Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Ørjan Carlsen

Merging Classical Control and Deep Reinforcement Learning for Dynamic Collision Avoidance for a Quadcopter

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
Co-supervisor: Thomas Nakken Larsen
June 2023

NTNU
Norwegian University of
Science and Technology

Ørjan Carlsen

Merging Classical Control and Deep Reinforcement Learning for Dynamic Collision Avoidance for a Quadcopter

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
Co-supervisor: Thomas Nakken Larsen
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Merging Classical Control and Deep Reinforcement Learning for Dynamic Collision Avoidance for a Quadcopter

Ørjan Carlsen

Master's Thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
Co-Supervisor: Thomas Nakken Larsen
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Abstract

Autonomous Unmanned Aerial Vehicles (UAVs), such as quadcopters, can improve the efficiency of deliveries, perform inspections of assets that are risky for humans, or even perform surveillance during rescue operations. A minimum requirement for achieving autonomy is the ability to follow an a priori known path while avoiding collision with unforeseen obstacles. Thus, the current work aims to solve the dual objective of path following and collision avoidance for a quadcopter by combining low-level and high-level control.

This thesis proposes a path following controller for a quadcopter that integrates Reinforcement Learning (RL) for collision avoidance to effectively exploit the strengths of classical and data-driven control design approaches. Using Deep Reinforcement Learning (DRL), an agent was trained to perform local navigation around obstacles using a Convolutional Neural Network (CNN)-based perception of LiDAR measurements. A geometrical path following controller was derived and implemented as a control abstraction for the DRL agent. The Proximal Policy Optimization (PPO) algorithm was applied to train the DRL agent in synthetic and stochastically generated environments. Ultimately, it was tested in scenarios of increasing complexity and in scenarios never encountered before.

The results obtained through software simulations show great potential for using classical control as an abstraction in complex control tasks. By rewarding the agent in the separate, competing problem domains of path following and collision avoidance, the trained agent has to a certain extent learned to follow the path in the absence of obstacles and perform evasive maneuvers when required. The performance has also generalized to previously unseen scenarios, where the agent acts intelligently to overcome the obstacles. However, there is still potential for improvement in scenarios with a high density of obstacles, and future research directions to improve the performance are suggested.

Sammendrag

Selvstyrte ubemannede luftfartøy (UAVs, engelsk: Unmanned Aerial Vehicles), som eksempelvis kvadrokoptere, kan forbedre effektiviteten ved leveranser, utføre risikofylte inspeksjoner av strukturelle eiendeler, eller til og med drive overvåking under redningsaksjoner. Et minimumskrav for å oppnå autonomi er evnen til å følge en a priori kjent bane og samtidig unngå kollisjon med uforutsette hindringer. Denne studien har derfor som mål å mestre baneføIging og kollisjonsunngåelse for et kvadrokopter ved å kombinere lavnivå- og høynivåkontroll.

Denne masteroppgaven presenterer en lavnivåregulator for et kvadrokopter som integrerer høy-nivå forsterkningslæring (RL, engelsk: Reinforcement Learning) for kollisjonsunngåelse for å effektivt utnytte styrkene til klassisk og datadrevet kontrolldesign. Ved bruk av dyp forsterkningslæring (DRL, engelsk: Deep Reinforcement Learning) ble en agent trent til å navigere rundt hindringer ved hjelp av konvolusjonelt nevralt nettverk (CNN, engelsk: Convolutional Neural Network)-basert sansing av LiDAR-målinger. En geometrisk baneføIgingsregulator ble utledet og implementert som en kontrollabstraksjon for DRL-agenten. Proximal Policy Optimization (PPO, engelsk: Proximal Policy Optimization)-algoritmen ble brukt til å trene DRL-agenten i syntetiske og stokastisk genererte miljøer. Til slutt ble den testet i scenarioer med økende vanskelighetsgrad og i scenarioer som aldri hadde blitt møtt tidligere.

Resultatene, som ble oppnådd gjennom simuleringer, viser stort potensial for bruk av klassisk reguleringsteknikk som en abstraksjon i komplekse kontrolloppgaver. Ved å belønne agenten i de separate, konkurrerende problemområdene baneføIging og kollisjonsunngåelse, har den trente agenten til en viss grad lært seg å følge banen i fravær av hindringer og utføre manøvrer for kollisjonsunngåelse når det er nødvendig. Prestasjonen har også generalisert til tidligere usette scenarier, der agenten handler intelligent for å overkomme hindringene. Det er imidlertid fortsatt potensiale for forbedring i scenarioer med mange hindringer, og fremtidige forskningsretninger for å forbedre agentens prestasjoner er foreslått.

Preface

This thesis concludes my Master's degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU) and is written with Professor Adil Rasheed as supervisor and Ph.D. Candidate Thomas Nakken Larsen as co-supervisor.

I want to express my gratitude towards Adil for his guidance, specifically the valuable feedback he provided for my writing. I am particularly grateful to Thomas for his immense help throughout the entire process. He aided me in selecting the thesis topic, debugging the code, and brainstorming implementation ideas. This thesis would not have been possible without the support from both of you.

The code implementations for this thesis build upon the DRL framework developed by Sundøen in [71]. Even though my Master's thesis is not a continuation of my specialization report [10], the theory related to reinforcement learning in Section 2.5 is built directly on the theory in the specialization report. Where applicable, the connection to the specialization project is cited. Otherwise, the thesis is mainly original work, and the author created all figures and illustrations.

Trondheim, June 2023
Ørjan Carlsen

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

Abbreviations

A2C	Advantage Actor-Critic
APE	Absolute Path Error
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DOF	Degree Of Freedom
DRL	Deep Reinforcement Learning
LQR	Linear Quadratic Regulator
MPC	Model Predictive Control
MDP	Markov Decision Process
NED	North-East-Down
PD	Proportional-Derivative
PID	Proportional–Integral–Derivative
PPO	Proximal Policy Optimization
QPMI	Quadratic Polynomial Interpolation
RL	Reinforcement Learning
SB3	Stable Baselines3
SNAME	Society of Naval Architects and Marine Engineers
TRPO	Trust Region Policy Optimization
UAV	Unmanned Aerial Vehicle
VTP	Virtual Target Point

Chapter 1

Introduction

Autonomous Unmanned Aerial Vehicles (UAVs), especially multi-rotors such as quadcopters, have the potential to improve the efficiency of deliveries, perform inspections of assets that are dangerous for humans, or even surveillance during rescue operations. Achieving this is a complex challenge, which requires the ability to follow a predefined path and make adjustments to avoid unforeseen obstacles. By implementing a path following controller and using Deep Reinforcement Learning (DRL) for collision avoidance, the current work aims at contributing to safe and effective quadcopter navigation.

This chapter provides a short motivation for the thesis before introducing an overview of previous research on path following and collision avoidance for quadcopters. Then, the research objectives for the study are defined, and the report outline is presented.

1.1 Motivation

Multi-rotors have numerous advantages over fixed-wing aircraft, with the ability for vertical take-off and landing, eliminating the necessity for runways and launchers. They can also hover over the ground, which is advantageous in urban or indoor environments with several obstacles like buildings, people, and furniture, or tight, constrained corridors where the vehicle must maneuver cautiously. Thus, home delivery in retail [27] or delivery of medical supplies [19] could be suitable multi-rotor applications. They can also inspect assets that prove difficult or dangerous for humans, like tanks, flair stacks, chimneys, and wind turbines [16].

In the case of quadcopters, developing a path following controller is a critical task to enable autonomous navigation in complex environments. However, ensuring safe operation is also a significant challenge due to the risk of collisions with obstacles or other vehicles. Therefore, integrating a collision avoidance system is crucial for achieving safe and reliable quadcopter navigation. Traditional approaches to collision avoidance rely on predefined rules and heuristics, which may not be adaptable to dynamic and unpredictable scenarios. In contrast, Reinforcement Learning (RL) is a promising approach that allows the quadcopter to learn from experience and improve its performance over

time. Thus, this thesis proposes a novel path following controller for a quadcopter that integrates RL for collision avoidance and simulates a LiDAR for obstacle detection. The results of this work can potentially contribute to the development of safe and effective quadcopter navigation systems.

1.2 Background

This section provides a short overview of the research done within the fields of path following and collision avoidance for quadcopters.

1.2.1 Path Following

Path following is the problem of following a predefined path without any time dependencies, as opposed to trajectory tracking, where a timed reference position is tracked [62]. As quadcopter systems are unstable, underactuated, and nonlinear, path following can be a challenging task [52]. There have been developed numeral algorithms to perform path following for quadcopters, and a few of the most common are presented in this section. A more comprehensive presentation of path following algorithms for quadcopters is provided by the authors of [61].

Several geometric guidance laws are developed for quadcopters. The missile guidance and control literature initially described these guidance laws, often steering the vehicle toward a Virtual Target Point (VTP) on the path. *Carrot-chasing* [49] is an algorithm that tries to steer the quadcopter to a VTP on the path, which is a fixed distance ahead of the vehicle's closest point of the path. Quite similarly, *line-of-sight* [2] seeks to steer the vehicle directly towards the closest point on the path with a set lookahead distance from the quadcopter. Both *pure pursuit* [3] and *trajectory shaping* [58] utilize a VTP moving with a constant velocity along the path. Whereas pure pursuit only tries to close down the VTP, the trajectory shaping algorithm takes the heading of the VTP into account, giving better path following abilities than the pure pursuit algorithm [45]. All the algorithms mentioned above calculate a commanded acceleration to guide the vehicle toward the VTP. In addition, the property of *differential flatness* [20, 46] can be used to design control laws, computing feedforward control terms from a reference path.

Optimal control theory can be used to minimize the error from the path and the control effort used to achieve this. This has been done using a *Linear Quadratic Regulator* (LQR) [39, 53, 74] and state feedback controller. The objective function to be minimized is on the form

$$J = \frac{1}{2}\sum_{t=0}^{\infty} \boldsymbol{x}^\top(t)\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}^\top(t)\boldsymbol{R}\boldsymbol{u}(t), \tag{1.1}$$

where the feedback control law is given by

$$\boldsymbol{u}(t) = -\boldsymbol{K}\boldsymbol{x}(t). \tag{1.2}$$

The optimal feedback matrix $\boldsymbol{K}$ can be computed by solving the Riccati equation. As the LQR name suggests, the state space representation of the quadcopter dynamics should be linearized. In addition to the LQR, there has been done research on the use of the optimization algorithm *nonlinear Model Predictive Control* (MPC) [51] for path

following. MPC does also minimize an objective function and allows to constrain the states and inputs. For each time step, the optimal control sequence is calculated for a fixed number of time steps forward in time, while only the first is applied. This typically leads to a high computational cost, but the energy usage for the quadcopter can be reduced as the control sequence is calculated for several time steps.

For nonlinear systems, *backstepping* [60] is a widely used control method, and in the case of quadcopters, it is traditionally used for trajectory tracking. However, global tracking cannot be assured because of the underactuated nature of quadcopters [57]. By eliminating the time dependencies of the tracking problem, i.e., turning the task into path following, it is possible to obtain a globally convergent backstepping controller [9]. In backstepping, the objective is to make predefined errors converge to zero. A Lyapunov function is defined for each error, whereas the system is stable and the errors converge to zero when their time derivatives are negative definite. Thus, the control input to the system ensures that all the Lyapunov functions are negative definite.

Feedback linearization [8] is a technique to linearize the system dynamics in a particular region, enabling application of linear control theory. The method has been applied on a path following problem, showing that the quadcopter converges to the path [60]. The linearization makes the control structure simple, yielding an advantage compared to, e.g., the optimization methods.

All the aforementioned methods are based on classical control theory, requiring knowledge of the model dynamics. In recent years, significant improvements have been made within the field of *learning-based methods*. The authors of [67] learned path following and collision avoidance for a quadcopter by using MPC as a supervisor to train a neural network. The drawback of this approach is that there is already a developed controller. The authors of [78] developed an iterative learning controller that was trained on a path following problem for a quadcopter at constant altitude. The policy was learned through repetitive flights to learn from experience. RL has also been used for waypoint tracking for a quadcopter [32]. This is a simplified problem compared to tracking a continuous path. The results were quite good, but the method was conservative and more stable than similar methods. In Sundøen's Master's thesis [71], RL was used to learn both path following and collision avoidance for a quadcopter. The thrust inputs were learned to follow a 3D path, with a success rate of 74%. The framework used by Sundøen is further developed in this thesis.

1.2.2 Collision Avoidance

In this thesis, collision avoidance is defined as the problem of avoiding crashing into obstacles while following a time-independent, predefined path. The obstacles are discovered locally, meaning that collision avoidance can be considered local path planning around the obstacles. In this section, some methods for collision avoidance are presented, while an in-depth survey of collision avoidance methods for UAVs is created by the authors of [76].

Geometric methods for collision avoidance try to keep a minimum distance between the UAVs and the obstacles based on their geometric attributes and velocities. One typical geometric method is called the *collision cone approach*, which is based on defining a cone between the vehicle and obstacle, and guiding the vehicle away from the cone. The authors of [12] presented this approach in a 2D case before they extended it to a 3D case

for aircraft [24]. Another geometric method is the *fast geometric avoidance algorithm* [26], where the probability of colliding with the obstacles' safety boundaries is calculated based on kinematics, and a tangential line for the UAV to follow is calculated.

Vector field methods [15, 34] generate a repulsive field around the obstacles. This could be combined with a vector field around the path, solving both the path following and collision avoidance problem. By letting the quadcopter follow the vectors, its position should converge to the path and be repulsed by the obstacles. These methods require known positions, velocities, and shapes of the obstacles. In the case of multiple collaborating vehicles, a repulsive field could be generated around each vehicle, like the authors of [70] did.

