path planning for the pursuing drone. This means the focus is on accurate tracking and stable pursuit of the target.

To achieve this primary objective, several specific objectives were considered during the algorithm's implementation, as detailed below:

- Establish stable communication with the drone while acquiring real-time frames from the camera and obtaining velocity and rotation states from the onboard state estimation system.
- Detect and track multiple drones, assigning a unique ID to each to distinguish them, and subsequently draw a 3D bounding box around each one. This allows the system to visually and programmatically differentiate between the various drones in the airspace.
- Estimate the pose of each detected drone relative to the pursuer drone.
- Design a control strategy that manipulates the drone control signals to send appropriate values, ensuring a stable pursuit of the selected tracked drone.

1.3. Thesis Outline

First, the materials and tools used to design and develop the algorithm for detecting multiple drones and pursuing one of them are described. This includes all hardware and software components and the environment in which the algorithm was developed.

Next, the design process of the algorithm is presented. This section details the organization of the algorithm into ROS2 packages, the purpose of each node, and the launch files found in those packages. Various methods and design choices are also discussed.

The thesis then covers the detection and tracking of an undefined number of drones using deep learning methods common in computer vision applications. This includes various types of Convolutional Neural Networks (CNNs) and image data processing techniques, which are

used not only to detect and segment the drones but also to differentiate between them across a sequence of frames, treating each as a unique instance.

Following the tracking, the process of obtaining the pose of one of the detected drones, as seen by the pursuer drone's camera, is explained. This is achieved by using the data from a bounding box drawn around the selected drone to be followed, which is received from the detection and tracking data.

Using the pose of the desired drone to be followed, a control strategy is developed. This strategy allows for the controlled movement of the pursuer drone, enabling it to perform a controlled pursuit in a bounded environment.

Subsequently, the results obtained by running the algorithm on the pursuer drone are discussed. This includes data acquisition using the available hardware and software components, the accuracy of pose estimation compared to the ground truth position of the followed drone, and the quality of the pursuit trajectory.

Finally, the thesis presents comprehensive conclusions based on the obtained results. These conclusions highlight the effectiveness and limitations of the developed algorithm for detecting and pursuing multiple drones, including the accuracy of pose estimation and the quality of the pursuit trajectory.

Additionally, the thesis proposes future work to improve the project. Suggestions include enhancing the deep learning models for better detection accuracy, developing more robust tracking methods, and exploring advanced control strategies to improve the pursuer drone's maneuverability.

2. Materials and Methods

This section will outline the main components used to design the algorithm and discuss their impact on the design process.

2.1. Hardware Components

2.1.1. Parrot Anafi

The ANAFI is a quadcopter drone from the Parrot company, equipped with a 4K HDR monocular camera (5). The primary reason for selecting this drone is its open-source Python library, Olympe, which allows for drone control via Wi-Fi. This drone will be used as the pursuer in the project.

Figure 2.1. Parrot ANAFI drone (6).

With a weight of 320 g and powered by four 60 W motors, the ANAFI drone can reach velocities of up to 15 m/s and has a flight range of up to 4 km from the control station using

the controller. It boasts a flight autonomy of 25 minutes on a full charge, supported by a 2700 mAh battery, which can be charged at a maximum rate of 25 W.

The drone is equipped with a Sony IMX230 camera featuring a 21MP resolution and 0.048 º/pixel. The camera has a focal length (*f*) of 4.04 mm and an aperture of *f*/2.39. Its video resolution can reach up to 4096x2160p at 24 FPS, though this is adjustable. For live video streaming, the resolution is up to 720x1280p at 24, 25, or 30 FPS, which is the mode in which the data will be acquired.

Although the ANAFI drone will be controlled with a desktop computer for this project, it can also be controlled with a mobile application named FreeFlight 6, which is compatible with both Android and iOS devices.

2.1.2. Parrot Bebop 2

The BEBOP 2 is another drone model from Parrot, and it will be the one pursued in this project (7). Since multiple drone tracking will be implemented, more than one drone will be used. Each BEBOP 2 drone used in this project will be controlled manually using the FreeFlight Pro phone app, which is also compatible with both Android and iOS devices.

Figure 2.2. Parrot BEBOP 2 drone (8).

2.1.3. Vicon Motion Capture System

Vicon is an enterprise founded in 1984 that focuses on designing optical motion capture systems for various applications, including life sciences, media and entertainment, location-based virtual reality, and engineering (9).

In this project, 10 Vicon Vero cameras will be used, strategically placed around the lab to capture the motion of the drones using passive optical markers. This data will be used to determine the ground truth position of each drone, which will subsequently be used to evaluate the performance of the tracking and pursuing of the BEBOP 2 drones.

Figure 2.3. Vicon Vero cameras at the Mechatronic Systems Lab.

To let the software identify the drones, small optical marker spheres will be attached to each drone. These markers will reflect the infrared light emitted by the cameras, enabling the cameras to track the drones' movements accurately.

Figure 2.4. Drones with markers attached.

The motion will be captured by a different computer from the one used to run the algorithm controlling the ANAFI drone. On this computer, a software named Vicon Tracker will acquire the data captured from each camera and convert it to 6DOF coordinates for each of the drones within the capture volume. This process will be discussed in section 2.2.4.

2.1.4. Desktop Computer Setup

The desktop computer controlling the ANAFI drone is equipped with an Intel Core i9-10940X CPU with 28 cores, 126 GB of RAM, and a Nvidia GeForce RTX 3090 GPU. The CPU will handle most data processing tasks, while the GPU will process frame data.

Since the computer lacks a built-in Wi-Fi module, a Netgear A6210 Wi-Fi USB adapter will be used to enable communication with the ANAFI drone.

2.2. Software Components

2.2.1. ROS 2

The Robot Operating System (ROS), is an open-source robotics middleware suite (10). ROS provides a publisher/subscriber communication system, a variety of libraries such as OpenCV and the Point Cloud Library (PCL), and numerous tools designed to assist software developers in creating robotics applications. This framework simplifies the process of developing and integrating complex robotic systems by offering reusable components and a structured communication protocol (11).

In this project, the ROS 2 Foxy Fitzroy version will be used. Although the primary focus is on the Foxy Fitzroy version, it has also been tested with the more recent Humble Hawksbill version. The choice of ROS 2 is motivated by its improved communication architecture, enhanced security features, and real-time capabilities, making it well-suited for complex and distributed robotic systems. Utilizing ROS 2 allows for more efficient development and integration of the project's components, facilitating seamless communication between different nodes and ensuring robust operation in diverse scenarios.

The basic functionality of ROS 2 consists of nodes, which are specific code blocks that execute certain tasks. To communicate between nodes, topics are used, which are a type of message that nodes can publish or subscribe to. These are the basics of ROS 2, and the complete guide can be found on its documentation webpage (12).

The algorithm developed to accomplish the purpose of this thesis will be composed of different nodes that communicate through topics. Each node will perform a specific function

and will publish and subscribe to information sent by other nodes. Additionally, premade packages will be utilized to accomplish various tasks.

To enable communication with the Vicon Tracker software, the *vicon_receiver* package has been used (13). This package retrieves data from Vicon Tracker sent through the lab's local network and publishes it on ROS 2 topics using a custom message type called *Position*. For this project, two specific topics will be published:

- */vicon/anafi/anafi*: This topic contains the positional data of the ANAFI drone.
- */vicon/bebop/bebop*: This topic contains the positional data of the BEBOP 2 drone.

Other packages and tools also have been used for data visualization and frame processing. For instance, *cv_bridge* is a package that converts between ROS2 Image messages and OpenCV images (14). Additionally, *rqt* is a graphical user interface (GUI) framework that implements various tools and interfaces in the form of plugins (15).

2.2.2. Parrot Olympe

Olympe is a component of the Parrot SDK that provides a Python programming interface designed for controlling Parrot ANAFI drones (16). This SDK allows users to connect to and manage a drone through a remote Python script running on a Linux-based computer. Olympe is compatible with a range of Parrot drones, including the ANAFI, ANAFI Thermal, ANAFI USA, and ANAFI AI.

To establish a connection with an ANAFI drone, users primarily use the *olympe.Drone* class. This class facilitates the creation of a Wi-Fi connection to the drone, requiring only the

drone's IP address. Through this connection, users can issue commands and receive telemetry data, enabling comprehensive control and monitoring of the drone's operations. The main features that will be used in this project are the following:

- Ardrone3.Piloting
- Ardrone3.PilotingState
- Ardrone3.PilotingSettingsState
- Ardrone3.SpeedSettingsState

However, to obtain the image data, the *streaming* method of the *olympe.Drone* class will be used. This method captures the video frames in YUV format and then processes them accordingly. By using the streaming functionality, real-time video data can be acquired from the drone, enabling tasks such as image analysis, object detection, and other computer vision applications. The YUV format is particularly useful as it separates the luminance and chrominance information, allowing for more efficient processing and higher quality results in various image processing tasks.

2.2.3. Parrot Sphinx

To test some parts of this project, Parrot Sphinx (17) will be used. Parrot Sphinx is a state-of-the-art drone simulation tool that provides an environment capable of simulating various types of drones, such as the ANAFI drone, in multiple settings, from an empty space to an industrial city, as shown in *Figure 2.5*.

Parrot Sphinx is built with Gazebo (18), an open-source physics engine, and Unreal Engine 4 (19), a graphics engine that allows for high-quality graphical simulations. This combination enables realistic physics interactions and visually detailed environments, making it an ideal platform for testing and validating drone behaviors and algorithms before deploying them in real-world scenarios.

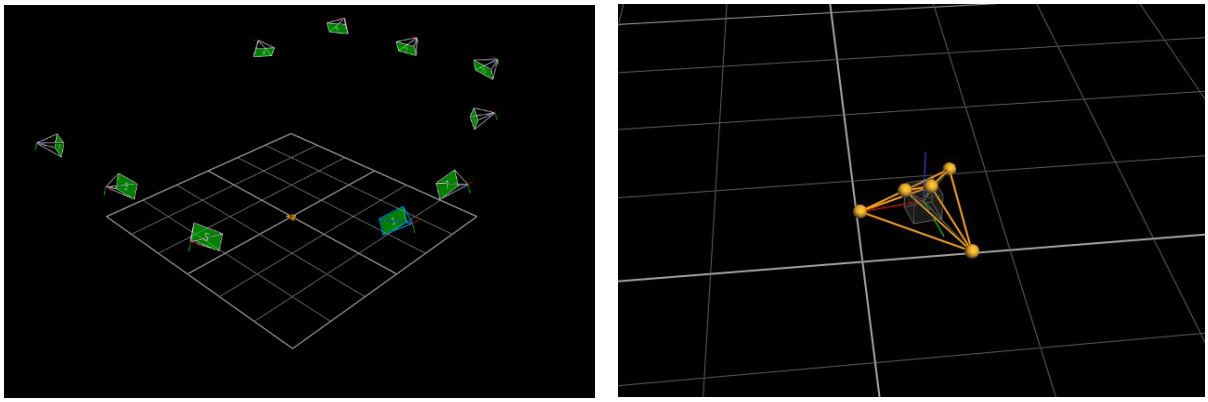
Figure 2.5. ANAFI drone simulation at Sphinx.

This tool offers various functionalities, including visualizing and recording flight data in real-time simulation, configuring different sensors and physical elements, and adding pedestrians and vehicles. However, this thesis will not explore the full range of tools provided by this software. Instead, it will focus on using the simulator to test both the manual and automatic control of the ANAFI drone.

2.2.4. Vicon Tracker

The cameras used in the lab to detect the markers send the information to the lab network, where a connected computer receives the data, as explained in section 2.1.3. Vicon Tracker, a software that collects this data, is then used to calculate the poses of the drones (20).

This software displays a GUI that shows the position of the markers. However, for proper operation, an object must be selected. If no objects are selected, the program crashes a few seconds after being turned on. In Tracker, an object is a group of markers with a specific relative position that references a target. To create an object, a set of markers must be selected using the *Ctrl* key and the left mouse button (if the *Alt* key is used as indicated in the user guide, the software crashes, at least on the computer used in this project) (21). Once selected, the markers are saved as an object with an appropriate name.



(a) (b)

Figure 2.6. Vicon Tracker lab environment (a) and ANAFI drone object (b).

After initiating the object tracking and subsequently acquiring the object's position, the system must be calibrated. The calibration process consists of three steps:

1. Create Camera Masks: First, a mask for each camera must be created to filter out unwanted reflections of infrared light emitted by the cameras. This step ensures that only the reflections from the markers are considered.

2. Calibrate the Cameras: Using the wand shown in *Figure 2.7*, the cameras are calibrated. This involves turning on the LEDs of the wand and moving it in front of the cameras to capture its movement.

3. Set Volume Origin: The origin of the coordinate system is established by placing the wand at the desired origin point.

Figure 2.7. Wand used in the calibration.

After proper calibration of the system, object tracking can begin. The position data of the tracked objects will be broadcast over the lab's local network.

2.2.5. OpenCV

The Open Source Computer Vision Library (OpenCV) offers functions to process images and videos for real-time computer vision applications. It is primarily written in C++, but it also provides language bindings for Python, Java, and MATLAB (22).

In this project, OpenCV will be used for general frame processing and for displaying the drones in a GUI, enabling the selection of which drone to pursue. Additionally, it will be used to calculate the position of the pursued drone.

2.2.6. PyTorch and CUDA

PyTorch is a library that provides machine learning tools based on the Torch library, used for computer vision and natural language processing applications. It defines a class called Tensor to store and manipulate homogeneous multidimensional rectangular arrays of numbers. PyTorch Tensors are similar to NumPy arrays but have the added capability of being operated on a CUDA-capable NVIDIA GPU (23, 24).

