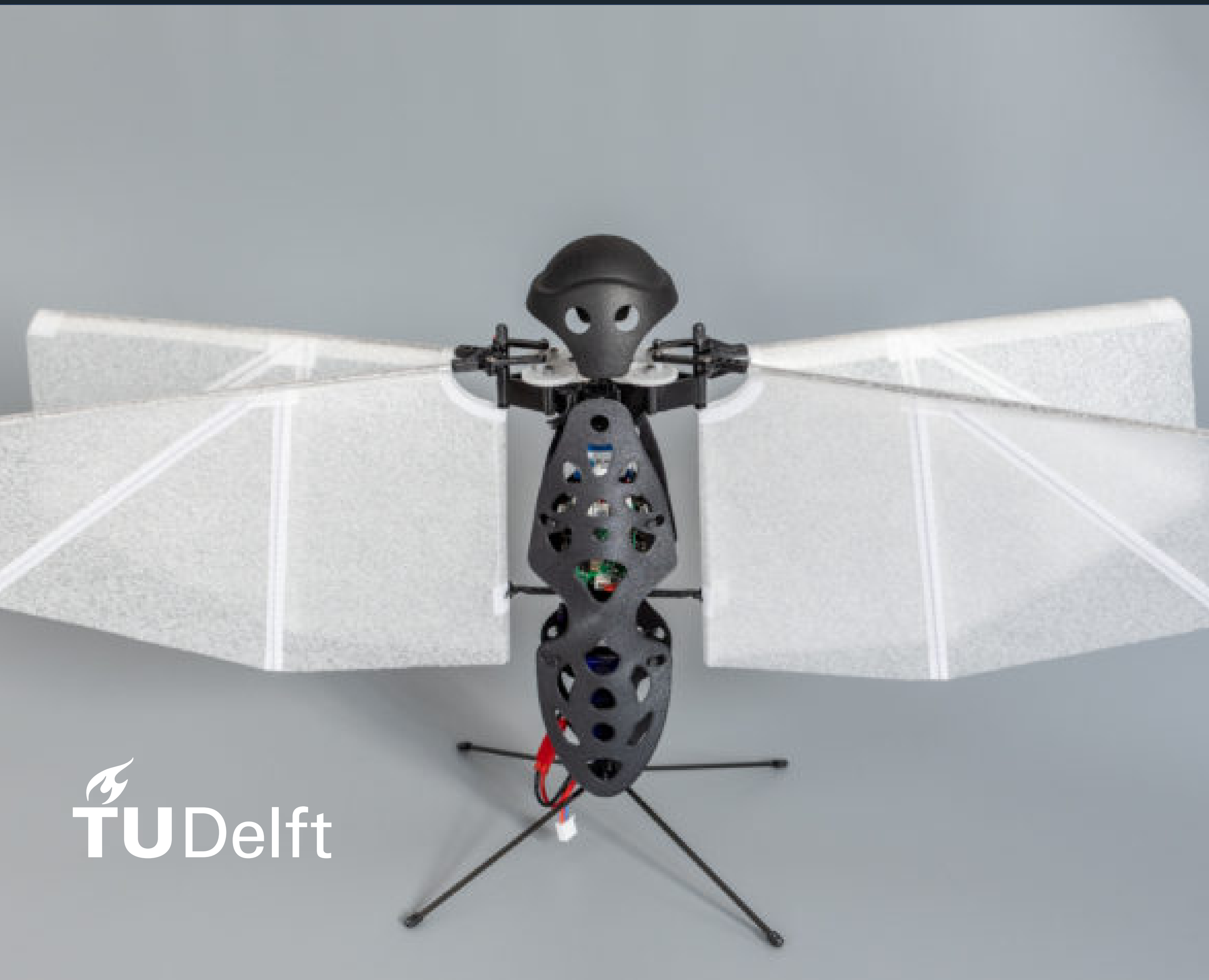
Autonomous Navigation for an Attitude-Stable Flapping Wing Air Vehicle

Obstacle Avoidance Strategies Using Time-of-Flight Sensors

Serban Blaga



Autonomous Navigation for an Attitude-Stable Flapping Wing Air Vehicle

Obstacle Avoidance Strategies Using Time-of-Flight Sensors

Thesis report

by

Serban Blaga

Supervisors:	Dr. G.C.H.E. (Guido) de Croon
	Dr.ir. C. (Christophe) de Wagter
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	April, 2024 - April, 2025
Student number:	4996666

Faculty of Aerospace Engineering · Delft University of Technology

Preface

This report marks the final step in my Master thesis project, carried out as part of the MSc Systems and Control program at TU Delft. The work focuses on autonomous navigation for a flapping wing aerial vehicle, a topic that is as challenging as it is exciting. Over the past months, this project has pushed me to grow both technically and personally. From dealing with unexpected setbacks to celebrating small breakthroughs, it has been an intense and rewarding journey.

I would first like to express my sincere gratitude to my supervisors, Guido de Croon and Christophe de Wagter, whose guidance, feedback, and encouragement throughout the project were invaluable. Their expertise and thoughtful insights consistently helped me navigate challenges and refine my ideas, and this thesis would not have been possible without their support. I am also deeply thankful to Shenqi Wang for his hands-on help with the hardware, and for patiently guiding me through the intricacies of the simulator and the underlying algorithms. His technical advice and willingness to assist at every stage of the process made a significant difference in my ability to move forward. I would also like to thank Mauro Gallo for his valuable insights and for consistently being present during meetings to help us stay on track and move toward our objective.

I am truly grateful to my friends who stood by me throughout this process. Whether it was through late-night calls, shared coffee breaks, or simple words of encouragement when things got overwhelming, your presence made a real difference. Thank you for keeping me grounded, helping me laugh when I needed it most, and reminding me to take breaks when I forgot how. I couldn't have done this without you.

To Roxi and Mara, thank you for always believing in me, loving me and for being the kind of friends anyone would be lucky to have. To Giorgio and Sergio, thank you for being like brothers to me these past years. You all mean the world to me.

I want to extend my deepest gratitude and love to my family for their unwavering support, patience, and belief in me. Your encouragement through every stage of this project, and beyond, has been my foundation. Thank you for giving me the space to grow, for understanding when I was too busy to call, and for always being there, no matter how far away. This achievement is as much yours as it is mine.

Delft, Tuesday 22nd April, 2025

Serban Blaga

Contents

Nomenclature

List of Abbreviations

AGV	Autonomous Ground Vehicle
CNN	Convolutional Neural Network
CPU	Central Processing Unit
EKF	Extended Kalman Filter
EMA	Exponential Moving Average
FLU	Forward-Left-Up (body frame convention)
FMAV	Flapping Micro Air Vehicle
FMCW	Frequency-Modulated Continuous Wave
FoV	Field of View
FWMAV	Flapping Wing Micro Air Vehicle
GAE	Generalized Advantage Estimation
GPU	Graphics Processing Unit
I^{2}C	Inter-Integrated Circuit
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging
MAV	Micro Air Vehicle
PID	Proportional–Integral–Derivative
PPO	Proximal Policy Optimization
RGB-D	Red Green Blue + Depth camera
RL	Reinforcement Learning
ROS	Robot Operating System
RTOS	Real-Time Operating System
SLAM	Simultaneous Localization and Mapping
SoC	System on Chip
SPI	Serial Peripheral Interface
ToF	Time-of-Flight
UAV	Unmanned Aerial Vehicle
URL	Uniform Resource Locator
VFH	Vector Field Histogram

List of Symbols

α	EMA smoothing constant
ΔE	Difference in EMA between left and right sensor regions
δ_t	TD error at time t
ϵ	Clipping threshold in PPO
γ	Discount factor
λ	Smoothing factor (for GAE)
$\mathbb{SO}(3)$	Special Orthogonal group (rotation space)
μ, σ	Mean and standard deviation (for Gaussian policies)
Ω	Angular velocity vector
ϕ	Roll angle
ψ	Yaw angle
τ	Torque vector
θ	Pitch angle
A_t	Advantage at time step t
b_x, b_y, b_z	Drag coefficients
d	Distance reading from ToF sensor
$e(t)$	Error at time t (in PID control)
K_d	Derivative gain
K_i	Integral gain
K_p	Proportional gain
R	Rotation matrix
R_t	Return (cumulative reward)
r_t	Probability ratio for PPO loss
T	Thrust
$u(t)$	Control input signal
$V(s)$	State value function
v_x, v_y, v_z	Velocity components in the body frame

List of Figures

List of Tables

1 Introduction

Quadrotors have been the standard in aerial robots for the past decade, and they have been used extensively in aerial photography, infrastructure inspection, agriculture, and military applications. Their ability to hover, make tight turns, and carry sensors of all types makes them an all-purpose device in research and industry. While they have been highly successful in open environments, new uses are emerging in more confined spaces, with greenhouses serving as a key example. Indoor production areas present their own unique set of spatial challenges, due to the closely spaced rows of plants and little room to maneuver between structures. Activities such as monitoring growth of crops, pest detection, or early recognition of diseases sometimes require flying close to the plant. In such scenarios, the spinning rotors of quadrotors can pose a risk, potentially damaging delicate plants upon contact. That implies a necessity for a more sheltered alternative. A flapping wing micro air vehicle (FWMAV) presents a promising solution. With its soft wings and inherently gentle flight modes, FWMAVs are able to carefully inspect without putting crops at risk. For efficient greenhouse operation, these platforms must also be compact in size, lightweight, and be able to navigate autonomously. This research focuses on enabling autonomous navigation in the Flapper Nimble+, an attitude-stable flapping wing air vehicle, through the use of Time-of-Flight (ToF) sensing combined with PID control and reinforcement learning.

To enable autonomous navigation on a flapping wing drone, this work first investigates practical sensor configurations that minimize onboard complexity. Due to strict limitations on payload and onboard computation, a minimal sensing setup using just two Time-of-Flight (ToF) sensors—one facing forward and one facing downward—is considered as the primary, lightweight configuration. In parallel, a more extensive setup using five ToF sensors arranged around the drone is also explored, primarily as a theoretical reference for understanding the potential benefits of increased environmental awareness. Both configurations use low-resolution 8×8 depth measurements with a four-meter range, providing only sparse and local observations. To evaluate these configurations and train navigation policies, a custom simulation environment is developed using Isaac Gym, NVIDIA's high-performance physics simulator. Based on the Aerial Gym[1] framework, the simulator models the dynamics of the Flapper Nimble+ and replicates realistic sensing constraints. Simulation plays a central role in this research, as flapping wing drones are sensitive to vibration and they are highly limited in terms of power supply and payload capacity. Within this environment, two control strategies are investigated. The first is a fully classical PID controller that governs both altitude and navigation. The second is a hybrid approach that uses PID to stabilize altitude while leveraging reinforcement learning (RL) for navigation and obstacle avoidance. This modular strategy enables RL to focus on high-level decision-making while relying on PID for consistent low-level control.

The report is structured as follows. Chapter 2 presents a literature review of MAV types and the various sensing and control strategies used for obstacle avoidance. Chapter 3 discusses the current state of the art in flapping-wing platforms and identifies the specific research gaps this work aims to address. Chapter 4 introduces the research question and frames the objectives. Chapter 5 describes the physical platform and sensor setup, while Chapter 6 covers the simulation tools and environment. Chapters 7 and 8 detail the two control approaches, classical PID and reinforcement learning, along with their implementation and rationale. Chapter 9 presents a thorough comparison of both approaches based on simulation results, and Chapter 10 concludes the work by summarizing key findings. Finally, Chapter 11 offers recommendations for future research and outlines potential real-world extensions of this work.

Part I

Literature Study

2 Current Models

This chapter provides a historical and technical overview of UAVs, tracing their progression into micro air vehicles and ultimately to flapping wing micro air vehicles. Emphasis is placed on the distinctions between tailed and tailless configurations, highlighting the unique challenges and advantages each presents. The goal is to frame the motivation for this work within the broader context of aerial robotics.

2.1. Background Information

Unmanned Aerial Vehicles (or in short UAVs) have been the subject of research for some time now. UAVs are highly versatile and they are deployed to perform a huge variety of tasks, from surveillance and reconnaissance, to agricultural applications, inspections and maintenance, photography and videography, all the way to entertainment.

They are aircraft systems without an onboard human pilot, operated remotely or autonomously. UAVs date back to the early 20th century, initially used for military applications. One of the earliest examples is the Kettering Bug, an experimental aerial torpedo developed during World War I.

The evolution of UAVs has been driven by advances in technology, including improvements in aerodynamics, propulsion systems, control mechanisms, and materials science. Significant progress was made during the Cold War, exemplified by the development of the Ryan Firebee, a target drone used for reconnaissance and surveillance [2].

Today, UAVs are used in a variety of applications, ranging from military and law enforcement to environmental monitoring, agriculture, and commercial delivery services. Technological advancements, particularly in sensor technology, data processing, and AI, have expanded their capabilities and applications [3].

2.2. Evolution of MAV

Classification of drones is based on several performance characteristics, including weight, wing span, range, and propulsion systems. Various classification schemes exist, offering different perspectives on categorizing drones. The operational purpose of the vehicle, materials used in fabrication, and complexity of the control system distinguish UAVs from other small drones like MAVs and NAVs.

UAVs come in a wide range of sizes and configurations, from those with a wing span comparable to a Boeing 737 to smaller radio-controlled drones. Mission capabilities play a significant role in categorizing UAVs, with distinctions made between HTOL, VTOL, hybrid models, helicopters, heli-wings, and unconventional types.

Micro UAVs (μUAVs), also known as small UAVs (SUAVs), mark a significant advancement in unmanned aerial vehicle (UAV) technology. These compact drones are designed to be easily portable, often launched by hand, and do not require a runway for takeoff. They represent a bridge between traditional UAVs and micro air vehicles (MAVs), offering increased mobility and versatility [4].

Micro UAVs come in various configurations, ranging from traditional fixed-wing designs to innovative models like ornithopters and cyclocopters. These configurations enable a diverse range of flight modes, including horizontal takeoff and landing (HTOL), vertical takeoff and landing (VTOL), hybrid models combining both, helicopters, and unconventional types [4].

Micro Air Vehicles (MAVs) represent a specialized category of small unmanned aerial vehicles (UAVs), characterized by their compact size and weight. Typically, MAVs are micro planes with lengths smaller than 100 cm and weights under 2 kg. These drones are classified into nine categories based on their design and functionality: fixed-wing, flapping wing, VTOL (Vertical Take-Off and Landing), rotary wing, tilt-rotor, ducted fan, helicopter, ornicopter, and unconventional types [4].

MAVs are equipped to carry a variety of sensors, including visual, acoustic, chemical, and biological sensors, making them versatile platforms for diverse applications. Their small size and capabilities have attracted attention from various disciplines, including aerospace, mechanical, electrical, and computer engineering. MAVs are designed for specialized missions in constrained environments where larger UAVs cannot operate effectively [5]. The aim in designing MAVs stands in achieving high agility, maneuverability and access to confined spaces. Their small size allows them to navigate indoor environments, dense vegetation, and urban areas with precision.

2.3. Flapping Wing Micro Air Vehicles

In general, the design of flapping wing MAVs (shortly FWMAVs) is inspired from birds, insects or organisms. Flapping wing micro air vehicles (MAVs) have flexible, lightweight wings, inspired by birds and insects, which are crucial for their aerodynamic efficiency and flight stability. These MAVs present more complex aerodynamics than fixed and rotary wing aircraft. Extensive research by biologists and aerospace engineers has focused on understanding the unique maneuverability advantages of flapping wings.

What is particularly remarkable in nature is its proficiency in the science of engineering design. Studies like [6] show that the insects' wings are one of the best examples of design perfection. This is due to their super-lightweight composite structures with complex configurations on both the macro and micro levels. These wings, which are the organs of flight, enable the insect to perform very smooth and stable flights with remarkable agility. Insect-inspired, hoverable FWMAVs present a low cost, low noise, and strong concealment and are flexible [7]. It performs very well in environmental monitoring due to its ability to enter narrow, complex and dangeroud areas [8].

Throughout the years, a variety of FWMAVs were developed. [9] introduced the *Microbat*, the first electronically powered palm-sized ornithopter. This is a tailed FWMAV that achieved a flight duration of 42 seconds using a radio control system. In a study by [10], the *Mentor* FWMAV was developed. This study presents a tailed flapper with a clap-fling mechanism that produces a higher value of thrust to power as the wings fling apart, carrying a so-called ”super circulation”. [11] introduced a tailless FWMAV, called the *Robobee*. This insect-scale flapping-wing robot is loosely modeled on the morphology of flies and demonstrated a tethered but unconstrained stable hovering and basic controlled flight. [12] introduced a small hovering ornithopter called the *Nano Hummingbird*. It has the ability to hover for several minutes and fly forward up to 6.7 m/s. It is also capable of transmitting live color video to a remote ground station. [13] developed the DelFly II, a tailed hovering FWMAV, with its successor, DelFly Nimble, developed by [14]. This is a programmable and agile autonomous free-flying robot, which presents no tail, making it more agile both around hover and in fast forward flight. Apart from these, a wide variety of FWMAVs were developed throughout the years, each with its distinctive characteristic. Table 2.1 presents a summary of these innovations.

2.3.1. Tailed vs Tailless FWMAV

As seen in the previous sections, one of the classifications that the FWMAVs can fall under is the tail configuration: if it is tailless or tailed. On one side, tailed flapping wing robots are passively stable (with the stabilization and control done by the tail) and vibration damped, but their flight envelope is limited to forward flight. On the other side, the tailless flapping wing robots have higher agility and a wide flight envelope from hovering to fast forward flight, but require active stabilization.

As discussed in Section 2.3, in nature both birds and insects flap their wings to produce flight forces. The absence of a tail in insects allow them to perform precise hovering flight, a feat most birds can not achieve. This feature allows tailless flapping wing robots to operate in confined spaces, such as collapsed buildings, hazardous facilities, greenhouses, etc. However, the absence of a tail makes the flight of the vehicle inherently unstable.

Therefore, since it is desired to achieve a insect-like behaviour in the FWMAVs with as much agility as

Table 2.1: Comparison of insect-inspired, hoverable FWMAVs.

Research	FWMAVs	Year	Wing Span (cm)	Mass (g)	Flight Endurance	Wings	Tail
[12]	Nano Hummingbird	2011	16.5	19	4 min	2	Yes
[15]	Robotic Hummingbird	2015	30	62	5 s	2	No
[16]	KU-Beetle	2016	18	21	40 s	2	No
[17]	Colibri	2017	22	21	20 s	2	No
[18]	NUSRoboticbird	2018	22	27	3.5 min	4	No
[19]	Insect-like FWMAV	2018	15	10.8	-	2	No
[13]	DelFly II	2018	33	28.2	5 min	4	Yes
[14]	DelFly Nimble	2018	17	20.4	20 s	2	No

possible and the possibility to operate in unfavourable conditions, the motivation is to develop a tailless FWMAV capable of detecting and avoiding obstacles.

3 Current State-Of-The-Art

This chapter presents a comprehensive review of the current state-of-the-art in obstacle avoidance techniques, sensor technologies, and control architectures, with a focus on approaches applicable to resource-constrained aerial platforms. The goal is to identify methods that offer robust navigation capabilities in complex and dynamic environments while satisfying the stringent size, weight, and power (SWaP) limitations inherent to FWMAVs. Various strategies—from classical rule-based algorithms to modern learning-based methods—are explored and compared, providing the foundational context for the control approach proposed in this thesis.

3.1. General Information

The development of micro aerial vehicles, particularly flapping wing MAVs, has opened new frontiers in aerial robotics, offering unique advantages in maneuverability and efficiency inspired by natural flyers. As these MAVs increasingly find applications in diverse and complex environments, from urban search and rescue missions to environmental monitoring and agricultural surveillance, the ability to effectively detect and avoid obstacles becomes paramount. Obstacle avoidance is critical not only for the safety and longevity of these vehicles but also for the success and reliability of their missions. Effective obstacle avoidance enhances the MAV's operational capabilities, allowing it to maneuver through cluttered environments, adapt to dynamic changes, and perform precise tasks without human intervention. This capability is particularly vital in environments where GPS signals are weak or unavailable, and in missions requiring close-proximity navigation. Thus, there is an obvious need to choose the most suitable obstacle avoidance method to maximize the efficiency of the MAV and allow it to perform its tasks in a smooth manner.

The obstacle avoidance task can be broken down in several subtasks, as Figure 3.1 shows. The first focus is choosing the type of control architecture, which will be discussed in Section 3.1.1, choosing the appropriate sensors and extracting the information from the chosen sensors to accurately represent the (immediate) surrounding environment.

3.1.1. Control Architecture

Figure 3.2 shows a simple overview of the different control architectures that can be used in obstacle avoidance.

A reactive avoidance control strategy is preferable over deliberative and hybrid approaches for a tailless flapping wing micro air vehicle, limited in computational power and equipped with a time of flight sensor, due to its real-time responsiveness and simplicity. Reactive control systems enable immediate reactions to obstacles based on direct sensor inputs, making them well-suited for dynamic environments where obstacles can appear suddenly or move unpredictably. Deliberative systems, which require extensive planning and computational resources to generate avoidance strategies based on a map or predictive models, would be impractical given the vehicle's constraints. Hybrid approaches, while combining elements of both reactive and deliberative control, still necessitate significant computational overhead and may not provide tangible benefits that justify their complexity in this context. Therefore, a reactive approach ensures efficient and effective object detection and avoidance while optimizing the use of available resources for the micro air vehicle's operational success.

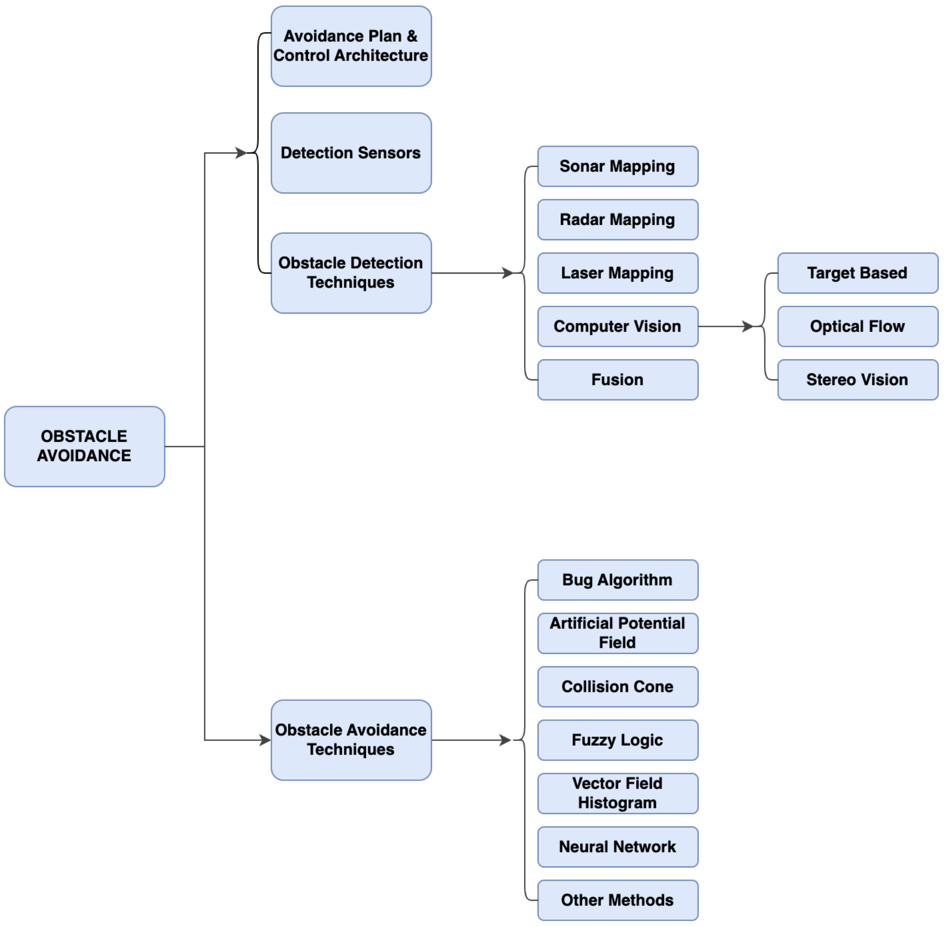


Figure 3.1: Obstacle Avoidance Scenario Group [20]

3.1.2. Detection Sensors

The selection of appropriate sensors for obstacle detection is a critical design decision for UAVs, especially when considering the diverse environmental conditions in which they may operate—such as varying lighting—and the constraints imposed by size, weight, and power (SWaP). Sensors are responsible for collecting information about the UAV's surroundings, which is used to determine obstacle distance and plan a safe flight path. Typically, this data includes the obstacle's size, shape, and position relative to the UAV [21].

Sensor types can be broadly categorized along two functional axes [22]: proprioceptive vs. exteroceptive and passive vs. active. Proprioceptive sensors measure internal states of the vehicle, such as position, orientation, and velocity. These include sensors like accelerometers, gyroscopes, tilt sensors, and heading indicators. Exteroceptive sensors, on the other hand, are used to sense the external environment, capturing data on nearby obstacles through technologies such as Time-of-Flight (ToF), LiDAR, sonar, radar, lasers, and cameras. These sensors play a pivotal role in enabling the UAV to interpret and react to its surroundings [20].

A second classification distinguishes between passive and active sensors. Passive sensors rely on ambient environmental energy—such as sunlight—to gather information. Vision-based sensors, for example, detect natural light or reflected wavelengths. In contrast, active sensors emit their own signals (e.g., light or sound waves) and detect their reflections to interpret the environment. Examples include ToF, sonar, radar, and laser sensors. Although active sensors often provide more consistent performance through controlled signal emission, they are susceptible to interference from external energy sources, which can introduce measurement errors [20].

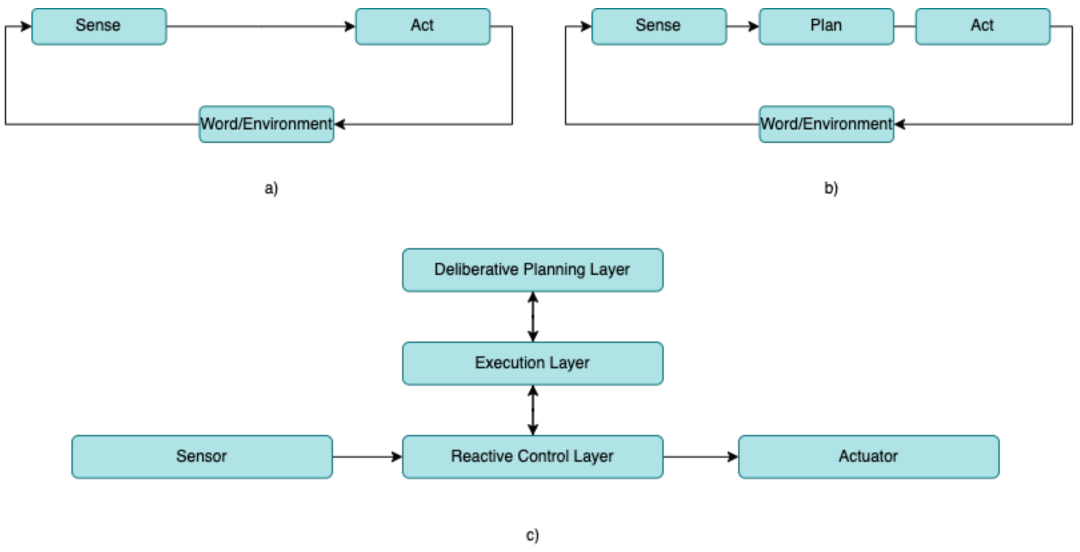


Figure 3.2: (a) reactive control; (b) deliberative planning; (c) hybrid control [20]

3.1.3. Obstacle Detection Technologies

Depending on the choice of sensor, the system gets various information about its surrounding obstacles from light or sound wave reflection. This information mainly contains the size, distance, shape, color and direction of the obstacle. The main concern of the UAV is the position and size of the obstacle. Some of the major obstacle detection methods are presented below.

Sonar Mapping

Sound navigation and ranging (sonar) mapping systems work on the principle of acoustic wave reflection from the surrounding objects to make an image or diagram of the environment. Elfes [31] creates a 2D map by using the sonar-based surrounding mapping, detecting the obstacle's range from different points of view. Steckle and Peremans [32, 33] use a 3D sonar sensor and BatSLAM (a biomimetics SLAM) for localization and mapping of objects in the surrounding environment. Other studies [34, 35, 36] use a round array setup of ultrasonic sensors in a small quadrotor UAV for obstacle mapping.

Radar Mapping

Radar Mapping, also known as radio detection and ranging, employs the same time-of-flight or echo-ranging principles as sonar but uses radio wave signals. Radar is capable of long-range detection and is extensively used in aerospace and military technologies. For smaller vehicles with lower computational capabilities, simpler radar systems like the one proposed by Giubbolini [37], which uses multiple 13–24 GHz radars around a vehicle, are suitable. Other technologies such as Comprehensive Sensing (CS) [38], microwave Doppler radars [39], and Frequency-Modulated Continuous Wave (FMCW) radar sensors [40] are also employed for obstacle detection and mapping in UAVs. Radar systems offer the advantage of functioning under adverse conditions like rain, dust, and strong sunlight.