As with path following, optimization methods can be used for collision avoidance, calculating an avoidance trajectory from geographical information. These algorithms often have a high computational cost. Thus, there are developed a variety of methods, including the use of *ant colony optimization* [55], *Bayesian policy optimization* [4] and *cooperative predictive control* [7]. MPC has also been utilized for collision avoidance, where a 2D bicycle model achieved excellent results in simulations [1].

Learning-based methods have been applied to achieve collision avoidance in recent years. In this case, the controller must weigh the importance of following the path and reducing the risk of hitting obstacles. The authors of [48] used Proximal Policy Optimization (PPO) to achieve path following and collision avoidance for an autonomous surface vehicle, showing that RL is an applicable method. This work was later extended to a 3D case for an autonomous underwater vehicle by the authors of [28], achieving excellent results both with regard to path following and collision avoidance. When this work was extended to a quadcopter by Sundøen in [71], collision avoidance cannot be claimed to have been solved completely.

1.3 Contribution, Research Objectives and Research Questions

1.3.1 Contribution

The present work's main contribution is a framework for solving the dual objective of path following and collision avoidance for a quadcopter in synthetic environments. The proposed solution consists of a control law for path following and a DRL agent for local navigation around obstacles, using a simulated LiDAR for obstacle detection and a Convolutional Neural Network (CNN) for dimensionality reduction.

Although various solutions to solve the dual objective of path following and collision avoidance for a quadcopter have been proposed in previous research, these have not combined low-level classical control with high-level RL to the extent of the author's knowledge. This work aims to effectively exploit the strengths of low-level and high-level control design approaches by establishing a suitable abstraction between them. Classical control has proven to handle the path following problem effectively. By utilizing the properties of RL to generalize to dynamic and unpredictable scenarios, this thesis seeks to develop a navigation system with strong path following abilities that perform in environments without relying on predefined rules and heuristics.

1.3.2 Research Objectives

In order to guide this study, a set of research objectives were established. Furthermore, research questions that will help achieve these objectives have been articulated.

Primary Objective: Obtain a fitting abstraction between low-level classical control and high-level RL to solve the dual objective of path following and collision avoidance for a quadcopter.

Secondary Objectives:

- Implement a controller for following a time-independent, continuous, and stochastically generated trajectory in three dimensions.
- Train a DRL agent to achieve intelligent decision-making regarding avoidance maneuvering, simulating a LiDAR for obstacle detection.

1.3.3 Research Questions

The guiding questions governing the research can be stated as:

- Can the path following controller be used as a control abstraction for the DRL agent and still achieve path following?
- Can the DRL agent learn to perform evasive maneuvers for collision avoidance when using a CNN-based perception of LiDAR measurements?
- Has the DRL agent generalized into acting intelligently in previously unseen scenarios?

1.4 Structure of the Thesis

The thesis is divided into five chapters. The introduction in Chapter 1 briefly motivates the thesis, summarizes the background research, and specifies the contributions of this thesis. Chapter 2 introduces the relevant theory for understanding the thesis. The quadcopter model is derived, and its associated assumptions are outlined. The theory required to derive the path following controller and the DRL agent for collision avoidance is also explained. Chapter 3 presents the research methodology and justifies design choices such that the study is reproducible. The obtained results are presented and discussed in Chapter 4. Finally, the thesis' conclusion is presented in Chapter 5, which also outlines possible directions for future work.

Chapter 2

Theory

This chapter provides the relevant theory for understanding the thesis. Firstly, the quadcopter model is derived, and its associated assumptions are outlined. Then, the basic concepts of feedback control are explained, which is essential for understanding the path following controller. A method for generating G^2-continuous paths is presented before the fundamentals of CNNs are introduced. Finally, the theoretical foundation of RL is described.

2.1 Quadcopter Model

In this section, the dynamics of the quadcopter model are derived. The resulting model is illustrated in Figure 2.1. This model has some deviations from the model derived by Sundøen in [71], in terms of a more detailed modeling of the quadcopter kinetics and the chosen reference frames.

2.1.1 Reference Frames

The position of the quadcopter must always be given relative to a reference frame. This thesis defines a world and a body frame with an orthonormal basis, both illustrated in Figure 2.1. The world frame is denoted $\{w\} = (\boldsymbol{x}_w, \boldsymbol{y}_w, \boldsymbol{z}_w)$, where $\boldsymbol{x}_w$ and $\boldsymbol{y}_w$ forms a tangent plane relative to the earth's reference ellipsoid, and $\boldsymbol{z}_w$ points upwards normal to the surface. A North-East-Down (NED) reference frame is often used for local navigation [21] and could have been used in this thesis, but it was disregarded as the chosen world frame appears to be the standard in most quadcopter models [20, 33, 35, 44, 75]. Other reference frames should have been considered in the case of terrestrial navigation. The body reference frame is fixed to the quadcopter and denoted $\{b\} = (\boldsymbol{x}_b, \boldsymbol{y}_b, \boldsymbol{z}_b)$, where $\boldsymbol{x}_b$ is the transversal axis, $\boldsymbol{y}_b$ is the longitudinal axis and $\boldsymbol{z}_b$ is the upwards normal axis. The origin of the body frame was chosen at the center of mass to simplify calculations.

2.1.2 State Variables

There are used 12 variables, presented in Table 2.1, to represent the quadcopter with motion in six Degrees Of Freedom (DOFs). The generalized position vector is defined

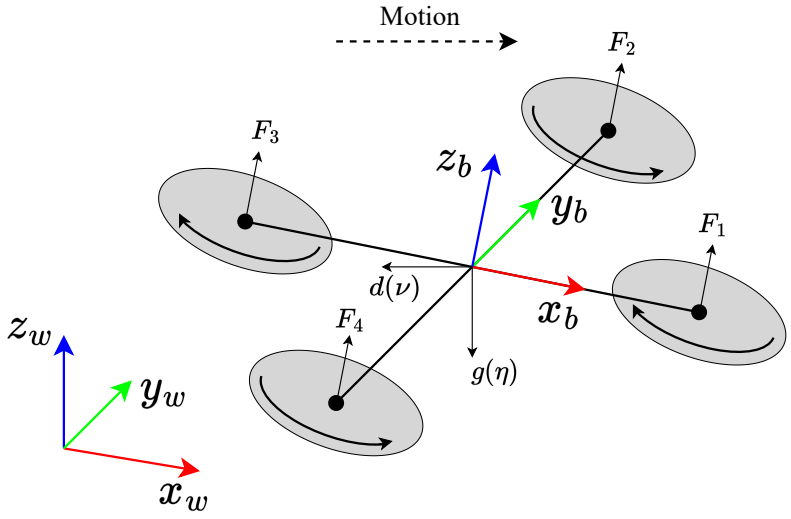


Figure 2.1: The quadcopter is illustrated with both world and body-fixed coordinate frames. Of the four rotors, two rotate clockwise, and the other two rotate counter-clockwise, each producing a force. The gravitational force acts through the center of mass in the $-\boldsymbol{z}_w$ direction. The drag force acts in the opposite direction of the quadcopter's velocity.

as $\boldsymbol{\eta} = [\boldsymbol{p}^w_{wb}, \boldsymbol{\Theta}_{wb}]^\top$, where $\boldsymbol{p}^w_{wb} = [x^w, y^w, z^w]^\top$ is the quadcopter's position in the world frame, while $\boldsymbol{\Theta}_{wb} = [\phi, \theta, \psi]^\top$ describes the rotation of the body frame relative to the world frame. The velocity vector is defined as $\boldsymbol{\nu} = [\boldsymbol{v}^b_{wb}, \boldsymbol{\omega}^b_{wb}]^\top$, where $\boldsymbol{v}^b_{wb} = [u, v, w]^\top$ are the linear velocities along the axes of the body frame, while $\boldsymbol{\omega}^b_{wb} = [p, q, r]^\top$ are the angular rates of the body frame's rotation relative to the world frame.

Table 2.1: State variables for the quadcopter following Society of Naval Architects and Marine Engineers (SNAME) notation [66], except that the presented world frame is used instead of NED.

DOF			Body Velocities and Angular Rates	World Position and Euler Angles
1	surge	- motion in $\boldsymbol{x}_b$-direction	u	x^w
2	sway	- motion in $\boldsymbol{y}_b$-direction	v	y^w
3	heave	- motion in $\boldsymbol{z}_b$-direction	w	z^w
4	roll	- rotation about the $\boldsymbol{x}_b$-axis	p	ϕ
5	pitch	- rotation about the $\boldsymbol{y}_b$-axis	q	θ
6	yaw	- rotation about the $\boldsymbol{z}_b$-axis	r	ψ

2.1.3 Kinematics

Kinematics describes the geometrical aspects of motion without considering the underlying forces. In the context of a quadcopter, kinematics is understanding the relationship between the generalized position of the quadcopter and its velocity vector. As the velocity vector is given in body frame, it must be transformed into the world frame for comparison with the generalized coordinates. The derivations in this section follow the

same methodology as the authors of [21]. In the following equations, c(·), s(·) and t(·) denote cos(·), sin(·) and tan(·), respectively.

In general, a rotation β about the $\boldsymbol{\zeta}$ axis is given by

$$\boldsymbol{R}_{\zeta,\beta} = \boldsymbol{I}_3 + \mathrm{s}(\beta)\boldsymbol{S}(\boldsymbol{\zeta}) + (1 - \mathrm{c}(\beta))\boldsymbol{S}^2(\boldsymbol{\zeta}), \tag{2.1}$$

where $\boldsymbol{S}(\boldsymbol{\zeta})$ denotes the skew-symmetric matrix of $\boldsymbol{\zeta}$. The three rotation matrices $\boldsymbol{R}_{x,\phi}$, $\boldsymbol{R}_{y,\theta}$ and $\boldsymbol{R}_{z,\psi}$ can then be obtained by substituting $\boldsymbol{\zeta}$ with the three unit vectors and β with ϕ, θ and ψ, respectively. Subsequently, the rotation from body frame to world frame can be found as $\boldsymbol{R}(\boldsymbol{\Theta}_{wb}) = \boldsymbol{R}_{z,\psi}\boldsymbol{R}_{y,\theta}\boldsymbol{R}_{x,\phi}$, giving

$$\boldsymbol{R}(\boldsymbol{\Theta}_{wb}) = \begin{bmatrix} \mathrm{c}(\theta)\mathrm{c}(\psi) & \mathrm{s}(\phi)\mathrm{s}(\theta)\mathrm{c}(\psi) - \mathrm{c}(\phi)\mathrm{s}(\psi) & \mathrm{c}(\phi)\mathrm{s}(\theta)\mathrm{c}(\psi) + \mathrm{s}(\phi)\mathrm{s}(\psi) \\ \mathrm{c}(\theta)\mathrm{s}(\psi) & \mathrm{s}(\phi)\mathrm{s}(\theta)\mathrm{s}(\psi) + \mathrm{c}(\phi)\mathrm{c}(\psi) & \mathrm{c}(\phi)\mathrm{s}(\theta)\mathrm{s}(\psi) - \mathrm{s}(\phi)\mathrm{c}(\psi) \\ -\mathrm{s}(\theta) & \mathrm{s}(\phi)\mathrm{c}(\theta) & \mathrm{c}(\phi)\mathrm{c}(\theta) \end{bmatrix}. \tag{2.2}$$

Equation (2.2) can be used to express the relation between the quadcopter's position and its linear velocities, such that

$$\dot{\boldsymbol{p}}^{w}_{wb} = \boldsymbol{R}(\boldsymbol{\Theta}_{wb})\boldsymbol{v}^{b}_{wb}. \tag{2.3}$$

As rotation matrices satisfy $\boldsymbol{R} \in \mathrm{SO}(3)$, the rotation from world frame to body frame is found as

$$\boldsymbol{R}(\boldsymbol{\Theta}_{bw}) = \boldsymbol{R}(\boldsymbol{\Theta}_{wb})^{\top}. \tag{2.4}$$

The angular velocities do also have to be transformed from body frame to world frame, as $\boldsymbol{\omega}^{b}_{wb}$ cannot be integrated directly to obtain $\boldsymbol{\Theta}_{wb}$. The inverse of the transformation matrix can be derived like

$$\boldsymbol{\omega}^{b}_{wb} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \boldsymbol{R}^{\top}_{x,\phi} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \boldsymbol{R}^{\top}_{x,\phi}\boldsymbol{R}^{\top}_{y,\theta} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \coloneqq \boldsymbol{T}^{-1}(\boldsymbol{\Theta}_{wb})\dot{\boldsymbol{\Theta}}_{wb}, \tag{2.5}$$

such that $\boldsymbol{T}(\boldsymbol{\Theta}_{wb})$ is defined as

$$\boldsymbol{T}(\boldsymbol{\Theta}_{wb}) = \begin{bmatrix} 1 & \mathrm{s}(\phi)\mathrm{t}(\theta) & \mathrm{c}(\phi)\mathrm{t}(\theta) \\ 0 & \mathrm{c}(\phi) & -\mathrm{s}(\phi) \\ 0 & \mathrm{s}(\phi)/\mathrm{c}(\theta) & \mathrm{c}(\phi)/\mathrm{c}(\theta) \end{bmatrix}. \tag{2.6}$$

This gives

$$\dot{\boldsymbol{\Theta}}_{wb} = \boldsymbol{T}(\boldsymbol{\Theta}_{wb})\boldsymbol{\omega}^{b}_{wb}, \tag{2.7}$$

which transforms the angular velocities into world frame. Note that $\boldsymbol{T}(\boldsymbol{\Theta}_{wb})$ contains singularities at $\theta = \pm\frac{\pi}{2}$. An operating quadcopter should never reach these orientations, so this should not introduce problems. However, possible solutions could be switching between two Euler angle representations with different singularities or using the unit quaternion representation. Finally, the 6-DOF kinematic equations of the quadcopter are expressed as

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}_{\boldsymbol{\Theta}_{wb}}(\boldsymbol{\eta})\boldsymbol{\nu}, \tag{2.8}$$

where

$$\boldsymbol{J}_{\boldsymbol{\Theta}_{wb}}(\boldsymbol{\eta}) = \begin{bmatrix} \boldsymbol{R}(\boldsymbol{\Theta}_{wb}) & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{T}(\boldsymbol{\Theta}_{wb}) \end{bmatrix}. \tag{2.9}$$

2.1.4 Kinetics

Kinetics refers to the study of motion and the forces that cause it. For a quadcopter, the relevant forces are the thrust from the rotors, the gravitational force, and the drag force. Most of the model used in this thesis is derived by the authors of [35], while drag is modeled by the authors of [44]. To simplify, the following assumptions are made:

- The quadcopter is axis-symmetrical, resulting in the system inertia being a diagonal matrix and $I_x = I_y$.
- The center of pressure coincides with the center of mass, meaning that the linear terms of the drag force act through the center of mass.
- There are no external forces, implying that the quadcopter is flying indoors or in calm weather.
- There exists a function to map the thrust of a propeller, F_i, to the required power of the propeller, W_i.
- The thrust of a propeller is applied instantly, with no inertia or time delay.