This is why PyTorch will be used with CUDA, also known as Compute Unified Device Architecture. CUDA is a parallel computing platform and an API that allows software to leverage certain types of NVIDIA GPUs for accelerated general-purpose processing. Using CUDA with PyTorch enhances computational efficiency, enabling faster processing times for complex machine learning tasks (25, 26).

Both these tools will be used to extract key points from each detected drone, that subsequently will allow to calculate the coordinates of the drones.

2.2.7. Detectron2

Detectron2 is a newer version of Detectron, a state-of-the-art platform for object detection research. This platform is built in PyTorch and provides a variety of prebuilt models that can be trained for specific purposes, such as aerial drones detection. To train these models, the datasets must be in COCO format, where the labeled data is stored in JSON files (27, 28).

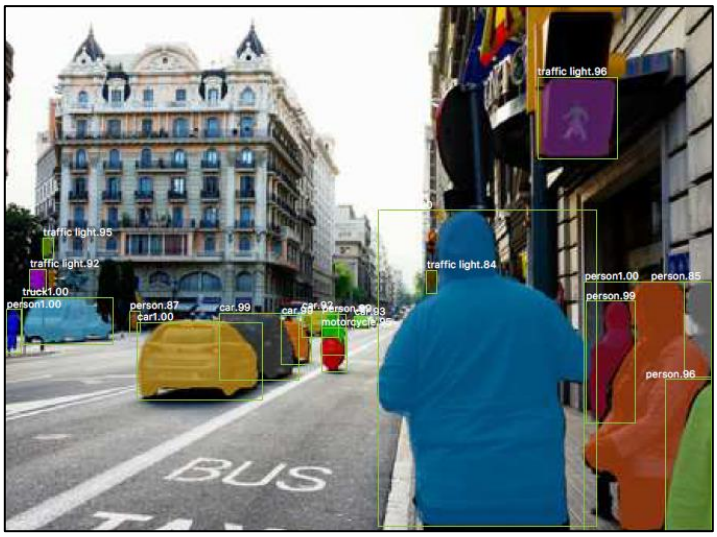


Figure 2.8. Mask R-CNN results on a COCO test data set (29).

To detect and segment drones, a prebuilt Detectron2 Mask R-CNN model has been used. Mask R-CNN is a type of convolutional neural network that extends Faster R-CNN by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression. This makes Mask R-CNN particularly effective for tasks that require precise object detection and segmentation, such as identifying and delineating drones within an image (29).

2.2.8. MATLAB

MATLAB is a platform developed by MathWorks, primarily used for algorithm development from a mathematical perspective. It is widely used for solving complex equations and analyzing large datasets due to its rapid computation capabilities. MATLAB is written in C and C++ and features its own programming language (30).

In this project, MATLAB has been utilized to analyze data, optimize keypoint detection, acquire position data, and design the control strategy for the ANAFI drone. Additionally, MATLAB tools such as Simulink and PID Tuner have been employed to accomplish these tasks.

2.2.8.1. Simulink

Simulink is a block-based environment used for modeling and simulating dynamic systems. It can retrieve data from MATLAB files and use it to create theoretical state-space models, enabling the visual implementation of control strategies. Simulink also offers a library of blocks, which includes various source signals that can be sent to a plant, allowing the performance of the output to be analyzed using a scope. This functionality is the main purpose of using Simulink in this project.

2.2.8.2. PID tuner

To tune the control loops that steer the drone, the PID Tuner tool has been used. This tool utilizes state-space models defined in MATLAB files to design a PID control strategy that

meets the specified requirements. However, only Single Input Single Output (SISO) plants can be tuned, which compromises the overall control strategy design. Additionally, only a unit step signal can be used as the reference signal, limiting the tuning design of the control.

It must be noted that PID Tuner does not account for the limitations of the control signals, such as their maximum and minimum values. Therefore, after obtaining the PID parameters, the response must be simulated in Simulink, applying these limitations to observe the real performance of the output signal. If necessary, the parameters are then modified to achieve the desired system performance.

3. Designed System

This chapter will explain how the tracking and pursuit system for the BEBOP 2 drones works using the components described in Chapter 2. It begins with the design and organization of the ROS 2 node structure and its packages. Then, it details the three main stages achieved to fulfill the purpose of this thesis: Multi-Object Detection and Tracking (MOT), detection of the pose of the desired drone to follow, and the design of the ANAFI's control system.

3.1. Algorithm Architecture

As mentioned in Section 2.2.1, the algorithm's structure is based on ROS 2 nodes, which are organized into packages. Three different packages were created for distinct purposes: *anafi_ros2*, *anafi_msg* and *sphinx_ros2*. All the packages are available in Appendix A.

3.1.1. anafi_ros2

The *anafi_ros2* package is the main package, containing the nodes required to track the BEBOP 2 drones and control the ANAFI as it pursues one of them in the lab environment. Although there are 9 nodes within this package, only 5 are used to build the MOT and pursuer algorithm. These nodes are executed together using a ROS 2 launch file, allowing the algorithm to run from a single file and providing the option to configure certain parameters to change the system's behavior.

3.1.1.1. af_pursuer

This node sends the control signals to drive the drone by managing the roll, pitch, and yaw angular velocity, as well as the “Gaz” (vertical axis) linear velocity. It is the only node that establishes a Wi-Fi connection with the ANAFI drone. The node retrieves the current rotation and linear speed of the drone along its body-fixed axes and publishes this data to a topic named *anafi/state*. Additionally, it transmits the frames captured by the drone’s camera and publishes them to the *anafi/frames* topic.

For manual control of the drone, the computer keyboard will be used. To enable the ANAFI to pursue a BEBOP 2 drone, switching to the pursuing mode is required. In this mode, the node subscribes to the *anafi/control* topic, which provides the control signals necessary for the ANAFI to pursue the selected BEBOP 2 drone.

3.1.1.2. af_tracker

This node subscribes to the *anafi/frames* topic, which contains the frames captured by the camera, and detects the BEBOP 2 drones within each frame. An algorithm is employed to identify and track each drone, drawing a square around them and assigning them an ID. This data is then published to the *anafi/bbox2D* topic.

3.1.1.3. af_3D_bbox

By subscribing to the *anafi/bbox2D* and *anafi/frames* topics, this node extracts 8 key points to draw a 3D bounding box around each detected target drone and publishes the data to the *anafi/bbox3D/kp* topic. Additionally, it displays a GUI that shows the detected drones in red

bounding boxes and allows the user to select the drone to be pursued, highlighting it with a green bounding box.

3.1.1.4. af_pnp

This node subscribes to the *anafi/bbox3D/kp* topic and uses the 8 points data to estimate the position of the target drone to be pursued. This position data is then published to the *anafi/pnp* topic.

3.1.1.5. af_control

Using the estimated position received from the *anafi/pnp* topic, this node sends control signals to move the drone while pursuing the desired target. It also subscribes to the *anafi/state* topic to obtain real-time velocity and orientation data of the ANAFI drone.

There are two control modes within this node, which can be switched by changing a parameter in the launch file. The default mode sends control signals to track the desired drone while it is moving. The other mode, designed for steady-state pursuit, tracks the BEBOP's initial detected position. This setup allows for testing both dynamic and steady-state pursuing. In both cases, the control signals will be published to the *anafi/control* topic.

Moreover, a GUI for configuring the PID variables can be displayed by adjusting a parameter in the launch file.

3.1.1.6. Launch file

The *pursuer_launch.py* file is the launch file that contains all the previously mentioned nodes and runs the complete algorithm for tracking and pursuing the BEBOP 2 drones. It includes several parameters that allow the user to configure the behavior of the system and display different visualization modes. The structure of the complete system is shown in Figure 3.1.

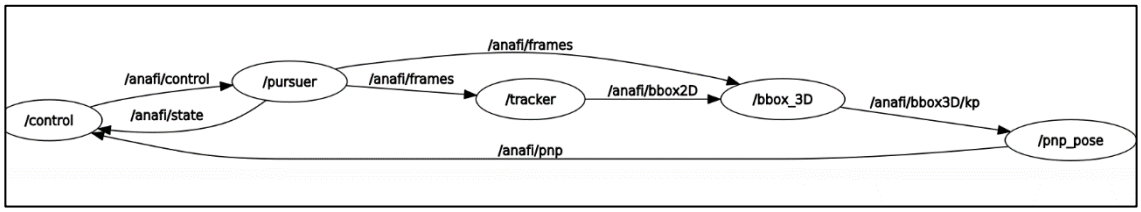


Figure 3.1. Structure of the ROS 2 node-based algorithm.

3.1.2. anafi_msg

This package does not contain any nodes. Instead, it provides custom message formats utilized by the *anafi_ros2* and *sphinx_ros2* nodes for subscribing to and publishing data. This design choice was necessary because the *anafi_ros2* package is built in Python, while ROS 2 only permits the definition of custom messages in C++ based packages.

3.1.3. sphinx_ros2

This package has been created to test the control of the ANAFI drone and gather data to design the control strategy in a simulated environment using Sphinx. The nodes in this package are similar to those in the *anafi_ros2* package but are adapted to the firmware of the simulated drone in Sphinx.

The main file for testing the control is the *pursuer_sim_launch.py* file. This file has been used to test the control strategy by controlling the drone to move it near the origin of the simulated world. This testing was conducted to validate the control strategy before flying the drone in a real environment.

3.2. Multiple Object Detection and Tracking

The first step in designing the algorithm is to detect and track the drones within view of the ANAFI's camera. Tracking involves detecting each drone and following its trajectory across frames, ensuring the algorithm can distinguish between different drones. To achieve this, various algorithms and types of Convolutional Neural Networks (CNNs) have been employed.

3.2.1. Drone Segmentation

To first detect the drones from the raw frames acquired through the *anafi/frames* topic, a premade Mask R-CNN from Detectron2 will be used as seen in section 2.2.7. The selected Mask R-CNN from the Detectron2 model zoo is the R50-FPN-3X (31).

This R-CNN model is made based on the original ResNet-50 model, from the Deep Residual Networks repository (32, 33). This pre-trained CNN model has an inference time of 0.043 seconds per image, indicating the time required for object detection. This results in a data acquisition frequency of 23.25 Hz, matching the frames per second (FPS) of the display window that shows the detected drones, using the four points that create a 2D box to contain each drone.

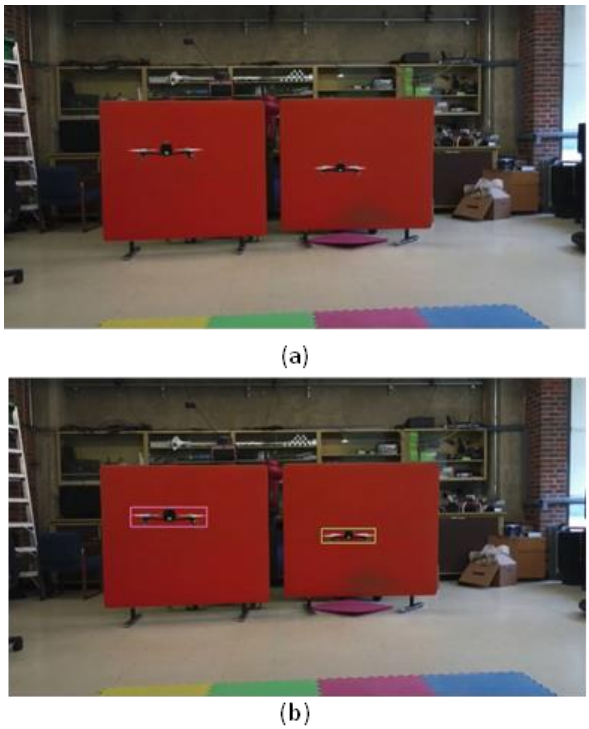


Figure 3.2. Raw image (a) and output of the Mask R-CNN (b).

The data obtained, including the four points that create the aforementioned box, is subsequently used to track the drones.

3.2.2. Multiple Object Tracking Algorithm

The Multiple Object Tracking (MOT) problem is essentially a data association challenge. In a single image, when multiple objects are detected and marked, the detection process only provides the position of each object within the frame, indicating that each object is a distinct instance. However, during a sequence of frames, such as in a video, it is necessary to associate and identify each object throughout the sequence.

Currently, the system assumes that each drone detected in a new frame is different from those detected in previous frames. For instance, if two drones are detected in one frame, and 0.05 seconds later another frame is received with the same two drones, the system treats these drones as new and distinct from those previously detected. Therefore, an algorithm is needed to assign a unique ID to each drone in every frame to maintain continuity and accurate tracking.

To achieve this, the Simple Online and Realtime Tracking (SORT) algorithm has been employed (34, 35). This method creates a model for each target, assigning a unique identifier to them. The states of each target are modeled as:

$$X(\mathrm{k}) = [\, u,\ v,\ s\,,\ r,\ \dot{u},\ \dot{v},\ \dot{s}\,] \qquad \text{(eq. 1)}$$

Where u and v denote the horizontal and vertical pixel locations of the target's center, and s and r represent the scale and the ratio of the bounding box, respectively. These states are extracted from the input to the SORT function which consists of the coordinates of the top-left vertex (*x1, y1*) and bottom-right vertex (*x1, y1*) of the detected drone's bounding box.

For each existing target, its next position is predicted using Kalman Filters considering the states calculated. This is done by using the current state estimate and a motion model to predict the state in the next time step, as shown in equation 2. The motion model assumes that the object's velocity remains constant over the short period between detections:

$$X(k+1) = \mathrm{F} \cdot \mathrm{X(k)} + \mathrm{B} \cdot \mathrm{u(k)} \qquad \text{(eq. 2)}$$

Where *F* is the state transition matrix, which models how the state evolves from one time step to the next, *u(k)* is the control input vector, representing external commands or inputs that affect the state, and *B* is s the control input matrix, which maps the control input u to the state space.