Laser Mapping

Laser imaging employs single or multiple laser sensors to create detailed images or maps of the surroundings. The primary technology used in laser imaging is LiDAR (Light Detection and Ranging), which operates similarly to radar but uses laser light for distance measurements. Single-point lasers provide individual distance measurements, while multi-point or 2D lasers generate arrays of point clouds for more comprehensive area mapping. A notable example of multi-point laser application is the obstacle detection algorithm proposed by Yu and Zhang [41], utilizing a four-layer laser scanner for autonomous land vehicles. Researchers like Demantké et al. [42] have developed methods to analyze geometric structures within LiDAR point clouds, optimizing the surrounding size for each point. Zheng [43] proposed a point cloud correction and clustering algorithm based on relative distance and density (CBRDD).

Table 3.1: Obstacle Detection Sensors for UAVs [20]

Sensor Types	**Ultrasonic**	**Laser/Initiative Infrared Sensor**	**ToF**	**Millimeter-Wave Radar**	**Monocular Vision**	**Stereo Vision**
Range	<10	<50	<10	<250	<10	<100
Size	Small	Medium	Small	Medium	Small	Medium
Weight	Low	Medium	Low	Medium	Low	Medium
Power	Low	Medium	Low	High	Low	Medium
Cost	Low	Medium	Medium	High	Medium	High
Precision	Short Range	Very High	Medium	Low	Low	Medium
Resolution	Low	Very High	Low	Low	Medium	Medium
Reliability	Low	High	High	High	Medium	Medium
Liquid Droplet Influence	Yes	Yes	Yes	No	Yes	Yes
Sound Influence	High-Pitched Sound Interference	No	No	No	No	No
Light Influence	No	Direct Sunlight May Influence Infrared	Yes	No	No	No
Temperature Influence	Yes	No	No	No	No	No
Light Need	No	No	No	No	Yes	Yes
Single Point Measurement Reliability	No	-	-	No	Suitable for Static Obstacle	-
Process	Fast	Fast	Fast	Fast	Large Amount of Data Need Faster Processor	Large Amount of Data Need Faster Processor
Reference	[23]	[23, 24, 25]	[23, 26]	[23, 27, 28]	[23, 29]	[23, 30]

Computer Vision
Computer vision is a prevalent method for obstacle detection, utilizing various types of cameras such as monocular, stereo, binocular, and infrared cameras [44, 45]. This technique encompasses obstacle detection, recognition, and distance measurement, with SLAM (Simultaneous Localization and Mapping) [46] being a common application. Initially, computer vision was implemented using binocular stereo configurations, but due to the high cost of multi-camera systems, monocular vision has become more popular. Computer vision is particularly useful in complex environments where data acquisition by active sensors is challenging, although it requires significant computational resources. The rapid development of microcomputers is addressing this challenge.

Target-based detection utilizes a single camera to identify obstacles using known features, offering low computational demand but lacking precise distance measurement. Optical flow-based detection examines frame-by-frame pixel movements to detect motion and changes in grayscale images, which is versatile but less effective for stationary or slow-moving obstacles. Stereo-vision-based detection, or binocular stereo-vision, employs multiple cameras to generate depth information, providing more accurate data for tasks such as vehicle identification and UAV navigation, though it requires higher computational resources.

3.1.4. Obstacle Avoidance Techniques
After selecting the appropriate detection technique, the focus switches towards the avoidance techniques. The need for avoidance is obvious: it is desired that the mobile robot manages to navigate in a complex environment without colliding with objects. For this, a number of different algorithms were investigated from literature.

Bug Algorithms

One of the most simple obstacle avoidance methods is the bug method. The first development of this method was developed by Lumelsky and Stepanov [47] following bug's movement. Traditional path planning algorithms like Dijkstra's and A* require prior knowledge of the environment, including obstacle positions and target locations. In contrast, bug algorithms were designed to navigate without such pre-existing information, inspired by maze-solving strategies where wall-following techniques guarantee finding an exit in enclosed spaces. Since introducing this avoidance technique, many variations of the same idea have been developed. The current bug algorithms can be divided into three main categories: Angle-bugs, M-line-bugs, Range-bugs [30]. Figure 3.3 gives an overview of general bug algorithms.

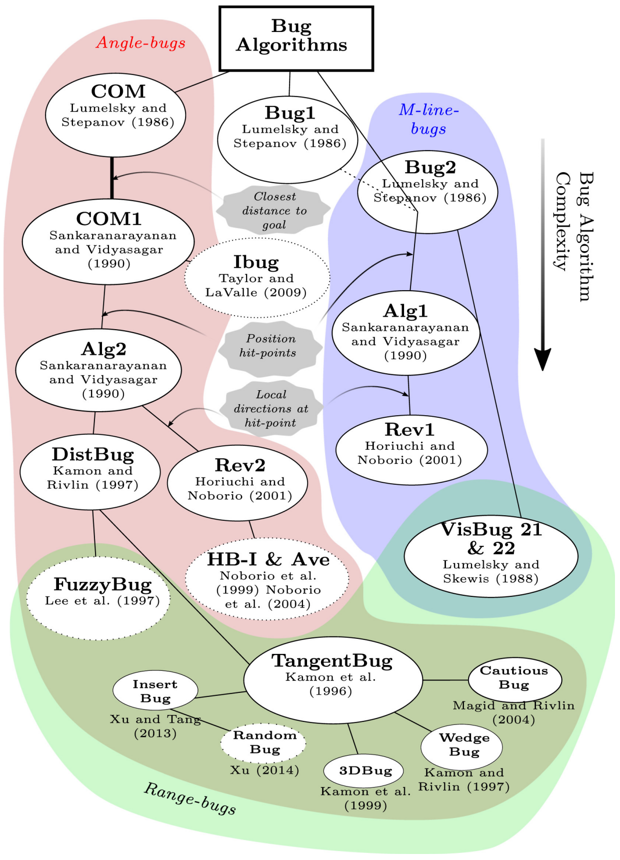

Bug Algorithms
Angle-bugs
M-line-bugs
Range-bugs
Bug Algorithm Complexity
COM Lumelsky and Stepanov (1986)
Bug1 Lumelsky and Stepanov (1986)
Bug2 Lumelsky and Stepanov (1986)
Closest distance to goal
COM1 Sankaranarayanan and Vidyasagar (1990)
Ibug Taylor and LaValle (2009)
Alg1 Sankaranarayanan and Vidyasagar (1990)
Position hit-points
Alg2 Sankaranarayanan and Vidyasagar (1990)
Local directions at hit-point
Rev1 Horiuchi and Noborio (2001)
DistBug Kamon and Rivlin (1997)
Rev2 Horiuchi and Noborio (2001)
VisBug 21 & 22 Lumelsky and Skewis (1988)
HB-I & Ave Noborio et al. (1999) Noborio et al. (2004)
FuzzyBug Lee et al. (1997)
TangentBug Kamon et al. (1996)
Cautious Bug Magid and Rivlin (2004)
Insert Bug Xu and Tang (2013)
Random Bug Xu (2014)
Wedge Bug Kamon and Rivlin (1997)
3DBug Kamon et al. (1999)

Figure 3.3: An overview of the Bug Algorithms [30].

Out of all the variations of the bug algorithm, as Figure 3.3 show, the ones that are the most relevant for this research are the range-bugs. This is due to the sensing technology that they are using, which rely on range sensors. Bug algorithms with range sensors enhance traditional Bug algorithms by allowing the robot to detect obstacles from a distance, thus acting before making physical contact. The VisBug algorithms, VisBug 21 and VisBug 22, are based on Bug2 and incorporate range sensors to identify ”short-cuts” to the next obstacle, reducing the overall path length. TangentBug, developed by Kamon et al. [48], constructs a local tangent graph (LTG) within the sensor's maximum range, representing the borders of the detectable obstacle field. The robot follows the LTG edge that offers the quickest path to the target but switches to following the obstacle boundary if the path becomes less efficient, based on a locally stored minimum distance [30].

In some cases, the TangentBug algorithm can become quite computationally expensive due to the need to generate and update LTGs and make real-time navigation decisions. In a computationally limited drones, this can strain processing resources, potentially slowing down response times and affecting overall performance. Therefore, the other bug algorithms are still considered.

Artificial Potential Field Algorithm

The artificial potential field algorithm (APF) was proposed by Khatib [49], which is a unique real-time obstacle avoidance approach for mobile robots. The working principle of this algorithm is setting an artificial potential field to every point from the known areas and moving towards the lower possible area, where the target point is the lowest possible area [20]. To reach the target point, the vehicle is always going towards the lowest possible area. [50] uses APF algorithm to avoid other obstacles and form obstacles.

There are, however, limitations to the APF algorithm. Two of the main limitations are local minimum point problem and dead point problems. In a study by Chen et al [51], a reconstructed APF constrained optimization solves the traditional dead point problem. Fan et al. [52] proposed an improved APF algorithm to solve the local minima and the target's inaccessibility.

Collision Cone Method

Another reactive method for obstacle avoidance was first proposed by Chakravarthy and Ghose [53]. The idea behind is to see every obstacle as a circular area, and by using the UAV's velocity vector, the position to the obstacle area is calculated within a collision cone. Watanabe et al. [54] used passive vision sensors and collision cone approach to examine obstacles from a critical distance. Park and Baek [55] proposed the Tangent Plane Coordinate (TPC) algorithm for real-time collision avoidance using stereo-vision sensors with limited field of view to detect unknown obstacles. Their method calculates collision cones by applying affine transformations to lines tangent to ellipsoids representing obstacles, ensuring these lines pass through the quadrotor's current position. This approach enables the quadrotor to navigate safely through environments with multiple obstacles by dynamically adjusting its path to avoid collisions based on real-time sensor data.

Tijmons et al. [56] proposed a similar method to the collision cone method, denoted the "Droplet" strategy.The "Droplet" strategy for obstacle avoidance continuously checks for a clear turn space ahead of the vehicle. This space is represented by a red circle with radius R_{total}, centered ahead of the vehicle so it fits within the combined field-of-view of the cameras. The green area within this view must be free of obstacles, allowing the vehicle to safely execute circular avoidance maneuvers. This method ensures continuous, infinite collision-free navigation, unlike others that only ensure avoidance up to a certain point.

Fuzzy Logic
In 1965, Zadeh [57] proposed the fuzzy logic, which, following its name, uses fuzzy controllers. The idea behind it is to create fuzzy sets that are used for obstacle avoidance or navigation by assigning a set of data or knowledge to the mobile robot. The set value is usually between two traditional logics, (0, 1), (Low, High), or (Cold, Hot), etc [20]. The fuzzy logic has been use extensively for obstacle avoidance in unknown environments in the past. The paper of Faisal et al. [58] utilizes fuzzy logic for obstacle avoidance in mobile robots by leveraging its capability to handle uncertainty and imprecision, akin to human decision-making. Sensors detect obstacles, providing inputs such as distance and angle, which are processed through a fuzzy inference system. This system applies predefined if-then rules to determine appropriate actions, such as turning or slowing down. The fuzzy outputs are then defuzzified into precise commands for the robot's actuators. This approach allows the robot to navigate efficiently and smoothly in dynamic and unknown environments, adapting to obstacles and ensuring robust, flexible control. Reignier [59] used fuzzy logic techniques to build a reactive navigation system and avoid obstacles, while other papers use the fuzzy logic for path planning.

Vector Field Histogram
The vector field histogram obstacle avoidance is a real-time method that was developed in 1991 by Johann Borenstein and Yoram Koren [60]. This obstacle avoidance method operates through three key steps. Firstly, the robot constructs a two-dimensional sensory histogram, mapping its surrounding environment within a specified range. This histogram is continuously updated with real-time data from the robot's sensors, ensuring an accurate representation of nearby obstacles. Secondly, the two-dimensional histogram data is transformed into a one-dimensional histogram. This simplification process involves condensing the spatial information into a linear format that highlights the density of obstacles in different directions. Lastly, the robot analyzes the one-dimensional histogram to identify regions with lower obstacle density, known as lower polar dense areas. Based on this analysis, the robot calculates the optimal direction for movement, allowing it to navigate safely and efficiently by steering towards the path with the least resistance. This method enables the robot to dynamically adapt to its environment, ensuring effective obstacle avoidance in real-time. Both [61] and [62] make use of the VFH algorithm for mobile robots equipped with lidar sensors for obstacle avoidance, the first operating on a hexa-copter while the latter on automated guided vehicles.

Neural Networks
Neural network algorithms, inspired by the human brain, train on data to recognize patterns and predict outcomes for new, similar data. These computational models iteratively train until achieving optimal results. A dynamic neural network can adjust its structure automatically in response to changes in the vehicle's environment. It maps the relationship between the vehicle's state and its obstacle avoidance decisions in real-time, thereby reducing computational load and improving efficiency.

Huang et al. [63] proposed a reinforcement learning approach towards obstacle avoidance. Using a vision-based neural network, Yadav et al. [64] created a controller for obstalce avoidance in a 3D environment. Chi and Lee [65] proposed a neural network control system for guiding mobile robots through mazes, avoiding obstacles. Kim and Chwa [66] used a fuzzy neural network for obstacle avoidance in wheeled mobile robots, incorporating fuzzy sets into the neural network layer. Back et al. [67] developed a vision-based system for UAVs to follow trails and avoid obstacles using Convolutional Neural Networks (CNN). Dai et al. [68] also utilized CNNs for obstacle avoidance in unknown environments for quadrotor UAVs. However, neural networks require extensive training data but excel in real-time obstacle avoidance.

A recent and very promising advancement in reinforcement learning comes from the KAN (Kolmogorov–Arnold Networks) [69]. Kolmogorov-Arnold Networks (KANs) are proposed as alternatives to Multi-Layer Perceptrons (MLPs). Unlike MLPs, which use fixed activation functions on neurons, KANs place learnable activation functions on the edges (weights), replacing linear weights with univariate functions parameterized as splines. This structural change enables KANs to outperform MLPs in terms of accuracy and interpretability, especially in small-scale AI tasks. KANs can efficiently learn both compositional structures and univariate functions, addressing the curse of dimensionality and enhancing performance in data fitting.

Comparison
Before performing the comparison between the obstacle avoidance techniques, it is important to understand the challenges and limitations the FWMAV faces. One of the main challenges in obstacle avoidance is the weather and lightning conditions. Fog or poor lightning conditions limit the detection capability of the sensors and renders the processed data inadequate for obstacle avoidance. A multiple sensor configuration can solve this problem, at the cost of complexity and power consumption.

Neural networks have demonstrated strong capabilities in learning obstacle avoidance behaviors, especially in complex and uncertain environments. However, they typically require large volumes of preprocessed data and rely on extensive offline training, often in simulated environments. This reliance introduces a potential reality gap between training and deployment, which can reduce the robustness of the final system. In contrast, reinforcement learning (RL) methods can offer a more direct and flexible alternative, as they learn optimal behaviors through interaction with the environment and do not require preprocessed datasets. This makes RL particularly appealing in scenarios with sparse or noisy sensory input. Nevertheless, both neural networks and RL approaches are computationally demanding and may require significant resources during training. To complement these methods and better understand the trade-offs involved, a simple Proportional–Integral–Derivative (PID) control strategy will also be investigated. A PID-based approach, with its low computational overhead and interpretability, may still offer effective navigation performance under the sensing and control constraints. Table 3.2 presents a comparative overview of the discussed obstacle avoidance techniques.

Table 3.2: Comparison of Avoidance Techniques for Flapping Wing Air Vehicles

Avoidance Technique	**Features**	**Flapping Wing MAV's Insights**
Bug Algorithms	• Easy and Convenient • It follows the obstacle's exact outline • It changes the heading when bypassing the obstacle • It doesn't detect the edge of the obstacle, which may take a longer path sometimes	This algorithm may be suitable for environments where the MAV needs to navigate tight spaces. However, due to the dynamic and unpredictable nature of a flapping wing MAV, the time to adjust headings for each obstacle edge might be increased. This could lead to higher energy consumption and longer flight times.
Artificial Potential Field	• A simple approach for implementation • Easy to find the shorter edge of the obstacle • Local minima problem can cause process failure	This method can quickly determine shorter avoidance paths, which is beneficial for a flapping wing MAV needing to conserve energy. However, the local minima issue can be problematic in cluttered environments, causing the MAV to get stuck.
Collision Cone	• Creates simpler avoidance path • Uses vehicle dynamics to create avoidance path • The minimum effort of guidance control • Doesn't consider the shape of the obstacle	Given the MAV's agility, this method can effectively reduce collision risks by leveraging dynamic flight adjustments. However, not considering the obstacle's shape could lead to inefficient paths around complex structures. Incorporating real-time shape analysis could improve path planning accuracy.
Fuzzy Logic	• Robust and suitable for dynamic environments • Needs multisensory system • Polyhedral shape may increase computational calculation	The use of multisensory input can enhance the MAV's ability to adapt to dynamic environments, which is crucial for unknown terrain. Despite potential computational overhead, the flexibility offered by fuzzy logic can result in smoother, more adaptive flight paths.
Vector Field Histogram	• A better method for detecting obstacle's shape identification • Require longer time to 2D map the obstacle • High computational requirement • Doesn't consider the vehicle's dynamics	For a flapping wing MAV, the ability to detect and map obstacles accurately is critical. However, the high computational requirements and longer mapping times could be a drawback in fast-paced environments.
Neural Network	• Good for known or unknown obstacle environment • Better performance for real-time avoidance • Needs training data before performance	For a flapping wing MAV, continuously updating the neural network with real-time data could enhance adaptability and performance. This approach allows the MAV to learn and adapt on-the-fly, resulting in more robust and efficient navigation.

The final architecture of the obstacle detection and avoidance for the FWMAV is presented in Figure 3.4.

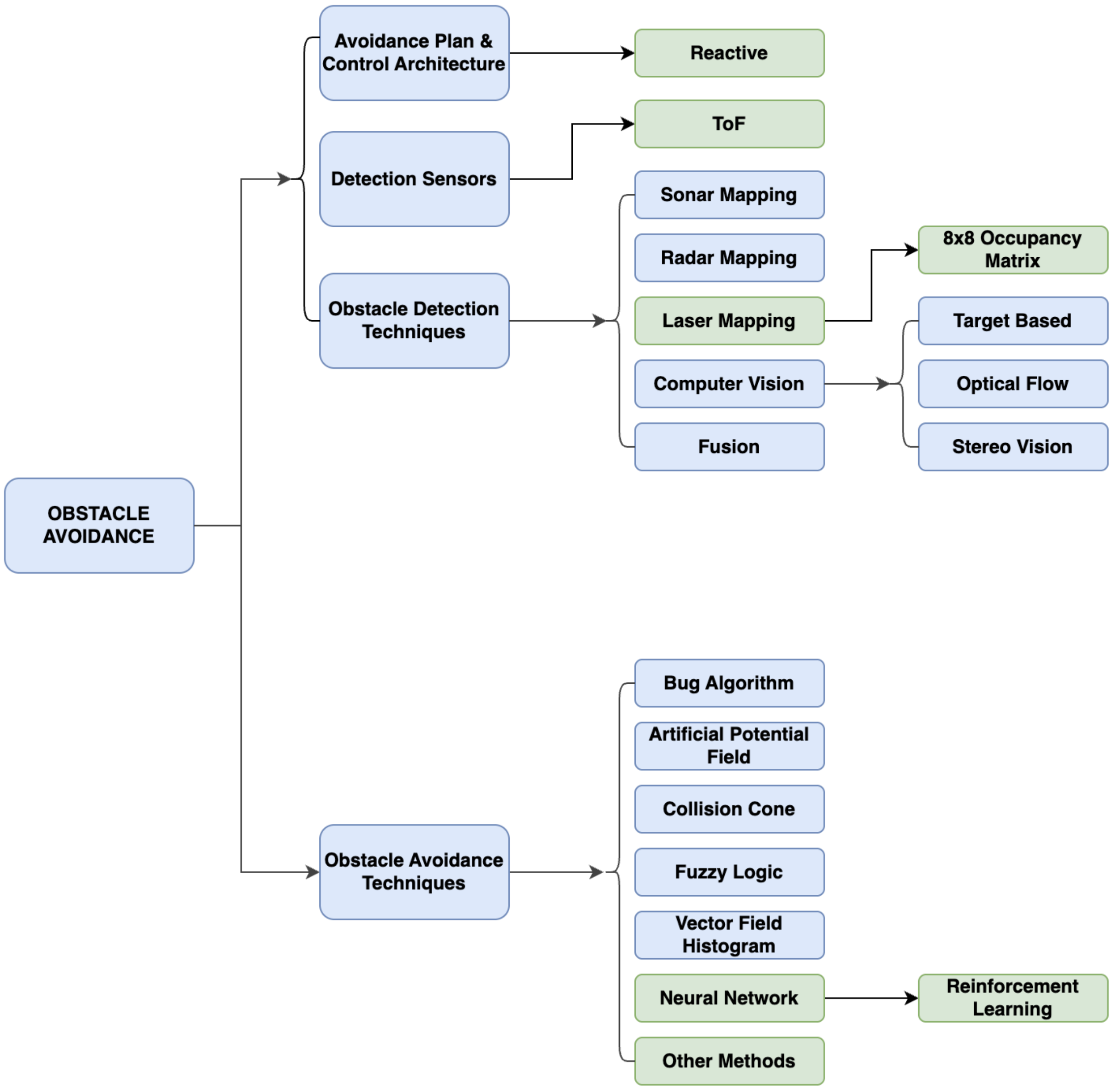


Figure 3.4: Obstacle Avoidance Scenario Group Result

4 Research Proposal

This chapter defines the research problem addressed in this thesis and outlines the rationale behind the selected approach. Following a review of current literature on obstacle avoidance in micro aerial vehicles, several critical challenges have been identified that hinder the deployment of fully autonomous navigation systems for flapping wing platforms—particularly those using tailless configurations. These challenges are synthesized into a set of research gaps and a central research question, supported by specific subquestions that guide the technical exploration.

4.1. Research Gap

Despite rapid progress in the development of flapping wing micro aerial vehicles, several open challenges remain in enabling effective onboard obstacle avoidance. The flapping mechanism itself induces vibrations that can degrade the performance of lightweight sensors. At the same time, such platforms face stringent constraints on weight, power consumption, and computational capacity, severely limiting the number and complexity of sensors that can be integrated. Sensor positioning is further complicated by the limited surface area and mechanical interference caused by the flapping wings, especially in tailless configurations.

A key gap identified in the literature is the lack of fully autonomous navigation capabilities for tailless FWMAVs. Although these platforms offer superior agility and are well-suited for navigation in cluttered or constrained environments, their unstable dynamics demand active stabilization, and existing navigation systems are either manually controlled or rely on heavy sensing and localization infrastructure not compatible with their limited payload. Furthermore, most classical avoidance algorithms fail to adapt to sparse depth input, while learning-based strategies, although promising, remain underexplored in this context.

The literature also points toward several decisions in sensor and controller design. Time-of-Flight sensors offer a favorable trade-off between size, energy consumption, and robustness to lighting conditions, making them suitable for onboard use. Reactive control architectures are preferred over deliberative or hybrid ones due to the limited onboard processing capabilities of FWMAVs. Among the avoidance methods, reinforcement learning stands out as a promising strategy due to its ability to learn directly from raw sensor data while classical methods like PID offer simplicity, transparency, and reliable performance under constrained conditions.

4.2. Research Question

In light of the identified gaps and technological constraints, the central research question of this thesis is defined as:

How can obstacle avoidance be achieved in an attitude-stable flapping wing air vehicle equipped with Time-of-Flight sensors?

To address this overarching question, the following subquestions are posed:

- What is the minimum sensor configuration required to achieve reliable obstacle avoidance under real-world hardware constraints?
- How does control performance differ between a classical PID-based navigation strategy and a learning-based reinforcement learning approach?

- How does sensor configuration (e.g., number and placement of ToF sensors) affect navigation quality, trajectory smoothness, and coverage in cluttered environments?
- To what extent can reactive control strategies generalize across different environmental layouts with limited sensing and no global positioning?

The investigation focuses on a flapping wing air vehicle that is assumed to be attitude-stable and controllable in thrust, roll, pitch, and yaw. The vehicle is equipped with a minimal set of directional ToF sensors providing sparse depth data in specific directions. No absolute positioning, global mapping, or visual odometry is available, making the task inherently reactive and partially observable.

Two complementary control strategies are developed and tested: a classical approach based on PID regulators combined with a confidence-driven yaw selection mechanism, and a reinforcement learning policy trained end-to-end to infer navigation actions from raw sensor input. These approaches are evaluated in a high-fidelity simulation environment under varying levels of sensor input and environmental complexity.

The goal is to determine not only whether obstacle avoidance is achievable under strict constraints, but also which control approach and sensor configuration offer the best balance between safety, exploration, and system simplicity.

Part II

Autonomous Navigation

5 Hardware

This chapter provides an overview of the hardware components used to enable autonomous flight on the drone. It begins with an explanation of the flapper drone platform chosen, highlighting its bio-inspired flight dynamics and unique control challenges. Next, the selection and integration of the Time-of-Flight sensors are introduced, including their location and performance under various conditions. Finally, the auxiliary processing unit, Raspberry Pi Zero 2, and the system architecture in general are presented, demonstrating how the sensor data is processed and translated into flight commands.

5.1. The Flapper Nimble+

The air vehicle used for this research is the Flapper Nimble+[1] by the Flapper Drones company, which is shown in Figure 5.1.

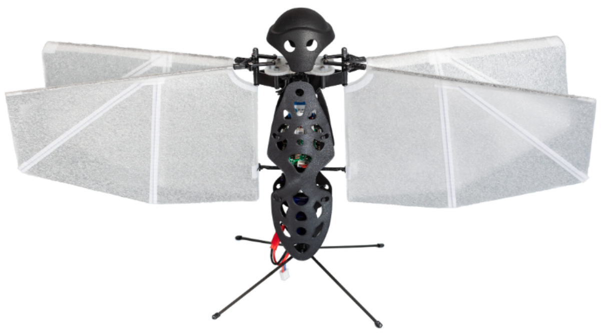


Figure 5.1: The Flapper Nimble+[1]

The drone is a bio-inspired flapping-wing drone designed for agile flight and maneuverability. Unlike conventional rotorcraft or fixed-wing drones, this aerial robotic platform mimics the flight mechanisms of insects and birds. It features a tailless design, meaning it relies entirely on wing motion adjustments for stability and control.

The drone utilizes a counter-phase wing flapping, where two wings on each side move in opposition to generate thrust. This setup includes a clap-and-peel mechanism, similar to what is observed in natural flyers, which improves the lift generation and making it energy-efficient. It achieves control through:

- **Thrust**: Increasing flapping frequency leads to increasing thrust, leading to lift generation, while reducing it lowers the thrust, enabling controlled descent.
- **Yaw Torque**: Adjusting the wing root angles to tilt the thrust vectors in opposite directions.
- **Pitch Control**: Modifying the dihedral angle of the wings to shift thrust distribution relative to the center of mass.

[1] `https://flapper-drones.com/wp/nimbleplus/`

- **Roll Control**: Alternating stroke frequencies between the left and right wing pairs to create a differential lift force.

The bio-inspired mechanism enables high agility, which is further improved by the tailless configuration which allows the flapper to perform sharp turns, hovering, and forward flight. However, the Flapper Nimble+ is inherently **attitude-stable**, meaning it can regulate its orientation (yaw, pitch, and roll) effectively through wingbeat adjustments. However, it lacks velocity and position stability, as it does not autonomously maintain a fixed location or velocity without active control inputs. Unlike conventional drones with GPS or barometer-based stabilization, the Flapper Nimble+ requires continuous corrections to counteract drift and external disturbances, making precise navigation and obstacle avoidance more challenging.

Due to its flapping-wing design, the Flapper Nimble+ experiences significant periodic vibrations resulting from the rapid oscillatory motion of its wings. The flapping frequency, approximately 12 Hz when hovering and 20 Hz at maximum throttle[1], induces mechanical oscillations throughout the structure, affecting both the airframe and onboard electronics. These vibrations are an inherent characteristic of flapping-wing drones and can influence flight stability, particularly at higher wingbeat amplitudes. Additionally, they introduce challenges in control precision, as they can cause small perturbations in attitude that require continuous corrections. While the drone remains attitude-stable, these vibrations can contribute to minor variations in its flight trajectory, necessitating compensatory adjustments in real-time control strategies.

The Flapper Nimble+ has a maximum payload of 25 grams and LiPo 2S 300 mAh battery, meaning that there is not much freedom in the amount of sensors it can equip.

5.2. Sensors

In order to achieve obstacle avoidance, the most important part of the system is the sensor suite, which enables environmental perception for obstacle avoidance. After analyzing different types of sensors and performing a trade-off based on a variety of criteria (see Section 3.1.3), it was concluded that the most suitable sensor for this research is the **Time-of-Flight (ToF)** sensor. Since there are a wide variety of tof sensors, the focus fell on finding variant that is most compatible with the aim of the research and with the current hardware. This sensor overview is presented in Table 5.1

Sensor Model	Max Range (m)	Field of View (FoV)	Multi-Target Detection	Measurement Speed (Hz)	Interface
VL53L0X	2	25°	No	Up to 50	I²C
VL53L1X	4	Adjustable	No	Up to 50	I²C
VL53L3CX	3	25°	Yes	Up to 30	I²C
VL53L4CD	1.3	18°	No	Up to 100	I²C
VL53L5CX	4	63°	Yes	Up to 60	I²C
VL53L8CX	4	65°	Yes	Up to 60	I²C, SPI

Table 5.1: Comparison of VL Series Time-of-Flight Sensors [70, 71, 72, 73, 74, 75].