The quadcopter model, with equations of motion about the center of gravity, can be written as

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu}) + \boldsymbol{d}(\boldsymbol{\nu}) + \boldsymbol{g}(\boldsymbol{\eta}) = \boldsymbol{B}\boldsymbol{u}, \tag{2.10}$$

where $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{C}(\boldsymbol{\nu})$ is the Coriolis-centripetal vector, $\boldsymbol{d}(\boldsymbol{\nu})$ is the drag force, $\boldsymbol{g}(\boldsymbol{\eta})$ is the gravitational force, and $\boldsymbol{B}$ is the input matrix mapping the control inputs $\boldsymbol{u}$ to generalized forces, such that $\boldsymbol{\tau} = \boldsymbol{B}\boldsymbol{u}$.

As the equations of motion are about the center of gravity and the quadcopter is axis-symmetrical, the mass matrix becomes a diagonal matrix

$$\boldsymbol{M} = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_x & 0 & 0 \\ 0 & 0 & 0 & 0 & I_y & 0 \\ 0 & 0 & 0 & 0 & 0 & I_z \end{bmatrix}. \tag{2.11}$$

The Coriolis-centripetal forces are given as

$$\boldsymbol{C}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & 0 & (I_z - I_y)qr & (I_x - I_z)pr & 0 \end{bmatrix}^\top, \tag{2.12}$$

and occur due to the rotation of the body-fixed frame relative to the world frame. Note that the final term equals zero as $I_x = I_y$.

The drag force is an aerodynamic force caused by friction between the drone and surrounding air. It opposes the motion of the quadcopter and is given as

$$\boldsymbol{d}(\boldsymbol{\nu}) = \begin{bmatrix} d_u u & d_v v & d_w w & d_p p^2 & d_q q^2 & d_r r^2 \end{bmatrix}^\top, \tag{2.13}$$

where d_u, d_v and d_w are the linear drag coefficients, while d_p, d_q and d_r are the rotational drag coefficients.

The gravitational force acts through the center of mass, thus not participating with torques. As the force acts downwards in the world frame, its contribution to the linear

dynamics can be written as $\boldsymbol{g}^w = [0, 0, -mg]^\top$, where $g = 9.81\,\mathrm{m/s^2}$ is the gravitational acceleration. However, as the equations of motion are written in body frame, the gravitational force must be rotated, such that $\boldsymbol{g}^b = \boldsymbol{R}(\boldsymbol{\Theta}_{bw})\boldsymbol{g}^w$. Moving the term to the left-hand side of the equation yields

$$\boldsymbol{g}(\boldsymbol{\eta}) = \begin{bmatrix} -mg\mathrm{s}(\theta) & mg\mathrm{c}(\theta)\mathrm{s}(\phi) & mg\mathrm{c}(\theta)\mathrm{c}(\phi) & 0 & 0 & 0 \end{bmatrix}^\top. \tag{2.14}$$

The input matrix is given as

$$\boldsymbol{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & -l & 0 & l \\ -l & 0 & l & 0 \\ -\rho & \rho & -\rho & \rho \end{bmatrix}, \tag{2.15}$$

and specifies how the propeller thrusts

$$\boldsymbol{u} = \begin{bmatrix} F_1 & F_2 & F_3 & F_4 \end{bmatrix}^\top, \tag{2.16}$$

map into the state space. The third row of $\boldsymbol{B}$ is the thrust force, $f_z = F_1 + F_2 + F_3 + F_4$, which always works in the same direction as $\boldsymbol{z}_b$. The roll torque in the fourth row, $\tau_\phi = l(F_4 - F_2)$, is proportional to the difference between the thrust of the second and fourth propeller, while the pitch torque in the fifth row, $\tau_\theta = l(F_3 - F_1)$, is proportional to the difference between the thrust of the first and third propeller. The yaw torque in the sixth row, $\tau_\psi = \rho(-F_1 + F_2 - F_3 + F_4)$, is proportional to the difference of thrust force generated by all propellers.

2.2 Feedback Control

This section briefly introduces feedback control, a method within the field of control systems, and the theory is based on the work of the authors of [5].

The purpose of control systems is to regulate the behavior of systems. There are two primary methods: forward connection and feedback connection. With a forward connection, a disturbance is measured before it affects the process, and the controller considers this disturbance to minimize the effects. This approach is model-based, as the change in control input must be calculated from the disturbance. Feedback connection does, on the other hand, not require a realistic model of the system. The system states are measured and compared to reference values in a feedback controller. The states can be driven toward the reference values by designing a control input from the deviations from the desired states. A simple feedback controller of a mono-variable system is illustrated in Figure 2.2. A mono-variable system has only one reference signal and one measured state, while a multi-variable system control has several reference signals, several measured states, or both. This section only considers mono-variable systems for simplicity of notation, but the presented principles can easily be applied to multi-variable systems.

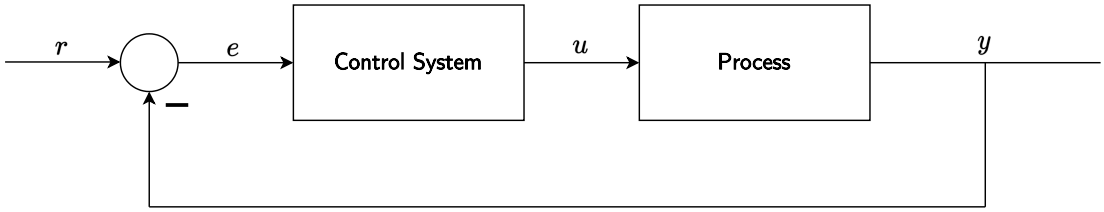


Figure 2.2: Illustration of feedback control for a process. The measurement of the state to be controlled is denoted y and has a reference value, r. The difference, $e = r - y$, between the measured and desired values is passed to the control system, which calculates the control input, u.

2.2.1 PID Controller

The Proportional-Integral-Derivative (PID) controller is perhaps the most common feedback controller and over 90% of industrial controllers use PID principles [42]. As the name suggests, the controller has incorporated three different effects, each serving a different purpose. The complete controller is given by

$$u = K_p e + K_i \int_0^t e(\tau)\mathrm{d}\tau + K_d \dot{e}, \tag{2.17}$$

where K_p, K_i, and K_d are the proportional-, integral-, and derivative gains. The transfer function can then be written on *parallel form* as

$$U(s) = K_p + K_i \frac{1}{s} + K_d s, \tag{2.18}$$

or on *ideal form* given by

$$U(s) = K_p \left(1 + \frac{1}{T_i s} + T_d s\right), \tag{2.19}$$

where s is the Laplace variable, and T_i and T_d are the integral- and derivative time constants.

The proportional gain of the controller generates the overall control action, which is proportional to the reference error. By increasing K_p, the system's time constant is reduced, meaning a more aggressive controller is achieved. Likewise, K_p can be reduced to get a slower response. By setting the gain too high, there might be fluctuations from the reference value, or in the worst case; the closed-loop system could become unstable. Thus, there is a trade-off between high stability and short transient.

The second functionality of the PID controller is the integral effect. With only a proportional gain, there will be a stationary deviation from the reference value when the system converges to a steady-state. By taking the integral of the error, this steady-state error will disappear. There is an intuitive explanation for why the steady-state error arises with only the proportional gain without diving too deep into the math. Imagine a proportional controller is trying to keep a car's cruise control at 60 km/h. If the car's speed reaches 60 km/h, then $e = 0$. This again yields $u = K_p e = 0$, meaning that the controller no longer provides any thrust to the motor. Subsequently, the car's speed will decrease due to drag forces. Thus, the proportional controller will only be able to keep a speed slightly lower than 60 km/h, with the size of the steady-state error dependent

on the value of K_p. However, introducing the integral effect will give a control action $\neq 0$ even though $e = 0$, allowing the cruise control to stay steady at 60 km/h. In addition to eliminating steady-state errors, the integral effect counteracts disturbances in the system. This could, e.g., be unmodeled wind gusts. Thus, the integral effect could be particularly important in a real-world application, where not all dynamics necessarily are modeled correctly.

The final functionality of the PID controller is the derivative effect, which contributes to the control input by a scalar multiplied by the change in error. The derivative term will improve the transient response by damping. When the state value approaches the reference value, the derivative of the error is negative, meaning that the term contributes with a negative control input. The reduced control input will dampen the transient and counteract possible fluctuations from the reference value. The derivative term can also cause an unstable system to become stable. These effects will improve if K_d increases and worsen if K_d decreases. Note that the transient response will be slower with the derivative term.

2.2.2 Anti-Windup

Often, the actuators of a system have an effective range limit for the actions. In a quadcopter's case, a maximum thrust force can be applied to the rotors. If the control input reaches the range limit, the integrator will saturate. That is, the error will continue being integrated up, even though no larger control input can be applied to the system. This can cause large overshoots and settling times for the system, and instability in the worst case. *Anti-windup* is a measure to prevent these effects.

Several techniques exist to perform anti-windup, whereas some are presented by the authors of [6]. In this thesis, anti-windup is performed as *conditional integration*, meaning that integration is stopped depending on a specific condition. The condition could, e.g., be that the integrator is saturated. With *limited integration*, the integrator value is limited to a linear range, and this will effectively prevent the value from becoming too high. A method called *tracking anti-windup* takes the difference between the saturated and unsaturated control input and generates a feedback signal to act upon the integrator input. The methods studied by the authors of [6] all reduced overshoot and thus proved to work as wanted. However, the use of limited integration showed to be the least desirable method.

2.3 Path Generation

This section presents an algorithm to generate a continuous path for the quadcopter. The Quadratic Polynomial Interpolation (QPMI) path-smoothing method was first introduced by the authors of [31] and [13] for 2D paths before it was extended to 3D paths [14]. When generating a path for a quadcopter to follow, it must be continuous in velocity and acceleration [14]. A G^2-continuous path shares a center of curvature and has the same second-order derivatives, meaning it is continuous in velocity and acceleration.

First, all the n_w waypoints the path should go through are defined. Each path will start

in $\boldsymbol{w}_1 = [0, 0, 0]^\top$, while the subsequent waypoints are generated randomly as

$$x_m = x_{m-1} + d\cos(\chi_m)\cos(\nu_m), \tag{2.20a}$$
$$y_m = y_{m-1} + d\sin(\chi_m)\cos(\nu_m), \tag{2.20b}$$
$$z_m = z_{m-1} + d\sin(\nu_m), \tag{2.20c}$$

where $m = 2, ..., n_w$ is the m^{th} waypoint, d is the linear distance between each waypoint, and χ_m and ν_m are randomly generated variables to define the curvature of the path.