However, the control input is typically not used because the motion of objects is generally assumed to be governed by a constant velocity model without external commands. Therefore, in the basic implementation of SORT, the control input *u* and the control input matrix *B* are set to zero, hence:

$$X(k+1) = \mathrm{F} \cdot \mathrm{X(k)} \qquad \text{(eq. 3)}$$

Additionally, the SORT algorithm predicts the error covariance matrix, which represents the uncertainty in the state prediction, shown as:

$$\mathrm{P}(k+1) = \mathrm{F} \cdot \mathrm{P(k)} \cdot \mathrm{F^T} + \mathrm{Q} \qquad \text{(eq. 4)}$$

Where *P(k+1)* is the error covariance matrix, and *Q* models the process noise, accounting for any uncertainties in the model, such as unexpected accelerations.

Once the new predicted states and the error covariance matrix are acquired, a cost assignment matrix is computed as the IoU distance between the bounding boxes of each detection and all predicted bounding boxes from the existing targets:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad \text{(eq. 5)}$$

The Hungarian algorithm is then used to solve the assignment problem. This algorithm finds the optimal assignment that minimizes the total cost. The output of the Hungarian algorithm is a set of pairs *(i, j)* indicating that the *i*-th predicted track is associated with the *j*-th detection.

Finally, after performing data association, if there are unmatched detections, it indicates that a new drone has been detected, and a new state for that detection is initialized to track it. Conversely, if a track remains unmatched for several consecutive frames, it is considered lost and removed.

To ensure robust and reliable tracking, several key parameters of the SORT function have been carefully adjusted and fine-tuned:

- *max_age*: The maximum number of frames a track can persist without associated detections. This is set to 200.
- *min_hits*: The minimum number of detections required before a track is initialized. This is set to 20.
- *iou_threshold*: The intersection over union (IOU) value needed for a match, representing the percentage of the area shared between the boxes of two detections in consecutive frames, set to 0.1.

Figure 3.3. Target drones with their ID number.

3.2.3. Key points acquisition

To extend the analysis from 2D to 3D bounding boxes for the tracked drones, the Keypoint R-CNN model trained by Amir Hossein Ebrahimnezhad in his previous thesis has been utilized (36). The model employed is the KEYPOINT_RCNN_RESNET50_FPN provided in Torch (37). This pre-trained model generates an output with eight key points for each detected drone.

However, the existing implementation is designed for a single target. To obtain key points for each detected drone, the data obtained from the SORT algorithm, which provides 2D bounding boxes, must be processed individually. For each detected drone, the raw frame is reshaped to isolate the drone. This reshaping process is repeated for the number of detected drones. Subsequently, key points are extracted from each reshaped image containing an individual drone.

As a result, the R-CNN model has to be executed as many times as there are detected drones. This increases the time required to estimate all the key points, taking between 0.06 and 0.1 seconds per drone, thus affecting the frequency at which the keypoint data is obtained.

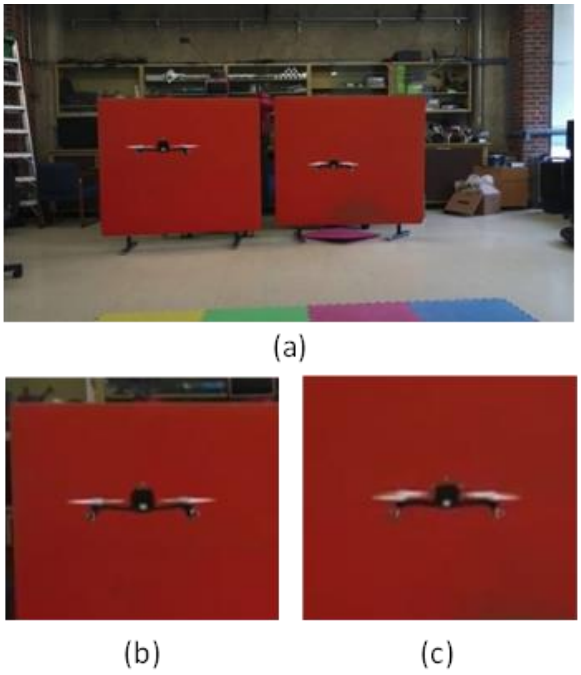


Figure 3.4. Raw Image (a) and reshaped images containing drones (b)(c).

To address oscillations caused by the tolerance of the CNN model, a low-pass filter has been applied to each of the extracted key points for the drones. Specifically, a second-order digital Butterworth filter with a cutoff frequency of 2.3 Hz was used. Considering that the average sampling frequency for receiving the 2D bounding box data is 14 Hz, the resulting cutoff frequency is 0.33 Hz, as shown in Equation 6.

$$fc = \frac{f_{cutoff}}{f_s / 2} = \frac{2.3}{14 / 2} = 0.33\ Hz \qquad \text{(eq. 6)}$$

As multiple drones are detected, each target will have its own associated group of filters linked to the target's ID number. Each group will contain eight filters, one for each key point. The parameters of the filter are shown in Equation 7.

$$H(z) = \frac{0.1468 + 0.2935\, z^{-1} + 0.1468\ z^{-2}}{1.0 - 0.6634\, z^{-1} + 0.2505\ z^{-2}} \quad \text{(eq. 7)}$$

After applying the filters, the key points required to form a 3D bounding box are finally obtained. The low-pass Butterworth filters effectively reduce noise in the data, ensuring more stable and accurate key point data. With these refined key points, we can now construct precise 3D bounding boxes for each detected drone, enhancing the overall accuracy and reliability of the tracking system.

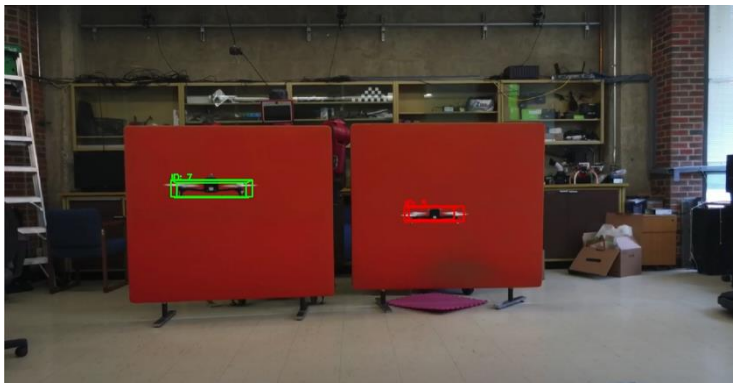


Figure 3.5. 3D bounding boxes result.

3.3. Pose estimation

The next step is to determine the relative position of the selected BEBOP 2 drone with respect to the ANAFI drone. To achieve this, the keypoints of the selected drone are used to estimate its position using the Perspective-n-Point (PnP) method. This method utilizes a set of points from an image for which the corresponding real-world dimensions are known.

3.3.1. Camera Matrix

Before delving into the position estimation method, it is essential to understand the camera model and the associated matrix utilized by the camera. The camera matrix, also known as the projection matrix or calibration matrix, is a 3x3 matrix that defines the internal parameters of a camera used in computer vision and photography (38). This matrix contains the necessary information to map real-world coordinates to image coordinates.

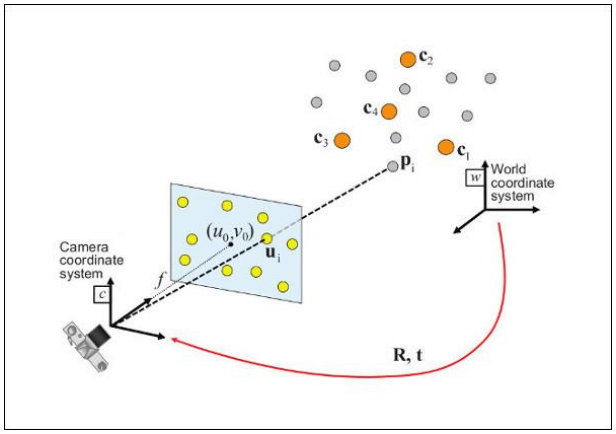


Figure 3.6. Transformation between world frame to image frame coordinates (39).

The projection matrix is composed of two matrices: the intrinsic parameters matrix and the extrinsic parameters matrix. The intrinsic matrix A is defined as follows:

$$A = \begin{bmatrix} f_x & 0 & u_o \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(eq. 8)}$$

This matrix represents the properties of the camera itself, where f_x and f_y are the focal lengths, and u_o and v_o are the horizontal and vertical origin points of the image frame, respectively. The extrinsic parameters matrix *[R, t]* is expressed as:

$$[R,t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \quad \text{(eq. 9)}$$

This matrix describes the transformation between the world coordinate system and the camera coordinate system, containing both rotation and translation. This allows the mapping of points from the world frame to the camera frame.

The combination of these matrices forms the camera's projection matrix, which is used for tracking. This transformation allows the conversion of coordinates from the world frame X_w to the image coordinates Y_c, represented as pixels.

$$Y_c = \begin{bmatrix} x_c \\ y_c \\ w_c \end{bmatrix} = A\,[R,t]X_w = A\,[R,t] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \quad \text{(eq. 10)}$$

While the extrinsic matrix can be obtained by determining the pose transformation between the camera and the world frame, the intrinsic matrix must be obtained through camera calibration. For this project, the intrinsic matrix previously obtained for the same ANAFI drone will be used (36). The parameters of the intrinsic matrix are presented in equation 11.

$$A = \begin{bmatrix} 941.35 & 0 & 638.96 \\ 0 & 948.55 & 375.83 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(eq. 11)}$$

3.3.2. Perspective-n-Point method

The Perspective-n-Point method or PnP, is an extension of the P3P problem initially developed by Quan et al. in 1999 (40) and further refined by Gao et al. in 2003 (41). It involves estimating the position of an object from an image frame, given the coordinates of three control points that are projected.

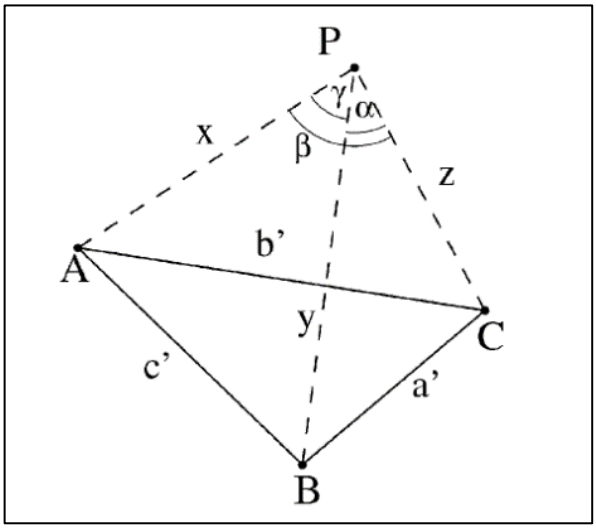


Figure 3.7. Scheme of the P3P problem (41).

Focusing on the P3P problem scheme in Figure 3.7, let $P \in R^3$ be the center of perspective of the camera, and $[A, B, C] \in R^3$ be the control points with 3D coordinates known in real-life. Then:

$$p = 2cos\alpha \; ; \; q = 2cos\beta \; ; \; r = 2cos\gamma \qquad \text{(eq. 12)}$$

The P3P problem can be formulated as:

$$\begin{cases} Y^2 + Z^2 - YZp - a'^2 = 0 \\ Z^2 + X^2 - XZq - b'^2 = 0 \\ X^2 + Y^2 - XYr - c'^2 = 0 \end{cases} \quad \text{(eq. 13)}$$

By reformulating the problem with the assumptions:

$$X = x\,Z \;;\; Y = y\,Z \;;\; Z = |AB|/\sqrt{v} \quad \text{(eq. 14)}$$

Hence,

$$v = x^2 + y^2 - xyr > 0 \quad \text{(eq. 15)}$$

This leads to the equations:

$$\begin{cases} p1 = (1 - a)y^2 - ax^2 - py + arxy + 1 = 0 \\ p2 = (1 - b)x^2 - bx^2 - qx + brxy + 1 = 0 \end{cases} \quad \text{(eq. 16)}$$

However, due to the two quadratic equations, four feasible physical solutions to the pose are typically obtained. Therefore, more than three points are needed to find a unique solution, leading to n≥4 PnP methods. In the OpenCV library, several methods can resolve the PnP problem, such as the EPnP method (42) or using the RANSAC scheme (43).

In this project, an iterative method using the *cv2.solvePnP* function from OpenCV has been employed to achieve this purpose (44). This method uses a non-linear Levenberg-Marquardt optimization scheme (45) and requires a minimum of six points for non-planar objects, such

as our drone. Our drone provides eight keypoints through the *anafi/bbox3D/kp* topic, making this method suitable.

The process involves an iterative method in which the reprojection error is reduced. The reprojection error is the sum of squared distances between the observed projections in the image plane and the projected points of the object in the world frame. This is achieved using the following equations, derived from equation 10:

$$\hat{u} = \frac{x_c}{w_c} \; ; \; \hat{v} = \frac{y_c}{w_c} \qquad \text{(eq. 17)}$$

Minimizing the reprojection error for *N* points is written as:

$$E = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} [\, (\hat{u}_i - u_i)^2 + (\hat{v}_i - v_i)^2] \qquad \text{(eq. 18)}$$

where (*u*, *v*) are the coordinates of one of the observed points in the image in pixels (46).

3.3.3. Pose Filtering

Noise effects during the estimation of the pose of the drones along the frame sequence has required filtering the obtained pose. However, since only the translation along the X, Y, and Z axes and the rotation around the Z axis (known as yaw) are used, only four filters are necessary. The filters used are second-order low-pass filters, similar to those described in section 3.2.3. Table 3.1 presents the frequencies of the filters used in this study. It includes the sample frequency at which the estimated pose data is obtained, the cutoff frequency, and

the resulting cutoff frequency of each filter normalized by the Nyquist frequency, calculated using equation 6.