The sensor that was chosen is the VL53L8CX, which has a good range of 4 meters. One of the key advantages of this sensor is its support for both I²C and SPI interfaces, providing greater flexibility in communication. The I²C interface is beneficial for simpler integration with microcontrollers and low-speed applications, while the SPI interface allows for higher-speed data transfer, making it more suitable for real-time processing in navigation and obstacle avoidance tasks. This versatility ensures that the sensor can be efficiently integrated into different processing architectures. The sensor is presented in Figure 5.2, already mounted on the Polulu microcontroller-board. For reference, Figure 5.3 shows the dimensions of the sensor plus the board.

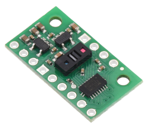


Figure 5.2: VL53L8CX ToF sensor on the Polulu microcontroller-board[76]

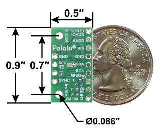


Figure 5.3: The Dimensions of the VL53L8CX ToF sensor[76]

5.2.1. Detection Procedure

The idea behind using ToF is quite straightforward. These sensors are not power-hungry and they are relatively small, making them a good choice when each added gram matters and when there is an issue with the space availability on the drone. These sensors work as follows: they emit fully invisible 940 nm IR light which then gets reflected. They then output two sets of values: the range (from 0 meters to 4 meters) and the validity of the reading, in an 8x8 depth matrix. This matrix then contains a representation about the distance from the sensor to detected objects within its field of view. This schematic can be seen in Figure 5.4. Here, the range values are converted to a gray-scale image, where white means detections that are larger or equal to 4, while black means detections that are at 0 meters. Note that a white cell can also represent an invalid reading.

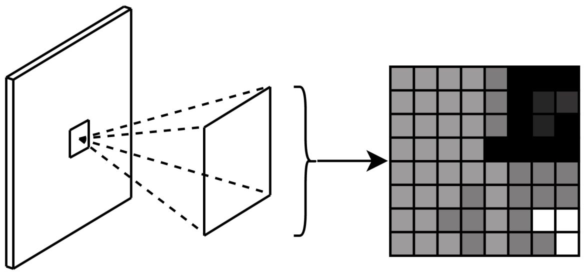


Figure 5.4: Depth Matrix from the ToF sensor

What is important to note is that the focus here is not to accurately describe what type of object was detected by the sensor, but more to assess that there is in fact an obstacle at a certain distance from the sensor. This, in theory, is enough to make sure that collision with the obstacle can be avoided.

As mentioned previously, besides the range values from the sensor, the VL53L8CX also outputs the validity of the reading. These values are described in Table 5.2.

Target Status	Description
0	Ranging data are not updated
1	Signal rate too low on SPAD array
2	Target phase
3	Sigma estimator too high
4	Target consistency failed
5	Range valid
6	Wrap around not performed (Typically the first range)
7	Rate consistency failed
8	Signal rate too low for the current target
9	Range valid with large pulse (may be due to a merged target)
10	Range valid, but no target detected at previous range
11	Measurement consistency failed
12	Target blurred by another one, due to sharpener
13	Target detected but inconsistent data. Frequently happens for secondary targets.
255	No target detected (only if number of target detected is enabled)

Table 5.2: Target Status Codes and Their Descriptions[77]

One of the main challenges of using the Flapper Nimble+ equipped with ToF sensors for obstacle avoidance is the impact of wing vibrations on sensor accuracy. The rapid flapping motion generates mechanical vibrations that affect ToF sensors, leading to quite a substantial noise in the measurements. For this, the detection capability of the sensor was analyzed to determine its effectiveness in various real-world scenarios. This included evaluating which objects and surfaces could be reliably detected and identifying materials or conditions that resulted in poor or inconsistent detections. Additionally, we examined the impact of vibrations on the sensor's performance, assessing how oscillations from the flapping wings influenced measurement stability, accuracy, and noise levels. The analysis aimed to quantify potential limitations in detection reliability.

To assess the reliability of the sensor, the drone was tested in the CyberZoo of TU Delft under 3 different conditions: **steady** (to assess the detection capabilities of the sensor and its range at different distances from an obstacle), **holding** (by holding the drone and approaching an obstacle, to mimic a non-vibrating flight) and **flight** (to assess the detection of the sensor under the induced vibrations from the drone).

The setup is presented in Figure 5.5. The drone was placed at 5 meters from the obstacle, and the obstacle was placed at 2 meters from the wall.

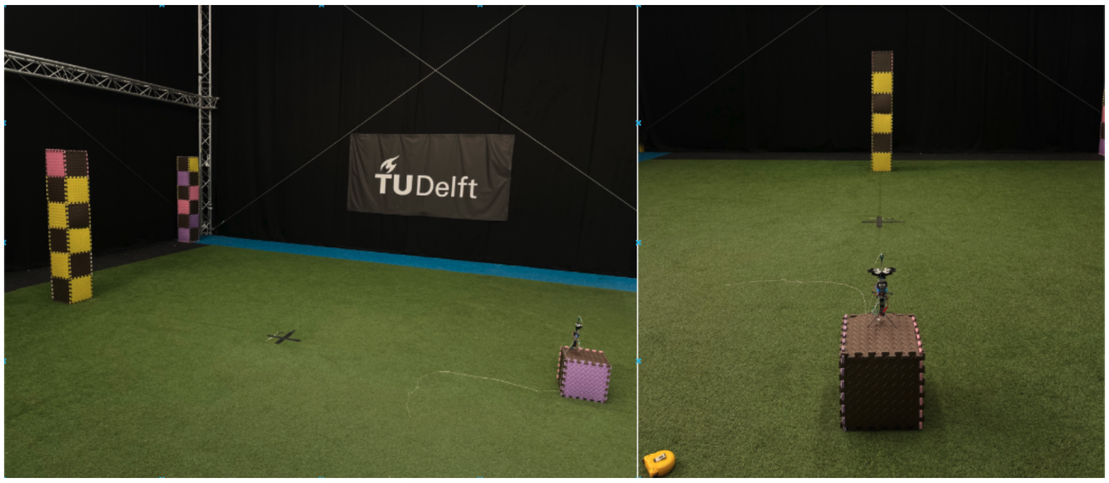


Figure 5.5: Test Setup

Steady Test

To assess the detection capabilities of the sensor, a steady test was performed first. Here, the drone was progressively moved towards the obstacle from 5 meters by 1 meter steps all the way to 0.5 meters from the obstacle. The outputs are presented in Figure 5.6 and Figure 5.7.

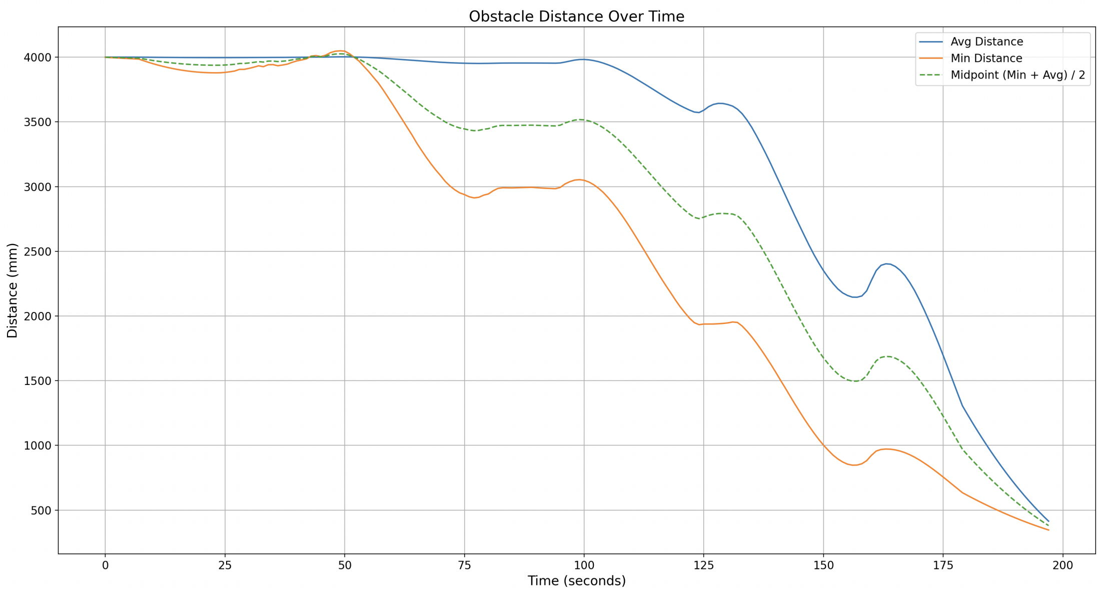


Figure 5.6: Steady - Minimum, Mean and Middle Distances

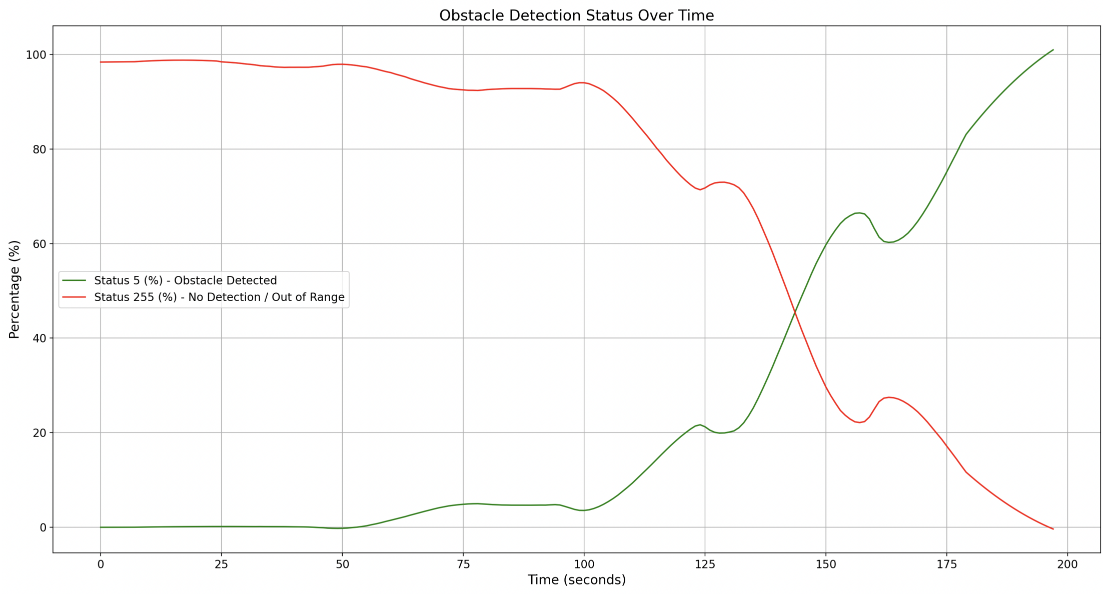


Figure 5.7: Steady - Detection Performance

The results from Figure 5.6 indicate that as the drone approaches an object, the minimum detected distance (orange line) decreases, which aligns with expectations, as closer proximity naturally results in smaller distance readings. Similarly, the average detection distance (blue line) also decreases, but an interesting pattern can be seen: a noticeable upward shift in its trend.

This upward shift suggests that when the drone is further from the obstacle, the large distance values dominate the averaging process. As a result, the average detection distance remains relatively high initially and only begins to decrease after the minimum distance has already dropped. This delay occurs because while the closest sensor readings quickly register the presence of the obstacle, the broader distribution of sensor values, which includes farther distances, causes the average to lag behind in reflecting the change.

The detection performance from Figure 5.7 follows the expected trend, demonstrating a clear relationship between obstacle proximity and detection accuracy. When the drone is positioned far from the obstacle, a significant number of sensor readings return status 255 (red line), which indicates “no detection.” This occurs because the obstacle is beyond the sensor’s maximum range, resulting in unreliable or missing distance measurements. As the drone moves closer to the obstacle, the proportion of valid detections (status 5, green line) increases. This is because the obstacle enters the sensor’s effective detection range, allowing for more reliable distance measurements. The transition from status 255 (no detection) to status 5 (valid detection) highlights how the sensor’s field of view and range constraints influence obstacle perception.

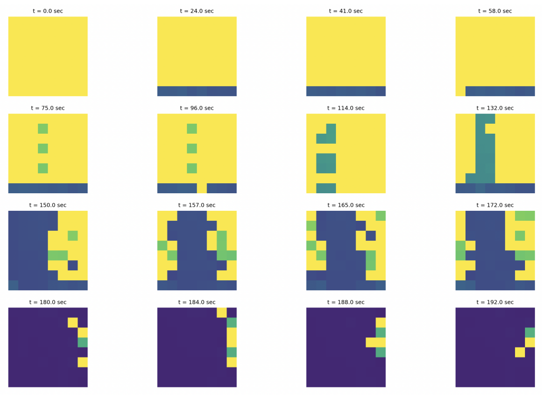


Figure 5.8: Steady Test - Heatmap Distances

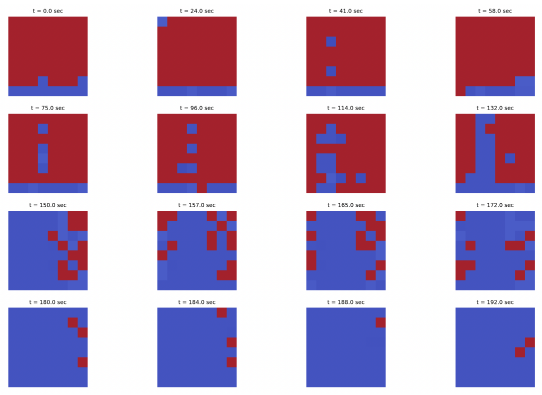


Figure 5.9: Steady Test - Heatmap Statuses

Figure 5.8 and Figure 5.9 show the recreated output from the sensors in form of heatmaps.

Holding Test
In this scenario, the drone is held and it is moved towards the obstacle. Figure 5.10 and Figure 5.11 presents outputs from the sensor.

Similar trends were noted in the minimum and average detected distances, where the minimum detection (orange line) decreased as expected when the drone got closer to the object. The average detection (blue line) also followed a decreasing trend, but again, a slight upward shift was observed due to the dominance of larger distances when the drone was farther from the obstacle. Detection performance also aligned with previous findings. At greater distances, the sensor frequently returned status 255 (no detection) due to the obstacle being outside the detection range. As the drone was brought closer, status 5 (valid detection) became increasingly dominant, confirming that the sensor reliably detects obstacles once they enter its effective range. Near the object, detection accuracy reached nearly 100%, verifying the sensor's functionality in a controlled setting.

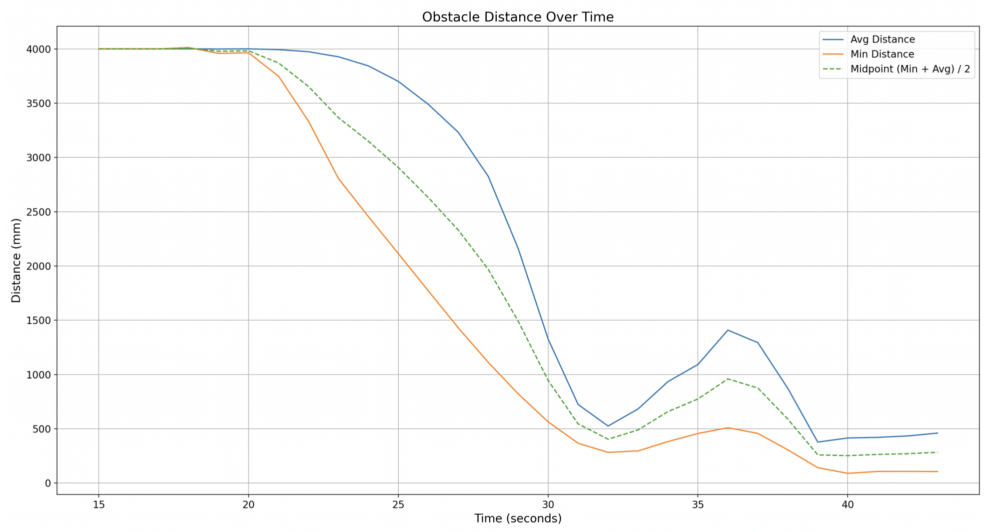


Figure 5.10: Holding - Minimum, Mean and Middle Distances

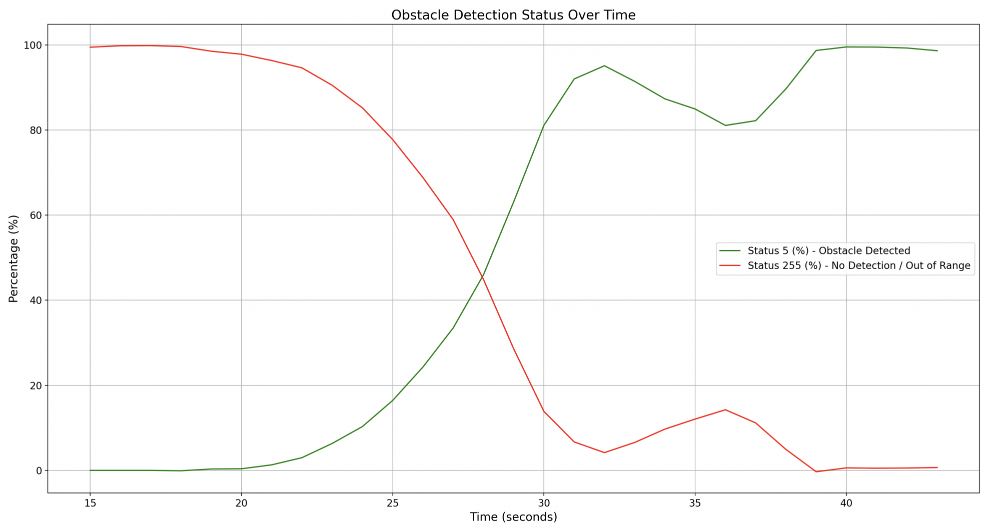


Figure 5.11: Holding - Detection Performance

The images from Figure 5.12 and Figure 5.13 show the heatmaps of the readings from the sensor roughly every 2 seconds.

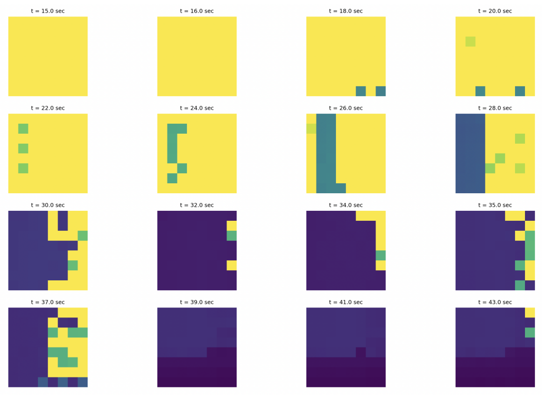


Figure 5.12: Holding Test - Heatmap Distances

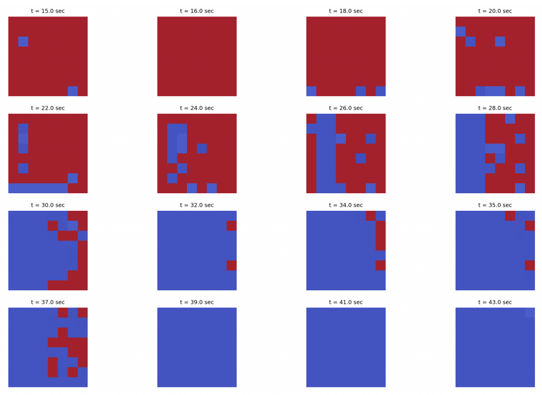


Figure 5.13: Holding Test - Heatmap Statuses

Flying Test

In this scenario, the drone was flown towards the obstacle. Figure 5.14 and Figure 5.15 show the outputs from the sensor.

The same test was conducted with the drone in actual flight conditions, flying toward an obstacle while recording sensor readings. The overall trend observed in the minimum and average detected distances remained consistent with previous tests in steady and holding conditions. However, the data exhibited increased noise, with a steep decay in distance values compared to previous tests.

During the flight test, the behavior of both the average and minimum distance curves was consistent with expectations. As the drone approached the obstacle, both metrics showed a steady and continuous decline, indicating increased proximity to nearby surfaces. The minor rise observed towards the end of the trial is attributed to the trajectory of the drone exiting the obstacle's vicinity and does not reflect a failure in the detection logic. Overall, the distance and status trends validate the expected perception behavior—closer proximity results in lower measured distances and an increase in valid detection counts (status 5), while non-detections (status 255) correspondingly decrease.

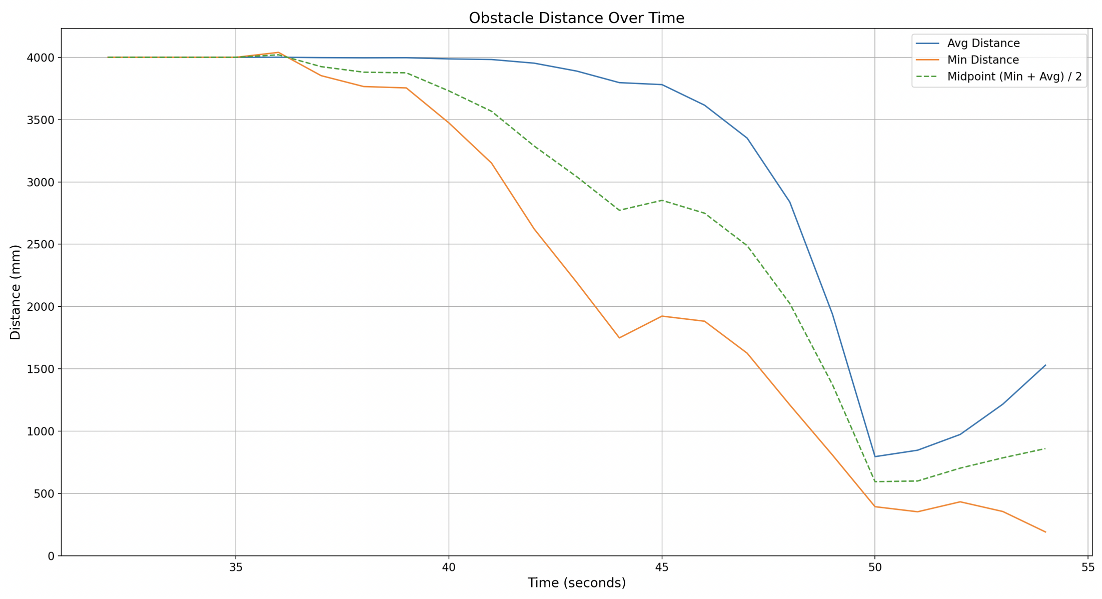


Figure 5.14: Flying - Minimum, Mean and Middle Distances

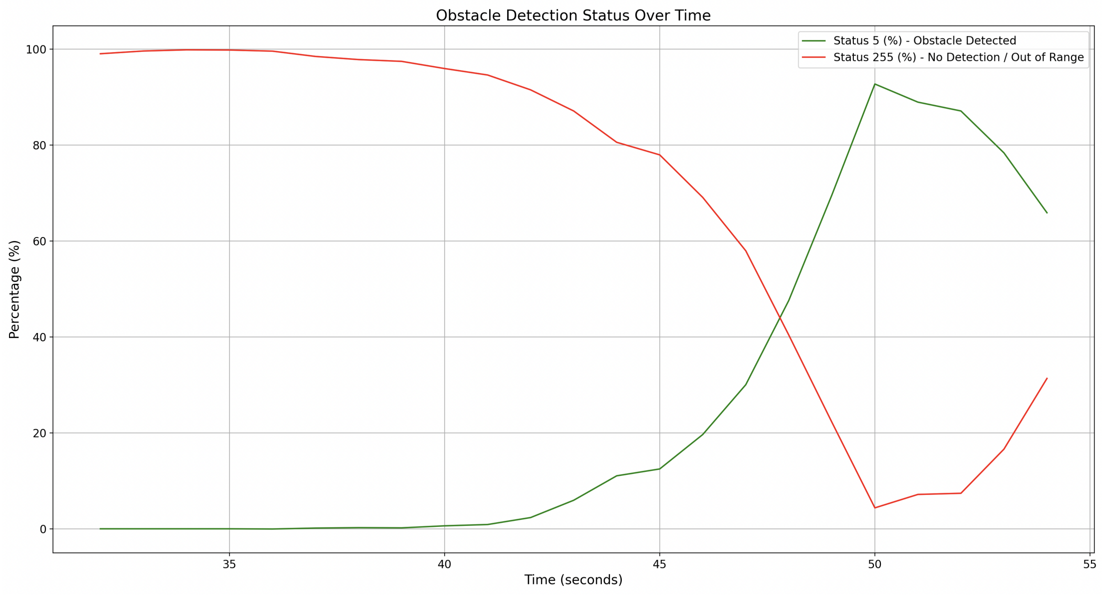


Figure 5.15: Flying - Detection Performance

An interesting observation emerged when comparing the steady test, holding test, and flying test results. In both the steady-state and holding test, the obstacle was detected but did not appear clearly in the sensor output until the drone was relatively close to it. The detection pattern remained somewhat diffuse, with large portions of the depth matrix still showing greater distances until the drone approached the obstacle directly.

However, during actual flight, the obstacle was noticeably more prominent and distinct in the depth matrix output from the sensor, even when the drone was further away. This improvement in detection clarity can be attributed to the vibrations present during flight. As the drone flaps its wings, small oscillations cause the sensor to cover a larger frontal area, effectively “scanning” a wider region ahead. This results in more nuanced and varied depth readings, making the obstacle more visible at greater distances compared to when the drone remains still.

The images from Figure 5.16 and Figure 5.17 show the heatmaps of the readings from the sensor roughly every 2 seconds.

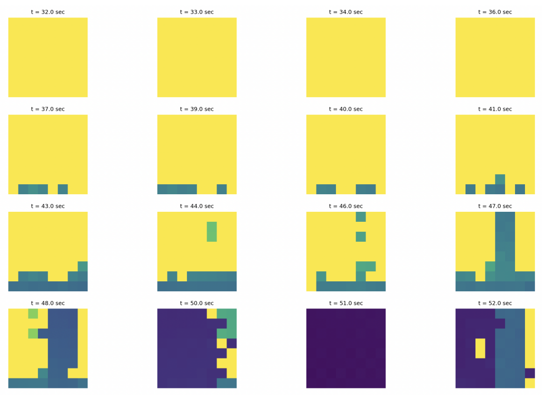


Figure 5.16: Fly Test - Heatmap Distances

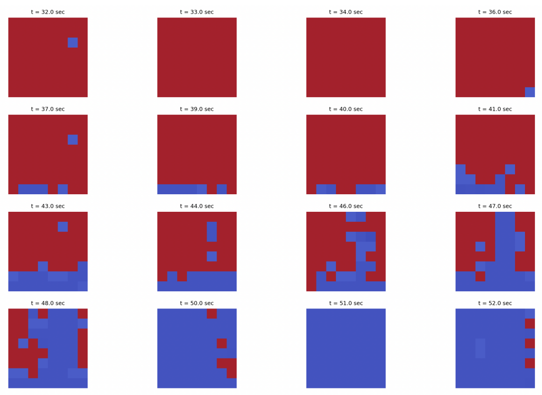


Figure 5.17: Fly Test - Heatmap Statuses

These findings suggest that the natural vibrations of flight contribute to a more detailed perception of obstacles, allowing for earlier recognition and possibly aiding in obstacle avoidance strategies.

Limitation
One limitation in using the ToF sensor, as mentioned before, is detecting glass surfaces, such as windows. This limits the environments in which the drone equipped with a ToF sensor can operate. Figure 5.18 and Figure 5.19 show the output from the sensor when directed towards a window. It can be seen that the window is virtually not detected at all, returning the max. range value (4 meters), meaning that the drone will try to go through it, mimicking a fly behaviour.

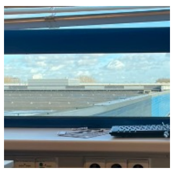


Figure 5.18: Window Image

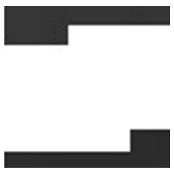


Figure 5.19: Window Detection

5.2.2. Placement

Integrating sensors onto the Flapper Nimble+ presents several challenges due to its flapping-wing design, high-frequency vibrations, and limited mounting space. The wing represent the first main concern, as they span 49 cm. This introduces a series of problems:

- Intermittent occlusions where the wings temporarily block the sensor's detection range.
- Limited optimal mounting positions due to the dynamic movement of the structure.
- Effect of pitch and roll on the sensor orientation.

Therefore, to make sure that the sensors can utilize the entirety of their field of view, placing them strategically is important. In Figure 5.20, a top-view simple drawing showcases the obstruction that comes from having the wings fully extended, while Figure 5.21 showcases the real-life scenario. At the same time, Figure 5.22 and Figure 5.23 presents the front-views of the Flapper Nimble+ for a better understanding of the challenges imposed by the large wings when it comes to sensor placement.