The parametric path can be expressed as

$$P(u) : (x(u), y(u), z(u)), \tag{2.21}$$

where

$$x_m(u) = a_{x_m}u^2 + b_{x_m}u + c_{x_m}, \tag{2.22a}$$
$$y_m(u) = a_{y_m}u^2 + b_{y_m}u + c_{y_m}, \tag{2.22b}$$
$$z_m(u) = a_{z_m}u^2 + b_{z_m}u + c_{z_m}, \tag{2.22c}$$

with $m = 2, ..., n_w - 1$, and u is the linear distance. Equations (2.22a) to (2.22c) can be inverted to obtain the parameters of the path

$$\begin{bmatrix} a_{x_m} \\ b_{x_m} \\ c_{x_m} \end{bmatrix} = \begin{bmatrix} u_{m-1}^2 & u_{m-1} & 1 \\ u_m^2 & u_m & 1 \\ u_{m+1}^2 & u_{m+1} & 1 \end{bmatrix}^{-1} \begin{bmatrix} x(u_{m-1}) \\ x(u_m) \\ x(u_{m+1}) \end{bmatrix}, \tag{2.23a}$$

$$\begin{bmatrix} a_{y_m} \\ b_{y_m} \\ c_{y_m} \end{bmatrix} = \begin{bmatrix} u_{m-1}^2 & u_{m-1} & 1 \\ u_m^2 & u_m & 1 \\ u_{m+1}^2 & u_{m+1} & 1 \end{bmatrix}^{-1} \begin{bmatrix} y(u_{m-1}) \\ y(u_m) \\ y(u_{m+1}) \end{bmatrix}, \tag{2.23b}$$

$$\begin{bmatrix} a_{z_m} \\ b_{z_m} \\ c_{z_m} \end{bmatrix} = \begin{bmatrix} u_{m-1}^2 & u_{m-1} & 1 \\ u_m^2 & u_m & 1 \\ u_{m+1}^2 & u_{m+1} & 1 \end{bmatrix}^{-1} \begin{bmatrix} z(u_{m-1}) \\ z(u_m) \\ z(u_{m+1}) \end{bmatrix}. \tag{2.23c}$$

To generate a path of n_w waypoints, $3(n_w - 2)$ polynomials are required, as groups of polynomials connect three and three waypoints. The final path can then be denoted as

$$P(u) : (X(u), Y(u), Z(u)), \tag{2.24}$$

where

$$X(u) = \begin{Bmatrix} x_2(u), & u_1 \leq u \leq u_2 \\ \mu_{r,m}(u)x_{m+1} + \mu_{f,m}(u)x_m(u), & u_2 \leq u \leq u_{n_w-1} \\ x_{n_w-1}(u), & u_{n_w-1} \leq u \leq u_{n_w} \end{Bmatrix}, \tag{2.25a}$$

$$Y(u) = \begin{Bmatrix} y_2(u), & u_1 \leq u \leq u_2 \\ \mu_{r,m}(u)y_{m+1} + \mu_{f,m}(u)y_m(u), & u_2 \leq u \leq u_{n_w-1} \\ y_{n_w-1}(u), & u_{n_w-1} \leq u \leq u_{n_w} \end{Bmatrix}, \tag{2.25b}$$

$$Z(u) = \begin{Bmatrix} z_2(u), & u_1 \leq u \leq u_2 \\ \mu_{r,m}(u)z_{m+1} + \mu_{f,m}(u)z_m(u), & u_2 \leq u \leq u_{n_w-1} \\ z_{n_w-1}(u), & u_{n_w-1} \leq u \leq u_{n_w} \end{Bmatrix}. \tag{2.25c}$$

$\mu_{r,m}$ and $\mu_{f,m}$ are increasing and decreasing membership functions, respectively, with value spans between zero and one. They represent the transition from one polynomial to another and are defined as

$$\mu_{r,m} = \frac{u - u_m}{u_{m+1} - u_m}, \tag{2.26a}$$

$$\mu_{f,m} = \frac{u_{m+1} - u}{u_{m+1} - u_m}, \tag{2.26b}$$

where $m = 2, ..., n_w - 1$. It is worth noticing that $\mu_{r,1} = 1$ and $\mu_{f,1} = 0$ for the first polynomial and that $\mu_{r,n_w-1} = 0$ and $\mu_{f,n_w-1} = 1$ for the final polynomial.

The path is G^1-continuous if the first-order derivatives match at each waypoint and G^2-continuous if the second-order derivatives match at each waypoint. The first-order derivatives of the path, $\frac{\mathrm{d}X(u)}{\mathrm{d}u}$, $\frac{\mathrm{d}Y(u)}{\mathrm{d}u}$ and $\frac{\mathrm{d}Z(u)}{\mathrm{d}u}$, can be calculated by inserting

$$\frac{\mathrm{d}x_m(u)}{\mathrm{d}u} = 2a_{x_m}u + b_{x_m}, \tag{2.27a}$$

$$\frac{\mathrm{d}y_m(u)}{\mathrm{d}u} = 2a_{y_m}u + b_{y_m}, \tag{2.27b}$$

$$\frac{\mathrm{d}z_m(u)}{\mathrm{d}u} = 2a_{z_m}u + b_{z_m}, \tag{2.27c}$$

into Equations (2.25a) to (2.25c). Finally, the second-order derivatives of the path, $\frac{\mathrm{d}^2X(u)}{\mathrm{d}u^2}$, $\frac{\mathrm{d}^2Y(u)}{\mathrm{d}u^2}$ and $\frac{\mathrm{d}^2Z(u)}{\mathrm{d}u^2}$, can be calculated by inserting

$$\frac{\mathrm{d}^2x_m(u)}{\mathrm{d}u^2} = 2a_{x_m}, \tag{2.28a}$$

$$\frac{\mathrm{d}^2y_m(u)}{\mathrm{d}u^2} = 2a_{y_m}, \tag{2.28b}$$

$$\frac{\mathrm{d}^2z_m(u)}{\mathrm{d}u^2} = 2a_{z_m}, \tag{2.28c}$$

into Equations (2.25a) to (2.25c).

2.4 Convolutional Neural Networks

This section gives an introduction to CNNs, as this is used for the perception of the quadcopter. A more thorough description of CNNs is presented by the authors of [23].

CNNs are a type of neural network that is commonly used for image classification [68], object detection [69] and segmentation [77]. CNNs do typically consist of several different layers: convolutional layers, pooling layers, and fully connected layers. The input to the CNN is often an image, possibly with several channels, where each pixel has a value. The image is passed through the CNN, which finally provides an output depending on the use case.

The *convolutional layer* convolves the input with a learnable kernel, illustrated in Figure 2.3. This outputs a feature map, representing the features of the input. Thus, different kernels will detect different features of the input. Adjusting the kernel allows emphasis on different aspects of the input. A larger kernel size enhances the detection of larger features while potentially disregarding some finer details. Stride is the kernel's

horizontal or vertical step size during convolution. Increasing the stride leads to a lower-dimensional feature map output. Padding can also be employed at the image edges to give greater significance to border size information. As with normal neural networks, activation functions can be applied to the output. The size of the outputted feature map can be read from the formula

$$W_O = \frac{W_I - K + 2P}{S} + 1, \tag{2.29}$$

where W_O is the output volume, W_I is the input volume, K is the kernel size, P is the padding and S is the stride. Thus, an image of size 55×35 convolved when $K = 5$, $P = 2$ and $S = 2$, will output an image of size 28×18.

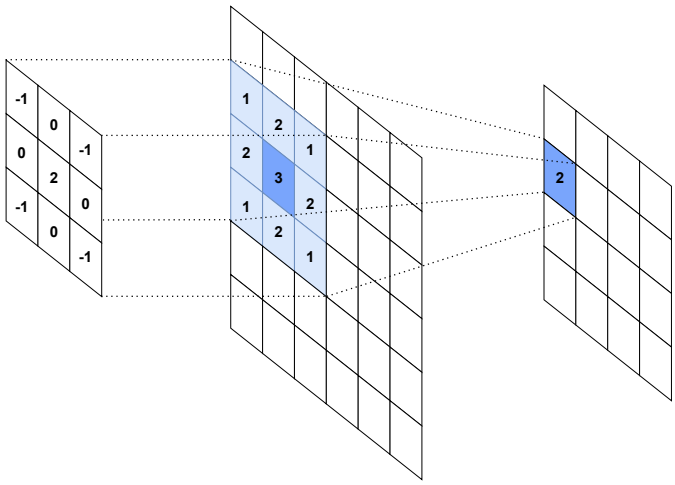


Figure 2.3: An example of an input of size 6×6 convolved with a kernel of size 3×3 and stride of one, resulting in a feature map of size 4×4.

Pooling layers can be used for dimensionality reduction of the feature maps by shrinking them to lower-sized feature maps. There are several different pooling operations, such as max pooling, min pooling, and average pooling. The operation is performed within a particular area and with a specified stride, e.g., max pooling could be used to divide the feature map into a grid with cells of size 2×2 and only keep the highest value in each cell. Pooling can allow only preserving the most dominant feature within each part of the image and ignore details such as variations in the position of the feature.

Fully connected layers are standard neural network layers, which are fed flattened feature maps. These layers allow the detection of relationships between the high-level features before they provide the output of the CNN.

2.5 Reinforcement Learning

This section is heavily based on the theory section of the specialization report [10]. There are made some slight changes to the explanations and Figure 2.4 is added, otherwise, the derived formulas are the same.

In this section, the basics of RL are briefly introduced, specifically the PPO [65] algorithm. A more exhaustive explanation of RL is given by the authors of [72]. PPO

is the only algorithm used in this thesis, as several studies conclude it surpasses other RL algorithms. The authors of [37] found that PPO outperforms Trust Region Policy Optimization (TRPO) [63] and Deep Deterministic Policy Gradient (DDPG) [43] in an attitude controller for a quadcopter. PPO has also been shown to perform better than Advantage Actor-Critic (A2C) [50] when stabilizing a quadcopter and moving to a target [30]. Further, the authors of [41] conclude that PPO is superior with respect to generalization, robustness to complexity changes, and changes in the reward function for path following and collision avoidance for an autonomous surface vessel.

RL is one of three primary learning paradigms in machine learning, alongside supervised learning and unsupervised learning [54]. Supervised learning is learning from a set of labeled data to detect relationships between input and output data, enabling it to yield correct labels on unseen data. On the other hand, unsupervised learning aims to detect hidden structures in unlabeled data. Sutton and Barto define RL as learning what to do - how to map situations to actions - to maximize a numerical reward signal [72].

The idea of RL is to let a learning agent interact with an environment over time to achieve a goal. By observing the state of the environment and taking actions to interact with the environment, the agent can affect the state. The actions taken by the agent will affect the reward received, both the immediate reward and the subsequent rewards. Thus, the agent must try to learn which actions will yield the highest cumulative reward. However, if the agent only makes actions it has already experienced to give the highest reward, it will never know if there are other, more effective actions. This introduces the exploration-exploitation dilemma. The agent must exploit previous experience to obtain a reward and explore other actions to take better actions in the future.

2.5.1 Preliminaries

In RL, the interaction between the agent and the environment can be modeled as a Markov Decision Process (MDP), illustrated in Figure 2.4, meaning that the next state depends only on the current state and action [54]. Formally, this is defined by the 5-tuple, $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where:

- $\mathcal{S}$ is the continuous state space, with the initial state denoted s_0,
- $\mathcal{A}$ is the continuous action space,
- $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the conditional transition probability distribution for the next state, s', such that $p(s'|s, a) = P(S_{t+1} = s'|S_t = s, A_t = a)$, with the initial state distribution denoted $p_0 : \mathcal{S} \to [0, 1]$,
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward received after action a transitions state s to state s',
- $\gamma \in [0, 1]$ is the discount factor for future rewards.

In this section, the state, action, and neural network parameters can be represented using vector notation. However, for the sake of simplicity, non-bold notation will be used instead. It is also worth noting that the observations are not necessarily equal to the state, but both observations and state will be denoted s in this section.

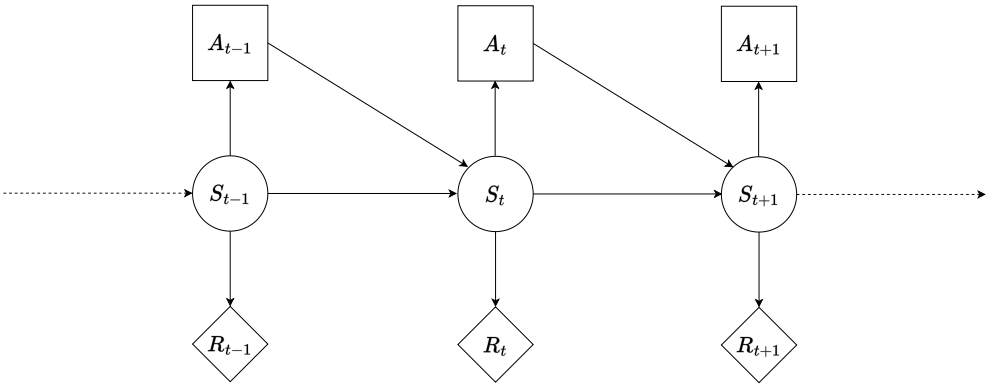


Figure 2.4: MDP showing how each state only is dependent on the previous state and action. The action is taken based on an observation of the current state, and the reward is received after an action transitions the state.

The policy function, π, defines how an action, a, at state, s, should be chosen. π specifies a conditional probability distribution to choose a possible action for each state, given by $\pi_\theta(a|s) : \mathcal{S} \times \mathcal{A} \to [0, 1] = P(A_t = a|S_t = s)$, where θ are the parameters of the neural network producing the output. As the policy always depends on θ, the policy is only denoted π from now on. By, at each time step t, observe the state s_t, draw an action a_t from the policy π, and observe the next state $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$ and reward r_t, there will be developed a trace in the environment

$$\tau_t = \{s_t, a_t, r_t, ...\}. \tag{2.30}$$

Further, the authors of [54] define the state-value function, $V^\pi(s_t)$, as the expected reward when standing in state, s, at time, t, and following policy, π, while the action-value function, $Q^\pi(s_t, a_t)$, is defined as the expected return when taking action, a, in state, s, at time t, and following policy, π. This is formalized by

$$V^\pi(s_t) = \mathbb{E}_{\tau_t \sim \pi}\left[\sum_{i=0}^{\infty} \gamma^i R(s_{t+i}, a_{t+i}) \middle| s_t\right], \tag{2.31}$$

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\tau_t \sim \pi}\left[\sum_{i=0}^{\infty} \gamma^i R(s_{t+i}, a_{t+i}) \middle| s_t, a_t\right], \tag{2.32}$$

where $R(s_t, a_t)$ is the reward at time t, and $\tau_t \sim \pi$ is the trace obtained from time, t, by following the policy, π.