Filtered signal	Sample freq. [Hz]	Cutoff freq. [Hz]	Resulting freq. [Hz]
X	5	0.5	0.2
Y	5	0.5	0.2
Z	5	0.1	0.04
Yaw	5	0.1	0.04

Table 3.1. Frequencies of the pose filters.

Consequently, the resulting filters are:

$$H_{\mathrm{X}}(z) = H_{\mathrm{Y}}(z) = \frac{0.0476 + 0.0953\, z^{-1} + 0.0476\ z^{-2}}{1.0 - 1.2948\, z^{-1} + 0.4854\ z^{-2}} \quad \text{(eq. 19)}$$

$$H_{\mathrm{Z}}(z) = H_{\mathrm{Yaw}}(z) = \frac{0.0036 + 0.0072\, z^{-1} + 0.0036\ z^{-2}}{1.0 - 1.8227\, z^{-1} + 0.8372\ z^{-2}} \quad \text{(eq. 20)}$$

Using these filters avoids sudden changes in pose detection, enhancing the stability of the signal values during the pursuing process.

3.4. Control

This section will describe the process that has been followed to design the control system, and the state space modelling of the drone behaviour.

3.4.1. State-space model

Before delving into the control scheme, it is necessary to design a state-space model of the drone to comprehend how the drone is controlled and to simulate and test the design. The structure of the model is as follows:

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t) \quad \text{(eq. 21)}$$

$$y(t) = C \cdot x(t) + D \cdot u(t) \quad \text{(eq. 22)}$$

Where *x(t)* is a Nx1 vector containing the states of the model, *u(t)* is an Mx1 vector containing the control signals that influence the system behavior, and *y(t)* is a vector with the outputs of the system (47).

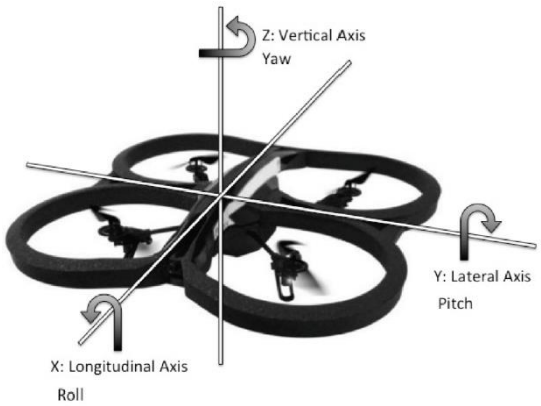


Figure 3.8. Primary axes of a drone (48).

To design the dynamic model of the ANAFI drone, it is essential to first understand the drone's movement behavior. The signal controlling the drone's movements is sent by the

PCMD message of the Parrot Olympe library, as shown in section 2.2.2. This message has four main parameters: roll, pitch, yaw, and gaz:

- The roll and pitch signals control the percentage of the maximum tilt for roll and pitch, with values ranging between [-100, 100].
- The yaw signal control the percentage of the maximum angular velocity for the yaw, with values ranging between [-100, 100].
- The gaz signal controls the percentage of the maximum linear velocity along the Z axis, with values also ranging between [-100, 100].

The maximum absolute values for these parameters are shown in Table 3.2.

Parameter	Maximum absolute value
Roll	10 deg
Pitch	10 deg
Yaw	56 deg/s
Gaz	1 m/s

Table 3.2. Maximum absolute values of the PCMD parameters (49).

Despite that, not all these parameters directly impact the drone's movement. The yaw and gaz parameters do have a direct impact on the angular position around the Z axis and the linear position along the Z axis, respectively, by controlling the corresponding angular and linear velocities. However, the roll and pitch parameters affect the linear acceleration along the Y and X axes, not the velocity. When the pitch and roll angles increase or decrease, the drone's acceleration along these axes increases or decreases accordingly.

Assuming this, the dynamic model of the drone has been designed using SISO plants for each parameter in the PCMD function. This design strategy is employed because each parameter influences movement through or around a specific axis, and the distinct nature of each parameter justifies the use of SISO plants. Additionally, using SISO plants simplifies the tuning of the PID parameters, which is discussed in section 3.4.3.

Then, as the pitch and roll parameters impact linear acceleration, their state-space models have the following structure as shown in equations 23 and 24:

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & a\end{bmatrix} \cdot \begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}0\\ b\end{bmatrix} \cdot u(t) \quad \text{(eq. 23)}$$

$$y(t) = \begin{bmatrix}1 & 0\end{bmatrix} \cdot \begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + 0 \cdot u(t) \quad \text{(eq. 24)}$$

And the yaw and gaz models have a structure shown in equations 25 and 26:

$$\dot{x}(t) = 1 \cdot x(t) + b \cdot u(t) \quad \text{(eq. 25)}$$

$$y(t) = 1 \cdot x(t) + 0 \cdot u(t) \quad \text{(eq. 26)}$$

Where $x_1(t)$ is the position and $x_2(t)$ the velocity along the Y axis for the roll and the X axis for the pitch. For the gaz and yaw, x(t) is the position along the Z axis and the rotation around the Z axis, respectively.

The *a* and *b* parameters in both model structures are unknown and will be determined using MATLAB. To accomplish this, an iterative algorithm has been developed, utilizing the

position data of the ANAFI while it is moving controlled by a step signal. This data is acquired using the Vicon System and the *vicon_receiver* package, as described in sections 2.1.3 and 2.2.1. The algorithm will estimate these parameters by iterating using the position and velocity acquired as the states of the model *x(t)*, and the parameters of the PCMD message sent to the ANSFI as the control signal *u(t)*. The algorithms used are shown in Appendix B.

Nevertheless, those models are in continuous time, and the control used will operate with discrete data. Therefore, the models described by equations 23, 24, 25, and 26 will be discretized using the *c2d()* function in MATLAB, employing the Zero-Order Hold method (50).

With the a and b parameters acquired, the models for roll, pitch, yaw, and gaz are shown in the following equations, from equation 27 to 34.

3.4.1.1. Roll Model

Figure 3.9 illustrates the real velocity and the velocity state of the drone's roll model along the Y axis, in response to a step signal applied to the roll parameter of the PCMD message.

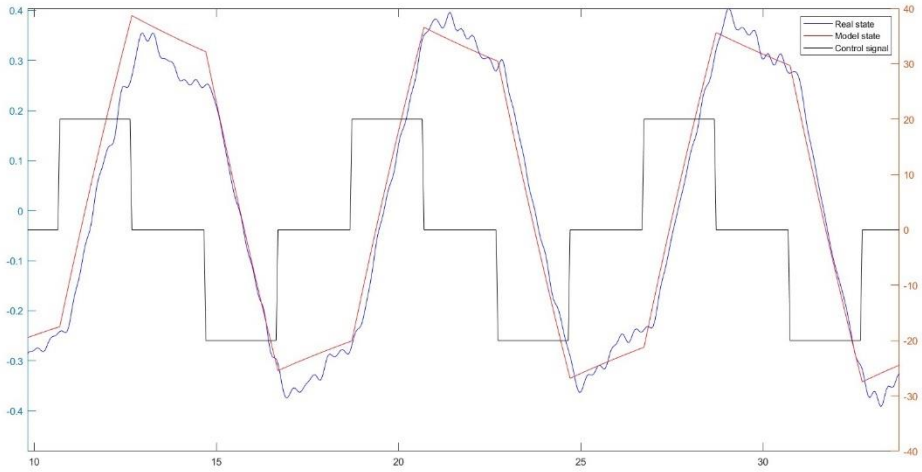


Figure 3.9. Step signal on the control of the roll (Left Axis) and the consequent real and model velocities states in m/s (Right Axis).

The state-space model that defines the displacement along the Y axis is:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 0.05 \\ 0 & 0.9928 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0.0000206 \\ 0.0008223 \end{bmatrix} \cdot u(k) \quad \text{(eq. 27)}$$

$$y(t+1) = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + 0 \cdot u(t) \quad \text{(eq. 28)}$$

3.4.1.2. Pitch Model

Figure 3.10 illustrates the real velocity and the velocity state of the drone's pitch model along the X axis, in response to a step signal applied to the pitch parameter of the PCMD message.

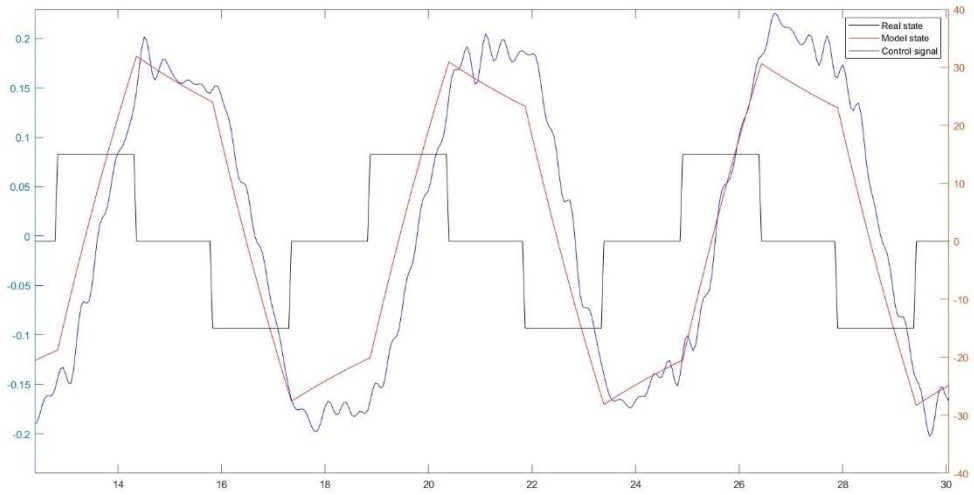


Figure 3.10. Step signal on the control of the pitch (Left Axis) and the consequent real and model velocities states in m/s (Right Axis).

The state-space model that defines the displacement along the X axis is:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 0.05 \\ 0 & 0.9947 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0.0000181 \\ 0.0007244 \end{bmatrix} \cdot u(k) \quad \text{(eq. 29)}$$

$$y(k+1) = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + 0 \cdot u(k) \quad \text{(eq. 30)}$$

3.4.1.3. Yaw Model

Figure 3.11 illustrates the real orientation and the orientation state of the drone's yaw model around the Z axis, in response to a step signal applied to the yaw parameter of the PCMD message.

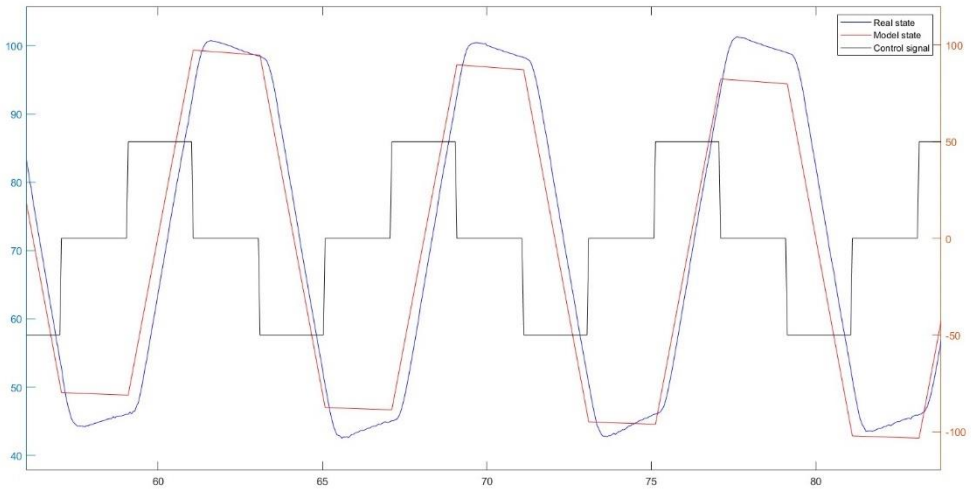


Figure 3.11. Step signal on the control of the yaw (Left Axis) and the consequent real and model orientation states in degrees/s (Right Axis).

The state-space model that defines the orientation around the Z axis is:

$$x(k+1) = 1 \cdot x(k) + 0.02618 \cdot u(k) \quad \text{(eq. 31)}$$

$$y(k+1) = 1 \cdot x(k) + 0 \cdot u(k) \quad \text{(eq. 32)}$$

3.4.1.1. Gaz Model

Figure 3.12 illustrates the real position and the position state of the drone's gaz model along the Z axis, in response to a step signal applied to the gaz parameter of the PCMD message.

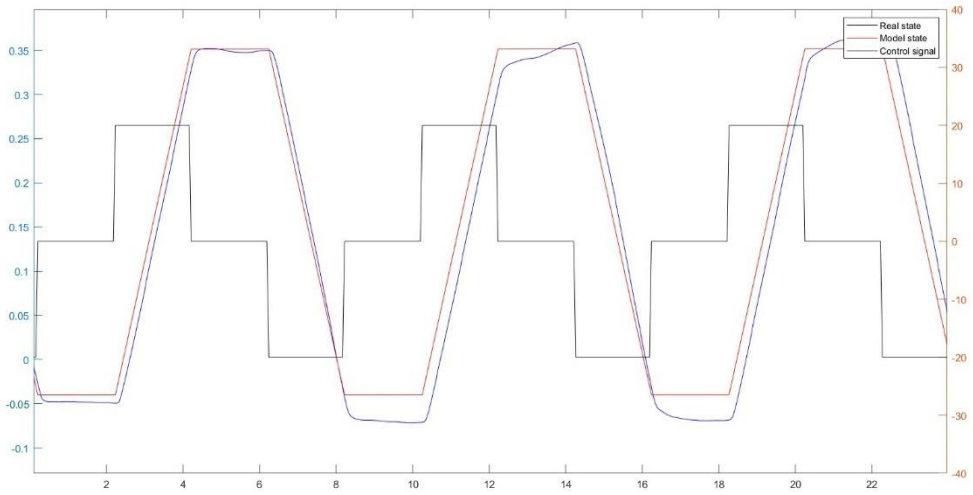


Figure 3.12. Step signal on the control of the gaz (Left Axis) and the consequent real and model position states in m/s (Right Axis).