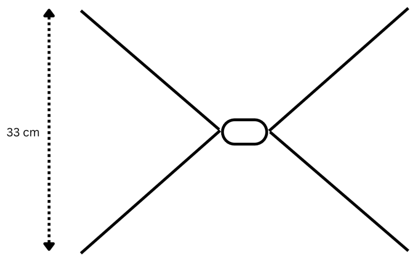


Figure 5.20: Top-view drawing of the extended wings.

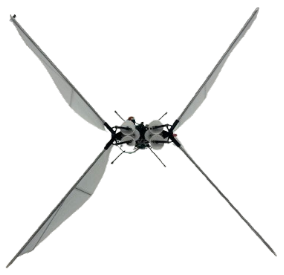


Figure 5.21: Top-view real image of the extended wings.

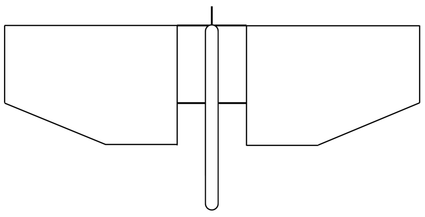


Figure 5.22: Front-view drawing of the Flapper Nimble+.

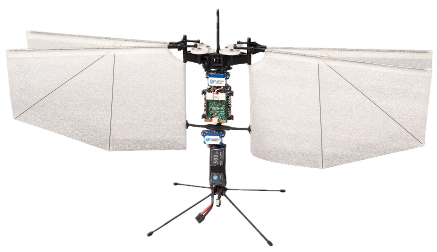


Figure 5.23: Front-view real image of the Flapper Nimble+.

Another factor in sensor placement is the vibration induced by the drone's flapping mechanism. Since the ToF sensor relies on transmitting laser pulses and measuring their reflections, these mechanical oscillations can introduce noise into the depth readings and affect measurement stability. The bottom section of the drone, in particular, experiences higher vibration amplitudes, which can pose challenges for sensors requiring consistent alignment and stability. However, experimental results suggest that these vibrations may also have a beneficial side effect: by causing slight oscillations in the sensor's field of view, they effectively enable a broader scanning area. This can improve obstacle visibility and make obstacles more prominent in the depth matrix, especially during flight. Therefore, while vibration presents a source of noise, it can also enhance environmental perception under certain conditions.

Finally, it should be noted that since the sensors are to be fixed to the structure of the drone, attitude changes will mean that the sensors will also experience a tilt, making it challenging to see obstacles that are directly in front of the drone while it is pitching or rolling.

Considering all the limitations, the most suitable place to mount the sensors was found to be the bottom section of the drone. This is because it is virtually the only place where the wings will not obstruct the lateral sensors. A sensor facing downwards must be placed at the very end of the bottom section, to be able to always keep track of the altitude at which the drone is flying. Figure 5.24 shows the placement of the downwards facing sensor. In the end, a forward-facing sensor will also be placed on the bottom part of the drone, but facing forwards instead of downwards.

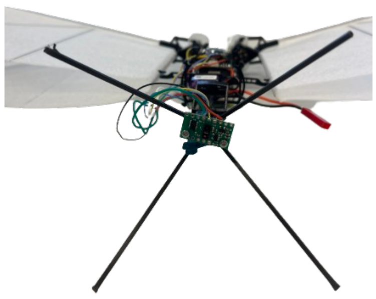


Figure 5.24: Down ToF Sensor

5.3. Raspberry PI

Connecting the ToF sensors directly to the Flapper Nimble+ drone was not feasible due to several limitations in communication interfaces, namely that the Flapper Nimble+ was not able to recognize the sensors. Instead, an external Raspberry Pi was used as an intermediary processing unit.

Connecting the sensors to the Raspberry Pi Zero 2 provides several advantages. By offloading the depth value analysis to the Raspberry Pi, the computational load is completely removed from the drone's onboard electronics. The Pi processes the raw depth data, applies necessary filtering and noise reduction, and then translates these readings into high-level attitude commands. These commands, such as roll, pitch, yaw or thrust adjustments, can then be directly transmitted to the drone via UART, ensuring a streamlined control loop.

5.4. Overall Configuration and Interaction

The system is designed to efficiently handle obstacle detection by integrating ToF sensors directly onto the Flapper Nimble+. These sensors continuously collect depth data and transmit it to a Raspberry Pi Zero 2, which serves as the processing unit for perception and decision-making. The Pi analyzes the depth information and converts the data into attitude and thrust adjustment commands. These commands, determining necessary yaw, pitch, roll and thrust corrections, are then sent to the drone via UART. This interaction is presented in Figure 5.25.

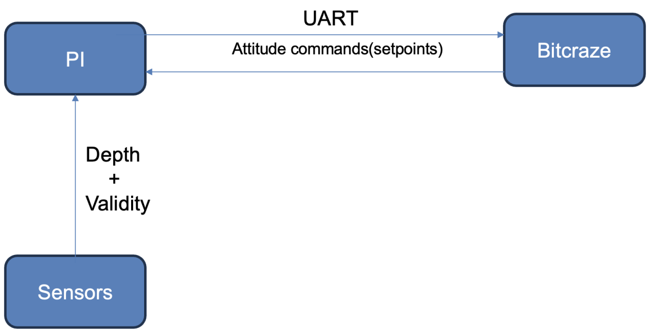


Figure 5.25: Sensor - Raspberry PI - Bitcraze interaction

6 Simulator

This chapter presents the simulation framework used to develop and evaluate control strategies for the Flapper Nimble+. A simulation provides a safe, efficient, and scalable environment for algorithm development, in which different applications and settings can be tested before real-world implementation. The chapter begins by introducing the Isaac Gym simulator and the Aerial Gym extension, detailing why this platform was chosen for its GPU-accelerated physics and reinforcement learning support. It then describes the modeling of the Flapper Nimble+, including its dynamics, drag behavior, control structure, and sensor emulation. Finally, it outlines the simulated environments used to assess obstacle avoidance capabilities under different navigation scenarios.

6.1. Set-Up

The simulator used throughout this research is Isaac Gym, a physics-based simulation platform developed by NVIDIA. Isaac Gym was chosen due to its ability to provide high-performance physics simulations with GPU acceleration, making it well-suited for training and testing reinforcement learning (RL) algorithms in real-time. Unlike traditional simulators that rely on CPU-based physics engines, Isaac Gym leverages NVIDIA GPU parallelism, enabling the simulation of thousands of environments simultaneously. This parallelization significantly accelerates the training process, allowing for faster policy optimization and more robust RL model development.

In this work, the simulation environment was built upon the Aerial Gym Simulator[1], an open-source, high-fidelity physics-based simulator specifically designed for training MAV platforms such as multirotors. Aerial Gym provides realistic physics modeling, sensor emulation, and customizable environmental conditions, making it an ideal testbed for reinforcement learning-based flight control. It integrates seamlessly with Isaac Gym, utilizing the GPU-accelerated physics engine to simulate aerial robotics scenarios with high efficiency. The key features of Aerial Gym are:

- **Modular & Extendable Design** – Easily create custom environments, robots, sensors, tasks, and controllers, with dynamic parameter modification.
- **Rewritten from the Ground-Up** – Provides extensive control over simulation components for full customization.
- **High-Fidelity Physics Engine** – Utilizes NVIDIA Isaac Gym for realistic multirotor physics, with support for custom physics engines and rendering pipelines.
- **Parallelized Geometric Controllers** – GPU-based controllers enable simultaneous control of thousands of multirotor vehicles.
- **Custom Rendering Framework** – Built on NVIDIA Warp, allowing custom sensor design and parallelized kernel-based operations.
- **RL-Based Control & Navigation** – Supports reinforcement learning policies for robot learning tasks, with scripts for training custom robots

[1] `https://github.com/ntnu-arl/aerial_gym_simulator`

6.2. Drone Model

For a simulation to be reliable, the drone must be modeled as accurately as possible to match its real-world counterpart. This includes capturing the physical properties, aerodynamics, actuation mechanisms, and sensor characteristics to ensure realistic behavior within the virtual environment. However, the exact dynamic model of the Flapper Nimble+ is not publicly available. As a result, the simulation model was adapted to replicate the key behavioral characteristics of the platform rather than its full dynamics. Specifically, the simulation captures the absence of egomotion and position feedback, the high-frequency vibrations typical of flapping wing flight, and the significant drag effects resulting from the large wing surface. These elements were prioritized to reflect the operational constraints of the real drone and to ensure that control strategies trained in simulation would generalize to the physical platform.

6.2.1. Controller

The Flapper Nimble+ is an attitude-stable drone, meaning it can regulate its orientation and maintain a desired target for pitch, roll, or yaw rate. However, a key limitation is its lack of ego-motion awareness: it cannot directly measure its velocity, position, or acceleration. This makes stabilization and hovering particularly challenging. To replicate these constraints in the simulation, a Lee Attitude Controller was implemented for this research.

The Lee Attitude Controller [78] is a geometric control method designed to regulate the orientation of a rigid body directly on the SO(3) manifold. It employs a proportional-derivative (PD)-like feedback law to track desired roll, pitch, and yaw rate commands while ensuring smooth convergence and stability. The controller takes as inputs the current robot state and desired command actions, and outputs the thrust and torque needed to maintain or transition to the desired orientation.

The first step involves computing the current orientation of the drone, represented as a quaternion, which is then converted to a rotation matrix $R \in \mathrm{SO}(3)$. This matrix is further decomposed into ZYX Euler angles to extract roll, pitch, and yaw:

$$(\phi, \theta, \psi) = \mathrm{EulerZYX}(R)$$

The desired orientation is constructed by keeping the current yaw ψ unchanged, while commanding the desired roll and pitch from user input:

$$(\phi_d, \theta_d, \psi_d) = (\mathrm{cmd}_{\mathrm{roll}}, \mathrm{cmd}_{\mathrm{pitch}}, \psi)$$

This is converted back into a desired rotation matrix $R_d \in \mathrm{SO}(3)$.

To track this desired orientation, the controller computes the angular velocity $\mathbf{\Omega}_d \in \mathbb{R}^3$ required in the body frame. Since Euler angle rates are not directly usable for control, a transformation matrix $T(\phi, \theta) \in \mathbb{R}^{3\times3}$ is used to convert desired Euler rates to body-frame angular velocity:

$$\mathbf{\Omega}_d = T(\phi, \theta)\dot{\mathbf{\Theta}}_d$$

where $\dot{\mathbf{\Theta}}_d = [0, 0, \dot{\psi}_d]^T$ and $T(\phi, \theta)$ is:

$$T(\phi, \theta) = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix}$$

The rotation error $\mathbf{e}_R \in \mathbb{R}^3$ is defined using the vee-map of the skew-symmetric part of the rotation difference:

$$\mathbf{e}_R = \frac{1}{2}(R_d^\top R - R^\top R_d)^\vee$$

This error vector represents the shortest angular displacement needed to align R with R_d.

Next, the angular velocity error $\mathbf{e}_\Omega \in \mathbb{R}^3$ is computed by mapping both actual and desired angular velocities into the same frame:

$$\mathbf{e}_\Omega = \mathbf{\Omega} - R^\top R_d \mathbf{\Omega}_d$$

The final control torque $\boldsymbol{\tau} \in \mathbb{R}^3$ is computed via the following feedback law:

$$\boldsymbol{\tau} = -K_R \mathbf{e}_R - K_\Omega \mathbf{e}_\Omega + \boldsymbol{\Omega} \times \boldsymbol{\Omega}$$

where:

- K_R is the rotation error gain,
- K_Ω is the angular velocity error gain,
- the cross product $\boldsymbol{\Omega} \times \boldsymbol{\Omega}$ accounts for gyroscopic effects (simplified here with unit inertia).

The thrust command is given as:

$$T = u_{\text{thrust}} + 1$$

where u_{thrust} is the normalized user-specified thrust (in simulation, centered around 0). This normalized thrust and the computed torque vector $\boldsymbol{\tau}$ are passed to the low-level motor controllers to generate the required rotor forces and moments.

6.2.2. Drag Model

The Flapper Nimble+ experiences significant horizontal drag due to its large wings, which substantially limit its horizontal velocity. The drag forces acting in the x and y directions are governed by the equations derived by [79], as presented in Equation 6.1 and Equation 6.2.

$$v_{x,k+1} = v_{x,k} + \mathrm{d}t \left(\tan\theta_k \cdot |g| - \cos^2\theta_k \cdot b_x v_{x,k} \right) \tag{6.1}$$

$$v_{y,k+1} = v_{y,k} + \mathrm{d}t \left(-\frac{\tan\phi_k}{\cos\theta_k} |g| - \cos^2\phi_k \cdot b_y v_{y,k} \right) \tag{6.2}$$

According to the findings of [79], the drag coefficients were determined to be $b_x = 4.2$ and $b_y = 1.8$.

In a more simplified form, the drag model could be reduced to Equation 6.3[79]. However, b_z is unknown, and thus will be considered 0 during this research.

$$\boldsymbol{f}_d^b = \begin{bmatrix} -b_x v_x^b \\ -b_y v_y^b \\ -b_z v_z^b \end{bmatrix} \tag{6.3}$$

6.2.3. Sensors Placement

The only way to ensure that the drone has some environmental perception is by equipping it with some sensors. As previously discussed, the sensors used are Time-of-Flight sensors, specifically the VL53L8CX. These sensors have a detection range of maximum 4 meters, and they output an 8x8 depth matrix. These sensors were simulated in the IsaacGym environment.

In Figure 6.1, the drone detects a thin object using its front-facing sensor. White pixels indicate the absence of detection, while gray pixels signify the presence of an obstacle in close proximity. The specific depth values that generated this detection are shown in Figure 6.2.

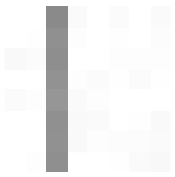
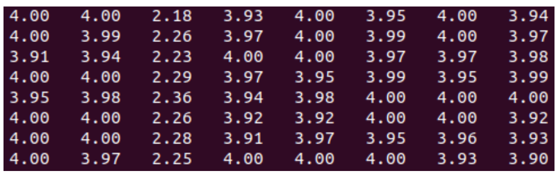


Figure 6.1: Simulated Sensor Detection



```
4.00   4.00   2.18   3.93   4.00   3.95   4.00   3.94
4.00   3.99   2.26   3.97   4.00   3.99   4.00   3.97
3.91   3.94   2.23   4.00   4.00   3.97   3.97   3.98
4.00   4.00   2.29   3.97   3.95   3.99   3.95   3.99
3.95   3.98   2.36   3.94   3.98   4.00   4.00   4.00
4.00   4.00   2.26   3.92   3.92   4.00   4.00   3.92
4.00   4.00   2.28   3.91   3.97   3.95   3.96   3.93
4.00   3.97   2.25   4.00   4.00   4.00   3.93   3.90
```

Figure 6.2: Depth Values of the Detection

To account for the vibrations present in the real drone, a significant amount of Gaussian noise was introduced to the sensor readings to simulate realistic conditions.

The sensor configuration follows two different approaches: one where the drone is equipped with a total of five sensors, and another where it is equipped with only two sensors. The objective is to determine the minimum number of sensors required to achieve effective obstacle avoidance.

I. Five-Sensor Configuration. In this setup, the drone has comprehensive coverage of its surroundings. It is equipped with a downward-facing sensor for altitude estimation, a front-facing sensor for detecting obstacles ahead, a back-facing sensor, and sensors on both the left and right sides for lateral obstacle detection. This configuration ensures full environmental awareness, as illustrated in Figure 6.3.

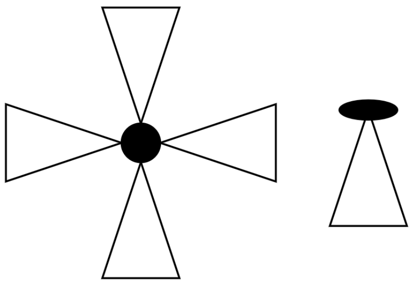


Figure 6.3: Five-Sensor Configuration

II. Two-Sensor Configuration. This setup reduces the number of sensors to the minimum required for basic navigation. The drone is equipped only with a downward-facing sensor to estimate altitude and a front-facing sensor for detecting obstacles. This configuration tests whether obstacle avoidance can still be achieved with a minimal sensor suite. A schematic representation of this setup is provided in Figure 6.4.

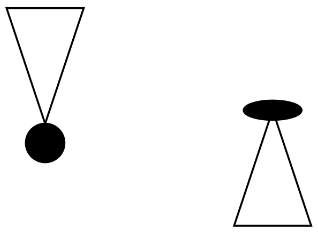


Figure 6.4: Two-Sensor Configuration

6.3. Simulated Environment

One of the primary motivations for this research is to enable the Flapper Drone to autonomously navigate greenhouse environments without colliding with crops. Greenhouses are highly cluttered spaces, densely populated with thin and irregularly placed obstacles, which makes reliable obstacle avoidance a non-trivial challenge. To evaluate the drone's navigation capabilities under such conditions, four distinct simulation environments were developed. Each environment targets specific aspects of autonomous navigation, ranging from general obstacle avoidance to precise maneuvering in constrained spaces.

1. Columns Obstacle Environment

The first environment features randomly distributed vertical columns acting as obstacles. These columns are placed throughout the simulation space without any structured layout. The goal is to test the drone's general obstacle detection and avoidance capabilities in a cluttered and unpredictable setting.

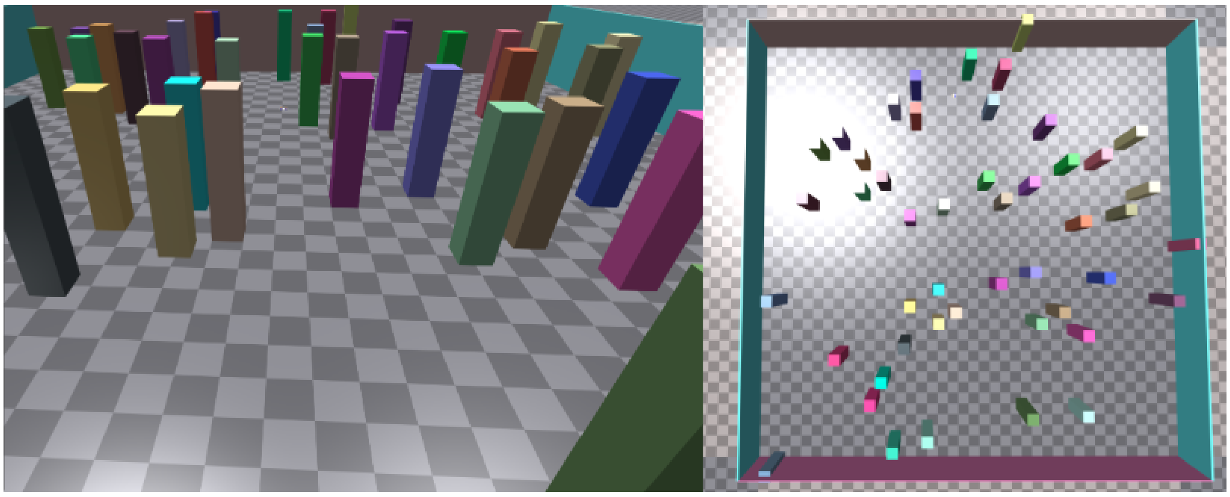


Figure 6.5: Columns Obstacle Environment

2. Maze-like Corridor Environment

The second environment presents a structured maze composed of narrow corridors and turns, mimicking tight indoor paths. Unlike the random nature of the previous scenario, this setup evaluates the drone's ability to perform precise maneuvers in confined spaces.

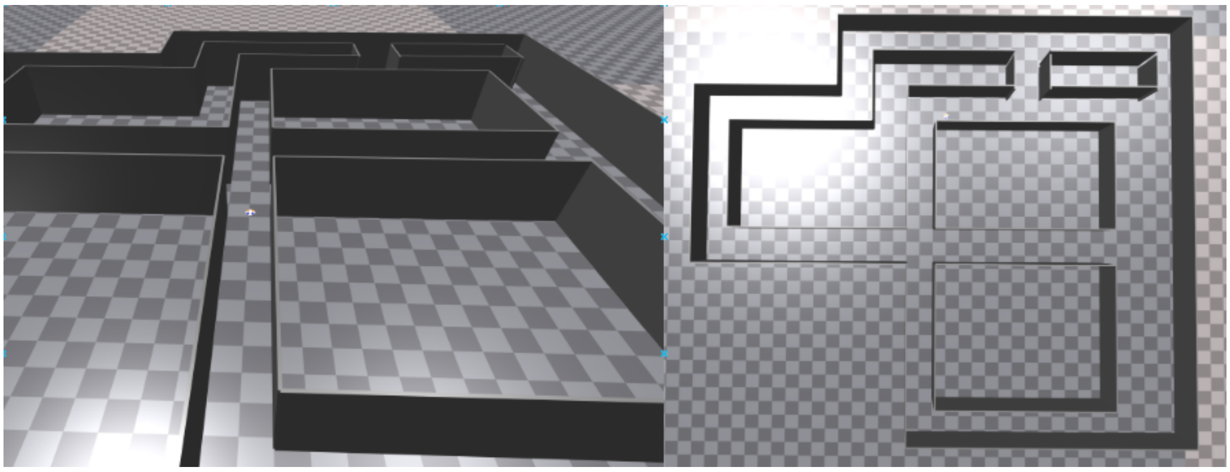


Figure 6.6: Maze-like Corridor Environment

Together, the first two environments test complementary aspects of flight control: the former focuses on adaptive real-time autonomous navigation, while the latter emphasizes path-following and precision in tight environments.

3. Tree-like Obstacle Environment

The third environment simulates a more chaotic and naturalistic scenario by introducing tree-like vertical obstacles with irregular spacing. This setup is designed to test the drone's ability to handle clutter that mimics real-world natural environments. The combination of narrow gaps and inconsistent distribution of obstacles poses a significant challenge for sensor-based navigation.

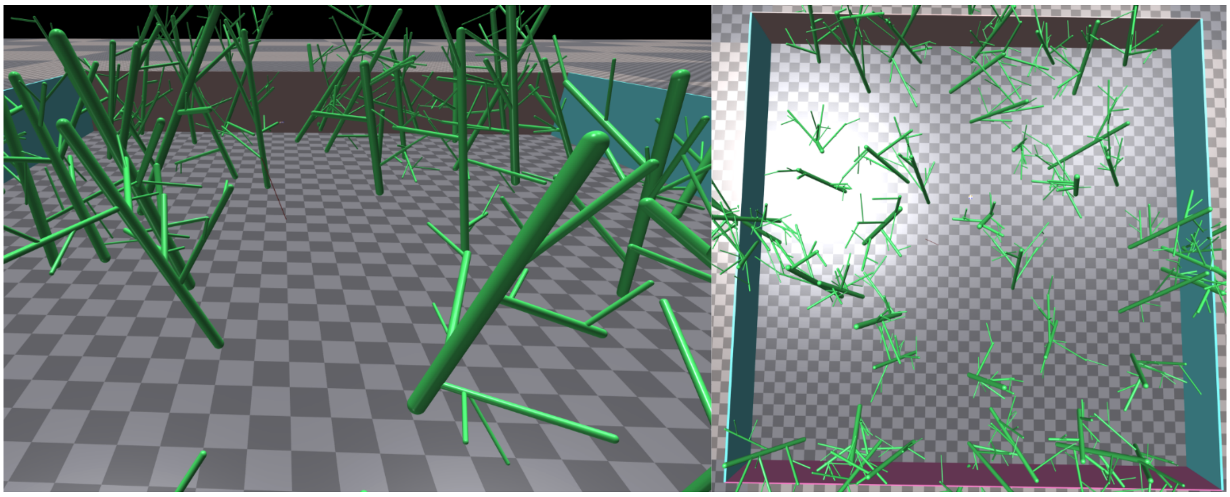


Figure 6.7: Tree-like Obstacle Environment

4. Wall Obstacle Environment

The fourth environment introduces inclined wall-like obstacles arranged in a staggered pattern. These walls create slanted surfaces that challenge the drone's perception and control, particularly when moving through sloped or partially obstructed paths.

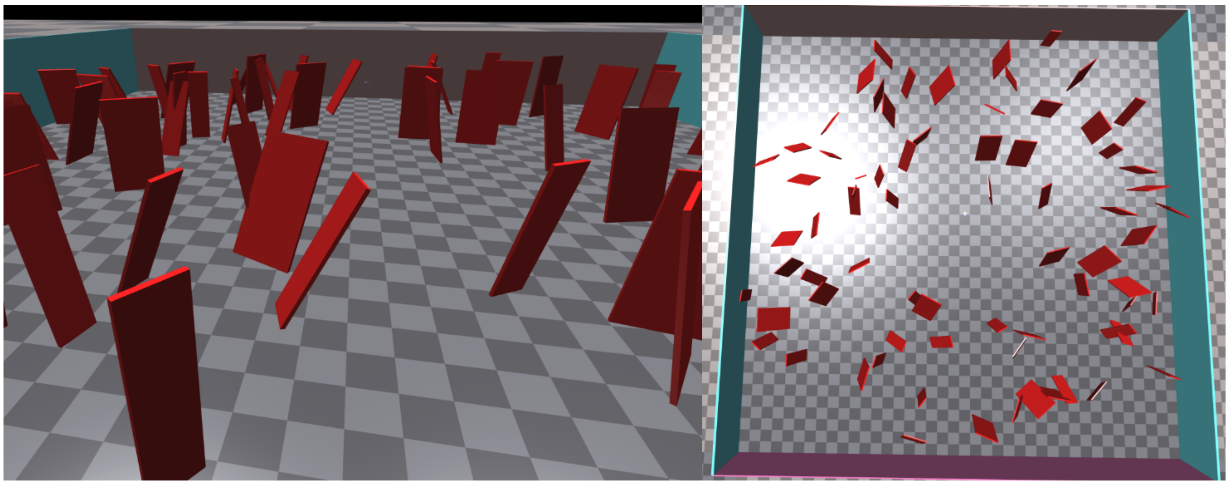


Figure 6.8: Inclined Wall Obstacle Environment

Each of these environments was carefully chosen to incrementally increase difficulty while evaluating different aspects of the drone's navigation pipeline. Success in all scenarios suggests that the proposed control and perception methods are robust enough for deployment in complex, real-world environments.

7 PID-Based Control

Proportional-Integral-Derivative (PID) control is one of the most widely used techniques in classical control theory, known for its simplicity, effectiveness, and versatility. In the context of this research, PID controllers are employed to regulate the drone's movements, enabling it to maintain stability and navigate autonomously. The primary objective of using PID control in this study is to enable reactive obstacle avoidance while maintaining stable flight. The controller adjusts the drone's thrust, roll, pitch, and yaw rate in response to sensor feedback, allowing it to navigate through unknown environments.

7.1. Fundamentals of PID Control

A PID controller consists of three fundamental components — Proportional (**P**), Integral (**I**), and Derivative (**D**) — each contributing uniquely to the system's response behavior:

- **Proportional (P) Control:** This component produces a control output that is directly proportional to the error — the difference between the desired and actual state. It provides immediate corrective action.

  $$P = K_p \cdot e(t) \tag{7.1}$$

  where K_p is the proportional gain, and $e(t)$ is the error at time t.

  Interpretation: This term reacts to the current error. A higher K_p results in stronger and faster corrections, but if too high, it can lead to oscillations or overshoot.

- **Integral (I) Control:** This component accumulates past errors to eliminate long-term steady-state offset.

  $$I = K_i \int e(t)\, dt \tag{7.2}$$

  where K_i is the integral gain.

  Interpretation: The integral term acts as a memory of past performance. It corrects biases (e.g., drift or steady error), but excessive K_i can cause overshoot or instability.

- **Derivative (D) Control:** This component predicts future error trends by responding to the rate of change of the error.

  $$D = K_d \frac{de(t)}{dt} \tag{7.3}$$

  where K_d is the derivative gain.

  Interpretation: The derivative term acts as a damping force. It reduces overshoot and helps stabilize the response, especially when approaching the target. However, it is sensitive to noise.

The combination of these three terms forms the complete PID control law, presented in Equation 7.4:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt} \tag{7.4}$$

Here, $u(t)$ represents the control input (for example, thrust adjustment, roll, pitch, or yaw correction) applied to the drone.

A block diagram of a generic PID controller is shown in Figure 7.1.

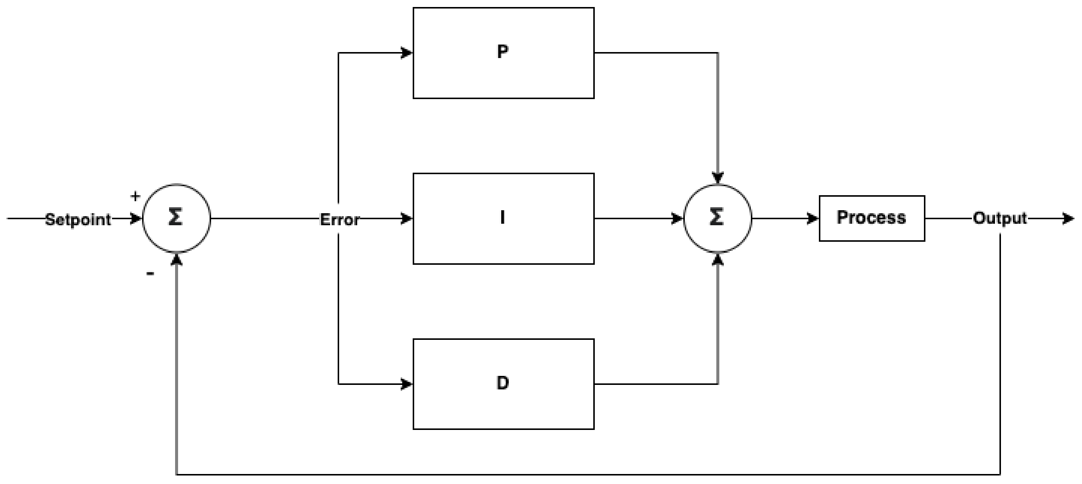


Figure 7.1: PID controller block diagram

7.2. Methodology

The drone was evaluated across all four simulated environments using the Two-Sensor Configuration. In this configuration, obstacle avoidance was managed using a PID controller, which allowed the drone to temporarily pause, analyze its surroundings, and then proceed with a navigational decision. As the avoidance strategy primarily relied on yaw adjustments (i.e., turning left or right), additional side-facing sensors would provide benefits, but they are not absolutely necessary. The forward-facing sensor alone was sufficient for detecting obstacles directly ahead and initiating avoidance maneuvers.