2.5.2 Policy Gradients

Both policy-based and value-based methods within RL aim to maximize the expected discounted reward. Policy-based methods find the explicit policy function. This differs from value-based methods that try to find the value function and use this to decide on the best possible action. However, in the case of continuous or large discrete action spaces, value-based methods can fail as it would be impractical to find the greedy policy [25].

The optimal policy can be derived by defining the objective function

$$J(\theta) = V^{\pi}(s_0), \tag{2.33a}$$

$$= \mathbb{E}_{\tau_0 \sim \pi}\left[\sum_{i=0}^{\infty} \gamma^i R(s_i, a_i) \middle| s_0\right], \tag{2.33b}$$

as this describes the long-term reward obtained from the start state s_0. The authors of [73] prove the policy gradient theorem, showing that the gradients can be found as

$$\nabla_\theta J(\theta) = \sum_s d^{\pi}(s) \sum_a \nabla_\theta \pi(a|s) Q^{\pi}(s, a), \tag{2.34}$$

where $d^{\pi}(s) = \lim_{t \to \infty} P(S_t = s|s_0)$, which is the stationary distribution of states under π. As explained by the authors of [22], the likelihood ratio trick

$$\nabla_\theta \pi(a|s) = \pi(a|s) \frac{\nabla_\theta \pi(a|s)}{\pi(a|s)}, \tag{2.35a}$$

$$= \pi(a|s) \nabla_\theta \log \pi(a|s), \tag{2.35b}$$

can be used to derive that

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi}(s, a)\right]. \tag{2.36}$$

To reduce the variance of the policy gradients, it is common to replace $Q^{\pi}(s, a)$ with the advantage function $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s, a)$. This will tell whether or not the action outperforms the policy's default behavior. However, the advantage function is unknown and must be estimated, which can be done using the Generalized Advantage Estimation (GAE), proposed by the authors of [64]. This finally yields the policy gradient estimator at time step t

$$\widehat{\nabla_\theta J(\theta)} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi(A_t|S_t) \hat{A}_t^{\pi}\right], \tag{2.37}$$

allowing to update the parameters with $\theta \leftarrow \theta + \alpha \widehat{\nabla_\theta J(\theta)}$. $\hat{\mathbb{E}}_t[...]$ is the empirical average over a finite batch of samples by alternating between sampling and optimization [65]. The estimator can then be obtained by differentiating the objective

$$J^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi(A_t|S_t) \hat{A}_t^{\pi}\right]. \tag{2.38}$$

2.5.3 Proximal Policy Optimization

PPO, first suggested by the authors of [65], optimizes a modification of the objective function in Equation (2.38). Defining the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, the authors suggest the modified surrogate objective function

$$J^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right], \tag{2.39}$$

where the hyperparameter ϵ is used to configure the size of the trust region. This objective function will guarantee policy improvements within a certain range, restricted by the term $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$. Clipping the probability ratio removes the incentive for moving r_t outside the trust region $[1-\epsilon, 1+\epsilon]$.

The pseudocode for the PPO algorithm is given in Algorithm 1. For each iteration, the N parallel actors collect T time steps of data. The surrogate loss for these NT time steps of data is calculated and optimized with either minibatch stochastic gradient descent or Adam [36] for K epochs.

Algorithm 1: Proximal Policy Optimization

for *iteration=1,2,...* **do**
  for *actor=1,2,...,N* **do**
    Run policy $\pi_{\theta_{old}}$ in environment for T time steps
    Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
  Optimize surrogate L wrt. θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$

Chapter 3

Methodology

This chapter covers the implementation of the path following controller and DRL agent. The simulation environment with the quadcopter, path, and obstacles is presented in Section 3.1. The path following controller, detailed in Section 3.2, calculates the control inputs to realize a desired acceleration. The DRL agent, presented in Section 3.3, learns local path planning around the obstacles by creating the next waypoint in an alternative path for the quadcopter to follow. The action space, observation space, policy network, and reward function to realize this are outlined. Details about the training and evaluation process are provided in Section 3.4 and Section 3.5, respectively. Further implementations for this thesis can be found in the gym-quad framework [11].

3.1 Simulation Environment

The training environment consists of three components: the quadcopter, a path, and obstacles. Training in simulated real-world environments could be computationally expensive because of the resolution of surroundings data. Also, it would not be appropriate to begin training with a real quadcopter, as this would lead to material damage and extremely long training times. Thus, simpler, synthetic environments are created for training, with a predefined path and spherical obstacles. Path and obstacle generation are done stochastically to ensure variety in the training environments and achieve generalization abilities for the agent. An example of these generated environments is provided in Figure 3.1.

All paths are generated by creating waypoints with a linear distance of $d = 50$ m between each other. The direction of the next waypoint is decided stochastically, with a uniform distribution for both the azimuth angle $\chi_m \sim U(-\frac{\pi}{4}, \frac{\pi}{4})$ and the elevation angle $\nu_m \sim U(-\frac{\pi}{4}, \frac{\pi}{4})$. Each path starts in $\boldsymbol{w}_1 = [0, 0, 0]^\top$, while the positions of the subsequent waypoints are decided by

$$x_m = x_{m-1} + d\cos(\chi_m)\cos(\nu_m), \tag{3.1a}$$
$$y_m = y_{m-1} + d\sin(\chi_m)\cos(\nu_m), \tag{3.1b}$$
$$z_m = z_{m-1} + d\sin(\nu_m), \tag{3.1c}$$

where $m = 2, ..., n_w$ is the m^{th} waypoint. The paths are generated with $n_w = 7$ waypoints, meaning the linear distance between the waypoints is 300 m in total. Then, a

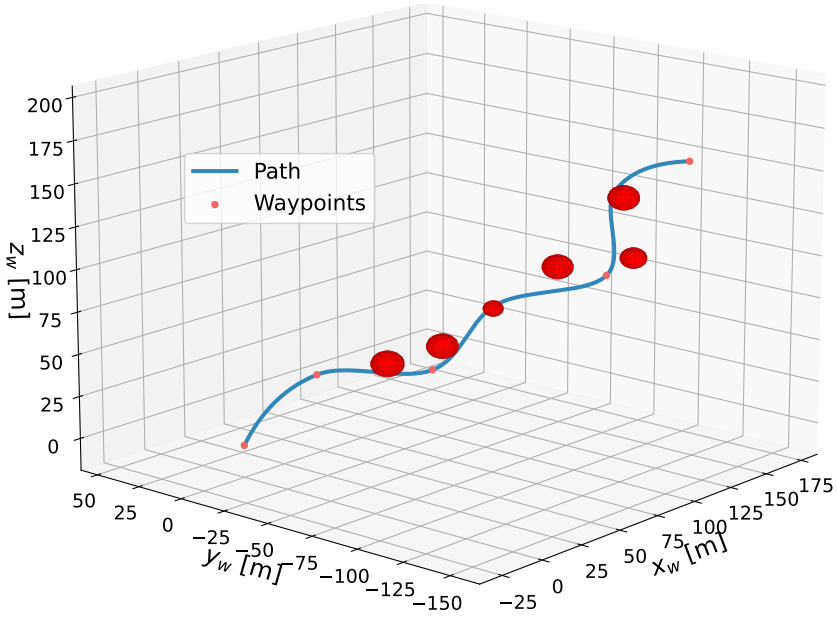


Figure 3.1: Example of training environment with a G^2-continuous path and six stochastically placed obstacles, whereas at least one obstacle is guaranteed to obstruct the path.

G^2-continuous path between the waypoints is generated, according to Section 2.3.

There are placed six spherical obstacles in the training environment. The obstacles' position and radius are decided randomly to ensure generalization. All obstacle radii are drawn from a uniform distribution between four and ten meters. The intention is that the training environments contain obstacles both on and off the path. If all obstacles are placed directly on the path, the quadcopter can learn to always keep a safe distance from the path and thus never correctly learn path following. Similarly, the quadcopter can risk not learning collision avoidance properly if very few obstacles are placed on the path. Therefore, one obstacle is placed halfway through the path, guaranteed to obstruct the path. In addition, there are placed five obstacles between $\frac{1}{6}$ and $\frac{5}{6}$ of the length of the path with a random position, meaning they can be placed both on and off the path. There are no obstacles at the beginning or end of the path to allow the quadcopter to converge to the path in the beginning and make it possible to reach the final waypoint.

The model parameters must be defined to realize the model dynamics of the quadcopter, which were derived in Section 2.1. The model parameters are taken from [44] and summarized in Table 3.1.

3.2 Control Law for Path Following

For this thesis, a path following controller for the quadcopter is derived and implemented, which is inspired by the approach for deriving the small angle control law presented by the authors of [47]. The control law is designed to provide the inputs, $\boldsymbol{u} = [F_1, F_2, F_3, F_4]^\top$, such that the quadcopter can track a time-independent trajectory in three dimensions. It is assumed that the attitude of the quadcopter is close to hover

Table 3.1: Parameters for the quadcopter model.

Parameter	Description	Value
g	Gravitational acceleration	9.81 m/s^2
h	Step size of simulation	0.01 s
m	Mass	0.5 kg
l	Arm length	0.5 m
ρ	Inflow ratio	0.08
F_{min}	Minimum rotor thrust	-6 N
F_{max}	Maximum rotor thrust	6 N
I_x	Moment of inertia about the $\boldsymbol{x}_b$-axis	0.005 kgm^2
I_y	Moment of inertia about the $\boldsymbol{y}_b$-axis	0.005 kgm^2
I_z	Moment of inertia about the $\boldsymbol{z}_b$-axis	0.01 kgm^2
d_u	Linear drag coefficient along the $\boldsymbol{x}_b$-axis	0.3729
d_v	Linear drag coefficient along the $\boldsymbol{y}_b$-axis	0.3729
d_w	Linear drag coefficient along the $\boldsymbol{z}_b$-axis	0.3729
d_p	Rotational drag coefficient about the $\boldsymbol{x}_b$-axis	$5.56 \cdot 10^{-4}$
d_q	Rotational drag coefficient about the $\boldsymbol{y}_b$-axis	$5.56 \cdot 10^{-4}$
d_r	Rotational drag coefficient about the $\boldsymbol{z}_b$-axis	$5.56 \cdot 10^{-4}$

state, which is reasonable for non-aggressive trajectories. An overview of the controller is presented in Figure 3.2.

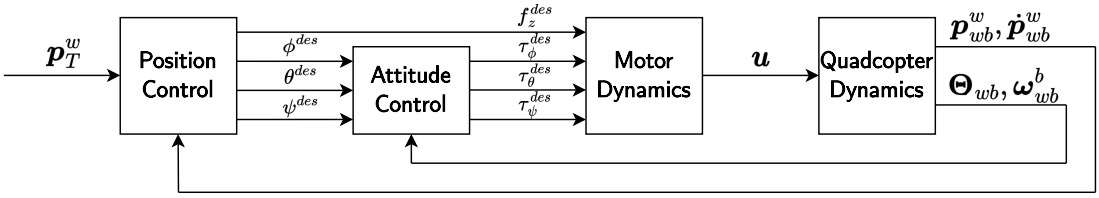


Figure 3.2: Illustration of the path following controller, consisting of nested control loops for position and attitude. The position controller calculates the desired thrust force, while the attitude controller calculates the desired torques. The input, $\boldsymbol{u}$, can then be calculated through the motor dynamics to track a three-dimensional trajectory.

The controller utilizes a nested control loop for position and attitude to calculate the input vector. The position controller calculates the desired thrust force, $f_z^{des} = F_1 + F_2 + F_3 + F_4$, while the attitude controller calculates the desired roll torque, $\tau_\phi^{des} = l(F_4 - F_2)$, desired pitch torque, $\tau_\theta^{des} = l(F_3 - F_1)$, and desired yaw torque, $\tau_\psi^{des} = \rho(-F_1 + F_2 - F_3 + F_4)$. The thrust inputs to the quadcopter can then be calculated by inverting the last four rows of the input matrix, $\boldsymbol{B}$, as follows

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -l & 0 & l \\ -l & 0 & l & 0 \\ -\rho & \rho & -\rho & \rho \end{bmatrix}^{-1} \begin{bmatrix} f_z^{des} \\ \tau_\phi^{des} \\ \tau_\theta^{des} \\ \tau_\psi^{des} \end{bmatrix}. \tag{3.2}$$

3.2.1 Position Controller

The position controller calculates the desired acceleration, $\boldsymbol{a}^{des} = [\ddot{x}^{des}, \ddot{y}^{des}, \ddot{z}^{des}]^\top$ required to follow the path. To realize this acceleration, the desired thrust force, f_z^{des}, is calculated and the desired attitude, $\boldsymbol{\Theta}_{wb}^{des} = [\phi^{des}, \theta^{des}, \psi^{des}]^\top$, is passed to the attitude controller for calculation of the desired torques. All this is derived from position errors, a velocity term, and a feedforward term of the trajectory acceleration.