The state-space model that defines the displacement along the Z axis is:

$$x(k+1) = 1 \cdot x(k) + 0.000505 \cdot u(k) \quad \text{(eq. 33)}$$

$$y(k+1) = 1 \cdot x(k) + 0 \cdot u(k) \quad \text{(eq. 34)}$$

3.4.2. Control system design

To control the drone's movement along the three axes, a PID control strategy has been designed. The control system uses the error between the actual position of the target drone

detected and the desired position of the target drone relative to the ANAFI drone. This desired position will always remain the same. Its values are shown in Table 3.3.

Position	**Value**
X displacement	2.5 m
Y displacement	0 m
Z displacement	0 m
Z orientation	0 deg.

Table 3.3. Desired position of the target BEBOP relative to the ANAFI.

Figure 3.13 illustrates how the control system functions. The error *e(k)* is used to calculate the control signal and is determined by the difference between the desired position value *r(k)* and the position of the target drone relative to the ANAFI *y(k)*.

However, as mentioned in section 3.1.1.5, two control systems have been developed: the steady state pursuing and the dynamic pursuing. The only difference between both control strategies is how the *y(k)* vector is obtained. In the dynamic pursuing, the *y(k)* vector is acquired from the estimated position obtained by the PnP method. In the steady state pursuing, only the first estimated position from the PnP method is used, and then the real-time position relative to the first estimated PnP pose is calculated by integrating the velocity of the ANAFI drone received through the *SpeedChanged* and *AttitudeChanged* messages of the Olympe library.

On the other hand, *u(k)* is the control signal sent by the PCMD message that controls the movement of the ANAFI. All these variables are vectors with four values referring to the target drone pose:

- *r(k)*: Desired position vector $[r_x, r_y, r_z, r_{yaw},]^T$.
- *y(k)*: Measured position vector of the target relative to the ANAFI $[y_x, y_y, y_z, y_{yaw},]^T$.
- *e(k)*: Error vector $[e_x, e_y, e_z, e_{yaw},]^T$.
- *u(k)*: Control signal vector $[u_{pitch}, u_{roll}, u_{gaz}, u_{yaw}]^T$.

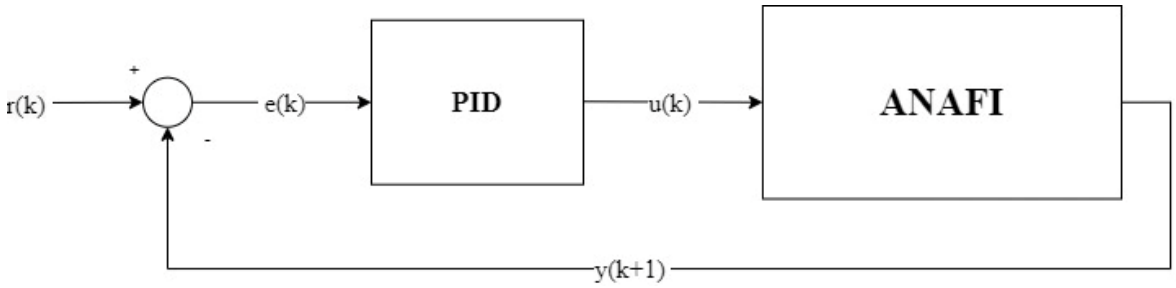


Figure 3.13. Control system scheme.

To get the *u(k)* vector, the PID controller uses the error vector and calculates the proportional e_p, integral e_i and derivative e_d errors for each value in the vector, as shown in equation 35:

$$e_p = e(k) \; ; \; e_i = \sum e(k) \; ; \; e_d = \frac{e(k) - e(k-1)}{t(k) - t(k-1)} \qquad \text{(eq. 35)}$$

Then, using the K_p, K_i, and K_d weights tuned in section 3.4.3, it calculates de values in the *u(k)* vector:

$$u(k) = K_p \cdot e_p + K_i \cdot e_i + K_d \cdot e_d \qquad \text{(eq. 36)}$$

However, the integral error e_i has been limited to values between [-50, 50] to avoid excessively high values. Additionally, due to the behavior of the parameters in the PCMD message, the control vector *u(k)* has been limited to values between [-100, 100].

3.4.3. PID parameters tuning

The tuning of the PID weights has been made by using the estimated models in section 3.4.1. Then, by using the PID Tuner as shown in section 2.2.8.2, the proportional, integral and derivative weights have been adjusted to fulfil the desired performance of the control signals.

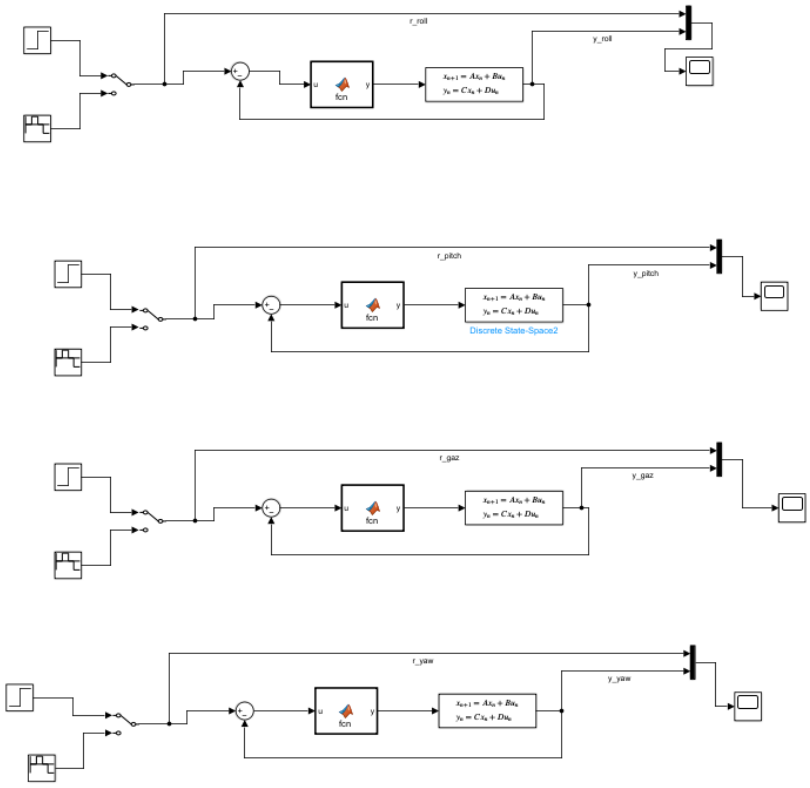


Figure 3.14. Simulation of the estimated models in Simulink.

As the PID Tuner does not take into account the limitations in the integral error and the limits of the PCMD function, the response of the system has been simulated in Simulink, as shown in section 2.2.8.1. For each control signal in the control vector, a simulated system has been created to check the response of each system.

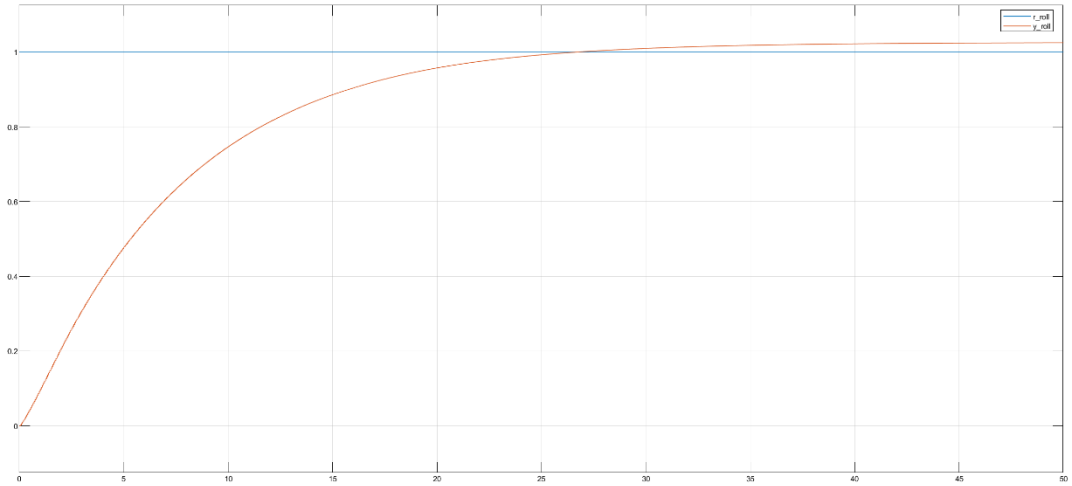


Figure 3.15. Response of the roll signal related mode in Simulink.

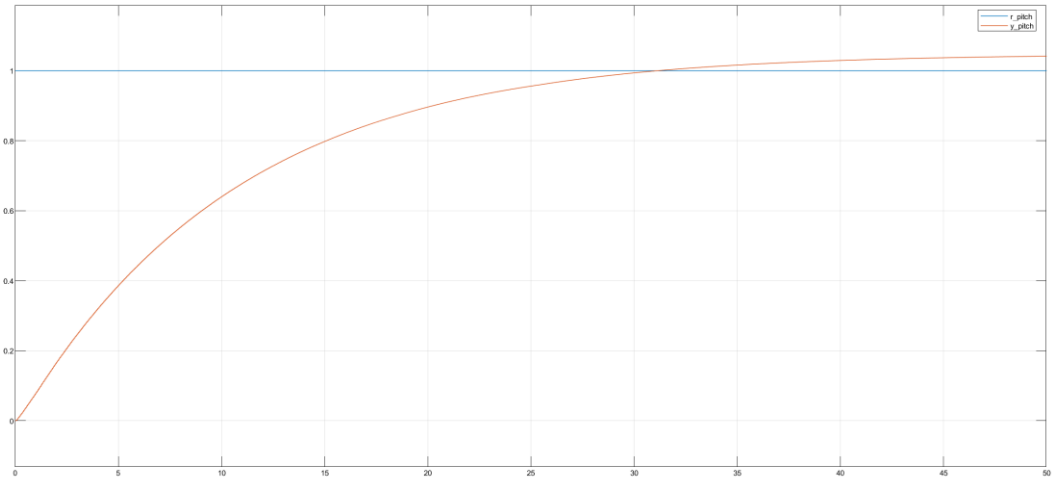


Figure 3.16. Response of the pitch signal related model in Simulink.

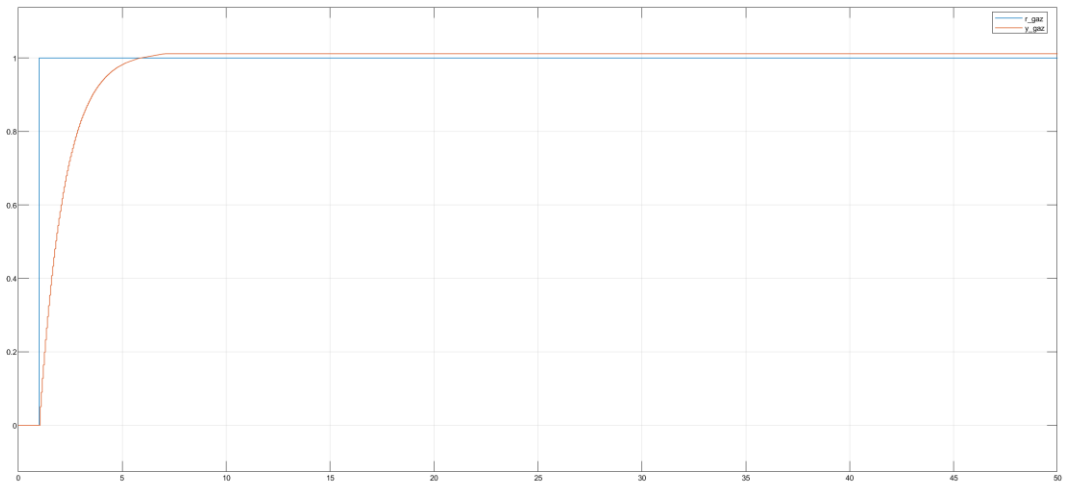


Figure 3.17. Response of the gaz signal related model in Simulink.

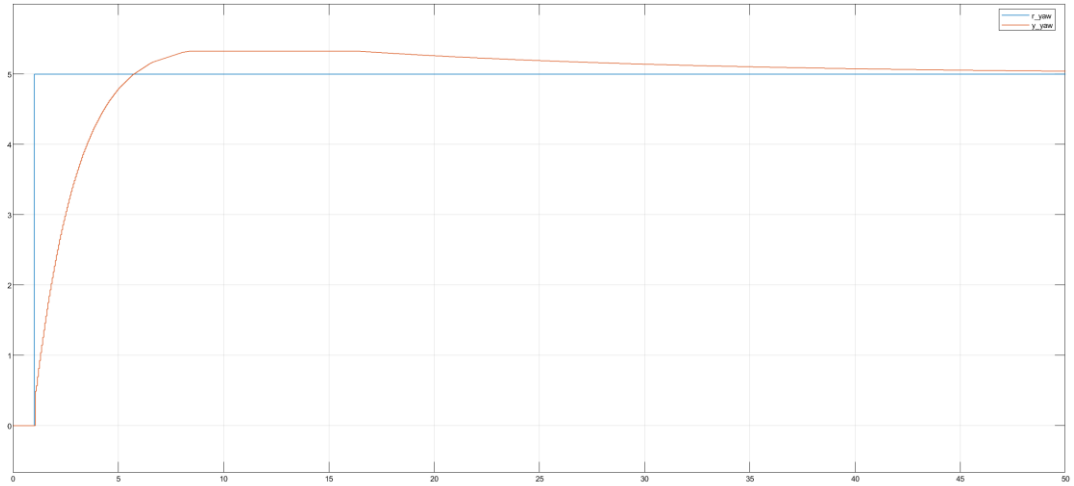


Figure 3.18. Response of the yaw signal related model in Simulink.