As described earlier, the drone receives four control inputs: thrust, roll, pitch, and yaw rate. Within this setup, a dedicated PID controller is used to regulate the thrust input in order to maintain a stable target altitude. This ensures that the drone neither drifts upward nor descends unnecessarily, helping maintain stable flight.

To avoid misinterpretation of the altitude due to the wide FoV of the ToF sensor, the downward-facing sensor's output is cropped to retain only the central 4×4 values of its 8×8 depth matrix (Figure 7.3). This focused cropping prevents peripheral depth values—such as those caused by nearby obstacles or slanted surfaces—from affecting the altitude control logic.

When the drone pitches forward during flight, the orientation of the downward-facing ToF sensor tilts with the body. As a result, the sensor no longer points directly downward, and it begins to detect points further away, often corresponding to the ground seen at an angle. This causes an artificial increase in the measured altitude. Since the PID controller operates solely on the reported sensor values, it interprets this increase as a genuine gain in height and responds by reducing thrust in an attempt to return to the target altitude. However, this correction is misleading, as the drone's actual altitude has not changed — only its orientation has. If left uncorrected, this behavior can lead to unintended descent and unnecessary oscillations.

To mitigate this effect, the raw altitude measurement is corrected by factoring in the drone's pitch angle. Specifically, the true vertical height is estimated by multiplying the measured depth by the cosine of the pitch angle, ensuring a more accurate representation of the drone's true altitude during dynamic maneuvers. The concept is illustrated in Figure 7.2.

Therefore, the altitude at each time step is calculated as Equation 7.5.

$$\text{True Altitude} = \text{Detected Altitude} \cdot \cos(\text{Pitch Angle}) \tag{7.5}$$

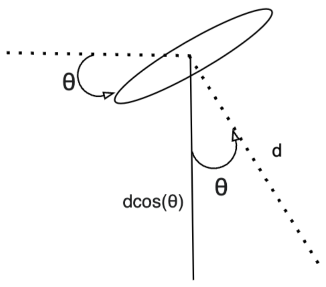


Figure 7.2: Altitude Correction

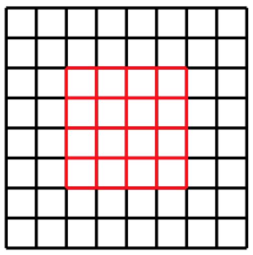


Figure 7.3: Altitude Reading Restriction

7.2.1. Obstacle Avoidance Setup

Although the Two-Sensor Configuration was found to be sufficient for safe navigation in most scenarios, it also imposes significant limitations. With perception restricted to only the forward and downward directions, the drone must rely entirely on forward vision to navigate safely. It continuously monitors the minimum distance detected in the forward direction to adjust its pitch and avoid frontal collisions, while the downward-facing sensor is used to regulate altitude. In the absence of lateral sensing, roll adjustments become unreliable, requiring the drone to rely solely on pitch and yaw commands to achieve successful navigation.

The PID coefficients shown in Table 7.1 were selected through an iterative trial-and-error process, guided by the drone's real-time behavior during test flights in simulation. The primary goal was to ensure smooth, responsive control while avoiding oscillations or overshooting

Table 7.1: PID Controller Parameters for Altitude, Pitch, and Roll

Controller	K_p	K_i	K_d	**Setpoint**
Altitude	0.007	0.0	0.005	1.4 meters
Pitch	0.05	0.0	0.01	2.0 meters
Roll	-	-	-	-

For altitude regulation, the proportional gain $K_p = 0.007$ and derivative gain $K_d = 0.005$ provide a gentle yet effective response to vertical disturbances. This conservative tuning prevents the controller from reacting too aggressively to sensor noise or sudden changes in perceived altitude — a common occurrence due to pitch-induced distortion in the downward-facing ToF readings. The integral term K_i was set to zero to avoid long-term drift compensation, which was deemed unnecessary given the short flight intervals and could lead to instability in the presence of sensor fluctuations.

The pitch controller was assigned a higher proportional gain $K_p = 0.05$ and a moderate derivative term $K_d = 0.01$, enabling the drone to make decisive forward tilts when navigating around obstacles. These values were tuned to provide fast pitch response without introducing oscillations. As with the altitude loop, the integral component was omitted to prioritize real-time responsiveness over long-term error correction.

The roll controller was intentionally disabled in this configuration, as the drone lacks lateral sensing. Without awareness of obstacles to its sides, roll-based corrections would be uninformed and potentially dangerous. Instead, lateral stability is passively maintained by fixing the roll command to zero, allowing pitch and yaw to serve as the primary steering mechanisms.

Avoidance Strategy

The limited sensory setup requires a more deliberate and adaptive strategy for navigating through an environment. While a basic reactive approach could be employed — where the drone stops, scans its surroundings, selects the freer direction, and resumes motion — this method is highly deterministic and inefficient. It often leads to abrupt stop-start behavior and increases the risk of the drone entering indecisive loops or getting stuck in cluttered regions.

To mitigate the limitations of this naïve approach, a more fluid navigation method was developed: Confidence-Based Decision Making (CBDM). Rather than reacting only when an obstacle is directly encountered, CBDM continuously evaluates the relative openness of the environment based on forward-facing sensor data. By gradually adjusting the yaw angle toward the clearer side, the drone is able to maintain smoother trajectories and avoid abrupt reorientations. This proactive steering behavior helps the drone anticipate and avoid potential bottlenecks.

If the drone does eventually become blocked and CBDM fails to identify a viable path, it reverts to a structured recovery behavior. In this mode, it performs an explicit yaw scan to assess left and right clearances, and then selects the most navigable direction.

The full sequence is as follows:

1. **Start Phase**
   The drone begins by thrusting to reach the target altitude.
2. **Pitching Phase**
   The drone starts pitching forward.
3. **Bias Yaw (Confidence-Based Decision Making)**
   It continuously evaluates obstacle clearance on the left and right using an Exponential Moving Average (EMA) of ToF depth readings. If one side consistently appears safer, the drone gradually biases its yaw toward that side.
4. **Check for Frontal Obstacle**
   The forward sensor checks for obstacles directly ahead.
   - If the path is clear, the drone continues pitching forward.
   - If an obstacle is detected, forward motion pauses.
5. **Yaw Scan (Recovery Step)**
   With forward motion halted, the drone performs a yaw scan — first left, then right — to assess potential side clearances.
6. **Select Left**
   If the scan indicates that the left path is unobstructed, the drone performs a left yaw to resume navigation.
7. **Select Right**
   If the left side is blocked but the right is clear, it yaws right instead.
8. **Emergency Exit (180° Turn)**
   If all directions are blocked (front, left, and right), the drone executes a 180-degree yaw turn to back out of the dead-end region.
9. **Loop**
   Resume from Step 1 to continue navigation.

Confidence-Based Decision Making
In this method, we analyze the depth data from the left and right sections of the forward-facing ToF sensor, denoted as D_L and D_R, which correspond to the leftmost and rightmost columns of the sensor's 8×8 depth matrix. To evaluate spatial clearance, we define a ”clearance intensity” by summing only those depth values that exceed a predefined obstacle threshold T, which indicates regions likely to be obstacle-free. These intensities are given by:

$$I_L = \sum_{d \in D_L,\, d>T} d, \quad I_R = \sum_{d \in D_R,\, d>T} d \tag{7.6}$$

To reduce noise and emphasize temporal consistency, we apply an Exponential Moving Average (EMA) to these intensities. The smoothed estimates at time step t are computed as:

$$E_L^t = \alpha I_L^t + (1-\alpha) E_L^{t-1}, \quad E_R^t = \alpha I_R^t + (1-\alpha) E_R^{t-1} \tag{7.7}$$

where $\alpha \in (0, 1)$ is the smoothing factor.

To track persistent trends, confidence counters C_L and C_R are incremented when the respective EMA is nonzero, and aggressively decayed otherwise. A dead zone threshold δ is applied to suppress yaw corrections in cases where the absolute difference between the EMAs is negligible. Specifically, if:

$$|E_R^t - E_L^t| < \delta, \tag{7.8}$$

then both confidence counters are reset.

When the difference exceeds the dead zone, a yaw correction command is computed as:

$$u_{\text{yaw}} = -\text{clip}(k \cdot \Delta E, -u_{\text{max}}, u_{\text{max}}), \quad \text{where} \quad \Delta E = E_R^t - E_L^t \tag{7.9}$$

Here, k is a scaling factor, and u_{max} is the saturation limit on the yaw adjustment. The negative sign ensures that if the right side appears clearer ($E_R > E_L$), the drone applies a leftward yaw correction, effectively steering toward the more open direction.

The avoidance process is depicted in Figure 7.4.

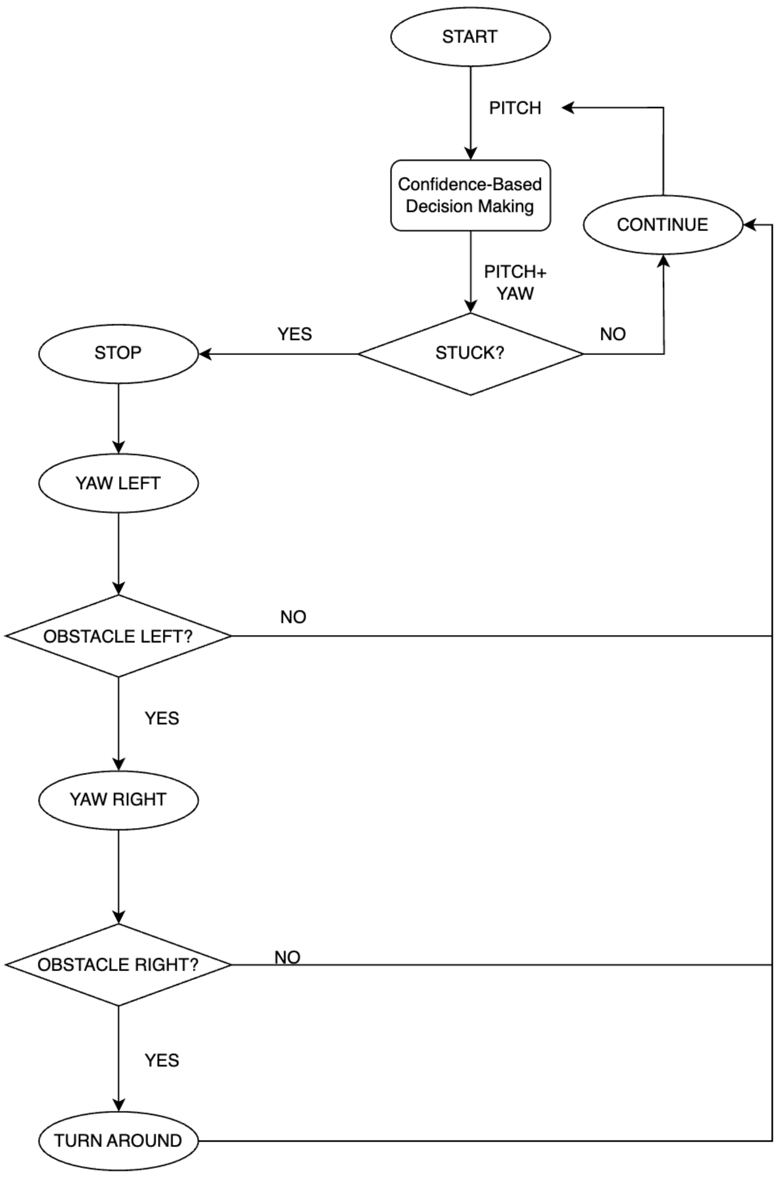


Figure 7.4: Decision-Making Diagram PID Control

8 Reinforcement Learning-Based Control

While classical control methods like PID offer deterministic and interpretable solutions for navigation, they are inherently limited in adapting to dynamic and unpredictable environments. Reinforcement Learning (RL) provides an alternative approach by allowing the drone to learn obstacle avoidance strategies through trial and error, rather than relying on pre-tuned control laws. By interacting with its environment, an RL agent can develop policies that optimize navigation efficiency while minimizing collision risks. This chapter explores how RL is applied to train a substitute model for the Flapper Nimble+ drone for autonomous navigation, focusing on designing state representations, defining reward structures, and selecting appropriate action spaces.

8.1. Objective

Unlike conventional control approaches that depend on manually tuned parameters, reinforcement learning provides an adaptive alternative by allowing the drone to learn effective obstacle avoidance strategies through interaction with the environment. Instead of relying on pre-defined rules, the agent discovers control policies that maximize performance based on reward feedback. This learning-based framework is particularly advantageous in dynamic and unpredictable environments where hard-coded behaviors often fail to generalize.

The primary objective of using RL in this study is to allow the drone to explore freely and avoid collisions without being constrained by fixed motion patterns. Rather than enforcing specific attitude behaviors, the agent learns to generate navigation decisions autonomously, guided only by its sensor inputs and the reward structure.

The goal is thus reduced to:

- **Explore the environment randomly** without predefined movement constraints.
- **Learn effective obstacle avoidance** purely from sensor feedback.
- **Adapt its control behaviour dynamically** without manually enforced (horizontal) control heuristics.

However, this approach does introduce several key challenges and limitations:

- **Limited Perception** – The drone is trained in two configurations:
  1. **Full perception (Five-Sensors Configuration)** – Allows omnidirectional awareness, making obstacle avoidance easier.
  2. **Limited perception (Two-Sensors Configuration)** – This configuration is much closer to the real flapper-wing drone but significantly restricts environmental awareness. The agent must learn to compensate for blind spots, relying only on limited depth data and reactive control to avoid collisions.
- **High Drag and Noisy Sensor Readings** – The drone experiences significant aerodynamic drag, affecting its ability to make rapid adjustments. Additionally, the ToF sensors introduce noise and uncertainty in depth measurements, making navigation more challenging.
- **Lack of Ego-Motion Awareness** – The drone does not have access to direct velocity, acceleration, or absolute position data. Unlike traditional navigation setups that rely on odometry or GPS, the agent must rely solely on ToF depth readings and past interactions to infer its motion state.

- **Stable and Interpretable Learning** – RL policies can exhibit unpredictable behaviors, especially when trained with limited perception. Without proper constraints, the agent might exploit loopholes in the reward function, leading to unnatural behaviors such as excessive yaw oscillations or hesitation near obstacles.

Thrust control remains managed by a PID controller. This decision is based on the PID controller's effectiveness in maintaining altitude stability using depth measurements from the downward-facing sensor. By delegating altitude control to PID, the RL agent is relieved of the additional burden of learning altitude stabilization, allowing it to focus solely on obstacle avoidance and navigation. This relationship is represented in Figure 8.1

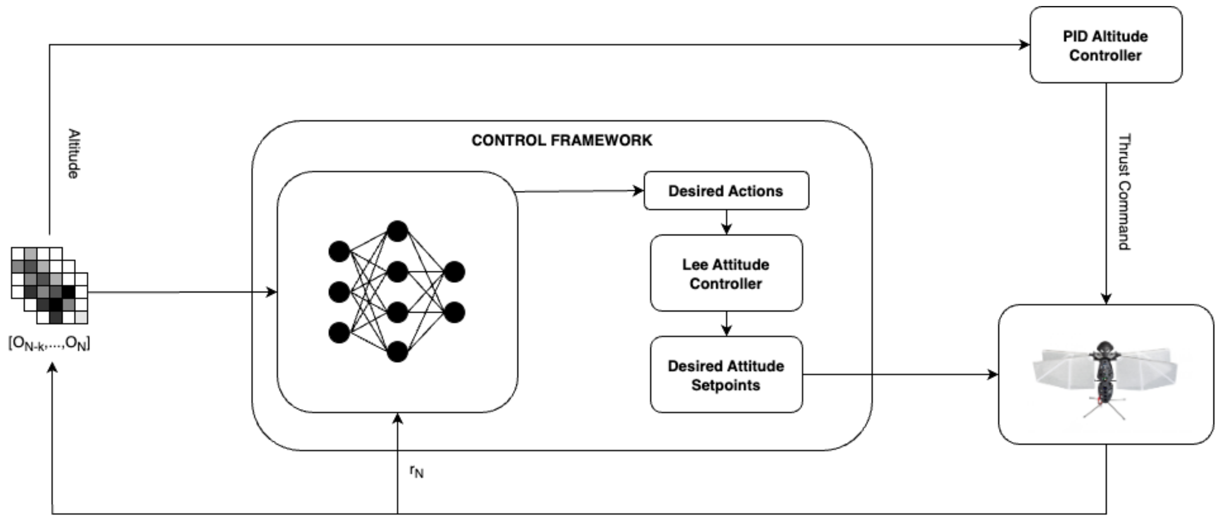


Figure 8.1: Reinforcement Learning Framework

8.2. Problem Formulation

Before applying Reinforcement Learning (RL) to train the drone, it is essential to define the information available to the agent and how it interacts with its environment. This involves specifying three key components: the **state space** (what the agent perceives), the **action space** (how it can control the drone), and the **reward function** (how it is incentivized to behave optimally).

8.2.1. State Space

To ensure stable learning and a balanced feature contributions state variables were normalized by scaling them to a common range. This prevents features with larger magnitudes, such as ToF sensor readings in meters, from dominating smaller-valued features like pitch angles in radians. Specifically, logarithmic scaling was applied to depth measurements to emphasize small distances and normalize them to [0,1], while angular values were scaled to [-1,1] to maintain consistency. This normalization helps stabilize training, reduces variance, and ensures that the RL agent explores the environment effectively without being biased toward any single input feature.

For distance perception, the ToF sensors are used. What was important here was to make sure that only relevant information is gathered from the sensors, as using the entire 8x8 output would increase the observation space significantly. Relying solely on the minimum ToF reading can cause excessive reactivity to single noisy measurements, leading to unstable navigation. Conversely, using only the mean ToF value provides a smoother representation but may overlook the nearest obstacles, increasing the risk of collisions. To balance sensitivity to close objects with robustness against noise, a combined approach using Equation 8.1 is applied. This method ensures that the agent remains aware of nearby hazards while also considering a broader view of the environment.

$$D_{\text{balanced}} = \frac{\text{min}(D) + \mu(D)}{2} \tag{8.1}$$

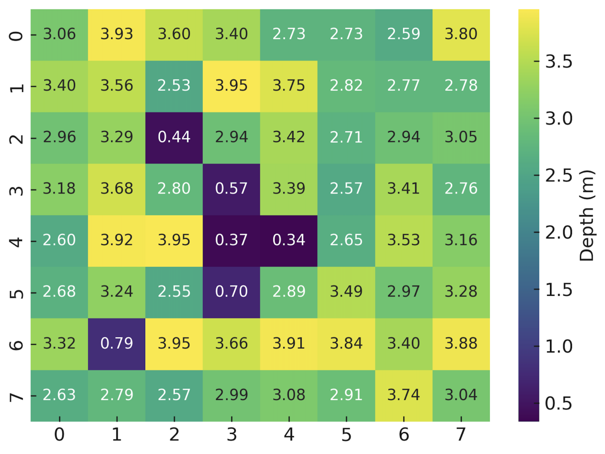


Figure 8.2: Heatmap ToF Readings Example

To exemplify this, look at Figure 8.2. In this scenario, $min(D)$ is equal to 0.34 and the mean, $\mu(D)$ is 2.94. This will output a balanced distance, D_{balanced} of 1.64. Although the balanced depth estimate D_{balanced} may not reflect the exact position of the nearest obstacle, it serves as a feature for decision-making. The goal is not to interpret this value as a hard measurement of proximity, but rather to provide the agent with a smoother observation that captures both immediate threats and overall spatial context.

When this feature is used as input to a reinforcement learning policy, the agent is not expected to act on the raw value directly. Instead, it learns through experience how different values of D_{balanced} correlate with safe or unsafe situations. If the drone frequently encounters low D_{balanced} values before collisions, the agent will learn to treat such readings as dangerous and take evasive actions. Over time, this leads to a policy that understands the implications of the observation, rather than relying on an exact geometric interpretation.

Including history points in the state representation improves the RL agent's decision-making by providing a temporal context. Since the drone lacks direct velocity or acceleration measurements, a single ToF reading only provides depth information at the current timestep, making it difficult to infer whether the drone is approaching or moving away from obstacles. By incorporating past ToF readings, pitch angles, and yaw rates, the agent can estimate motion trends and adjust its actions accordingly. This helps reduce overly reactive behavior, as the agent can distinguish between consistent obstacle proximity and momentary sensor noise. Additionally, history points allow the agent to develop a longer-term understanding of its movement, improving obstacle avoidance by detecting gradual depth changes instead of reacting to isolated observations. Furthermore, explicit history points act as a pseudo-memory, enabling the agent to approximate temporal relationships without requiring an internal state.

Normalization

Logarithmic normalization (Equation 8.2) is used for ToF sensor readings to enhance the RL agent's sensitivity to nearby obstacles while compressing larger distances. This is good for obstacle avoidance, as small distance changes near obstacles are more important than distant ones. Log scaling ensures that depth values remain within a consistent range, preventing large gradients that could destabilize learning. Additionally, it mimics human depth perception, where closer objects appear more significant for decision-making.

$$\text{Log Normalization} = \frac{\log_{1p}(D_{\text{balanced}} + 0.02)}{\log(\text{Max. Range} + 1)} \tag{8.2}$$

The term $+0.02$ is added inside the logarithm function to prevent numerical instability and avoid undefined values when the sensor reading is zero. Since $\log(0)$ is undefined, adding a small constant ensures that even for very small or zero readings, the logarithm function remains valid and smooth. Dividing by $\log(\text{max range} + 1)$ ensures that all depth values are normalized within the range $[0, 1]$, regardless of the actual sensor limits.

Both the pitch and yaw are divided by a factor of π to ensure that they stay between $[-1, 1]$.

Five-Sensors Configuration

In this configuration, the RL agent receives observations from five ToF sensors, providing a 360-degree depth perception around the drone. The state space is given in Table 8.1. In the end, this configuration has 14 observations.

Table 8.1: State Space - Five-Sensors Configuration

Raw Value	**Observation**
Pitch Angle	$\frac{\text{pitch}}{\pi}$
Yaw Rate	$\frac{\text{yaw rate}}{\pi}$
Front ToF	$\frac{\log_{1p}(\text{Front ToF}_{\text{balanced}}+0.02)}{\log(4.0+1)}$
Left ToF	$\frac{\log_{1p}(\text{Left ToF}_{\text{balanced}}+0.02)}{\log(4.0+1)}$
Right ToF	$\frac{\log_{1p}(\text{Right ToF}_{\text{balanced}}+0.02)}{\log(4.0+1)}$
Back ToF	$\frac{\log_{1p}(\text{Back ToF}_{\text{balanced}}+0.02)}{\log(4.0+1)}$
History of Previous States	Normalized History

Two-Sensors Configuration
The approach in this configuration follows the same principles as in the Five-Sensors Configuration. However, with only a single front-facing ToF sensor available for horizontal perception, certain adjustments are required to compensate for the reduced field of view.

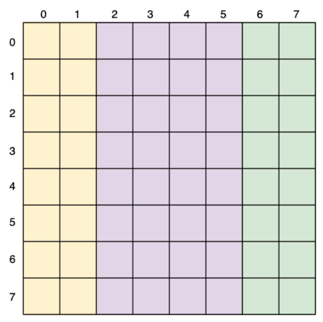


Figure 8.3: Regions of Front ToF

To extract more spatial information from the front-facing ToF sensor, the depth matrix was divided into three distinct regions: left, center, and right. This segmentation allows the RL agent to infer directional depth information despite having only one horizontal sensor. A visual representation of this division is shown in Figure 8.3, where each region captures depth values from a different section of the sensor's 8×8 grid.

Now, the yellow region is treated as the left sensor, the purple region as the front sensor and the green region as the right sensor. The same data pre-processing was done as in the previous configuration.

For each of the three regions, the same principle as before was considered (Equation 8.1).

Table 8.2 presents the state space of the Two-Sensors Configuration. It can be seen that the observations are almost the same as the Five-Sensors Configuration. The only observation missing is the input from the back sensor, which is not present here.

Table 8.2: State Space - Two-Sensors Configuration

Raw Value	**Observation**
Pitch Angle	$\frac{\text{pitch}}{\pi}$
Yaw Rate	$\frac{\text{yaw rate}}{\pi}$
Front ToF	$\frac{\log_{1p}(\text{Front ToF}_{\text{balanced}}+0.02)}{\log(4.0+1)}$
Left ToF	$\frac{\log_{1p}(\text{Left ToF}_{\text{balanced}}+0.02)}{\log(4.0+1)}$
Right ToF	$\frac{\log_{1p}(\text{Right ToF}_{\text{balanced}}+0.02)}{\log(4.0+1)}$
History of Previous States	Normalized History

8.2.2. Action Space

For both configurations, the network will output the same number of actions, as the objective remains obstacle avoidance by adjusting only the drone's pitch and yaw. Consequently, the network produces two action outputs: **pitch** and **yaw**. Rolling was explicitly restricted, as it is not necessary for navigation and would unnecessarily increase the network's complexity without providing additional control benefits.

Moreover, as mentioned before, the thrust is controlled solely by a PID controller. The way the thrust action is generated is explained in Section 7.2. Table 8.3 gives a summary of the action space.

Table 8.3: Action Space

Configuration	Action 1	Action 2	Action 3
Five-Sensors	Pitch Adjustment	Yaw Adjustment	Thrust from PID
Two-Sensors	Pitch Adjustment	Yaw Adjustment	Thrust from PID

8.2.3. Reward Function

Designing an effective reward function is a central aspect of Reinforcement Learning, as it directly influences the agent's ability to learn optimal strategies. The reward function serves as the primary feedback mechanism, guiding the RL agent toward behaviors that achieve collision-free flight while discouraging suboptimal actions. Given the constraints of the drone, the reward structure must be carefully designed to balance exploration, safety, and efficiency.

The reward function is structured to ensure the drone prioritizes avoiding collisions, maintains a small forward pitch, and primarily relies on yaw adjustments for obstacle avoidance. Additionally, an exploration incentive prevents the agent from staying in one location for extended periods, encouraging it to interact dynamically with its environment.

Potential Issues with Reward Design

One of the main challenges in reinforcement learning is balancing sparse and dense reward signals. Sparse rewards — such as penalizing only collisions or giving feedback only at episode termination — can hinder learning by making it difficult for the agent to associate actions with long-term outcomes. Conversely, dense rewards provide more frequent feedback, helping the agent reinforce desirable behaviors. For instance, issuing small penalties as the drone approaches obstacles encourages early avoidance rather than last-second corrections. However, if the reward is too frequent or overly detailed, the agent may focus on avoiding immediate penalties rather than developing a robust long-term navigation strategy. This effect, often referred to as "reward saturation," can lead to overly cautious behaviour.

Another risk is reward exploitation, where the agent maximizes the numerical reward without achieving meaningful navigation. For instance, a time-based survival bonus could lead to the agent hovering in place indefinitely. Similarly, if only collisions are penalized without any forward incentive, the agent may learn to spin in place to avoid obstacles, rather than moving through the environment. These behaviors highlight the need for auxiliary constraints, such as penalizing yaw oscillations or rewarding consistent forward motion.

This issue is exemplified in Figure 8.4.

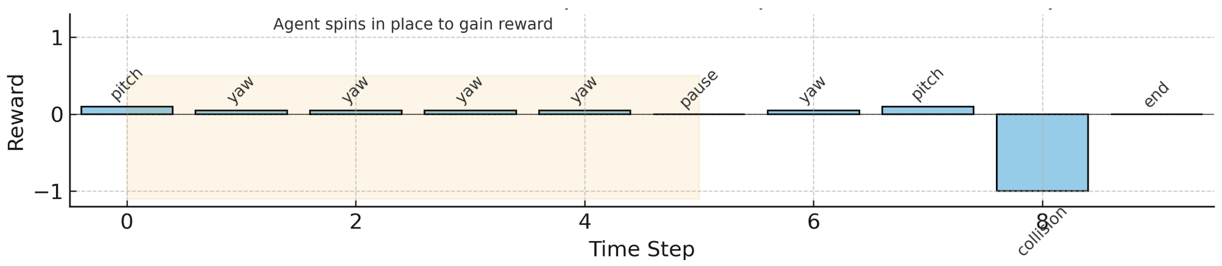


Figure 8.4: Reward Exploitation

The agent performs repeated yaw actions (i.e., turning in place) across several time steps without making meaningful progress toward navigation goals. Despite the lack of forward movement, it continues to receive small positive rewards. This unintended behavior demonstrates how an agent can exploit the reward structure by maximizing return without fulfilling the intended task. The episode eventually ends in collision, shown by the sharp penalty.