The quadcopter's distance from the path is controlled through a PID feedback controller, taking the position error, accumulated position error, and change of position error into account. This finally yields a positional contribution, $\boldsymbol{a}_{pos}^{des}$, to the desired acceleration. The position error is defined as the difference between the position of the closest point on the trajectory, $\boldsymbol{p}_T^w = [x_T^w, y_T^w, z_T^w]^\top$, and the position of the quadcopter, $\boldsymbol{p}_{wb}^w = [x^w, y^w, z^w]^\top$, given by

$$\boldsymbol{e}_p^w = \boldsymbol{p}_T^w - \boldsymbol{p}_{wb}^w. \tag{3.3}$$

The controller's integral term contributes to the desired acceleration dependent on the accumulated position errors of the relevant episode. The integral term is implemented to eliminate stationary position errors from the reference trajectory, as these were significant in the absence of the integral term. The integral term will also make the controller more prone to wind gusts and other unexpected disturbances [59]. As the controller is implemented in discrete time, the integral of the error is calculated as a sum of all position errors in the relevant episode multiplied by the step size h, as

$$\boldsymbol{e}_{p,acc}^w = \sum_{\tau=0}^{t} \boldsymbol{e}_p^w(\tau)h, \tag{3.4}$$

where $\boldsymbol{e}_p^w(\tau)$ is the position error at time τ. However, simply implementing this integral term leads to large accelerations when the quadcopter's position is far from the desired path. Therefore, anti-windup is implemented, meaning that $\boldsymbol{e}_{p,acc}^w$ is only updated if $\boldsymbol{a}^{des}$ at the previous time step was smaller than a chosen threshold. This threshold is chosen to be 0.5 $\mathrm{m/s^2}$ through trial and error. A derivative term is added to the controller to avoid fluctuations from the path and reduce the risk of instability. This contributes to the desired acceleration by considering how the position error has changed since the previous time step

$$\dot{\boldsymbol{e}}_p^w = \frac{\boldsymbol{e}_p^w(t) - \boldsymbol{e}_p^w(t-1)}{h}. \tag{3.5}$$

The positional contribution to the desired acceleration is then calculated as

$$\boldsymbol{a}_{pos}^{des} = \boldsymbol{K}_{p,a}\boldsymbol{e}_p^w + \boldsymbol{K}_{i,a}\boldsymbol{e}_{p,acc}^w + \boldsymbol{K}_{d,a}\dot{\boldsymbol{e}}_p^w, \tag{3.6}$$

where $\boldsymbol{K}_{p,a}$, $\boldsymbol{K}_{i,a}$ and $\boldsymbol{K}_{d,a}$ are diagonal matrices representing the proportional, integral, and derivative gains for the position. The values of the gains influence how much the acceleration should change to minimize the error, meaning that a higher value indicates a higher priority to minimize the error. In the implementation, $\boldsymbol{a}_{pos}^{des}$ is scaled down to a vector of length 1.5 in the case where the Euclidean norm is larger than 1.5. This is done as the quadcopter being far from the path does not necessarily mean the desired acceleration should be extremely large.

To make the quadcopter move along the path and not only move to its closest point and stay there, a velocity term is added to the position controller. A reference velocity is not directly provided through the trajectory, as it only consists of positions. However,

a velocity reference is created by considering the tangent of the trajectory at $\boldsymbol{p}_T^w$ and scaling it to a desired velocity, as follows

$$\boldsymbol{v}_T^w = v^{des} \frac{\boldsymbol{t}_T^w}{\|\boldsymbol{t}_T^w\|}, \tag{3.7}$$

where $\boldsymbol{t}_T^w$ denotes the tangent and $v^{des} = 2.5\text{m/s}$. $\boldsymbol{t}_T^w$ is calculated by inserting Equation (2.27) into Equation (2.25). Then, the velocity error is given as

$$\boldsymbol{e}_v^w = \boldsymbol{v}_T^w - \dot{\boldsymbol{p}}_{wb}^w, \tag{3.8}$$

where $\dot{\boldsymbol{p}}_{wb}^w = [\dot{x}^w, \dot{y}^w, \dot{z}^w]^\top$ is the quadcopter's velocity in world frame. Similarly, as with the positional contribution to the desired acceleration, the velocity error contributes to the desired acceleration with the term

$$\boldsymbol{a}_{vel}^{des} = \boldsymbol{K}_v \boldsymbol{e}_v^w, \tag{3.9}$$

where $\boldsymbol{K}_v$ is a diagonal matrix representing gains for the velocity term. $\boldsymbol{a}_{vel}^{des}$ is scaled down to a vector of length one in the case where the Euclidean norm is larger than one.

Finally, a feedforward term for the trajectory acceleration, $\boldsymbol{n}_T^w$, is added to the desired acceleration. In practice, this is the normal vector of the path, which is calculated by inserting Equation (2.28) into Equation (2.25). The purpose of this term is to make the controller resistant to sharp turns in the trajectory. This term is, however, probably not necessary as the trajectories in this thesis are not very aggressive. The resulting desired acceleration is then given by

$$\boldsymbol{a}^{des} = \boldsymbol{a}_{pos}^{des} + \boldsymbol{a}_{vel}^{des} + \boldsymbol{n}_T^w. \tag{3.10}$$

When the desired acceleration is calculated, the desired thrust force and attitude can be derived.

The desired thrust force is found by considering the acceleration in the $\boldsymbol{z}_w$-direction. Rewriting Equation (2.8) and Equation (2.10), the desired acceleration can be formulated as

$$m\ddot{z}^{des} + d_w w \mathrm{c}(\phi)\mathrm{c}(\theta) + mg = f_z^{des} \mathrm{c}(\phi)\mathrm{c}(\theta). \tag{3.11}$$

Close to hover state, ϕ and θ are small, giving the approximations $\mathrm{c}(\phi) \approx 1$ and $\mathrm{c}(\theta) \approx 1$. Then, the thrust force can be expressed as

$$f_z^{des} = m(\ddot{z}^{des} + g) + d_w w. \tag{3.12}$$

The desired attitude, $\boldsymbol{\Theta}_{wb}^{des} = [\phi^{des}, \theta^{des}, \psi^{des}]^\top$, must be derived, as it is passed to the attitude controller. ψ^{des} can simply be calculated by considering the yaw angle of the tangent at $\boldsymbol{p}_T^w$, like

$$\psi^{des} = \arctan\left(\frac{t_{T,y}^w}{t_{T,x}^w}\right), \tag{3.13}$$

where $t_{T,x}^w$ is the component of the tangent in $\boldsymbol{x}_w$-direction and $t_{T,y}^w$ is the component of the tangent in $\boldsymbol{y}_w$-direction. This will drive the heading of the quadcopter to follow the heading of the path. Similarly as with the calculation of f_z^{des}, the desired rotations ϕ^{des} and θ^{des} can be calculated by using the small-angle approximation on the expressions

for acceleration in the $\boldsymbol{x}_w$-direction and $\boldsymbol{y}_w$-direction. The equation of motion in $\boldsymbol{x}_w$-direction is

$$m\ddot{x} + d_u u(\mathrm{c}(\phi)\mathrm{s}(\theta)\mathrm{c}(\psi) + \mathrm{s}(\phi)\mathrm{s}(\psi)) = f_z(\mathrm{c}(\phi)\mathrm{s}(\theta)\mathrm{c}(\psi) + \mathrm{s}(\phi)\mathrm{s}(\psi)), \tag{3.14}$$

which can be simplified into

$$m\ddot{x}^{des} = (f_z^{des} - d_u u)(\theta^{des}\mathrm{c}(\psi^{des}) + \phi^{des}\mathrm{s}(\psi^{des})), \tag{3.15}$$

by assuming $\mathrm{c}(\phi) \approx 1$, $\mathrm{c}(\theta) \approx 1$, $\mathrm{s}(\phi) \approx \phi$ and $\mathrm{s}(\theta) \approx \theta$. Further, f_z^{des} can be inserted to find the expression

$$\ddot{x}^{des} = \left(\ddot{z}^{des} + g + \frac{d_w w - d_u u}{m}\right)\left(\theta^{des}\mathrm{c}(\psi^{des}) + \phi^{des}\mathrm{s}(\psi^{des})\right). \tag{3.16}$$

The same procedure can be repeated for the acceleration in $\boldsymbol{y}_w$-direction, resulting in

$$\ddot{y}^{des} = \left(\ddot{z}^{des} + g + \frac{d_w w - d_v v}{m}\right)\left(\theta^{des}\mathrm{s}(\psi^{des}) - \phi^{des}\mathrm{c}(\psi^{des})\right). \tag{3.17}$$

Finally, Equation (3.16) and Equation (3.17) are rewritten to find the desired rotations

$$\phi^{des} = \frac{\ddot{x}^{des}}{\ddot{z}^{des} + g + \frac{d_w w - d_u u}{m}}\mathrm{s}(\psi^{des}) - \frac{\ddot{y}^{des}}{\ddot{z}^{des} + g + \frac{d_w w - d_v v}{m}}\mathrm{c}(\psi^{des}), \tag{3.18}$$

$$\theta^{des} = \frac{\ddot{x}^{des}}{\ddot{z}^{des} + g + \frac{d_w w - d_u u}{m}}\mathrm{c}(\psi^{des}) + \frac{\ddot{y}^{des}}{\ddot{z}^{des} + g + \frac{d_w w - d_v v}{m}}\mathrm{s}(\psi^{des}). \tag{3.19}$$

3.2.2 Attitude Controller

The attitude controller uses the desired attitude to calculate the desired roll torque, τ_ϕ^{des}, desired pitch torque, τ_θ^{des}, and desired yaw torque, τ_ψ^{des}, from the errors in attitude and angular velocity. The attitude error is defined as the difference between the desired attitude and the attitude of the quadcopter, such that

$$\boldsymbol{e}_\Theta^b = \boldsymbol{\Theta}_{wb}^{des} - \boldsymbol{\Theta}_{wb}. \tag{3.20}$$

For simplicity, it is chosen that the desired angular velocities are zero, independently of the path. This means that the quadcopter should have small changes in attitude when moving along the path. Thus, the error in angular velocities is defined as

$$\boldsymbol{e}_\omega^b = -\boldsymbol{\omega}_{wb}^b, \tag{3.21}$$

where $\boldsymbol{\omega}_{wb}^b = [p, q, r]^\top$. The desired torque, $\boldsymbol{\tau}^{des} = [\tau_\phi^{des}, \tau_\theta^{des}, \tau_\psi^{des}]^\top$, is calculated using a Proportional-Derivative (PD) feedback controller, giving the following control law

$$\boldsymbol{\tau}^{des} = \boldsymbol{K}_{p,\tau}\boldsymbol{e}_\Theta^b + \boldsymbol{K}_{d,\tau}\boldsymbol{e}_\omega^b, \tag{3.22}$$

where $\boldsymbol{K}_{p,\tau}$ and $\boldsymbol{K}_{d,\tau}$ are diagonal matrices representing the proportional and derivative gains, respectively.

3.2.3 Tuning of Control Gains

The authors of [47] suggest relying on the dynamic model to tune the control law. It is chosen to do this for the attitude controller, but as the position controller is significantly more complicated than the controller in [47], it is more effective to tune the gains manually. All control gains are summarized in Table 3.2.

Deciding the values of $\boldsymbol{K}_{p,\tau} = \text{diag}(K_{p,\phi}, K_{p,\theta}, K_{p,\psi})$ and $\boldsymbol{K}_{d,\tau} = \text{diag}(K_{d,\phi}, K_{d,\theta}, K_{d,\psi})$ is done by deriving the equations of motion for rotation about the $\boldsymbol{x}_w$-, $\boldsymbol{y}_w$- and $\boldsymbol{z}_w$-axes. The equation of motion for rotation about $\boldsymbol{x}_w$ is given as

$$I_x\dot{p} + (I_z - I_y)qr + d_p p^2 = \tau_\phi, \tag{3.23a}$$
$$I_x\dot{p} + (I_z - I_y)qr + d_p p^2 = K_{p,\phi}(\phi^{des} - \phi) + K_{d,\phi}(p^{des} - p). \tag{3.23b}$$

By assuming $p \approx \dot{\phi}$ near hover state and that the terms qr and p^2 are small, the equation of motion for rotation can be written as

$$I_x\ddot{\phi} = -K_{p,\phi}\phi - K_{d,\phi}\dot{\phi}, \tag{3.24a}$$
$$I_x\ddot{\phi} + K_{d,\phi}\dot{\phi} + K_{p,\phi}\phi = 0, \tag{3.24b}$$

when the desired attitude is at hover. As this is a second-order system, it can be written on the form

$$\ddot{\phi} + 2\xi\omega_n\dot{\phi} + \omega_n^2\phi = 0, \tag{3.25}$$

giving $K_{p,\phi} = I_x\omega_n^2$ and $K_{d,\phi} = 2I_x\xi\omega_n$, where ξ is the relative damping ratio and ω_n is the natural frequency. The exact same approach is used to find the expressions for the remaining control gains. The control gains are given by $K_{p,\theta} = I_y\omega_n^2$, $K_{d,\theta} = 2I_y\xi\omega_n$, $K_{p,\psi} = I_z\omega_n^2$ and $K_{d,\psi} = 2I_z\xi\omega_n$. It is chosen to set $\xi = 1$ for the system to be critically damped and $\omega_n = 9\ \frac{\text{rad}}{\text{s}}$ as this was suggested as a starting point for tuning by the authors of [47].

Table 3.2: Values of control gains for the path following controller.