Although the simulated responses of the pitch and roll-related models may seem quite slow, this was done to accommodate the limited space inside the lab and to avoid accidents caused by high velocities and accelerations.

Even so, the weights of the roll and pitch PID parameters can be increased by adding a multiplier to those parameters. This adjustment can make the response faster without altering the relationship between the proportional, integral, and derivative weights.

After this, the control system was tested in Sphinx, as mentioned in section 2.2.3. Since it is not possible to simulate more than one drone in Sphinx, the pursuing behavior could not be simulated. Nevertheless, the control was simulated using the *sphinx_ros2* package, with the origin of the world frame in Sphinx representing the supposed pose of a detected BEBOP drone. This approach allowed for testing the functionality of the control strategy, which worked as expected, leaving us with the following PID weights and the corresponding multiplier values tested.

Control signal	**Kp**	**Ki**	**Kd**	**Multiplier Value**
Roll	6.2047	0.070493	43.2132	1
Pitch	5.5103	0.063691	52.0765	1
Gaz	85.9881	1.3046	3.8812	N/A
Yaw	6.1384	0.37509	1.194	N/A

Table 3.4. PID controllers' weights and multipliers.

4. Experiments and Results

This section presents the results obtained from running the system in a constrained environment at the Mechatronic Systems Lab of the Mechanical Engineering Building (MecE) at the University of Alberta. The ground truth positions of both the pursuing and pursued drones were obtained using Vicon Tracker software. The Vicon System was used to acquire the positions of both drones, which were then transmitted to the lab's local network, as detailed in sections 2.1.3 and 2.2.4. The experiments have been tested with 3 drones, an ANAFI as the pursuer and two BEBOP 2 as the targets and possible pursued drones.

The results include an evaluation of the accuracy of the pose estimation discussed in section 3.3 and the outcomes of the pursuit using both steady-state and dynamic pursuing control strategies, as detailed in sections 3.1.1.5. and 3.4.2.

4.1. Pose Estimation Accuracy

Using the eight keypoints extracted from the Keypoint R-CNN model of one of the tracked drones, the drone's pose is obtained using the Perspective-n-Point method, as described in section 3.3. However, time offsets occur due to the duration required to communicate and acquire frame data from the ANAFI drone, process the keypoint estimations and filtering, and obtain the pursued drone's pose. This results in a delay in pose acquisition due to the offset between the real-time position of the detected target and the system's response.

Additionally, there is a stationary error in the estimated position due to the discrepancy between the center position of the pursued drone considered by the Vicon Tracker software and the Perspective-n-Point method.

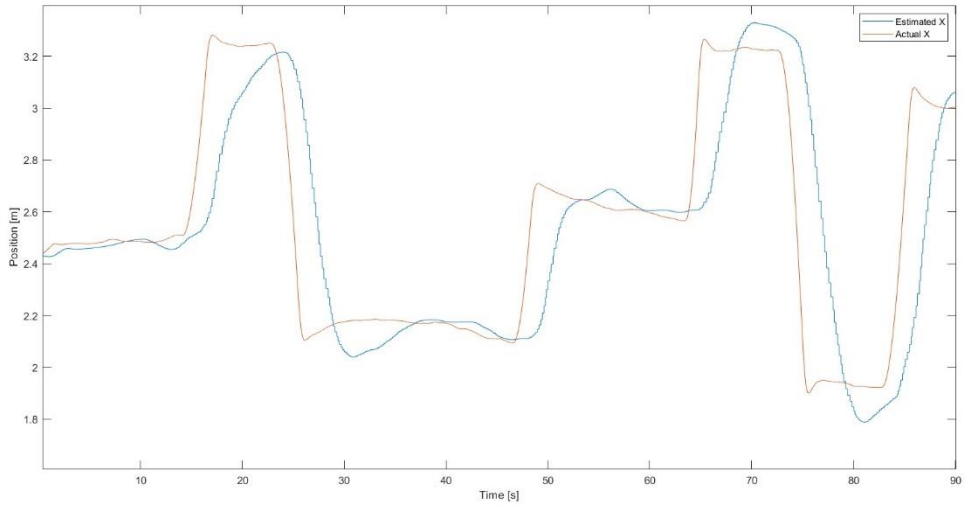


Figure 4.1. Estimated and real X position of the selected BEBOP drone.

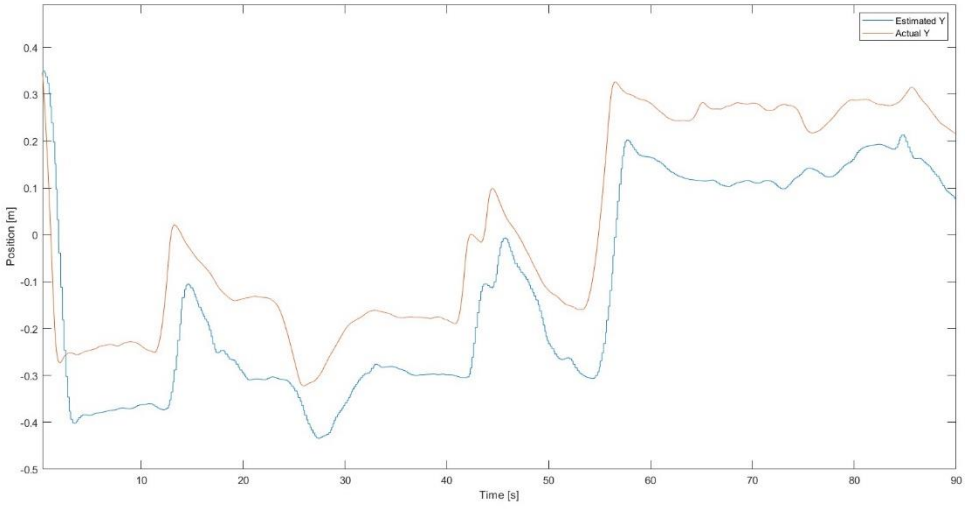


Figure 4.2. Estimated and real Y position of the selected BEBOP drone.

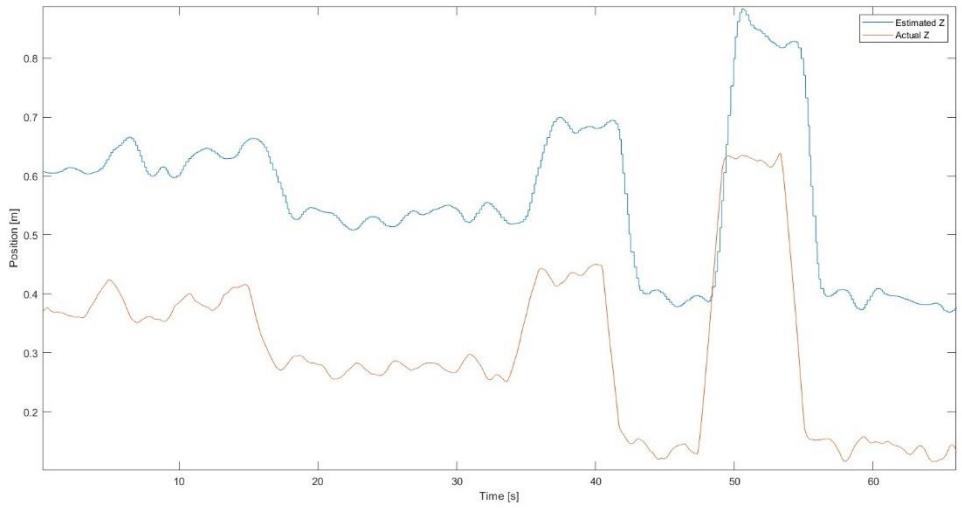


Figure 4.3. Estimated and real Z position of the selected BEBOP drone.

Assuming those offsets, Table 4.1 details the main absolute error of the pose estimation method, and the average delay time and offset position for each axis of the pose.

Axis	Average Time offset [s]	Average Position Error [m]
X	2.21	0.121
Y	2.21	0.040
Z	2.21	0.263

Table 4.1. Pose estimation results.

Considering these results, it is evident that the offset time between the real-time position of the pursued drone and the time taken to estimate that position is significant. This delay is caused by the detection of two drones and could increase if more drones enter the field of vision of the pursuing drone. The primary cause of this delay is the time required to extract the key points for each drone. Therefore, optimizing the Keypoint R-CNN to detect multiple drones more quickly should mitigate this problem.

4.2. Pursuing Assessment

In this section, both types of control systems will be tested while detecting to drones and then pursue one of them. However, due to the only position estimated is the position of the selected drone to be pursued, only the position of the pursued drone and the position of the pursuer drone will be taken into account. Additionally, the control of the yaw of the ANAFI drone will not be considered in the following experiments, as its sole purpose is to maintain a constant yaw value.

4.2.1. Steady State Pursuing

The steady-state pursuing control system estimates the initial position of the selected drone at the start of the pursuit. By utilizing the velocity states acquired by the ANAFI's sensors and integrating these velocities with the initial estimated position, the ANAFI is able to move toward the position of the target BEBOP drone. The results from this control strategy demonstrate the drone's ability to pursue a stationary drone and highlight the accuracy of the PID controller, as validated by the simulated results in Simulink. Additionally, the performance and reliability of the ANAFI sensors were confirmed.

It was observed that the drone experienced issues with controlling the yaw and accurately determining the current yaw orientation, leading to drifting while moving along the X, Y, and Z axes. Despite this, the control system was not significantly affected.

The next three figures demonstrate the results of applying this control strategy to the ANAFI drone while attempting to move to a position approximately 1 meter away from the pursued

drone along the ANAFI's three axes. However, for safety reasons, the ANAFI drone will maintain a safety distance along the X axis from the BEBOP drone.

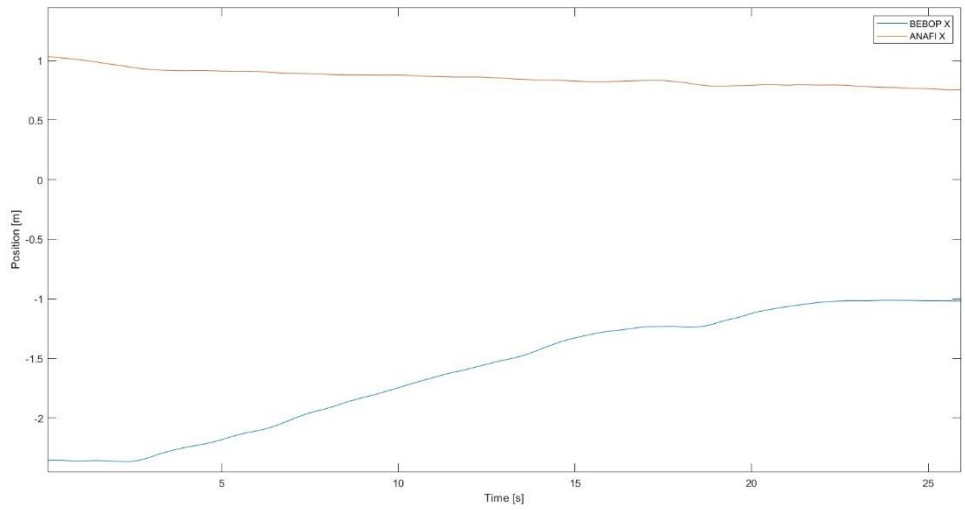


Figure 4.4. Position of the ANAFI (pursuer) and BEOP (pursued) along the X axis in a steady state pursuit.

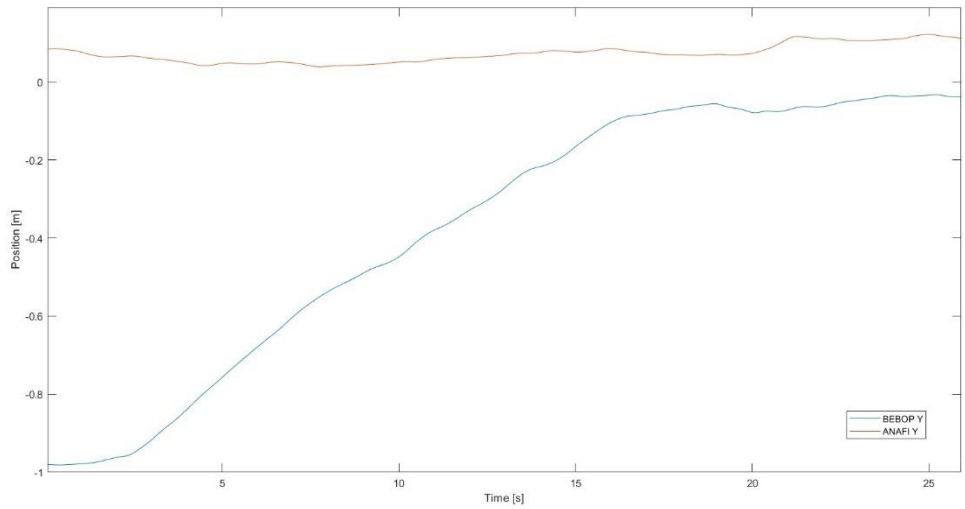


Figure 4.5. Position of the ANAFI (pursuer) and BEOP (pursued) along the Y axis in a steady state pursuit.