Expected Effects of the Reward Function
One of the central components of the reward structure is the collision penalty, which must be carefully tuned. If set too low, the agent may become reckless, frequently brushing or hitting obstacles without significant consequences. If too high, the agent may adopt overly conservative behaviors, avoiding exploration or becoming hesitant in tighter spaces. A balanced penalty, combined with early avoidance incentives, encourages both safety and maneuverability.

Yaw-based rewards also require careful calibration. While encouraging yaw to steer away from obstacles is important, overemphasizing it can lead to oscillatory behaviors or spinning in place. If the agent learns to favor turning over progressing forward, flight becomes inefficient and unstable. To address this, the reward function includes light penalties for abrupt or excessive yaw changes.

Five-Sensors Configuration
Since this configuration has a full perception of the environment, the reward function can be more relaxed. This being said, the drone is mainly rewarded for good behaviour and it is only penalized for colliding. This way, the RL will try to maximize the reward instead of being afraid of exploring.

Equation 8.3 shows the collision penalty, while Figure 8.5 shows a visual representation of this.

$$P_{\text{collision}} = \begin{cases} -10, & \text{if collision} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{8.3}$$

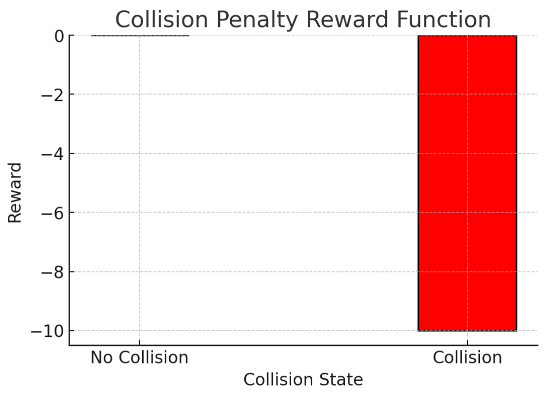


Figure 8.5: Collision Penalty Graph

To make the drone achieve a small, forward pitch, the reward function presented in Equation 8.4 was used.

$$R_{\text{pitch}} = e^{-5\cdot|\text{pitch}-0.1|} \tag{8.4}$$

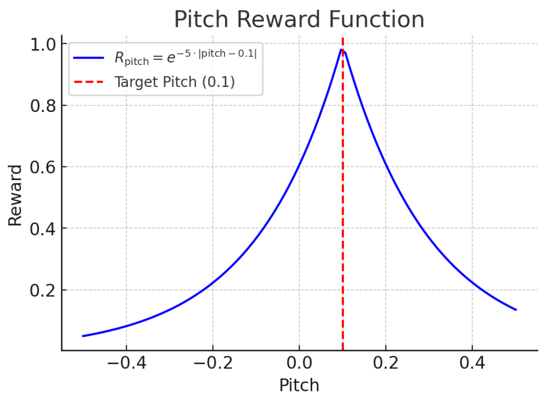


Figure 8.6: Pitch Reward Graph

To incentivize the drone to yaw when a potential obstacle is detected, Equation 8.5 was used. Note that 0.44 is approximately 1 meter normalized.

$$R_{\text{yaw}} = \begin{cases} 0.2 \cdot |\text{yaw rate}|, & \text{if front\_tof} < 0.44 \\ 0, & \text{otherwise} \end{cases} \tag{8.5}$$

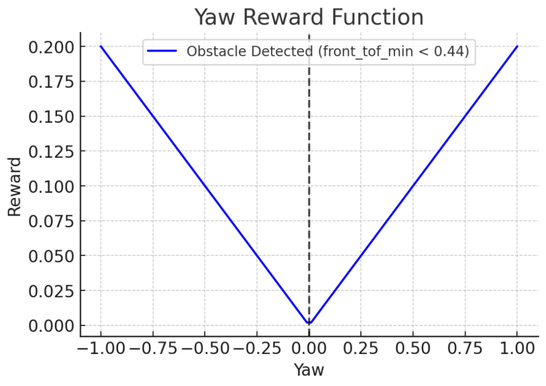


Figure 8.7: Yaw Reward Graph

Finally, to make the drone explore, a small reward for moving was given to the drone, as seen in Equation 8.6.

$$R_{\text{path}} = \begin{cases} 0.05, & \text{if distance traveled} > 0.02 \\ 0, & \text{otherwise} \end{cases} \tag{8.6}$$

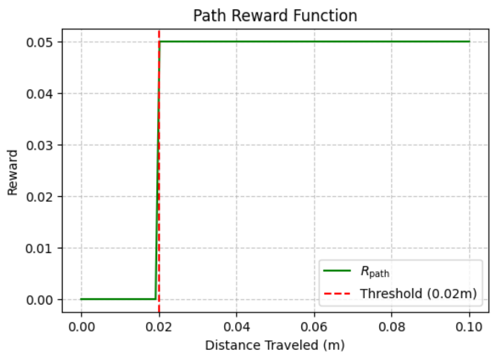


Figure 8.8: Path Reward Graph

In the end, at each step, the drone will obtain the total reward dictated by Equation 8.7.

$$R_{\text{total}} = P_{\text{collision}} + 0.3R_{\text{pitch}} + R_{\text{yaw}} + 0.5R_{\text{path}} \tag{8.7}$$

Two-Sensors Configuration

Since this configuration has limited perception, with only a front-facing ToF sensor, the reward function must be designed to strongly encourage forward motion while ensuring safe navigation. Unlike the full-perception setup, where the agent has access to a complete environmental view, this configuration introduces blind spots, particularly for obstacles behind the drone.

To address this limitation, the agent is given an even stronger incentive for pitching forward, as it helps avoid accidentally pitching backward into an unseen obstacle. While the reward structure still primarily reinforces good behavior, the absence of a rear sensor means that maintaining a slight forward pitch is even more important. The agent is also encouraged to explore by incorporating a yaw-based exploration reward, ensuring that it actively scans its environment rather than hesitating due to incomplete perception. Collision penalties remain strong to discourage reckless movement.

Equation 8.8 gives the drone a stronger penalty for collisions.

$$P_{\text{collision}} = \begin{cases} -25, & \text{if collision} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{8.8}$$

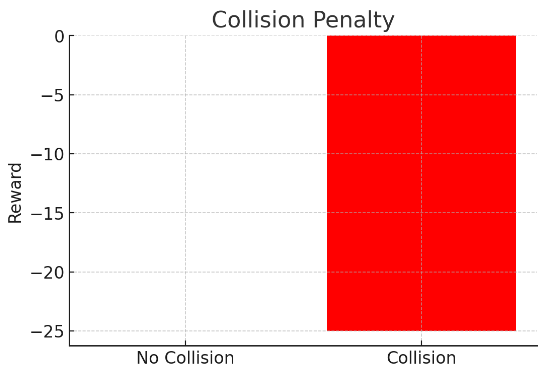


Figure 8.9: Collision Penalty Reward

The pitch reward is more aggressive, being more steep since it is even more important in this scenario to keep a small, forward pitch. The agent also does not get any reward for a slight backward pitch. Equation 8.9 shows this behaviour.

$$R_{\text{pitch}} = \begin{cases} e^{-6\cdot|\text{pitch}-0.07|}, & \text{if pitch} \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{8.9}$$

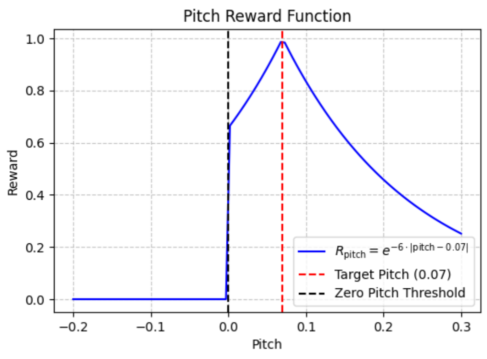


Figure 8.10: Pitch Reward Graph

Since yawing is even more important now do to the lack of lateral perception, the drone is pushed to yaw more, having a stronger reward for this action, as shown in Equation 8.10.

$$R_{\text{yaw exploration}} = \begin{cases} 0.2 \cdot |\text{yaw rate}|, & \text{if distance} \leq 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{8.10}$$

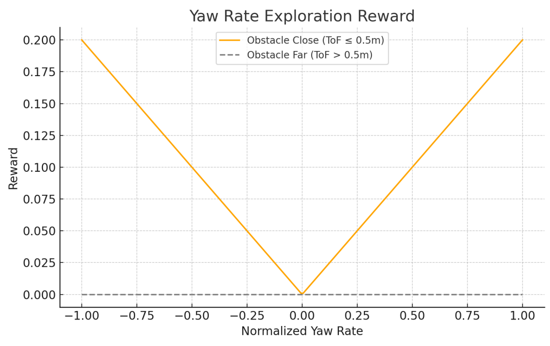


Figure 8.11: Yaw Reward Graph

To prevent the drone from getting stuck in corners and push her to yaw when stuck, Equation 8.11 gives a small reward for rotation. Equation 8.10.

$$R_{\text{yaw stuck}} = \begin{cases} 0.05, & \text{if distance - previous distance} > 0.2 \\ 0, & \text{otherwise} \end{cases} \tag{8.11}$$

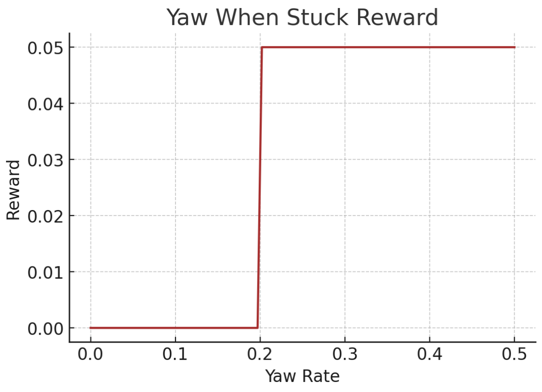


Figure 8.12: Yaw Stuck Reward Graph

In the end, the drone receives the reward dictated by Equation 8.12.

$$R_{\text{total}} = P_{\text{collision}} + 0.4 \cdot R_{\text{pitch}} + R_{\text{yaw exploration}} + R_{\text{yaw stuck}} \tag{8.12}$$

Reward Shaping Motivation
Each reward function component was selected to encourage specific behaviors in the agent. For pitch control, an exponential penalty was used to gently discourage small deviations while strongly penalizing large ones. This helped promote smooth, stable forward motion without overcorrection.

Yaw incentives were shaped using a linear absolute function, equally rewarding left and right turns. This encouraged active exploration while maintaining symmetry and avoiding erratic yaw behavior.

To promote meaningful movement, a step function was used: the agent receives a reward only if it moves more than 0.02 m in a time step. This prevents it from exploiting tiny oscillations and ensures forward progress.

Collisions incur a fixed penalty via a binary function, making crashes unambiguously bad. This discourages any strategy involving contact with obstacles.

Finally, a linear survival reward increases over time, reinforcing long-term stability and encouraging the agent to stay airborne as long as possible.

Reward Normalization
To ensure that the rewards remain withing reasonable range and do not destabilize training, reward normalization was applied (Equation 8.13).

$$R_{\text{final}} = 0.05 \times \text{clamp}\left(\frac{R - \mu_R}{\sigma_R + 1.0}, -2, 2\right), \quad \text{where} \quad \mu_R = \frac{1}{N}\sum_{i=1}^{N} R_i, \quad \sigma_R = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(R_i - \mu_R)^2 + 10^{-8}} \tag{8.13}$$

First, the mean reward (μ_R) and standard deviation (σ_R) are computed over a batch of rewards. The reward values are then centered by subtracting the mean and scaled by the standard deviation (with an added $+1.0$ for stability). This normalization ensures that rewards have a consistent magnitude, preventing extreme values from dominating training. Next, the result is clamped between $[-2, 2]$, limiting the impact of large rewards or penalties. Finally, the entire reward is scaled down by multiplying by 0.05, reducing its influence on the training process.

This normalization is a good addition because raw rewards can have high variance, leading to unstable policy updates in reinforcement learning. By keeping rewards within a controlled range, the training process becomes more stable, allowing the agent to learn effectively without being overly influenced by occasional large rewards or penalties. Clamping further prevents outliers from skewing learning, and scaling ensures smooth policy updates.

8.3. Reinforcement Learning Algorithm

The RL algorithm determines how the agent learns from its interactions with the environment. For the objective of this research, the chosen algorithm is the **Proximal Policy Optimization (PPO)**. The main reason behind this choice was that the PPO has the ability to handle continuous action spaces.

The architecture of the PPO algorithm is presented in Figure 8.13.

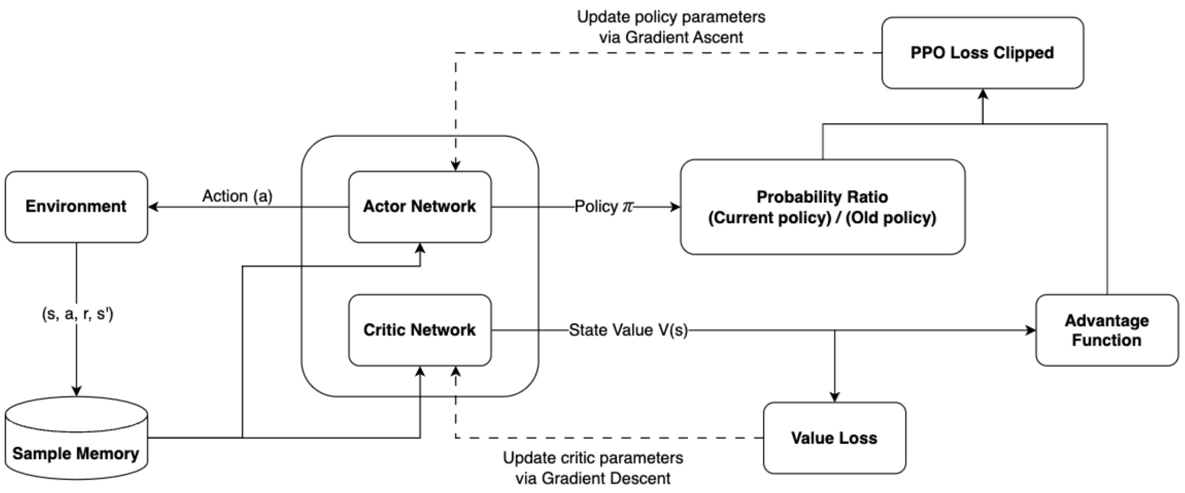


Figure 8.13: Proximal Policy Optimization (PPO) Architecture

PPO naturally supports continuous control by modeling actions using a Gaussian distribution, allowing smooth and gradual policy updates. Unlike discrete-action methods (e.g., DQN), PPO avoids discretization, which would limit the precision of movement.

Additionally, PPO uses a clipped objective function, preventing the policy from making large and unstable updates. This is essential because small changes in pitch or yaw can lead to significant deviations in drone behaviour, and PPO ensures controlled policy improvements. PPO is an on-policy algorithm, meaning it learns directly from its most recent interactions with the environment. This aligns well with the drone's real-time control requirements, where each action must be evaluated and adjusted in response to the latest sensor data.

The **Actor Network** and **Critic Network** are presented in Figure 8.16. Both of them have an input, 3 layers and an output. The activation function chosen was **tanh()**. The activation function **tanh()** was selected for both the Actor and Critic networks due to its desirable properties in continuous control tasks. Unlike ReLU, which can lead to unbounded outputs, **tanh()** squashes inputs to the range [-1, 1], promoting smoother gradients and bounded activations. This is particularly useful in reinforcement learning when the actions (Actor output) or value estimates (Critic output) need to remain within a specific, normalized range.

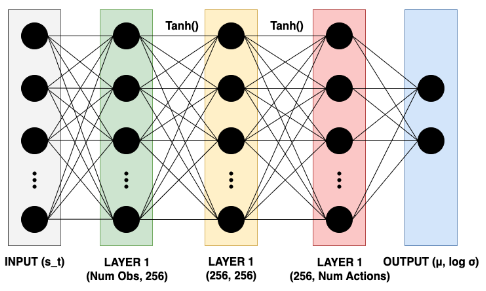


Figure 8.14: Actor Network

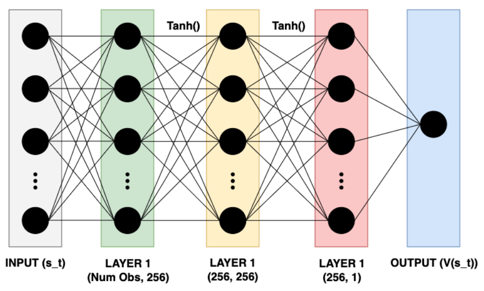


Figure 8.15: Critic Network

Figure 8.16: Actor-Critic Framework

Refer to Appendix A for a detailed explanation on how the algorithm works.

8.4. Hyperparameter Tuning

Hyperparameter tuning is at the base of an efficient reinforcement learning algorithm. They influence stability, convergence speed and overall performance. PPO introduces several hyperparameters that control different aspects of learning, such as policy updates, value function estimation, exploration strategies, and gradient optimization. They must be carefully tuned to balance exploration and exploitation.

The process of hyperparameter tuning involves running multiple training experiments with different parameter settings and evaluating the impact on the agent's learning performance. Metrics such as total episodic return, training stability, policy improvement rate, and sample efficiency help assess the effectiveness of different hyperparameter configurations.

The hyperparameters must be adjusted for both configurations. Table 8.4 presents the values chosen.

Table 8.4: Hyperparameter Tuning for PPO in Five-Sensors and Two-Sensors Configurations

Hyperparameter	**Five-Sensors Configuration**	**Two-Sensors Configuration**
Total Timesteps	30,000,000	30,000,000
Learning Rate	1.5×10^{-4}	1.5×10^{-4}
Num Steps	16	16
Anneal LR	True	True
Gamma (Discount Factor)	0.99	0.99
GAE Lambda	0.99	0.99
Num Minibatches	8	8
Update Epochs	4	4
Norm Advantage Off	False	False
Clip Coefficient	0.01	0.1
Clip Value Loss	True	True
Entropy Coefficient	0.008	0.007
Value Function Coefficient	2.0	2.0
Max Gradient Norm	1.0	1.0
Target KL	None	None

8.5. Training Process

The drone was trained entirely in a simulated environment, as described in Chapter 6, only under the Columns Obstacle Environment, and with significantly less obstacles. The objective during training is not to follow any predefined trajectory but to learn a general obstacle avoidance strategy that maximizes both survival time and environmental coverage. The agent starts with no prior knowledge and gradually improves its policy through trial and error.

To promote generalization and prevent overfitting to fixed obstacle patterns, each training episode begins with a randomized map configuration. Obstacle positions, orientations, and sometimes sizes are procedurally altered between episodes. This ensures that the agent encounters a variety of layouts and learns behaviors that are robust across environments.

The training loop proceeds as follows:

- At each timestep, the agent receives a state observation (see Table 8.1) and selects an action using its current policy.
- The environment responds with the next state and a reward, as described in Section 8.2.3.
- The transition is stored, and the policy is updated periodically using the PPO algorithm.

This loop continues for tens of thousands of episodes, with periodic evaluation checkpoints to assess policy performance on held-out environments not seen during training.

8.6. Policy Evaluation

To evaluate the performance of the trained policy, testing was conducted in the simulated environments described in Chapter 6. These environments were excluded during training to ensure that the agent's performance reflects its ability to generalize to new obstacle configurations.

Evaluation focused on two main criteria:

- The quality of exploration, assessed by analyzing the trajectory of the drone as it navigated the environment.
- The number of collisions, which provides a direct measure of the agent's ability to avoid obstacles.

These metrics allow for a qualitative and quantitative assessment of how well the policy performs in unfamiliar environments. The outcomes of this evaluation are discussed in Chapter 9.

9 Results

This chapter presents the results of the two control strategies evaluated in this study. First, the performance of the PID controller is assessed in terms of its ability to maintain a fixed altitude using only sensor feedback, and to navigate through the environment without collisions. Next, the focus shifts to the reinforcement learning-based approach, analyzing how the learned policies adapt to different environments. Finally, a comparison is made between the generalization capabilities of the RL-based method and the predefined control strategy.

9.1. PID-based Control

This section presents the results of the PID-based control approach used to regulate the drone's altitude and attitude (pitch, yaw and roll) while navigating through the simulated environment. The drone was tested in four environments:

1. **Columns Obstacle Environment**
2. **Maze-Like Corridor Environment**
3. **Trees Obstacle Environment**
4. **Walls Obstacle Environment**

Only one sensor configurations was evaluated: **Two-Sensors Configuration**

9.1.1. Altitude Control Performance

One of the primary objectives of the PID controller was to maintain a target altitude of **1.4 meters**, adjusting thrust based on real-time sensor feedback. Table 9.1 summarizes the key performance metrics, while Figure 9.1 provides a visualization of the altitude regulation over time.

Table 9.1: Altitude Control Performance Metrics

Performance Metric	Value
Rise Time	17 steps
Overshoot	8.50%
Peak Time	36 steps
Settling Time	47 steps

The *rise time* indicates how quickly the drone reaches a significant portion of its target altitude. At 17 steps, this reflects a reasonably responsive system with minimal delay. The *overshoot* of 8.5% remains within acceptable bounds, suggesting that the proportional gain is adequately tuned despite minor transient oscillations. The *peak time* of 36 steps corresponds to the point of maximum altitude, after which the controller gradually dampens the overshoot. Finally, the *settling time* of 47 steps demonstrates that the system stabilizes effectively without prolonged oscillations.

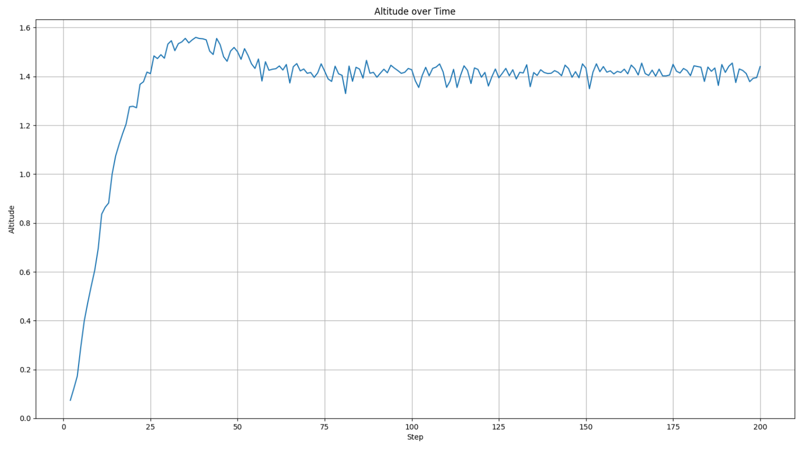


Figure 9.1: Altitude Control Performance

The altitude response in Figure 9.1 demonstrates a smooth initial rise as the drone ascends towards the target altitude. However, a slight overshoot is observed, with the altitude briefly exceeding 1.4 meters before stabilizing. This overshoot, measured at 8.5% in Table 9.1, indicates that the proportional gain provides sufficient responsiveness, though some initial momentum carries the altitude slightly beyond the desired level before correction occurs.

Once the altitude settles, the steady-state phase shows minor fluctuations around the intended 1.4-meter setpoint. These small variations are likely due to sensor noise and minor disturbances in the system. Nevertheless, the control system maintains the altitude within a narrow range, indicating stable and accurate performance.

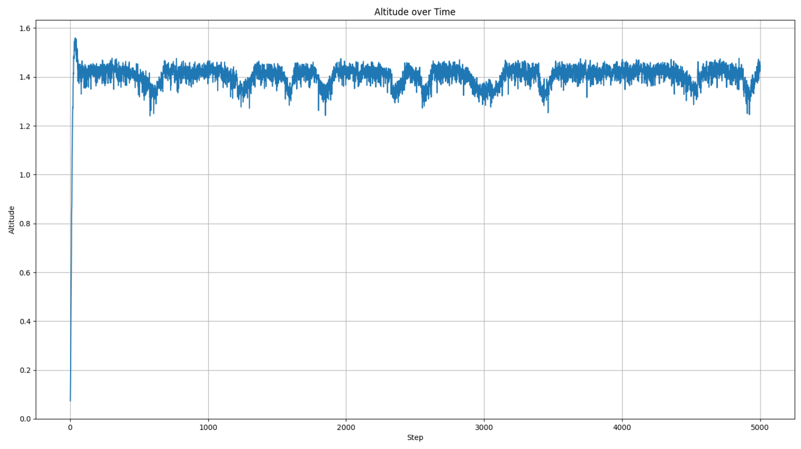


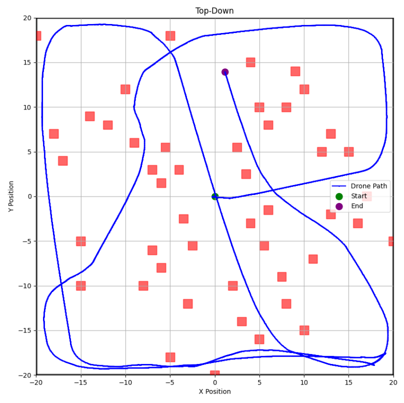
Figure 9.2: Altitude over Time

The altitude remains consistently within an acceptable range without large deviations. This indicates that the PID controller is successfully compensating for disturbances and maintaining flight at the desired height. The oscillations visible in the steady-state phase appear to be within reasonable bounds and do not seem to affect overall stability. This can be seen in Figure 9.2. Importantly, no significant oscillations or divergence are observed, indicating that the PID parameters were well-tuned to achieve a balance between responsiveness and stability. The system avoids excessive corrections that could lead to oscillatory behavior.

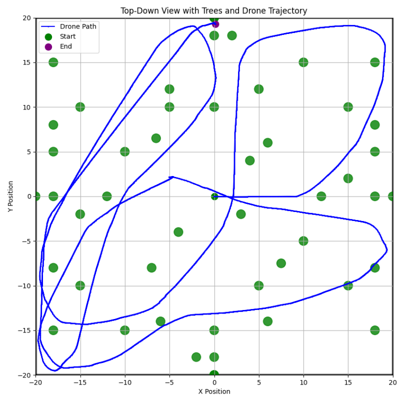
9.1.2. Obstacle Avoidance

The PID was used to ensure a collision-free navigation for the drone. The obstacle avoidance strategy relied on sensor feedback to adjust the drone's pitch and yaw, ensuring that it could maneuver around obstacles while maintaining stability.

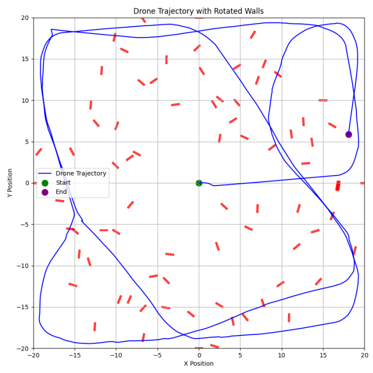
The drone was equipped with only a forward-facing sensor and a down-facing sensor, significantly limiting its perception of the environment and its ability to detect obstacles from all directions. This constraint made maintaining a safe distance from surrounding objects more challenging. Nevertheless, thanks to the implementation of Confidence-Based Decision Making, the drone was able to successfully navigate through all test environments in a remarkably smooth and safe manner. It consistently avoided collisions by making informed, cautious decisions based on the limited sensory input available. Notably, the drone adopted more conservative behavior, such as reduced speed near obstacles and delayed turning actions when visibility was low. Although this approach worked well in open or semi-structured environments, it faced minor challenges in tight spaces where lateral sensing would have provided critical context. Figure 9.3 gives the trajectory of the drone in the four environments.



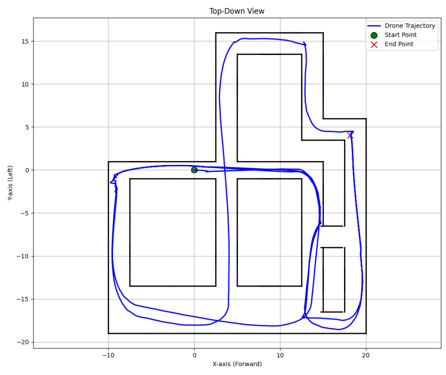
(a) Column Obstacle Environment Trajectory



(b) Trees Obstacle Environment Trajectory



(c) Walls Obstacle Environment Trajectory



(d) Maze-Like Environment Trajectory

Figure 9.3: Two-Sensors PID Control Trajectories

In the Column and Tree environments ((a) and (b)), the trajectories are smooth and continuous, with minimal abrupt turns or hesitation. This indicates that the agent was able to confidently infer safe forward motion using only frontal depth cues. In the Column environment, the agent achieved approximately 39% map coverage, while the Tree environment yielded around 42.5% coverage—suggesting moderate exploration despite the limited perceptual field. Importantly, no collisions were recorded in either scenario.