Parameter	Description	Value
$\boldsymbol{K}_{p,a}$	Proportional gains for the position controller	$\text{diag}(3, 3, 6)$
$\boldsymbol{K}_{i,a}$	Integral gains for the position controller	$\text{diag}(0.01, 0.01, 0.01)$
$\boldsymbol{K}_{d,a}$	Derivative gains for the position controller	$\text{diag}(1, 1, 1)$
$\boldsymbol{K}_{v}$	Velocity gains for the position controller	$\text{diag}(0.5, 0.5, 0.5)$
$\boldsymbol{K}_{p,\tau}$	Proportional gains for the attitude controller	$\text{diag}(0.405, 0.405, 0.81)$
$\boldsymbol{K}_{d,\tau}$	Derivative gains for the attitude controller	$\text{diag}(0.09, 0.09, 0.18)$

3.3 Deep Reinforcement Learning for Collision Avoidance

As the low-level control for path following is defined, the high-level control for collision avoidance must be derived. This is done by training an agent with DRL, using Stable Baselines3 (SB3) [29], a Python library with RL algorithms based on OpenAI Baselines [17]. The agent aims to learn local path planning around obstacles obstructing the path. As detailed in Section 2.5, the PPO algorithm has been shown to surpass other RL algorithms in control problems. Thus, PPO2 from SB3 is used in this thesis. The hyperparameters for the PPO2 algorithm are summarized in Table 3.3.

Table 3.3: Hyperparameters for PPO2.

Parameter	Description	Value
μ	Learning rate	$2.5 \cdot 10^{-4}$
γ	Discount factor	0.99
ϵ	Clipping parameter	0.2
c_{vf}	Value function coefficient for the loss calculation	0.5
c_{ent}	Entropy coefficient for the loss calculation	0.001
λ_{PPO}	Factor for trade-off of bias vs variance	0.95
T	Number of steps during training for each environment	1,024
T_{tot}	Number of total steps during training	65,000,000
t_{max}	Maximum amount of steps before episode ends	60,000
N_{MB}	Number of training minibatches per update	4
N_A	Number of parallel actors	8

The structure of the agent is summarized in Figure 3.3. The purpose of the policy network is to generate an action that maximizes the reward function based on the observations. All implementation details are explained further in this section.

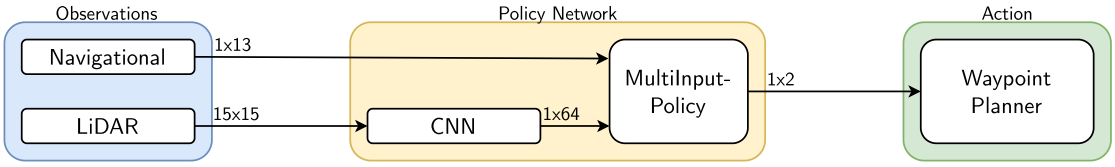


Figure 3.3: Structure of the DRL agent. The observation vector is divided in two, with both navigational observations and observations from a 360-degree LiDAR sensor. The policy network processes the observations, which outputs an action to plan the next waypoint.

3.3.1 Action Space

The action vector is the DRL agent's output, defining the quadcopter's next waypoint. The next waypoint is determined to be placed on a plane perpendicular to the original path at a certain lookahead distance, Δ, down the path. Placing the next waypoint a distance Δ down the path ensures that the quadcopter moves in the correct direction along the path. The agent takes an action every tenth time step of the quadcopter, meaning that the quadcopter is simulated for 0.1 s for each new path. A new path is not generated at each time step, as this is computationally heavy. Recalculating the path at even lower rates could also be problematic, as this gives infrequent updates to the policy network.

At each point on the path, there are defined three perpendicular vectors: the unit tangent vector, $\hat{\boldsymbol{t}}_T^w$, the unit normal vector, $\hat{\boldsymbol{n}}_T^w$, and the unit binormal vector, $\hat{\boldsymbol{b}}_T^w$. The unit tangent vector is the vector of the gradient at the point, scaled to size one. It is calculated by inserting Equation (2.27) into Equation (2.25) and dividing by the norm. Likewise, the unit normal vector is the vector defined as the scaled second-order derivative of the path, calculated by inserting Equation (2.28) into Equation (2.25) and dividing by the norm. Finally, the unit binormal vector is defined as the cross product between the tangent vector and normal vector, like $\hat{\boldsymbol{b}}_T^w = \hat{\boldsymbol{t}}_T^w \times \hat{\boldsymbol{n}}_T^w$.

$\boldsymbol{p}^w_{T,\Delta}$ denotes the point on the path a distance Δ further down the path from the closest point on the path from the quadcopter. The action vector

$$\boldsymbol{a} = \begin{bmatrix} c_n & c_b \end{bmatrix}, \tag{3.26}$$

defines where on the plane, spanned by $\hat{\boldsymbol{n}}^w_{T,\Delta}$ and $\hat{\boldsymbol{b}}^w_{T,\Delta}$, the next waypoint should be placed. $\hat{\boldsymbol{n}}^w_{T,\Delta}$ and $\hat{\boldsymbol{b}}^w_{T,\Delta}$ are defined to be the normal and binormal vectors, respectively, at point $\boldsymbol{p}^w_{T,\Delta}$ on the path. Then, the next waypoint in the alternative path for the quadcopter to follow is given by

$$\boldsymbol{w} = \boldsymbol{p}^w_{T,\Delta} + c_n \hat{\boldsymbol{n}}^w_{T,\Delta} + c_b \hat{\boldsymbol{b}}^w_{T,\Delta}. \tag{3.27}$$

By letting the DRL agent learn c_n and c_b, it should be able to avoid crashing into obstacles. In the absence of obstacles, c_n and c_b should be small to place the next waypoint of the alternative path on the original path. In the presence of obstacles, the agent should assign values to c_n and c_b such that the alternative path followed by the path following controller avoids the obstacle. The plane where the quadcopter can place the next waypoint within is illustrated in Figure 3.4.

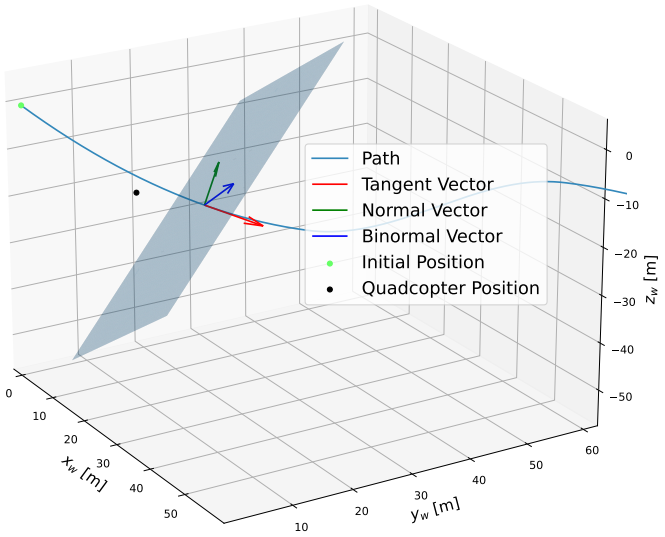


Figure 3.4: Visualization of how the agent places the next waypoint. In the figure, the tangent vector, $\boldsymbol{t}_{T,\Delta}$, normal vector, $\boldsymbol{n}_{T,\Delta}$, and binormal vector, $\boldsymbol{b}_{T,\Delta}$ of the path at the lookahead point $\boldsymbol{p}_{T,\Delta}$ are illustrated. Note that the vectors are scaled for visibility. The agent can place the next waypoint on the plane spanned by the normal vector and binormal vector, illustrated in the figure. The waypoint's exact position on the plane is determined by adjusting the size of c_n and c_b.

3.3.2 Observation Space

The observation space of the DRL agent consists of the navigational observations, $\boldsymbol{o}_n$, and the LiDAR observations, $\boldsymbol{o}_l$. The complete observation space is then given as

$$\boldsymbol{o} = \begin{bmatrix} \boldsymbol{o}_n & \boldsymbol{o}_l \end{bmatrix}, \tag{3.28}$$

and all related parameters are summarized in Table 3.4.

The navigational features for the DRL agent are all given in body frame. As the actions of the agent are directly related to the normal and binormal vectors at the point $\boldsymbol{p}^w_{T,\Delta}$, these vectors are observed as $\hat{\boldsymbol{n}}^b_{T,\Delta} = \boldsymbol{R}(\boldsymbol{\Theta}_{bw})\hat{\boldsymbol{n}}^w_{T,\Delta}$ and $\hat{\boldsymbol{b}}^b_{T,\Delta} = \boldsymbol{R}(\boldsymbol{\Theta}_{bw})\hat{\boldsymbol{b}}^w_{T,\Delta}$. In addition, the vector from the quadcopter's position to the lookahead point is observed, given by $\boldsymbol{p}^b_{\Delta} = \boldsymbol{R}(\boldsymbol{\Theta}_{bw})(\boldsymbol{p}^w_{T,\Delta} - \boldsymbol{p}^w_{wb})$. These enteties can be observed in Figure 3.4, where $\hat{\boldsymbol{n}}^b_{T,\Delta}$ is the normal vector, $\hat{\boldsymbol{b}}^b_{T,\Delta}$ is the binormal vector, and $\boldsymbol{p}^b_{\Delta}$ is the vector from the quadcopter's position to the base of $\hat{\boldsymbol{n}}^b_{T,\Delta}$ and $\hat{\boldsymbol{b}}^b_{T,\Delta}$. This yields the navigational observation space as

$$\boldsymbol{o}_n = \begin{bmatrix} \boldsymbol{\Theta}_{wb} & \hat{\boldsymbol{n}}^b_{T,\Delta} & \hat{\boldsymbol{b}}^b_{T,\Delta} & \boldsymbol{p}^b_{\Delta} \end{bmatrix}, \tag{3.29}$$

where $\boldsymbol{\Theta}_{wb}$ is the quadcopters rotation relative to world frame. The orientation of the quadcopter is included in the observations as the action are scalars of vectors in world frame, while all observations are in body frame.

The LiDAR observations are a 15×15 grid of rangefinder sensors simulating LiDAR measurements. The rangefinder sensors are employed with a 360-degree field of view with equal spacing, assuming there are no blind spots. Obstacles are found with a line search along each sensor, with a maximum range of d_{max}. Each measurement is converted to a closeness-quantity, as

$$c(d_{i,j}) = \text{clip}\left(1 - \frac{d_{i,j}}{d_{max}}, 0, 1\right), \tag{3.30}$$

where $d_{i,j}$ is the i'th and j'th rangefinder measurement and d_{max} is the LiDAR range.

Table 3.4: Parameters for the observation space of the DRL agent.

Parameter	Description	Value
Δ	Lookahead distance	5 m
d_{max}	LiDAR range	25 m
-	Sensor suite	15×15

3.3.3 Policy Network

The policy network should maximize the expected reward by finding the best actions from observations. By training the agent, the parameters of the policy network are changed to find the optimal policy $\pi_\theta(a_t|o_t)$. As seen in Figure 3.3, the policy network is divided into two parts: a CNN and *MultiInputPolicy*.

CNNs are designed to extract spatial information from location-based data. As the LiDAR measurements have this attribute, a CNN extracts features from the data. The complete CNN is illustrated in Figure 3.5. The network does only contain one convolutional layer before the resulting grid is flattened out. Sundøen, the writer of [71], used three convolutional layers with ReLUs for activation in-between. Initial tests showed better performance with a sparser network, which led to the decision to reduce the complexity of the CNN. Typically, a deeper CNN detects smaller features in the observed data. As keeping information related to the position and orientation of the data is important, a shallow network can be more effective. Training time is also reduced when the network contains fewer parameters. These results are also supported by [40], where CNNs with depth one, three, and four convolutional layers were tested for path following and collision avoidance for an autonomous marine surface vessel. The results showed that the CNN with one convolutional layer performed best.

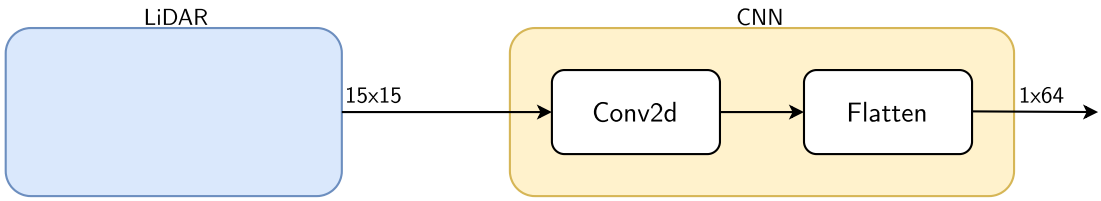


Figure 3.5: Illustration of the CNN feature extraction. The network does only consist of one convolutional layer and a flatten function.

The hyperparameters of the convolutional layer are presented in Table 3.5. Since only one convolutional layer is used, the input and output are limited to just one channel. A kernel size of five and both padding and stride equal to two give an outputted feature map of size 8×8. This can be calculated with Equation (2.29). Circular padding is used to encapsulate the sphere shape of the LiDAR measurements. In the horizontal axis, the left side is padded with the rightmost part of the measurements, and vice versa. Likewise, the upper left part of the image is padded with the upper right part of the image, and vice versa. This resolves the problem of discontinuity of the measurements along the edges. How the kernel processes the LiDAR measurements is visualized in Figure 3.6.

Table 3.5: Hyperparameters for the convolutional layer in the policy network.

Parameter	Value
in_channels	1
out_channels	1
kernel_size	5
stride	2
padding	2
padding_mode	circular

The navigational observations and the preprocessed LiDAR features are fed to *MultiInputPolicy*, a deep neural network. By using a "MultiInput"-policy, SB3 uses the *CombinedExtractor* feature extractor to convert the inputs into a single vector. This vector is then used as input to a fully-connected network that maps the features to actions and a value. Thus, the input vector is of size 1×77, and the output size is of size 1×2. The policy defines an actor-network and a value-network; in this thesis, both have a structure of three hidden layers of 128, 64, and 32 neurons, respectively. This is the same structure as Sundøen used in [71]. It is worth noticing that an extra linear layer is added on top of the specified layers to have the correct output dimensions and activation functions. Each layer uses hyperbolic tangent, $\tanh(\cdot)$, as activation functions.