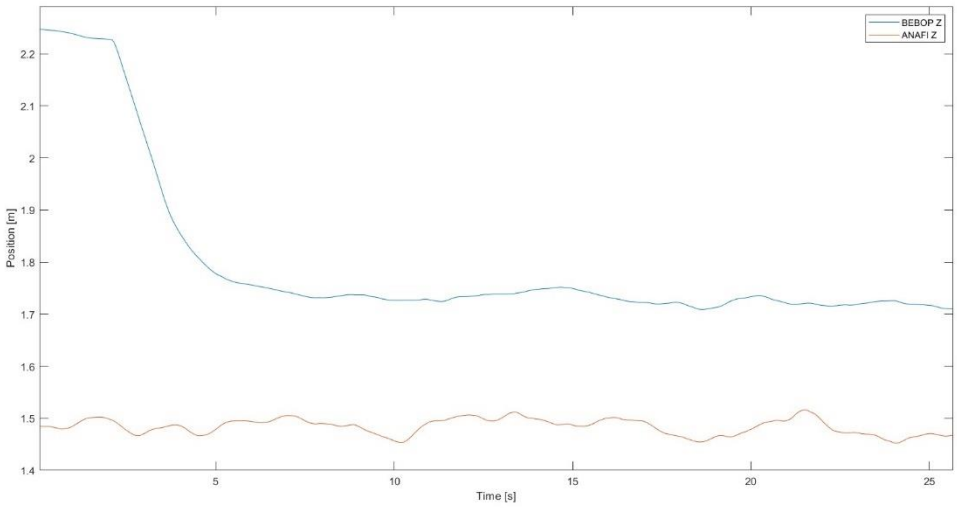


Figure 4.6. Position of the ANAFI (pursuer) and BEOP (pursued) along the Z axis in a steady state pursuit.

As observed, there is an offset position similar to what is described in section 4.1. Figure 4.4 illustrates how the ANAFI maintains a 2-meter safety distance from the pursued drone after reaching the desired position. In all three axes, the desired position is reached and maintained by the controller, indicating the correct performance of the system.

Comparing these results with those obtained in the simulations while testing the PID controllers in Simulink, both tests aimed to reach a position 1 meter away in each axis. We observe that the time taken to reach the desired position in both the simulation and reality is similar. This reflects the effective design and modeling of ANAFI's signal-related models, as discussed in section 3.4.1.

Table 4.2 compares the settling time in simulations seen in section 3.4.3, to the settling time in real-world tests. However, the real-world acquisition of the settling time is affected by the offset position, resulting in a reduction along the Z-axis due to a displacement of only 0.5

meters in the test. The results for a 1-meter displacement in the X and Y axes indicate a reduction in the settling time for this test.

Axis	Settling time [s]	
	Simulation	Reality
X	17.68	15.90
Y	23.25	15.21
Z	2.6	1.73

Table 4.2. Comparison of the settling time between the simulation and reality.

4.2.2. Dynamic Pursuing

The dynamic pursuit is the final stage of the results and the main focus of this thesis. It involves detecting and tracking multiple drones, followed by pursuing one of them while it moves in 3D space. Figure 4.7 shows the trajectory followed by the pursued (BEBOP) and pursuer (ANAFI) drones in the experiment.

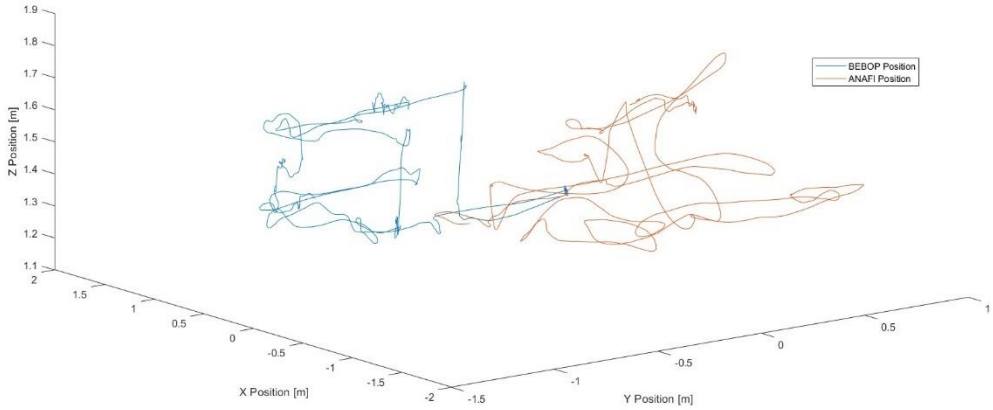


Figure 4.7. Trajectory followed by the ANAFI and the BEBOP during the experiment.

In the static state pursuit experiment, the ANAFI drone maintains a 2-meter safety distance from the BEBOP drone along the X-axis while attempting to match its position along the Y and Z axes. Due to the drift mentioned in the previous section, which caused the drone to rotate around the Z-axis unintentionally, and due to position errors and the time required for the drone to estimate its position, the stationary error becomes more pronounced in the dynamic pursuit experiment. This sometimes causes the drone to oscillate while maintaining its position.

This phenomenon is particularly noticeable in the control of the X-axis while maintaining the safety distance, as shown in Figure 4.8. It can be observed that when the drone tries to maintain the safety distance, a slight oscillation occurs. However, in the control of all three axes, when there is a significant change in the BEBOP's position, the ANAFI drone pursues effectively.

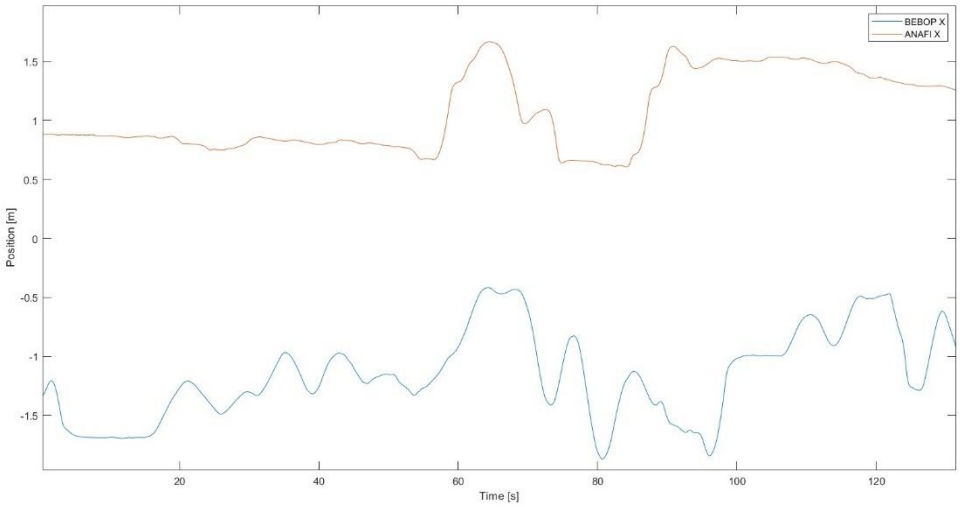


Figure 4.8. Position of the ANAFI (pursuer) and BEOP (pursued) along the X axis in a dynamic pursuit.

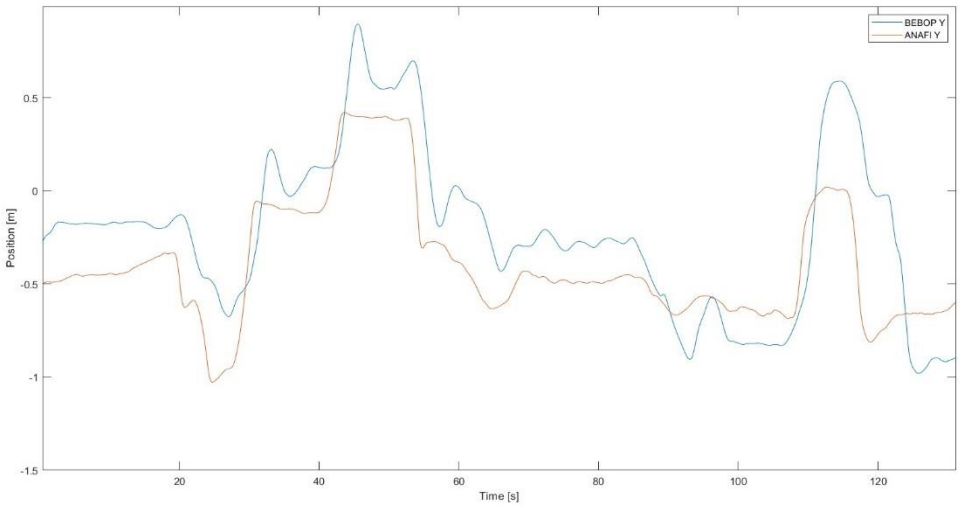


Figure 4.9. Position of the ANAFI (pursuer) and BEOP (pursued) along the Y axis in a dynamic pursuit.

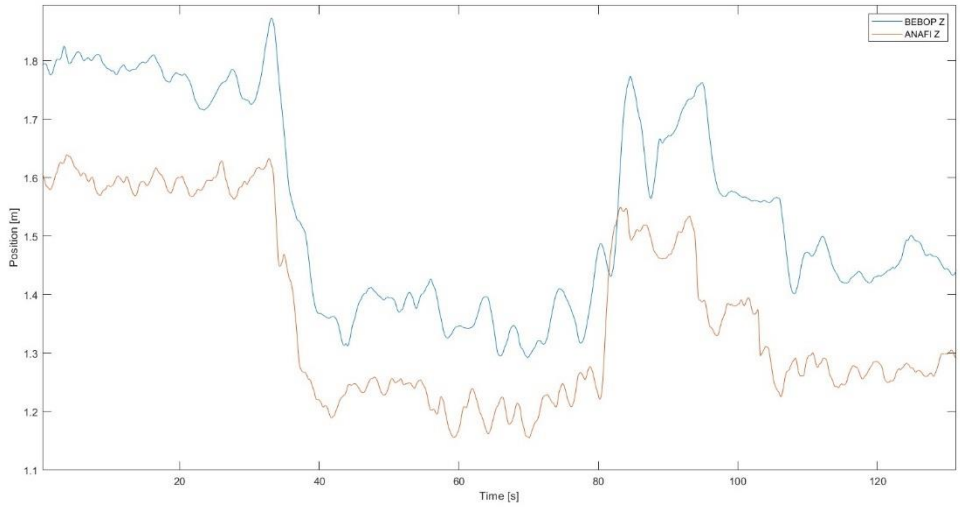


Figure 4.10. Position of the ANAFI (pursuer) and BEOP (pursued) along the Z axis in a dynamic pursuit.

Considering the previously obtained results, the average stationary absolute error (ASE) propagated by the offset and calculated by resting both signals, which prevents the pursuer

drone from tracking the exact position along the axes, as well as the average dynamic absolute error (ADE) calculated by comparing both signals and resting the ASE, caused by movement during the pursuit and the control system, and the maximum absolute error (ME), are presented in Table 4.3.

Axis	ASE [m]	ADE [m]	ME [m]
X	0.317	0.312	1.339
Y	0.266	0.205	0.995
Z	0.176	0.039	0.374

Table 4.3. Position errors during the dynamic pursuing experiment.

The results obtained show that while maintaining relatively low average stationary and dynamic errors, the error occasionally spikes to a high value that needs correction. Despite these errors, the pursuit is generally accurate and functions as expected most of the time.

5. Conclusion

This thesis has demonstrated the feasibility of implementing a real-time multi-drone tracking and pursuing algorithm using deep learning methods and a PID-based control system. The initial objectives have been achieved, yielding satisfactory results and exhibiting robust performance in a controlled environment.

Throughout the development of this thesis, numerous challenges were encountered. One of the most significant challenges was identifying each detected drone as distinct from others across a sequence of received frames. This issue was resolved using the SORT algorithm, which assigns IDs to each drone detected and segmented by a Mask R-CNN model, with positions estimated using Kalman filters. Additionally, a Keypoint R-CNN model was employed to extract specific points from each drone, allowing position estimation through the PnP method. This process culminated in the design of a PID controller to pursue the desired drones. All these tasks were integrated into ROS 2 nodes and executed together.

An exhaustive analysis of the system's performance was conducted by examining data obtained from the pursuing drone and the Vicon System. Despite minor errors during pursuit, the results confirm the algorithm's effective performance.

6. Future work

Despite the algorithm's correct performance, several improvements could be pursued in future work, ranging from the detection process to the control system.

Focusing on detection and tracking, a new Keypoint-RCNN model could be trained to detect multiple drones with varying appearances, as well as other types of flying objects, to differentiate between them. This training should utilize a dataset containing data from all viewpoints of these objects for optimal keypoint extraction. Additionally, a new method involving Kalman filters to track and assign IDs to drones using the extracted keypoints could be developed, resulting in more accurate tracking.

The control system could also be improved by implementing an observer-based control strategy, which involves accurately estimating the model of the drone used for pursuit. This approach would enable the design of a more precise control system.

An interesting area of research would be the generation of trajectories for multiple drones with the purpose of collision avoidance. This would involve considering the positions of all detected drones to create trajectories that avoid collisions or automatically select the nearest drone to pursue.

In conclusion, there are numerous improvements and intriguing research areas to explore, offering significant development opportunities in the field addressed by this thesis.