In the Walls environment ((c)), the presence of angled, elongated obstacles introduced more complex spatial constraints. The drone exhibited more frequent directional adjustments and sharper turns, especially when navigating around tight corners. These behaviors reflect the difficulty of accurately estimating obstacle edges and clearances using only forward-facing perception. Despite these challenges, the drone maintained a stable trajectory with zero collisions and achieved approximately 35.5% environment coverage, primarily exploring locally accessible regions.

The Maze-like Corridor environment ((d)) presented the most demanding setting, with narrow passages and frequent direction changes. Nonetheless, the drone successfully navigated through all corridors, closely following the maze structure from start to finish. The trajectory includes slight deviations near sharp corners, but no collisions were observed throughout the episodes. Remarkably, the agent achieved approximately 90% coverage of the environment—highlighting its ability to effectively traverse constrained spaces using only frontal sensing and reactive planning.

9.2. RL-based Control

This section presents the performance of the drone when controlled using a reinforcement learning policy. The agent, previously trained in a simplified obstacle environment, was evaluated across the four distinct environments introduced in Chapter 6. These evaluations assess how well the learned policy generalizes to novel scenarios, with particular focus on obstacle avoidance behavior, trajectory patterns, and adaptability to varied layout complexity.

The results show how well the drone, under both configurations, learned the control policy and how well it performs in terms of safe exploration, responsiveness, and adaptability across the different test conditions.

9.2.1. Five-Sensors Configuration

This configuration was trained for approximately 4 hours, during which a platoon started forming, meaning that the agent learned the most optimal policy given the reward function and the hyperparameters chosen.

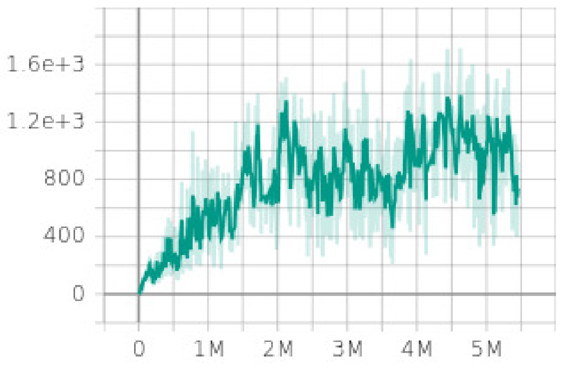


Figure 9.4: Episodic return during training

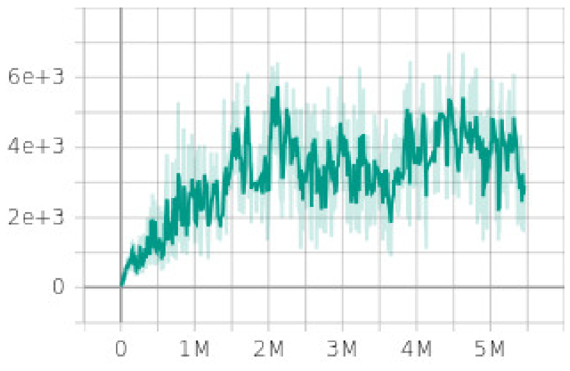


Figure 9.5: Episode length during training

The episodic return (Figure 9.4) increased rapidly during the early stages of training, indicating that the agent quickly learned behaviors that led to higher cumulative rewards. After approximately 2 million steps, the return plateaued, with the agent maintaining consistent high performance while still exhibiting variability due to exploration. This fluctuation suggests that the agent retained some stochasticity in its policy, due to entropy regularization and variations in environment dynamics.

The episodic length (Figure 9.5) also increased substantially early in training, reflecting the agent's improved ability to avoid collisions and prolong flight time. After roughly 1.5 million environment steps, the episode duration stabilized, with most episodes reaching durations between 3000 and 4000 steps. This indicates that the agent successfully learned reliable navigation strategies, while the remaining variance is expected due to ongoing exploratory behavior.

The resulted trajectories for 3 different test episodes can be seen in Figure 9.6, Figure 9.7, Figure 9.8 and Figure 9.9.

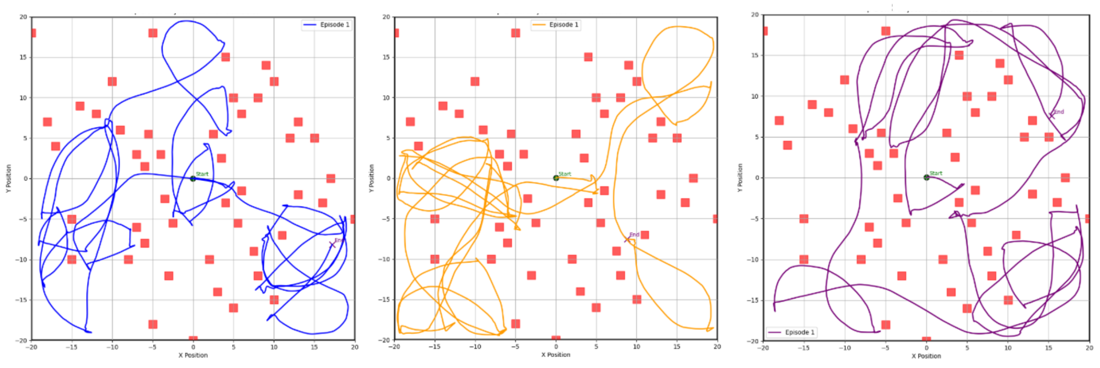


Figure 9.6: Five-Sensors Configuration - Column Obstacles Environment Trajectories

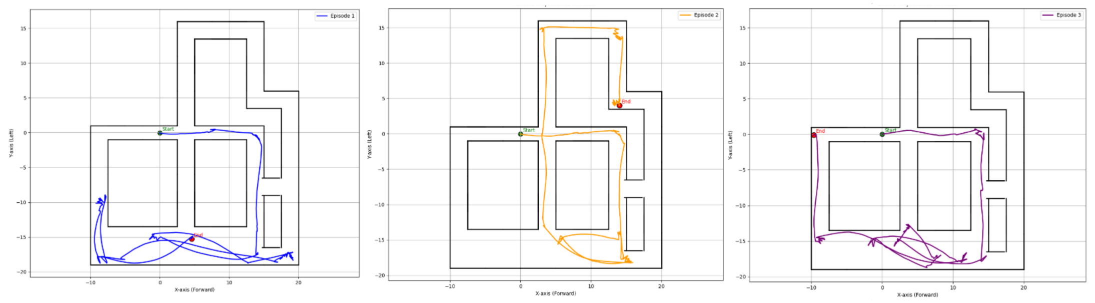


Figure 9.7: Five-Sensors Configuration - Maze-Like Environment Trajectories

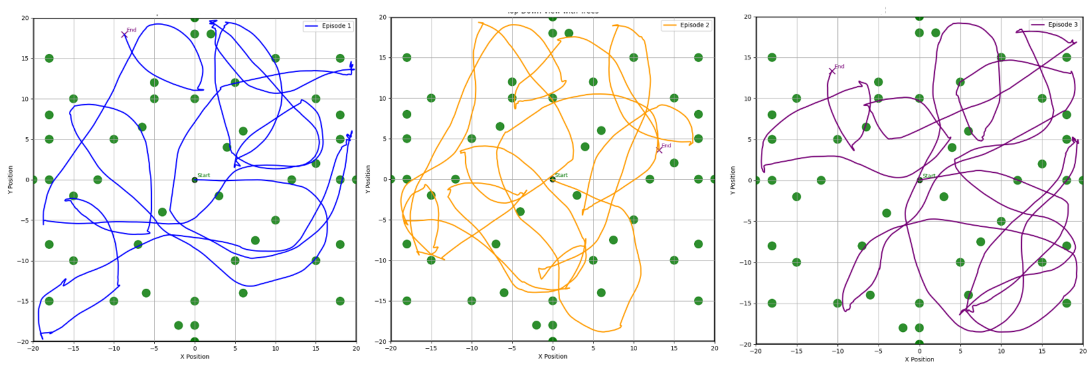


Figure 9.8: Five-Sensors Configuration - Trees Obstacle Environment Trajectories

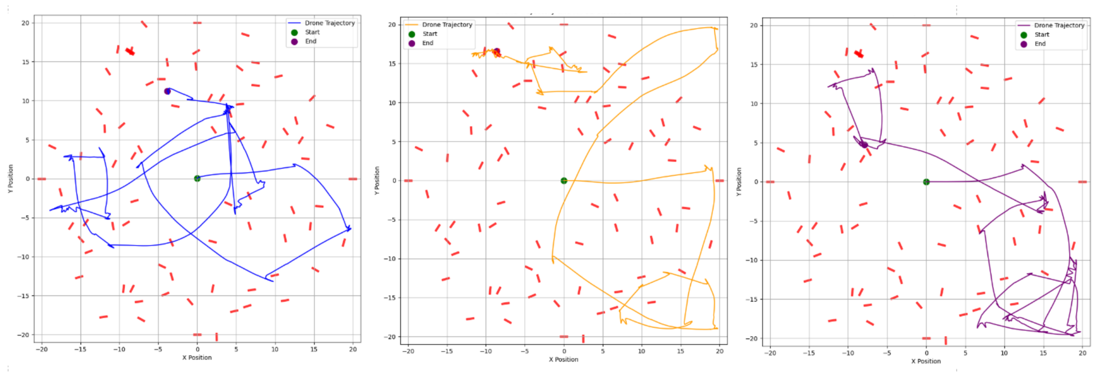


Figure 9.9: Five-Sensors Configuration - Walls Obstacle Environment Trajectories

Category	Column Obstacles	Maze-Like	Trees Obstacles	Wall Obstacles
Collisions	3.1	4.6	2.9	3.3
Coverage	57.74%	47.37%	67.66%	25.42%

Table 9.2: Five-Sensors Configuration Averages

In the Tree (Figure 9.8) and Column environments (Figure 9.6), the agent demonstrated consistent navigation. The trajectories were smooth and continuous, with few abrupt turns or signs of hesitation. This reflects an effective motion planning strategy capable of dealing with open environments. Notably, the agent explored large portions of the map and often revisited untraveled regions, resulting in the highest overall coverage—67.66% in the Trees environment and 57.74% in the Columns—paired with relatively low average collisions (3.1 and 2.9, respectively), as shown in Table 9.2.

In the Walls environment (Figure 9.9), the elongated partial barriers presented a moderate increase in local planning difficulty. The drone maintained stable behavior, but more frequent and sharper turns appeared near narrow passages, leading to a slightly higher collision rate (3.3) and the lowest coverage (25.42%). Despite this, the agent showed no signs of getting trapped or falling into oscillatory behavior, indicating short-term adaptability.

The Maze-like Corridor environment (Figure 9.7) posed the most complex challenge due to its tight turns and limited maneuvering space. The agent successfully completed navigation across all trials, avoiding deadlocks entirely. However, the average collision count was the highest among all environments (4.6), and the coverage was relatively modest (47.37%). The trajectories often favored wider corridors, with the agent reusing familiar paths. This suggests a conservative strategy focused more on safe traversal than on maximizing exploration or discovering novel paths.

Overall, these results indicate that the five-sensor configuration allows for effective obstacle avoidance across varying complexity levels. The agent adapts well to local geometry, with performance trade-offs between cautiousness (reflected in lower collision rates) and spatial exploration (reflected in higher coverage percentages).

To note, in the visualizations of the Walls and Trees environments (Figure 9.9 and Figure 9.8), the plotted trajectories occasionally appear to pass through obstacles. This is primarily a visualization artifact rather than a true failure in obstacle avoidance. Due to slight randomization in the orientation of walls and trees at each episode, the plotting system does not always precisely capture the rotated geometry of the obstacles, leading to apparent overlaps that do not reflect actual collisions during the simulation.

9.2.2. Two-Sensors Configuration

This configuration was trained for approximately 3 hours, during which the agent converged to an optimal policy, as evidenced by the emergence of stable and consistent behavior (platoon formation), given the defined reward function and selected hyperparameters.

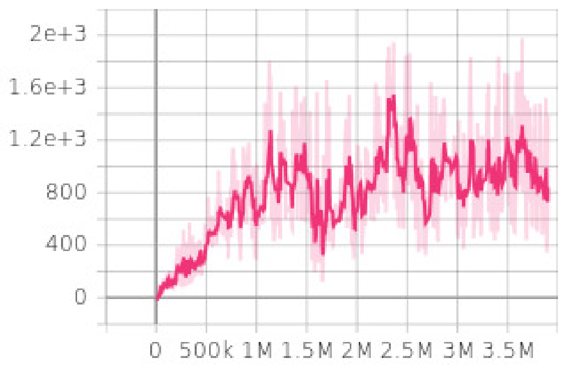


Figure 9.10: Episodic return during training

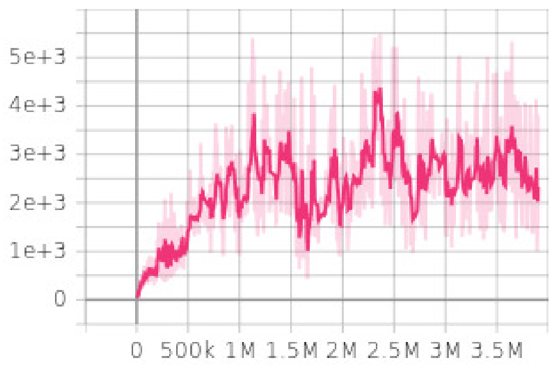


Figure 9.11: Episode length during training

The episodic return (Figure 9.10) increased steadily during the early stages of training, suggesting that the agent was able to discover reward-yielding behaviors despite the limited sensor input. After approximately 2 million environment steps, the return began to plateau, with consistent performance maintained thereafter. Although the return exhibits greater variability compared to the five-sensor case, this is expected due to the reduced environmental awareness provided by the two-sensor configuration.

The episodic length (Figure 9.11) follows a similar trend. The agent learns early on to extend flight duration, avoiding collisions with only partial perceptual coverage. Most episodes eventually stabilize between 2000 and 3000 steps, reflecting a reasonably reliable navigation strategy under sensory constraints. However, the higher variance and lower ceiling compared to the five-sensor case highlight the inherent limitations of operating with less information. These results still indicate effective learning, though with reduced consistency and control confidence.

The resulted trajectories for 3 different test episodes can be seen in Figure 9.12, Figure 9.13, Figure 9.14 and Figure 9.15.

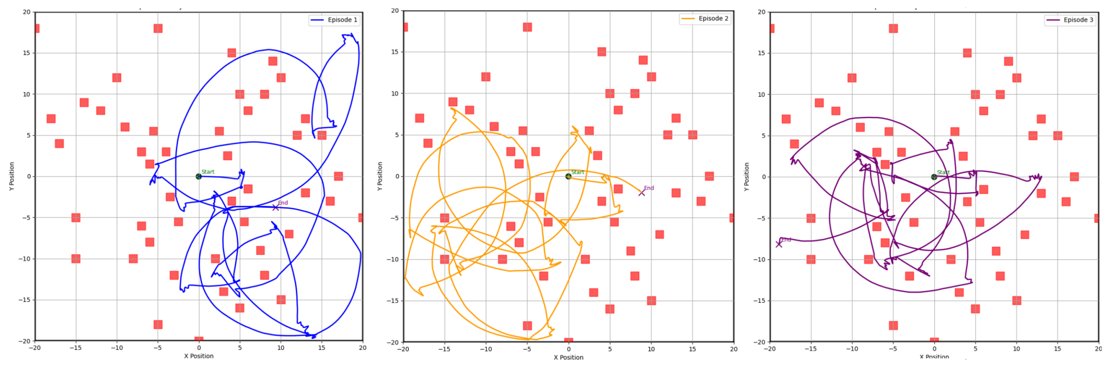


Figure 9.12: Two-Sensors Configuration - Column Obstacles Environment Trajectories

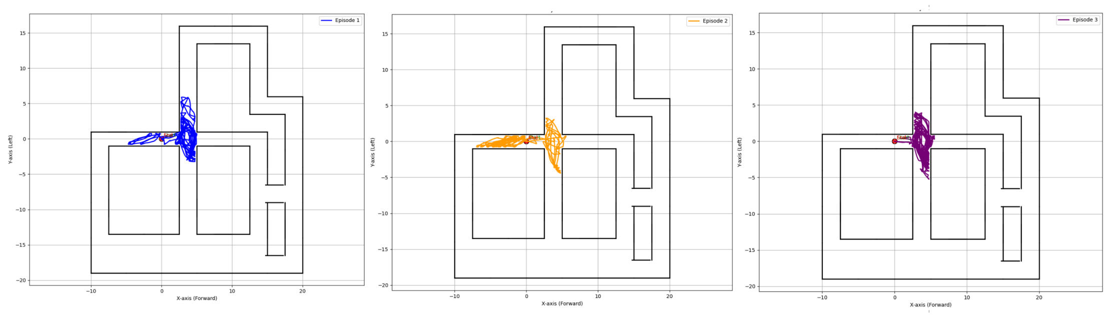


Figure 9.13: Two-Sensors Configuration - Maze-Like Environment Trajectories

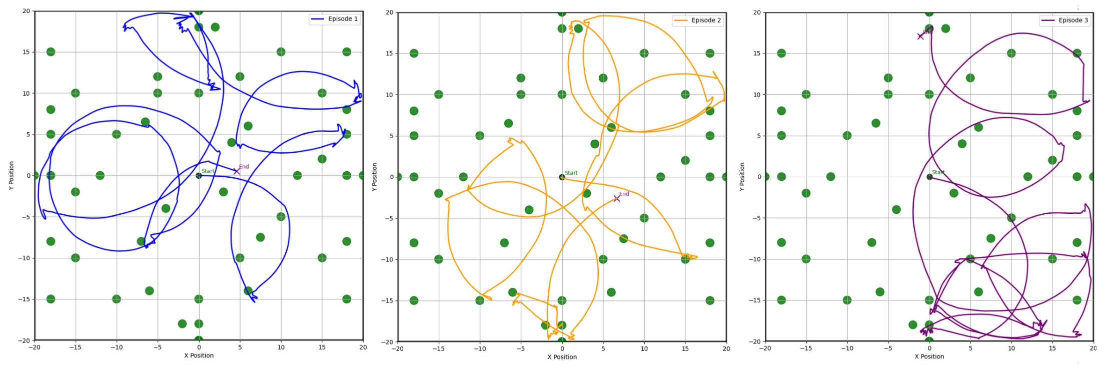


Figure 9.14: Two-Sensors Configuration - Trees Obstacle Environment Trajectories

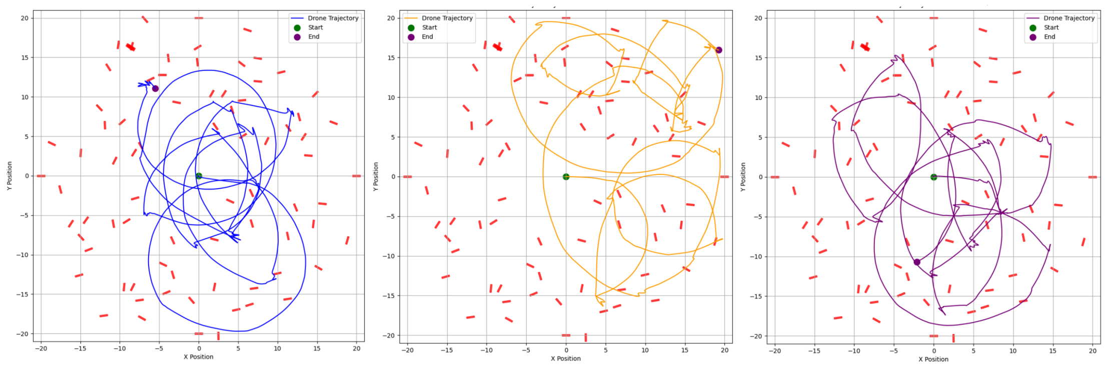


Figure 9.15: Two-Sensors Configuration - Walls Obstacle Environment Trajectories

Category	Column Obstacles	Maze-Like	Trees Obstacles	Wall Obstacles
Collisions	6.6	50+	6.4	6.8
Coverage	33.77%	6.22%	37.16%	38.58%

Table 9.3: Two-Sensors Configuration Averages

In the Column Obstacles environment (Figure 9.12), the RL policy exhibits basic yet consistent obstacle avoidance behavior. The drone successfully navigates the space without frequent collisions, maintaining looping trajectories that indicate local reactivity. However, the motion lacks global intent or directed exploration. This aligns with the reward structure, which emphasizes short-term survival and low-level stabilization (e.g., slight forward pitch and yaw reactivity), without incentivizing strategic, long-horizon behaviors. The coverage achieved in this setting was moderate at 33.77%, and the average collision count remained reasonably low at 6.6 (see Table 9.3).

A notable observation across all environments is a consistent bias toward clockwise yawing. Despite the reward function treating left and right turns symmetrically, the agent shows a strong preference for right turns. This is likely an emergent artifact of random policy initialization combined with sparse sensory input: early successful right turns may have been positively reinforced, and the agent lacked the perceptual tools to correct this drift later in training. With only forward and downward-facing sensors, the agent cannot sense lateral imbalances or plan corrective maneuvers, resulting in persistent asymmetric behavior.

The Maze-like Corridor environment (Figure 9.13) proved especially difficult. With a coverage of only 6.22% and a collision count exceeding 50, the agent failed to meaningfully enter or navigate through the maze. Instead, it remained clustered near the initial region, occasionally colliding and failing to align with corridor entries. This indicates that without lateral or backward depth perception, the agent struggles to escape tight spaces or reposition itself when trapped. The lack of long-range orientation awareness also prevents proactive adjustments, contributing to repeated failed attempts at maze traversal.

In contrast, the Trees environment (Figure 9.14) offered a more favorable setting for local reactivity. The circular, evenly spaced obstacles allowed the drone to maintain smoother loops and demonstrate modest exploration, achieving 37.16% coverage with 6.4 average collisions. While still relying on short-horizon reactivity, the agent was able to move more freely, occasionally crossing its own paths and forming large circular trajectories—again emphasizing the reactive rather than deliberative nature of the policy.

The Walls environment (Figure 9.15) features randomized, wall-like segments that present fragmented spatial constraints. In this setting, the drone exhibited relatively strong short-term adaptation, weaving between gaps and avoiding sudden obstacles. The average collision rate remained moderate at 6.8, and the agent achieved 38.58% map coverage. However, similar to other environments, the lack of broad perception prevented strategic exploration, resulting in mostly localized movement with repeated loops and limited coverage of the full space.

Overall, the two-sensor configuration demonstrates the agent's capacity for reactive avoidance and local stability, but at the cost of global spatial awareness. The reduced coverage and higher collision rates compared to the five-sensor configuration reflect the challenge of navigating with minimal perceptual input. Nonetheless, the agent was still able to learn safe and consistent behaviors in moderately structured environments.

9.3. Discussion

This section provides a comparative analysis of the two control strategies, PID-based and reinforcement learning-based, under both rich and minimal sensor configurations. By evaluating the drone's behavior across a range of environments, we highlight the trade-offs between stability, adaptability, and perception-driven control. The goal is not only to assess raw performance but to understand which method is better suited for different navigation tasks and sensing conditions.

PID Control Performance

The PID-based approach demonstrated impressive reliability across all environments. The agent maintained collision-free trajectories throughout the Columns, Trees, Walls, and even the Maze environment, relying on confidence-based thresholds to make conservative, cautious decisions. This reactive control ensured stability and safety.

Despite its limited perceptual awareness, the PID controller achieved reasonable coverage—around 39% in Columns and 42.5% in Trees. In the more fragmented and complex layouts like Walls and Maze, the controller still performed admirably, achieving 35.5% and 90% coverage, respectively. The high Maze coverage is especially notable, as it reflects the controller's ability to follow constrained paths when aligned with the corridor openings. This indicates that with well-tuned heuristics and stable control loops, even minimal sensor input can support effective navigation.

RL-Based Control Performance

In contrast, the RL-based controller showed the ability to learn navigation behaviors from raw sensory input without any explicit control logic. With the full five-sensor configuration, the learned policy exhibited strong adaptability, generating smooth trajectories in all environments. It achieved high coverage in Trees (67.66%) and Columns (57.74%), while maintaining low collision rates. Even in complex layouts like the Maze (47.37% coverage) and Walls (25.42%), the agent avoided deadlocks and exhibited consistent motion, adapting well to local geometry.

However, reducing the sensor count to two significantly impacted performance. In the Maze, the agent failed to make meaningful progress, with coverage dropping to just 6.22% and collision counts exceeding 50. In less structured settings like Trees and Walls, the RL policy maintained reactive loops but with limited spatial exploration and higher collision rates compared to the five-sensor case. The reduced observability and lack of lateral and rear perception left the agent vulnerable to local minima and repetitive patterns. A consistent clockwise yawing bias further indicated limitations in learning performance when the number of sensors is reduced.

Control Strategy Comparison

To assess the capabilities of each control method, the PID controller (two sensors) and reinforcement learning (two and five sensors) were evaluated across the four environments: Columns, Trees, Walls, and Maze. Their performance was analyzed based on collision avoidance, exploration coverage, adaptability, and computational requirements.

This comparison (Table 9.4) highlights clear trade-offs between the three approaches. The **PID controller**, while relying on a minimal sensor setup and requiring almost no computational resources, achieves **perfect collision avoidance** in all environments. It performs particularly well in structured spaces like the Maze, where it achieves 90% coverage. However, its exploration is limited in most open spaces.

The **RL policy with five sensors** demonstrates the most **balanced and capable navigation** overall. It achieves the **highest coverage** in Trees (67.66%) and Columns (57.74%) and maintains low collision counts across all scenarios. It also shows the greatest adaptability, learning to explore and generalize across complex geometries. This makes it the **best-performing solution** when both coverage and adaptability are key priorities.

On the other hand, the **RL agent with two sensors** struggles with limited environmental awareness. It

Metric	PID (2 Sensors)	RL (2 Sensors)	RL (5 Sensors)
Collision-Free Flight	All environments	High collisions number	Few collisions
Coverage (Columns)	39.0%	33.77%	57.74%
Coverage (Trees)	42.5%	37.16%	67.66%
Coverage (Walls)	35.5%	38.58%	25.42%
Coverage (Maze)	90.0%	6.22%	47.37%
Avg. Collisions (Columns)	0	6.6	3.1
Avg. Collisions (Trees)	0	6.4	2.9
Avg. Collisions (Walls)	0	6.8	3.3
Avg. Collisions (Maze)	0	>50	4.6
Adaptability	Low	Medium	High
Computational Load	Very Low	Moderate	High

Table 9.4: Comparison of Control Strategies Across Environments

suffers from high collision rates—particularly in the Maze—and shows low coverage in all environments. It is clear that this configuration is not suitable for environments requiring a large coverage or collision-free flight.

In conclusion, the **five-sensor RL controller is the most capable overall**, offering a strong balance between safety and exploration. It is best suited for tasks involving unknown environments, dynamic layouts, or when a high map coverage is desired. The **PID controller**, despite its limitations, remains the most reliable choice for low-complexity missions where collision avoidance and simplicity are the main focus.

Part III

Closure

10 Conclusion

This thesis investigated the feasibility of autonomous navigation for an attitude-stable flapping wing micro aerial vehicle (FWMAV) using a minimal sensing setup composed solely of Time-of-Flight (ToF) sensors. Operating under real-world constraints—such as limited payload, the absence of GPS or ego-motion estimation, and the vibration-heavy nature of flapping-wing flight—the focus was on achieving reliable obstacle avoidance using only sparse, local depth information. Two control strategies were explored: a classical Proportional–Integral–Derivative (PID) controller and a learning-based approach using Proximal Policy Optimization (PPO).

The PID-based controller demonstrated strong baseline performance, offering safe and predictable flight behavior across a variety of environments. Despite using only forward- and downward-facing ToF sensors, the drone successfully avoided collisions in all test scenarios. It maintained a stable altitude and employed a confidence-based decision-making mechanism to adjust its yaw conservatively near obstacles. This approach proved especially effective in structured environments such as maze-like corridors, where deliberate pauses and cautious maneuvering enabled successful navigation even without lateral sensing.

The reinforcement learning agent, particularly in the five-sensor configuration, showed superior adaptability and spatial exploration. It learned to generate smooth, forward-driven trajectories and achieved high coverage in both structured and open environments while maintaining low collision rates. When reduced to only two sensors, the RL agent retained basic avoidance capabilities but struggled in tightly constrained environments, often failing to progress beyond the starting area.

These findings highlight the viability of both classical and learning-based strategies for obstacle avoidance in resource-constrained flapping-wing platforms. The PID controller is well-suited for applications requiring high reliability, low computational demand, and predictable behavior. In contrast, RL policies offer greater flexibility and performance in environments that benefit from adaptive exploration—provided sufficient sensing is available.

11 Recommendations

This thesis demonstrated that autonomous navigation for a tailless flapping wing aerial vehicle is achievable using a minimal set of Time-of-Flight sensors and a combination of classical and learning-based control strategies. Building on this foundation, several directions are proposed for future work to extend the capabilities of such systems and support their transition from simulation to real-world deployment.

One important direction is the exploration of additional sensor configurations. This study focused on two specific layouts—a minimal two-sensor setup and a more comprehensive five-sensor configuration—but intermediate options, such as three or four sensors, remain unexplored. These configurations may offer a better balance between environmental awareness, weight, and computational cost. Moreover, alternative sensor placements, including angled or side-facing sensors, could improve perception and should be evaluated systematically.