Bibliography

1. VISONTAY, Elias. Australia building air traffic control system for drones ahead of influx of ‘flying taxis’ | Air transport | The Guardian. Online. 2024. [Accessed 16 June 2024]. Available from: https://www.theguardian.com/world/2024/mar/04/australia-building-air-traffic-control-system-for-drones-ahead-of-influx-of-flying-taxis

2. BALASUBRAMANIAN, Sai. Drones May Become ‘The Next Big Thing’ In Healthcare Delivery. Online. 2022. [Accessed 16 June 2024]. Available from: https://www.forbes.com/sites/saibala/2022/01/09/drones-may-become-the-next-big-thing-in-healthcare-delivery/

3. SAVAGE, Steven. Farm With A View: How Drone Technology Is Taking Agriculture To A New Level. Online. 2023. [Accessed 16 June 2024]. Available from: https://www.forbes.com/sites/stevensavage/2023/02/23/farm-with-a-view-how-drone-technology-is-taking-agriculture-to-a-new-level/

4. Amazon drones: Prime Air expands drone deliveries after FAA approval. Online. 2024. [Accessed 16 June 2024]. Available from: https://www.aboutamazon.com/news/transportation/amazon-drone-prime-air-expanded-delivery-faa-approval

5. Parrot ANAFI | Professional Drone Camera 4K HDR. Online. [Accessed 16 June 2024]. Available from: https://www.parrot.com/en/drones/anafi

6. Parrot Anafi 4K HDR Drone Review: Can’t Catch the Competition | WIRED. Online. [Accessed 16 June 2024]. Available from: https://www.wired.com/review/parrot-anafi-4k-hdr-drone/

7. Parrot Bebop drone downloads | Parrot Support Center. Online. [Accessed 16 June 2024]. Available from: https://www.parrot.com/en/support/documentation/bebop-range

8. Parrot Bebop 2 review: fun, fine, and fatally flawed - The Verge. Online. [Accessed 16 June 2024]. Available from: https://www.theverge.com/2016/1/22/10814282/parrot-bebop-2-drone-review

9. About Us | The Vicon Difference | Motion Capture Systems. Online. [Accessed 17 June 2024]. Available from: https://vicon.com/about-us/

10. Robot Operating System - Wikipedia. Online. [Accessed 17 June 2024]. Available from: https://en.wikipedia.org/wiki/Robot_Operating_System

11. MARUYAMA, Yuya, KATO, Shinpei and AZUMI, Takuya. Exploring the performance of ROS2. In : *Proceedings of the 13th International Conference on Embedded*

Software, EMSOFT 2016. Association for Computing Machinery, Inc, 1 October 2016. ISBN 9781450344852. DOI 10.1145/2968478.2968502.

12. ROS 2 Documentation — ROS 2 Documentation: Foxy documentation. Online. [Accessed 17 June 2024]. Available from: https://docs.ros.org/en/foxy/index.html

13. GitHub - OPT4SMART/ros2-vicon-receiver: Vicon Receiver for ROS2. Online. [Accessed 17 June 2024]. Available from: https://github.com/OPT4SMART/ros2-vicon-receiver/tree/master

14. ROS Package: cv_bridge. Online. [Accessed 17 June 2024]. Available from: https://index.ros.org/p/cv_bridge/

15. Overview and usage of RQt — ROS 2 Documentation: Foxy documentation. Online. [Accessed 17 June 2024]. Available from: https://docs.ros.org/en/foxy/Concepts/About-RQt.html

16. Olympe 7.7 - Olympe Documentation. Online. [Accessed 17 June 2024]. Available from: https://developer.parrot.com/docs/olympe/index.html

17. What is Parrot Sphinx - 2.15.1. Online. [Accessed 18 June 2024]. Available from: https://developer.parrot.com/docs/sphinx/index.html

18. Gazebo. Online. [Accessed 18 June 2024]. Available from: https://gazebosim.org/home

19. The most powerful real-time 3D creation tool - Unreal Engine. Online. [Accessed 18 June 2024]. Available from: https://www.unrealengine.com/en-US

20. Tracker | Delivering Precise Real-World Data | Motion Capture Software. Online. [Accessed 18 June 2024]. Available from: https://vicon.com/software/tracker/

21. Work with objects - Tracker 4.1 documentation - Vicon Help. Online. [Accessed 18 June 2024]. Available from: https://help.vicon.com/space/Tracker41/14320634/Work+with+objects

22. OpenCV - Open Computer Vision Library. Online. [Accessed 18 June 2024]. Available from: https://opencv.org/

23. PyTorch. Online. [Accessed 18 June 2024]. Available from: https://pytorch.org/

24. PyTorch - Wikipedia. Online. [Accessed 18 June 2024]. Available from: https://en.wikipedia.org/wiki/PyTorch

25. CUDA Toolkit - Free Tools and Training | NVIDIA Developer. Online. [Accessed 18 June 2024]. Available from: https://developer.nvidia.com/cuda-toolkit

26. CUDA - Wikipedia. Online. [Accessed 18 June 2024]. Available from: https://en.wikipedia.org/wiki/CUDA

27. Facebook open sources Detectron - Meta Research | Meta Research. Online. [Accessed 18 June 2024]. Available from: https://research.facebook.com/blog/2018/01/facebook-open-sources-detectron/

28. Detectron2. Online. [Accessed 18 June 2024]. Available from: https://ai.meta.com/tools/detectron2/

29. HE, Kaiming, GKIOXARI, Georgia, DOLLÁR, Piotr and GIRSHICK, Ross. *Mask R-CNN*. [no date].

30. MATLAB. Online. [Accessed 18 June 2024]. Available from: https://www.mathworks.com/products/matlab.html

31. detectron2/MODEL_ZOO.md at main · facebookresearch/detectron2 · GitHub. Online. [Accessed 19 June 2024]. Available from: https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md

32. HE, Kaiming, ZHANG, Xiangyu, REN, Shaoqing and SUN, Jian. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Online. 9 December 2016. Vol. 2016-December, p. 770–778. [Accessed 28 June 2024]. DOI 10.1109/CVPR.2016.90.

33. GitHub - KaimingHe/deep-residual-networks: Deep Residual Learning for Image Recognition. Online. [Accessed 28 June 2024]. Available from: https://github.com/KaimingHe/deep-residual-networks

34. GitHub - abewley/sort: Simple, online, and realtime tracking of multiple objects in a video sequence. Online. [Accessed 21 June 2024]. Available from: https://github.com/abewley/sort

35. BEWLEY, Alex, GE, Zongyuan, OTT, Lionel, RAMOS, Fabio and UPCROFT, Ben. Simple online and realtime tracking. *Proceedings - International Conference on Image Processing, ICIP*. Online. 3 August 2016. Vol. 2016-August, p. 3464–3468. [Accessed 21 June 2024]. DOI 10.1109/ICIP.2016.7533003.

36. EBRAHIMNEZHAD, Amir Hossein. *Deep Learning in Autonomous UAV Pursuit*. . 2023.

37. keypointrcnn_resnet50_fpn — Torchvision main documentation. Online. [Accessed 21 June 2024]. Available from: https://pytorch.org/vision/main/models/generated/torchvision.models.detection.keypointrcnn_resnet50_fpn.html

38. GUTIÉRREZ, Barranco, ISRAEL, Alejandro, DE JESÚS, José and JUÁREZ, Medel. Digital Camera Calibration Analysis Using Perspective Projection Matrix.

39. OpenCV: Perspective-n-Point (PnP) pose computation. Online. [Accessed 23 June 2024]. Available from: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html

40. QUAN, Long, LAN, Zhong-Dan and LAN, Zhongdan. Point Camera Pose Determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Online. 1999. Vol. 21, no. 8. DOI 10.1109/34.784291ï.

41. GAO, Xiao-Shan, HOU, Xiao-Rong, TANG, Jianliang and CHENG, Hang-Fei. *Complete Solution Classification for the Perspective-Three-Point Problem*. [no date].

42. LEPETIT, Vincent, MORENO-NOGUER, Francesc and FUA, Pascal. EPnP: An accurate O(n) solution to the PnP problem. *International Journal of Computer Vision*. February 2009. Vol. 81, no. 2, p. 155–166. DOI 10.1007/s11263-008-0152-6.

43. DERPANIS, Konstantinos G. *Overview of the RANSAC Algorithm*. 2010.

44. OpenCV: Camera Calibration and 3D Reconstruction. Online. [Accessed 23 June 2024]. Available from: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#gga357634492a94efe8858d0ce1509da869acbd7d9f9513a22a59412661a9d31ca3d

45. EADE, Ethan. *Gauss-Newton / Levenberg-Marquardt Optimization*. 2013. 1 Definitions Let x ∈ X be the state parameters to be optimized, with n degrees of freedom. The goal of the optimization is to maximize the likelihood of a set of observations given the parameters, under a specified observation model.

46. Multiple View Geometry in Computer Vision - Richard Hartley, Andrew Zisserman - Google Books. Online. [Accessed 24 June 2024]. Available from: https://books.google.ca/books?id=si3R3Pfa98QC&printsec=frontcover#v=onepage&q&f=false

47. HAMILTON, James D. *STATE-SPACE MODELS**. 1994.

48. DUARTE, Rafael. Low cost Brain Computer Interface system for AR.Drone Control. Online. 2017. DOI 10.13140/RG.2.2.21000.32008. The user has requested enhancement of the downloaded file.

49. Ardrone3.Piloting and SpeedSettings - 7.7. Online. [Accessed 24 June 2024]. Available from: https://developer.parrot.com/docs/olympe/arsdkng_ardrone3_piloting.html#olympe.messages.ardrone3.SpeedSettingsState.MaxPitchRollRotationSpeedChanged

50. Convert model from continuous to discrete time - MATLAB c2d. Online. [Accessed 24 June 2024]. Available from: https://www.mathworks.com/help/ident/ref/dynamicsystem.c2d.html#mw_53fc4689-2099-41d0-93b3-de1e51a174c1

Appendix A: ROS 2 Packages

The packages developed for the algorithm, including the nodes and launch files containing the code, are available on GitLab with the appropriate permissions:

https://gitlab.com/barczyk-mechatronic-systems-lab/anafi_ros2

Appendix B: Model estimation algorithms

```
%% STATES and INPUTS
inputs = [u_roll, u_pitch, u_gaz, u_yaw];
states = [x_pos, y_pos, z_pos, yaw, vel_x, vel_y, vel_z, vel_yaw];

%% PLANT DESIGN

Real_Pos = states(:,5);
Step_signal = inputs(:,1);

P0 = [0 0];

Result = fminsearch(@est_model,P0)

function r=est_model(P)
global Real_Pos Step_signal time Ts model_d
Ts = 0.05;

A = [1 0;
    0 P(1)];

B = [0;
    P(2)];

C = [0 1];

D = 0;

model = ss(A,B,C,D);
model_d = c2d(model, Ts)
y=lsim(model_d,Step_signal);

%---- plot------
yyaxis right;
hold off
plot(time,Step_signal,'k-');
ylim([-40, 40]);

yyaxis left;
plot(time,Real_Pos,'b-');
hold on

plot(time,y,'r-');
legend('Real state', 'Model state', 'Control signal');
drawnow()
%----end plot------

V=Real_Pos-y;
r=norm(V(1:end));

return

end

```

Figure A.1. MATLAB Algorithm for estimating the displacement along the Y axis (roll) of the drone.

```
%% STATES and INPUTS
inputs = [u_roll, u_pitch, u_gaz, u_yaw];
states = [x_pos, y_pos, z_pos, yaw, vel_x, vel_y, vel_z, vel_yaw];

%% PLANT DESIGN

Real_Pos = states(:,6);
Step_signal = inputs(:,2);

P0 = [0 0];

Result = fminsearch(@est_model,P0)

function r=est_model(P)
global Real_Pos Step_signal time Ts model_d
Ts = 0.05;

A = [1 0;
    0 P(1)];

B = [0;
    P(2)];

C = [0 1];

D = 0;

model = ss(A,B,C,D);
model_d = c2d(model, Ts)
y=lsim(model_d,Step_signal);

%---- plot------
yyaxis right;
hold off
plot(time,Step_signal,'k-');
ylim([-40, 40]);

yyaxis left;
plot(time,Real_Pos,'b-');
hold on

plot(time,y,'r-');
legend('Real state', 'Model state', 'Control signal');
drawnow()
%----end plot------

V=Real_Pos-y;
r=norm(V(1:end));

return

end
```

Figure A.2. MATLAB Algorithm for estimating the displacement along the X axis (pitch) of the drone.

```
%% CONTROL STATES
inputs = [u_roll, u_pitch, u_gaz, u_yaw];
states = [x_pos, y_pos, z_pos, yaw, vel_x, vel_y, vel_z, vel_yaw, acc_x, acc_y, acc_z,
acc_yaw];

%% PLANT DESIGN

Real_Pos = states(:,4);
Step_signal = inputs(:,4);

P0 = [0 0];

Result = fminsearch(@est_model,P0)

function r=est_model(P)
global Real_Pos Step_signal time Ts model_d
Ts = 0.05;

A = [P(1)];

B = [P(2)];

C = 1;

D = 0;

model = ss(A,B,C,D);
model_d = c2d(model, Ts)
y=lsim(model_d,Step_signal);

%---- plot------
yyaxis right;
hold off
plot(time,Step_signal,'k-');
ylim([-120, 120]);
legend('1');

yyaxis left;
plot(time,Real_Pos,'b-');
hold on

plot(time,y,'r-');
legend('Real state', 'Model state', 'Control signal');
drawnow()
%----end plot------

V=Real_Pos-y;
r=norm(V(1:end));

return

end

```

Figure A.3. MATLAB Algorithm for estimating the orientation around the Z axis (yaw) of the drone.

```
%% CONTROL STATES
inputs = [u_roll, u_pitch, u_gaz, u_yaw];
states = [x_pos, y_pos, z_pos, yaw, vel_x, vel_y, vel_z, vel_yaw, acc_x, acc_y, acc_z,
acc_yaw];

%% PLANT DESIGN

Real_Pos = states(:,3);
Step_signal = inputs(:,3);

P0 = [0 0];

Result = fminsearch(@est_model,P0)

function r=est_model(P)
global Real_Pos Step_signal time Ts model_d
Ts = 0.05;

A = [P(1)];

B = [P(2)];

C = 1;

D = 0;

model = ss(A,B,C,D);
model_d = c2d(model, Ts)
y=lsim(model_d,Step_signal);

%---- plot------
yyaxis right;
hold off
plot(time,Step_signal,'k-');
ylim([-120, 120]);
legend('1');

yyaxis left;
plot(time,Real_Pos,'b-');
hold on

plot(time,y,'r-');
legend('Real state', 'Model state', 'Control signal');
drawnow()
%----end plot------

V=Real_Pos-y;
r=norm(V(1:end));

return

end

```

Figure A4. MATLAB Algorithm for estimating the displacement along the Z axis (gaz) of the drone.