To guide the selection of sensor locations, future work may benefit from an automated optimization strategy. An evolutionary algorithm or similar search-based method could be employed to identify the most effective sensor placement for a given task or environment. This would be particularly valuable as aerodynamic interference and physical mounting constraints are non-trivial considerations.

Another recommendation is to improve the simulation environment by incorporating the full flapping dynamics of the vehicle. While the current model includes an accurate representation of the control inputs and sensor outputs, it does not yet simulate fully the aerodynamic forces and unsteady effects generated by the wing motion. Adding a physics-based flapping model would better reflect the flight characteristics of the real drone and support more accurate training and evaluation of control policies.

In addition, future studies could explore sensor fusion techniques. By combining ToF data with other modalities such as inertial measurements or visual cues, it may be possible to achieve more consistent and accurate perception. This could enhance the drone's ability to estimate motion and detect and avoid obstacles with a higher success rate. It can also allow the drone to have a memory of the visited places, such that it will avoid flying in circles and explore more of the map.

Finally, the most important step forward is to deploy the control strategies developed in this thesis on a real drone. Testing on the Flapper Nimble+ platform will enable the validation of trained policies under real-world conditions and reveal practical challenges not captured in simulation, such as sensor latency, actuator imperfections, and environmental variability. A particularly important difference lies in the low-level control interface: while the current simulator assumes direct access to thrust and attitude commands for simplicity, the real platform, running on the Bitcraze firmware, uses a Lee Controller in which all control inputs are mapped indirectly through modulation of the wing flapping frequency. This creates a nontrivial relationship between motor signals and vehicle dynamics, one that was not modeled in the simulated environment. As a result, additional work is needed to adapt the trained policies or control signals to the physical platform's actuation model.

References

[1] Ntnu-Arl. *GitHub - ntnu-arl/aerial_gym_simulator: Aerial Gym Simulator - Isaac Gym Simulator for Aerial Robots*. URL: `https://github.com/ntnu-arl/aerial_gym_simulator`.

[2] R. Austin. *Unmanned aircraft systems: UAVs design, development and deployment*. John Wiley & Sons, 2017.

[3] V. Kumar et al. “Opportunities and challenges with autonomous micro aerial vehicles”. In: *The International Journal of Robotics Research* 31.11 (Aug. 2012), pp. 1279–1291. DOI: `10.1177/0278364912455954`.

[4] M. Hassanalian et al. “Classifications, applications, and design challenges of drones: A review”. In: *Progress in Aerospace Sciences* 91 (2017), pp. 99–131. DOI: `10.1016/j.paerosci.2017.04.003`.

[5] Y. Mulgaonkar et al. “Design of small, safe and robust quadrotor swarms”. In: *IEEE International Conference on Robotics and Automation (ICRA)* (May 2015). DOI: `10.1109/icra.2015.7139491`.

[6] H. Rajabi et al. “Experimental investigations of the functional morphology of dragonfly wings”. In: *Chinese Physics B* 22.8 (Aug. 2013), p. 088702. DOI: `10.1088/1674-1056/22/8/088702`.

[7] S. Xiao et al. “A review of research on the mechanical design of hoverable flapping wing Micro-Air vehicles”. In: *Journal of Bionic Engineering* 18.6 (Nov. 2021), pp. 1235–1254. DOI: `10.1007/s42235-021-00118-4`.

[8] H. V. Phan et al. “Insect-inspired, tailless, hover-capable flapping-wing robots: Recent progress, challenges, and future directions”. In: *Progress in Aerospace Sciences* 111 (Nov. 2019), p. 100573. DOI: `10.1016/j.paerosci.2019.100573`.

[9] T. N. Pornsin-Sirirak et al. “Microbat: A Palm-Sized Electrically Powered Ornithopter”. In: 2001.

[10] P. Zdunich et al. “Development and testing of the Mentor Flapping-Wing Micro Air vehicle”. In: *Journal of Aircraft* 44.5 (Sept. 2007), pp. 1701–1711. DOI: `10.2514/1.28463`.

[11] K. Y. Ma et al. “Controlled Flight of a Biologically Inspired, Insect-Scale Robot”. In: *Science* 340.6132 (2013), pp. 603–607. DOI: `10.1126/science.1231806`.

[12] M. Keennon et al. “Development of the Nano Hummingbird: a Tailless flapping wing micro air vehicle”. In: *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition* (Jan. 2012). DOI: `10.2514/6.2012-588`.

[13] K. M. E. De Clercq et al. “Aerodynamic Experiments on DelFly II: Unsteady lift enhancement”. In: *International Journal of Micro Air Vehicles* 1.4 (Dec. 2009), pp. 255–262. DOI: `10.1260/175682909790291465`.

[14] M. Karásek et al. “A tailless aerial robotic flapper reveals that flies use torque coupling in rapid banked turns”. In: *Science* 361.6407 (Sept. 2018), pp. 1089–1094. DOI: `10.1126/science.aat0350`.

[15] D. Coleman et al. “Design, Development and Flight-Testing of a robotic Hummingbird”. In: *AHS International Forum 71* (May 2015).

[16] H. V. Phan et al. “Mechanisms of collision recovery in flying beetles and flapping-wing robots”. In: *Science* 370.6521 (Dec. 2020), pp. 1214–1219. DOI: `10.1126/science.abd3285`.

[17] A. Roshanbin et al. “COLIBRI: A hovering flapping twin-wing robot”. In: *International Journal of Micro Air Vehicles* 9.4 (Mar. 2017), pp. 270–282. DOI: `10.1177/1756829317695563`.

[18] Q.-V. Nguyen et al. “Development and flight performance of a biologically-inspired tailless flapping-wing micro air vehicle with wing stroke plane modulation”. In: *Bioinspiration Biomimetics* 14.1 (Dec. 2018), p. 016015. DOI: `10.1088/1748-3190/aaefa0`.

[19] D. Gong et al. “Design and experiment of string-based flapping mechanism and modulized trailing edge control system for insect-like FWMAV”. In: *2018 AIAA Information Systems - AIAA Infotech @ Aerospace* (Jan. 2018). DOI: 10.2514/6.2018-0987.

[20] S. Ahmed et al. “A State-of-the-Art Analysis of Obstacle Avoidance Methods from the Perspective of an Agricultural Sprayer UAV’s Operation Scenario”. In: *Agronomy* 11.6 (May 2021), p. 1069. DOI: 10.3390/agronomy11061069.

[21] N. Aswini et al. “UAV and obstacle sensing techniques – a perspective”. In: *International Journal of Intelligent Unmanned Systems* 6.1 (Jan. 2018), pp. 32–46. DOI: 10.1108/ijius-11-2017-0013.

[22] R. Siegwart et al. “Introduction to autonomous mobile robots”. In: *Choice Reviews* 49.03 (Nov. 2011), pp. 49–1492. DOI: 10.5860/choice.49-1492.

[23] L. Wang et al. “Applications and prospects of agricultural unmanned aerial vehicle obstacle avoidance technology in China”. In: *Sensors* 19.3 (Feb. 2019), p. 642. DOI: 10.3390/s19030642.

[24] H. Zhao et al. “Scene understanding in a large dynamic environment through a laser-based sensing”. In: *2010 IEEE International Conference on Robotics and Automation* (May 2010). DOI: 10.1109/robot.2010.5509169.

[25] A. Donges et al. *Laser Measurement Technology: Fundamentals and Applications*. Oct. 2014.

[26] S. M. Nejad et al. “Low-Noise High-Accuracy TOF Laser Range Finder”. In: *American Journal of Applied Sciences* 5.7 (July 2008), pp. 755–762. DOI: 10.3844/ajassp.2008.755.762.

[27] G. Rankin et al. “Millimeter wave array for UAV imaging MIMO radar”. In: *Proceedings of the 2015 16th International Radar Symposium (IRS)* (June 2015). DOI: 10.1109/irs.2015.7226217.

[28] S. L. Johnston. *Millimeter wave radar*. Jan. 1980.

[29] H.-C. Chen. “Monocular Vision-Based obstacle detection and avoidance for a multicopter”. In: *IEEE Access* 7 (Jan. 2019), pp. 167869–167883. DOI: 10.1109/access.2019.2953954.

[30] K. McGuire et al. “Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone”. In: *IEEE Robotics & Automation Letters* 2.2 (Apr. 2017), pp. 1070–1076. DOI: 10.1109/lra.2017.2658940.

[31] A. Elfes. “Sonar-based real-world mapping and navigation”. In: *IEEE Journal of Robotics and Automation* 3.3 (June 1987), pp. 249–265. DOI: 10.1109/jra.1987.1087096.

[32] J. Steckel et al. “BatSLAM: Simultaneous localization and mapping using biomimetic sonar”. In: *PLOS ONE* 8.1 (Jan. 2013), e54076. DOI: 10.1371/journal.pone.0054076.

[33] J. Steckel et al. “Spatial sampling strategy for a 3D sonar sensor supporting BatSLAM”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Sept. 2015). DOI: 10.1109/iros.2015.7353452.

[34] N. Gageik et al. “Obstacle detection and collision avoidance for a UAV with complementary Low-Cost sensors”. In: *IEEE Access* 3 (Jan. 2015), pp. 599–609. DOI: 10.1109/access.2015.2432455.

[35] N. Gageik et al. “Obstacle Detection and Collision Avoidance Using Ultrasonic Distance Sensors for an Autonomous Quadrocopter”. In: 2012.

[36] N. Gupta et al. “Obstacle detection and collision avoidance using ultrasonic sensors for RC multirotors”. In: *Proceedings of the 2015 International Conference on Signal Processing and Communication (ICSC)* (Mar. 2015). DOI: 10.1109/icspcom.2015.7150689.

[37] L. Giubbolini. “A multistatic microwave radar sensor for short range anticollision warning”. In: *IEEE Transactions on Vehicular Technology* 49 (2000), pp. 2270–2275. DOI: 10.1109/25.901901.

[38] R. Baraniuk et al. “Compressive Radar Imaging”. In: *IEEE Radar Conference* (2007), pp. 128–133. DOI: 10.1109/RADAR.2007.374175.

[39] A. Viquerat et al. “Reactive Collision Avoidance for Unmanned Aerial Vehicles Using Doppler Radar”. In: *Field and Service Robotics: Results of the 6th International Conference*. Ed. by C. Laugier et al.

Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 245–254. DOI: 10.1007/978-3-540-75404-6_23. URL: https://doi.org/10.1007/978-3-540-75404-6_23.

[40] R. Feger et al. “A 77-GHz FMCW MIMO Radar Based on an SiGe Single-Chip Transceiver”. In: *IEEE Transactions on Microwave Theory and Techniques* 57 (2009), pp. 1020–1035. DOI: 10.1109/TMTT.2009.2017294.

[41] C. Yu et al. “Obstacle Detection Based on a Four-Layer Laser Radar”. In: *Proceedings of the 2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2007, pp. 218–221. DOI: 10.1109/ROBIO.2007.4522185.

[42] J. Demantké et al. “Dimensionality Based Scale Selection in 3D Lidar Point Clouds”. In: *ISPRS International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 38 (2012), pp. 97–102. DOI: 10.5194/isprsarchives-XXXVIII-5-W12-97-2011.

[43] L. Zheng et al. “The obstacle detection method of UAV based on 2D LiDAR”. In: *IEEE Access* 7 (Jan. 2019), pp. 163437–163448. DOI: 10.1109/access.2019.2952173.

[44] X. Zhao et al. “Detection, Tracking, and Geolocation of Moving Vehicle From UAV Using Monocular Camera”. In: *IEEE Access* 7 (2019), pp. 101160–101170.

[45] A. Zaarane et al. “Distance measurement system for autonomous vehicles using stereo camera”. In: *Array* 5 (2020), p. 100016.

[46] J. Fuentes-Pacheco et al. “Visual simultaneous localization and mapping: A survey”. In: *Artificial Intelligence Review* 43 (2015), pp. 55–81.

[47] V. J. Lumelsky et al. “Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape”. In: *Algorithmica* 2.1-4 (Nov. 1987), pp. 403–430. DOI: 10.1007/bf01840369.

[48] I. Kamon et al. “A new range-sensor based globally convergent navigation algorithm for mobile robots”. In: *Proceedings of IEEE International Conference on Robotics and Automation* (Dec. 2002). DOI: 10.1109/robot.1996.503814.

[49] O. Khatib. “Real-Time obstacle avoidance for manipulators and mobile robots”. In: *The International Journal of Robotics Research* 5.1 (Mar. 1986), pp. 90–98. DOI: 10.1177/027836498600500106.

[50] O. Cetin et al. “Establishing Obstacle and Collision Free Communication Relay for UAVs with Artificial Potential Fields”. In: *Journal of Intelligent & Robotic Systems* 69.1-4 (Aug. 2012), pp. 361–372. DOI: 10.1007/s10846-012-9761-y.

[51] Y.-B. Chen et al. “UAV path planning using artificial potential field method updated by optimal control theory”. In: *International Journal of Systems Science* 47.6 (June 2014), pp. 1407–1420. DOI: 10.1080/00207721.2014.929191.

[52] X. Fan et al. “Improved Artificial Potential Field Method applied for AUV path planning”. In: *Mathematical Problems in Engineering* 2020 (Apr. 2020), pp. 1–21. DOI: 10.1155/2020/6523158.

[53] A. Chakravarthy et al. “Obstacle avoidance in a dynamic environment: a collision cone approach”. In: *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans* 28.5 (Jan. 1998), pp. 562–574. DOI: 10.1109/3468.709600.

[54] Y. Watanabe et al. “Vision-Based obstacle avoidance for UAVs”. In: *AIAA Guidance, Navigation and Control Conference and Exhibit* (June 2007). DOI: 10.2514/6.2007-6829.

[55] J. Park et al. “Collision avoidance of hexacopter UAV based on LiDAR data in dynamic environment”. In: *Remote Sensing* 12.6 (Mar. 2020), p. 975. DOI: 10.3390/rs12060975.

[56] S. Tijmons et al. “Obstacle Avoidance Strategy using Onboard Stereo Vision on a Flapping Wing MAV”. In: *IEEE Transactions on Robotics* 33.4 (Aug. 2017), pp. 858–874. DOI: 10.1109/tro.2017.2683530.

[57] L. A. Zadeh. “Fuzzy sets”. In: *Information and Control* 8.3 (June 1965), pp. 338–353. DOI: 10.1016/s0019-9958(65)90241-x.

[58] M. Faisal et al. "Fuzzy logic navigation and obstacle avoidance by a mobile robot in an unknown dynamic environment". In: *International Journal of Advanced Robotic Systems* 10.1 (Jan. 2013), p. 37. DOI: 10.5772/54427.

[59] P. Reignier. "Fuzzy logic techniques for mobile robot obstacle avoidance". In: *Robotics and Autonomous Systems* 12.3-4 (Apr. 1994), pp. 143–153. DOI: 10.1016/0921-8890(94)90021-3.

[60] J. Borenstein et al. "The vector field histogram-fast obstacle avoidance for mobile robots". In: *IEEE Transactions on Robotics and Automation* 7.3 (1991), pp. 278–288.

[61] I. P. Sary et al. "Design of Obstacle Avoidance System on Hexacopter Using Vector Field Histogram-Plus". In: *2018 IEEE 8th International Conference on System Engineering and Technology (ICSET)* (Oct. 2018). DOI: 10.1109/icsengt.2018.8606388.

[62] S. S. Bolbhat et al. "Intelligent obstacle avoiding AGV using vector field histogram and supervisory control". In: *Journal of Physics: Conference Series* 1716.1 (Dec. 2020), p. 012030. DOI: 10.1088/1742-6596/1716/1/012030.

[63] B.-Q. Huang et al. "Reinforcement Learning Neural Network to the Problem of Autonomous Mobile Robot Obstacle Avoidance". In: *Proceedings of the 2005 International Conference on Machine Learning and Cybernetics* (Jan. 2005). DOI: 10.1109/icmlc.2005.1526924.

[64] V. Yadav et al. "Neural network approach for obstacle avoidance in 3-D environments for UAVs". In: *Proceedings of the 2006 American Control Conference* (Jan. 2006). DOI: 10.1109/acc.2006.1657288.

[65] K.-H. Chi et al. "Obstacle avoidance in mobile robot using Neural Network". In: *Proceedings of the 2011 International Conference on Consumer Electronics, Communications and Networks (CECNet)* (Apr. 2011). DOI: 10.1109/cecnet.2011.5768815.

[66] C.-J. Kim et al. "Obstacle avoidance method for wheeled mobile robots using interval type-2 fuzzy neural network". In: *IEEE Transactions on Fuzzy Systems* 23.3 (June 2015), pp. 677–687. DOI: 10.1109/tfuzz.2014.2321771.

[67] S. Back et al. "Autonomous UAV Trail Navigation with Obstacle Avoidance Using Deep Neural Networks". In: *Journal of Intelligent & Robotic Systems* 100.3-4 (Sept. 2020), pp. 1195–1211. DOI: 10.1007/s10846-020-01254-5.

[68] X. Dai et al. "Automatic obstacle avoidance of quadrotor UAV via CNN-based learning". In: *Neurocomputing* 402 (Aug. 2020), pp. 346–358. DOI: 10.1016/j.neucom.2020.04.020.

[69] Z. Liu et al. *KAN: Kolmogorov-Arnold Networks*. 2024. arXiv: 2404.19756 [cs.LG].

[70] STMicroelectronics. *VL53L0X - Time-of-Flight Distance Sensor*. 2024. URL: https://www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html.

[71] STMicroelectronics. *VL53L1X - Time-of-Flight Distance Sensor*. 2024. URL: https://www.st.com/en/imaging-and-photonics-solutions/vl53l1x.html.

[72] STMicroelectronics. *VL53L3CX - Time-of-Flight Distance Sensor*. 2024. URL: https://www.st.com/en/imaging-and-photonics-solutions/vl53l3cx.html.

[73] STMicroelectronics. *VL53L4CD - Time-of-Flight Distance Sensor*. 2024. URL: https://www.st.com/en/imaging-and-photonics-solutions/vl53l4cd.html.

[74] STMicroelectronics. *VL53L5CX - Time-of-Flight Distance Sensor*. 2024. URL: https://www.st.com/en/imaging-and-photonics-solutions/vl53l5cx.html.

[75] STMicroelectronics. *VL53L8CX - Time-of-Flight Distance Sensor*. 2024. URL: https://www.st.com/en/imaging-and-photonics-solutions/vl53l8cx.html.

[76] *Pololu - VL53L8CX Time-of-Flight 8×8-Zone Distance Sensor Carrier with Voltage Regulators, 400cm Max*. URL: https://www.pololu.com/product/3419.

[77] *VL53L8CX - STMicroelectronics*. URL: https://www.st.com/en/imaging-and-photonics-solutions/vl53l8cx.html#documentation.

[78] Taeyoung Lee et al. “Geometric tracking control of a quadrotor UAV on SE(3)”. In: *2021 60th IEEE Conference on Decision and Control (CDC)* (Dec. 2010), pp. 5420–5425. DOI: 10.1109/cdc.2010.5717652. URL: https://doi.org/10.1109/cdc.2010.5717652.

[79] Sven Pfeiffer et al. “Three-dimensional relative localization and synchronized movement with wireless ranging”. In: *Swarm Intelligence* 17.1-2 (Dec. 2022), pp. 147–172. DOI: 10.1007/s11721-022-00221-0. URL: https://doi.org/10.1007/s11721-022-00221-0.

A PPO Algorithm

This appendix presents the components of the Proximal Policy Optimization (PPO) algorithm used during training. It explains each block from Figure 8.13 and the workflow of the algorithm.

A.0.1. Neural Network

PPO uses a combination of two neural networks: the **Actor** and the **Critic**. The Actor-Critic framework ensures the optimization of control policies while maintaining stable learning. Figure 8.16 shows the architecture of the network.

Actor Network

The Actor network is responsible for selecting actions based on the current state of the environment. It represents the *policy* $\pi(a|s)$ which defines a probability distribution over possible actions given a state. Since the simulation requires continuous control, the Actor outputs continuous action values instead of discrete choices. The actions correspond to the pitch and yaw adjustments necessary for the drone's navigation.

Inputs to the Actor Network:

1. An **observation** s_t from the environment.

Outputs of the Actor Network:

1. The **mean action** μ (the most probable action to take given s_t).
2. The **log standard deviation** $\log \sigma$ (the spread of a Gaussian distribution for stochastic action sampling).

By modeling actions as a Gaussian distribution, the policy remains stochastic, enabling exploration by allowing the agent to take slightly different actions rather than repeating the same exact behavior. The action is then sampled from this distribution, ensuring that the agent explores alternative maneuvers while still favoring actions that lead to high rewards.

Critic Network

While the Actor is responsible for choosing actions, it does not inherently evaluate how good those actions are. This is the role of the Critic network, which estimates the state value function $V(s)$. The value function represents the expected total future reward starting from a given state and following the learned policy.

Inputs to the Critic Network:

1. An **observation** s_t from the environment.

Outputs of the Critic Network:

1. The **state value estimate** $V(s_t)$ (estimated return from state s_t).

The Critic is trained to minimize the error between its predicted state values and the actual observed returns from the environment. This is done using Mean Squared Error (MSE) loss, which compares the Critic's value estimate V(s) to a more accurate TD-Target (bootstrapped return):

$$L_{\text{value}} = \frac{1}{2}(V(s_t) - R_t)^2 \tag{A.1}$$

where R_t is the expected return computed from rewards collected in the episode. The Critic becomes better at predicting long-term rewards, allowing it to provide more accurate feedback to the Actor.

Interaction Between Actor and Critic
The Actor and Critic work together to optimize the agent's performance. The Critic's role is to provide an evaluation of states, while the Actor's role is to use that evaluation to select better actions. The key connection between them is the Advantage Function, which determines how much better an action was compared to the Critic's baseline estimate:

$$A_t = R_t - V(s_t) \tag{A.2}$$

A.0.2. Environment and Sample Memory

The environment and sample memory are responsible for generating experiences and storing data used for updating the policy.

Environment
The environment represents the simulation (Chapter 6) in which the agent operates. It defines the state space, action space, and reward structure. The agent interacts with the environment by taking actions, which lead to state transitions and rewards.

Inputs to the Environment:

1. **Action** a_t from the Actor Network based on current state s_t.

Outputs from the Environment:

1. The **next state** s_{t+1}.
2. The **reward** r_t measuring effectiveness of the action.
3. If episode ends, **termination signal** is sent.

Sample Memory
The sample memory stores the agent's experiences, which are later used to update the policy. The reason behind using a sample memory is because PPO performs batch updates, meaning it collects multiple steps before updating the network.

Stored Data in Sample Memory:

1. s_t: **current state**.
2. a_t: **action taken**.
3. r_t: **reward received**.
4. s_{t+1}: **next state** after taking the action.

A.0.3. Probability Ratio

The Probability Ratio is a component of PPO's policy optimization process. In short, it determines how much the **new policy** has changed compared to the **previous policy** when selecting actions. This is essential to ensure controlled updates and to prevent shifts in policy.

In PPO, policy updates are based on the idea of trust region optimization, which ensures that the new policy does not diverge too much from the old policy. The probability ratio helps regulate this by measuring the relative likelihood of taking the same action under the new and old policies.

Inputs to the Probability Ratio:

1. **Current policy probabilities** $\pi_\theta(a_t|s_t)$: The probability of taking action a_t under the new, updated policy.
2. **Old policy probabilities** $\pi_{\theta_{\text{old}}}(a_t|s_t)$: The probability of taking the same action under the old policy, which was stored in Sample Memory.

Outputs of the Probability Ratio:

1. The **computed ratio** r_t.

The ratio outputted is calculated using Equation A.3. The ratio determines how much the policy update should be adjusted.

$$r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{A.3}$$

A.0.4. Advantage Function

The Advantage Function plays the role of determining how much better or worse an action was compared to the expected state value. This information is used for updating the Actor network, as it helps the policy learn to prefer better-than-expected actions while discouraging suboptimal ones.

Inputs to the Advantage Function:

1. **Bootstrapped Return** R_t.
   - This is computed using discounted rewards and the value function: $R_t = r_t + \gamma V(s_{t+1})$
   - The return represents what actually happened in the environment.
2. **State Value Estimate** $V(s_t)$ from the Critic Network.
   - This is the Critic's estimate of how good state s_t is, but before the action was taken.

Outputs of the Advantage Function:

1. The output is the **Advantage Value** $A_t = R_t - V(s_t)$.
   - If $A_t > 0$: The action was better than expected, so the policy should be reinforced.
   - If $A_t < 0$: The action was worse than expected, so the policy should be discouraged.

A.0.5. PPO Loss Clipped

The PPO Loss Clipped Block is the core of PPO's policy update mechanism. It ensures that the policy does not change too drastically in a single update, maintaining stability while still allowing the policy to improve. This block controls how much the policy is updated based on the Advantage Function and Probability Ratio.

Inputs to the PPO Loss Clipped:

1. **Probability Ratio** r_t.
2. **Advantage** A_t.
3. **Clipping Threshold** ϵ. Limits how much r_t can deviate from 1.

Outputs of the PPO Loss Clipped:

1. The **final policy loss**: $L_{\text{PPO}} = \min\left(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon) A_t\right)$. It is used to update the Actor network.

A.0.6. Value Loss

In PPO, the Critic Network learns to approximate the state-value function $V(s)$, which predicts the expected return from a given state. The Value Loss measures how far off these predictions are from the actual observed returns.

Inputs to the Value Loss:

1. **Predicted Value** $V(s_t)$.
2. **Bootstrapped Return** R_t.

Outputs of the Value Loss:

1. The **Value Loss** $L_{\text{value}} = \frac{1}{2}\left(V(s_t) - R_t\right)^2$. This is backpropagated to update the Critic Network.

A.0.7. Workflow

At each training step, the agent collects experience by executing actions in the environment and storing the resulting state transitions in memory. Each transition consists of the state s_t, the action taken a_t, the received reward r_t, and the next state s_{t+1}. The actor network generates a probability distribution over actions, from which an action is sampled. To ensure stable updates, PPO first computes the probability ratio between the new policy and the old policy, given by

$$r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{A.4}$$

If this ratio deviates too much from 1, it indicates that the policy has changed significantly, which PPO aims to control.

To determine how much an action contributes to improving the policy, PPO computes the Advantage Function, which represents the difference between the actual observed return and the value estimate given by the critic. The advantage is computed as

$$A_t = R_t - V(s_t) \tag{A.5}$$

where R_t is the bootstrapped return, defined as

$$R_t = r_t + \gamma V(s_{t+1}) \tag{A.6}$$

where γ is the discount factor. A positive advantage means the action was better than expected, while a negative advantage means it was worse. Instead of using raw advantages, PPO often applies Generalized Advantage Estimation (GAE) to smooth variance, given by

$$A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{A.7}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the Temporal Difference (TD) error, and λ is a smoothing factor that controls the trade-off between bias and variance.

With the probability ratio and advantage function computed, PPO then applies its clipped objective function to prevent excessive updates. Instead of maximizing the standard policy gradient, PPO restricts how much the probability ratio can change with

$$L_{\text{PPO}} = \mathbb{E}_t \left[\min\left(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon) A_t\right)\right] \tag{A.8}$$

where ϵ is a clipping threshold (typically 0.2). If the probability ratio r_t exceeds this range, the update is constrained, preventing the policy from shifting too drastically. This clipping ensures that PPO maintains stable policy learning, avoiding issues seen in older policy gradient methods that allowed for unbounded policy updates.

Alongside optimizing the policy, PPO trains the critic network to improve its estimation of state values. The critic is trained using a value loss function, which minimizes the error between the predicted state value and the actual return:

$$L_{\text{value}} = \frac{1}{2}(V(s_t) - R_t)^2 \tag{A.9}$$

To further enhance training stability, PPO can use clipped value loss, which prevents large changes in value function predictions:

$$L_{\text{value, clipped}} = \frac{1}{2}\max((V(s_t) - R_t)^2, (V_{\text{clipped}}(s_t) - R_t)^2) \tag{A.10}$$

where $V_{\text{clipped}}(s_t)$ is a slightly modified version of $V(s_t)$, ensuring that updates do not drastically change the value estimates.

To prevent the policy from collapsing into a deterministic strategy, PPO incorporates an entropy regularization term, which encourages exploration by maximizing the entropy of the policy distribution. The entropy is given by

$$H(\pi_\theta) = -\sum_a \pi_\theta(a|s) \text{log } \pi_\theta(a|s) \tag{A.11}$$

and is included in the final PPO objective to promote diverse action selection.

Combining these components, the final PPO loss function consists of three terms: the policy loss (clipped PPO loss), the value loss, and the entropy bonus:

$$L_{\text{total}} = L_{\text{PPO}} + c_1 L_{\text{value}} - c_2 H(\pi_\theta) \tag{A.12}$$

where c_1 and c_2 control the weight of the value function loss and entropy regularization, respectively.

At each training step, PPO computes the gradients of this total loss and updates the Actor and Critic networks using gradient-based optimization. The policy gradient update follows:

$$\theta \leftarrow \theta + \alpha \nabla_\theta L_{\text{PPO}} \tag{A.13}$$

where α is the learning rate, ensuring that the policy improves incrementally. Similarly, the Critic network is updated by minimizing the value loss:

$$\phi \leftarrow \phi - \alpha_v \nabla_\phi L_{\text{value}} \tag{A.14}$$

where ϕ represents the Critic parameters and α_v is the critic learning rate.