3.3.4 Reward Function

The objective of the RL algorithm is to find a policy to maximize future rewards. Thus, the reward function is used to define what priorities the agent should make. To solve the dual objective of path following, there is given a reward for staying close to the path, r_t^{pf}, while closeness to obstacles, r_t^{ca}, and collisions, r_t^{c}, are penalized. The rewards are calculated at each time step, but the reward is only provided to the policy network every

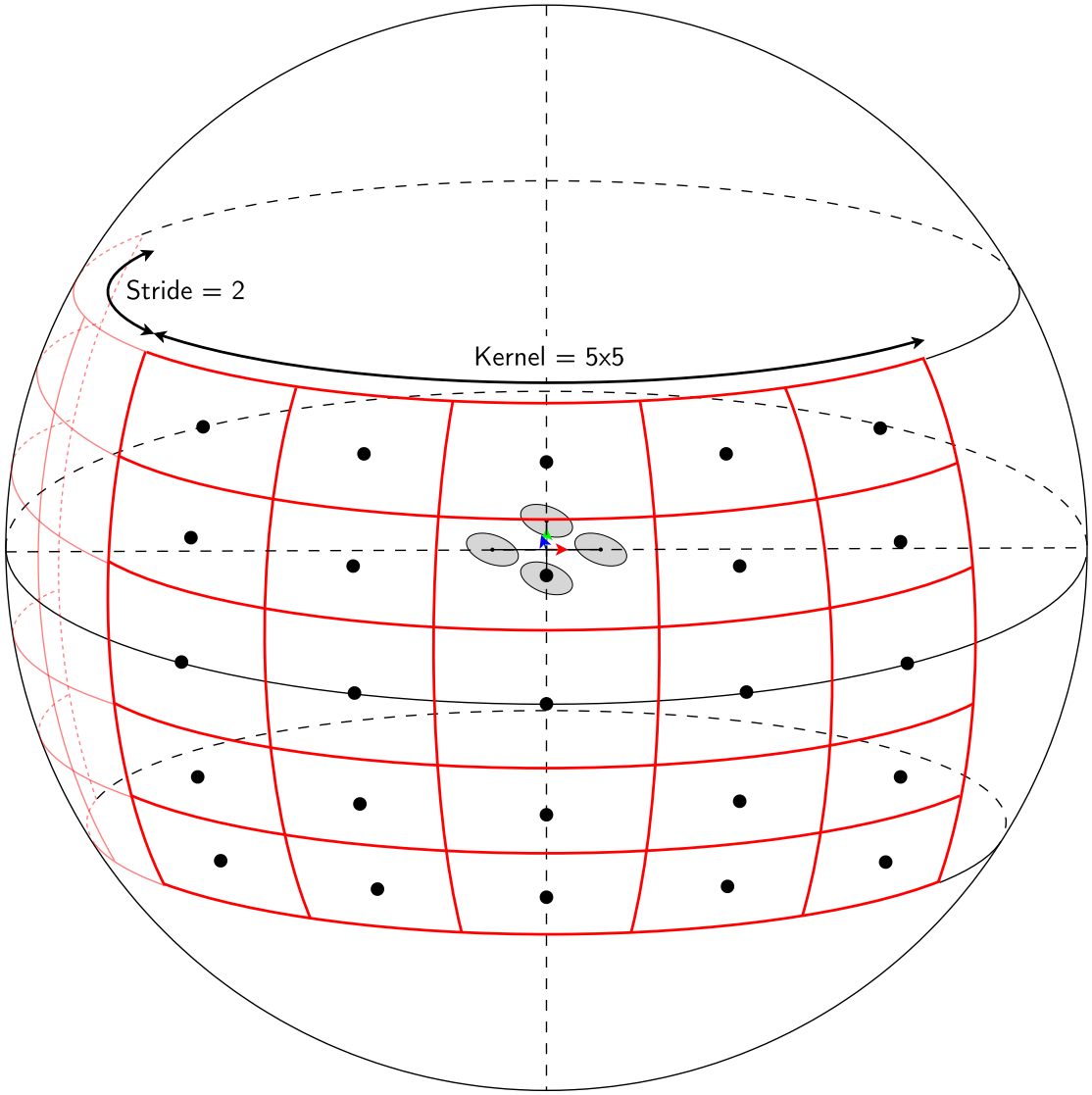


Figure 3.6: Visualization of how the convolutional filter processes the LiDAR measurements. The kernel strides across $8 \times 8 = 64$ feature activations. Circular padding ensures continuous overlap.

tenth time step when a new action is applied, and the new alternative path is calculated. Thus, the reward at time step t is given as

$$r_t = \lambda r_t^{pf} + (1-\lambda) r_t^{ca} + r_t^c, \tag{3.31}$$

and the reward given to the policy network is

$$r = \sum_{\tau=t-9}^{t} r_\tau, \tag{3.32}$$

for the past ten time steps. The weighting coefficient $\lambda \in [0,1]$ is introduced to regulate the trade-off between the competing objectives path following and collision avoidance.

The path following reward is given by

$$r_t^{pf} = \frac{\min\left(-\ln\left(\|\boldsymbol{p}_T^w - \boldsymbol{p}_{wb}^w\|\right), -\ln\left(d_{pf}\right)\right)}{-\ln\left(d_{pf}\right)}, \tag{3.33}$$

where $\|\boldsymbol{p}_T^w - \boldsymbol{p}_{wb}^w\|$ is the distance from the quadcopter to its closest point on the path, and d_{pf} is the distance from the path yielding the highest possible path following reward. The natural logarithm ensures that if the quadcopter is already far from the path, there is no need to apply a higher penalization if the quadcopter moves slightly further away. By taking the minimum of this value and $-\ln\left(d_{pf}\right)$, the maximum reward is upper-bounded when the quadcopter is within 10 cm away from the path. A maximum reward is provided as a 10 cm deviation from the path is considered insignificant.

The term for penalizing obstacle closeness is inspired by the authors of [28] and is given by

$$r_t^{ca} = -2\ln\left(1 - \gamma_{ca}\frac{\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\beta_{ca}(\theta_j,\psi_i)(\gamma_c \max((1-c(d_{i,j}))^2,\epsilon_c))^{-1}}{\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\beta_{ca}(\theta_j,\psi_i)}\right), \tag{3.34}$$

where i and j specify the sensor sector, ϵ_c is a constant to avoid singularities when obstacle closeness is one, and γ_c and γ_{ca} are scaling parameters. The obstacle closeness, $c(d_{i,j})$, defined in Equation (3.30), is used to increase the penalization as the distance to the obstacle decreases. The penalization term is written as a weighted average to allow for different sensor configurations. The vehicle-relative scaling factor is defined as

$$\beta_{ca}(\theta_j,\psi_i) = (1-2|\theta_i|/\gamma_v)(1-2|\psi_j|/\gamma_h) + \epsilon_{ca}, \tag{3.35}$$

where ϵ_{ca} is used to penalize obstacles at the edge of the configuration. This orientation factor aims to scale the reward according to the vehicle-relative orientation. Obstacle closeness likely to result in a collision should result in a lower reward than obstacles not on a collision course. Figure 3.7 illustrates how the 2D LiDAR image is weighted according to sector importance, given by β_{ca}. Obstacles right in front of the quadcopter will yield the largest penalties.

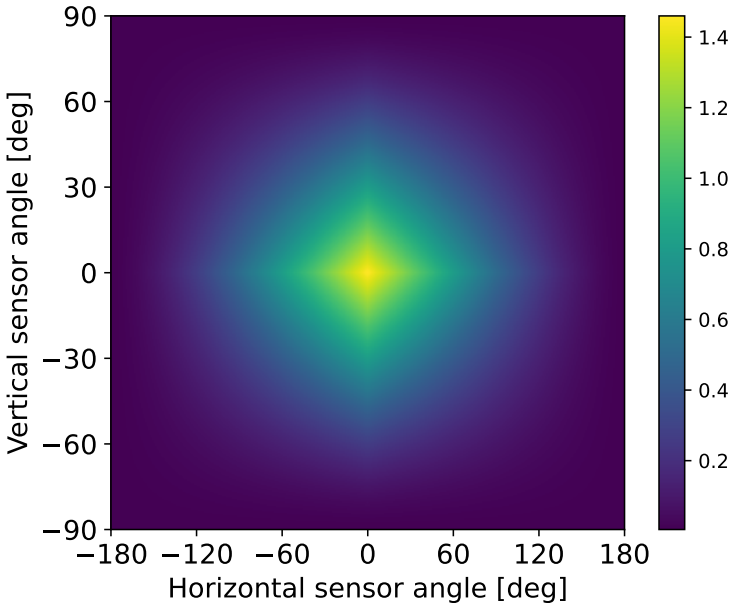


Figure 3.7: Scaling of collision avoidance reward according to where the LiDAR measurements are made, relative to the quadcopter's heading. Obstacle detection right in front of the quadcopter yields higher penalization than detections on the side or behind. Note that the grid illustrated is much finer than the 15×15 sensor suite used during the simulation.

Finally, there is provided a constant, negative reward for collisions, given by

$$r_t^c = r_{collision}. \tag{3.36}$$

Table 3.6: Parameters for reward function.

Parameter	Description	Value
λ	Weighting between path following and collision avoidance	0.5
d_{pf}	Maximum distance from path to yield highest possible path following reward	0.1 m
ϵ_c	Minimum obstacle penalty closeness	0.05
ϵ_{ca}	Minimum vehicle-relative scaling	0.05
γ_c	Obstacle closeness penalty scaling	12.5
γ_{ca}	Collision avoidance penalty scaling	-20
γ_v	LiDAR span vertical apex angle	180
γ_h	LiDAR span horizontal apex angle	360
$r_{collision}$	Negative reward for collision	-1,000
r_{min}	Minimum reward in an episode	-20,000

3.4 Training Process

The training is done with an Intel® Core™ i7-10700 CPU @ 2.90 GHz in eight parallel environments. The agent is trained for 65,000,000 time steps, making it possible to

achieve desired performance levels. Note that this means the quadcopter is simulated for 65,000,000 time steps, and the policy network is updated 6,500,000 times, as updates are made every tenth time step.

For each episode, there are formulated four termination criteria:

(a) reaches the final waypoint,

(b) collides with an obstacle,

(c) $r < r_{min}$,

(d) $t > t_{max}$.

Reaching the final waypoint of the path counts as a success. An acceptance distance is defined, $d_{accept} = 1$ m, around the final waypoint. If the quadcopter reaches this area, the episode is terminated. Collision with an obstacle will also lead to termination of the episode. In addition, $r_{min} = -20,000$ is chosen as a minimum reward for an episode, and $t_{max} = 60,000$ is the maximum number of time steps. If either of these two criteria is met, it means that the quadcopter is probably behaving very disadvantageously, and it is not worthwhile to continue the learning process.

3.5 Evaluation

To evaluate the performance, both the path following controller and DRL agent are assessed. It is possible to assess the path following controller independently from the DRL agent by creating a path without obstacles, while the DRL agent is assessed in environments both with and without obstacles. Note that the performance of the DRL agent is directly impacted by the performance of the path following controller. To conclude if combining classical and data-driven control can improve the results, the performance of the DRL agent is compared with the performance obtained by Sundøen in [71], using an end-to-end trained agent in the same gym-quad framework.

Testing scenarios ranging from *beginner* to *expert* complexity are constructed to evaluate performance in different complexities. In the *beginner* scenario, there is only a path and no obstacles. The *intermediate* level has an obstacle at the path's halfway mark. *Proficient* is the second-highest complexity level and has two additional obstacles placed stochastically around and possibly on the path. In the final *expert* level, one obstacle is guaranteed to be placed on the path, and a total of five obstacles are on or around the path. Example environments from each test level are shown in Figure 3.8. The path following controller is only tested on the *beginner* level, as it is not able to handle obstacles. The DRL agent is tested in all four complexity levels. Both are simulated for 100 episodes in each scenario to get enough test data to draw conclusions from the performance.

The DRL agent is also tested in four unseen environments to evaluate the ability to generalize and adapt to other situations. In the *helix* scenario, a spherical obstacle with a radius of 100 m is centered in origin, and the path is designed as a helix with two revolutions with a radius of 110 m around the obstacle. The *dead-end* test has the obstacles configured as a half-sphere with a radius of 15 m. Finally, the *vertical* and *horizontal* scenarios are designed with obstacles stacked vertically and horizontally. The agent was simulated for 100 episodes in each environment with a random initial position

to get statistically significant results. These unseen test environments are presented in Figure 3.9.

The distance from the quadcopter to the path is considered to evaluate the path following performance. The Average Path Error (APE) of an episode is defined as the average distance from the quadcopter to the closest point on the path over each time step until the episode is terminated. APE is calculated as

$$APE = \sum_{t} \sqrt{e_t^2 + h_t^2}, \tag{3.37}$$

where e_t and h_t are the cross-track and vertical-track errors at time t. A low APE implies that the quadcopter stays close to the path.

In the presence of obstacles, APE is not enough to evaluate the performance, as the obstacles might drive the quadcopter away from the path to avoid colliding. Thus, both failure and collision rates are used to evaluate the performance. The failure rate is defined as the number of times an episode terminated without the quadcopter reaching the final waypoint, as

$$FR = \frac{\#\ of\ failed\ episodes}{N}, \tag{3.38}$$

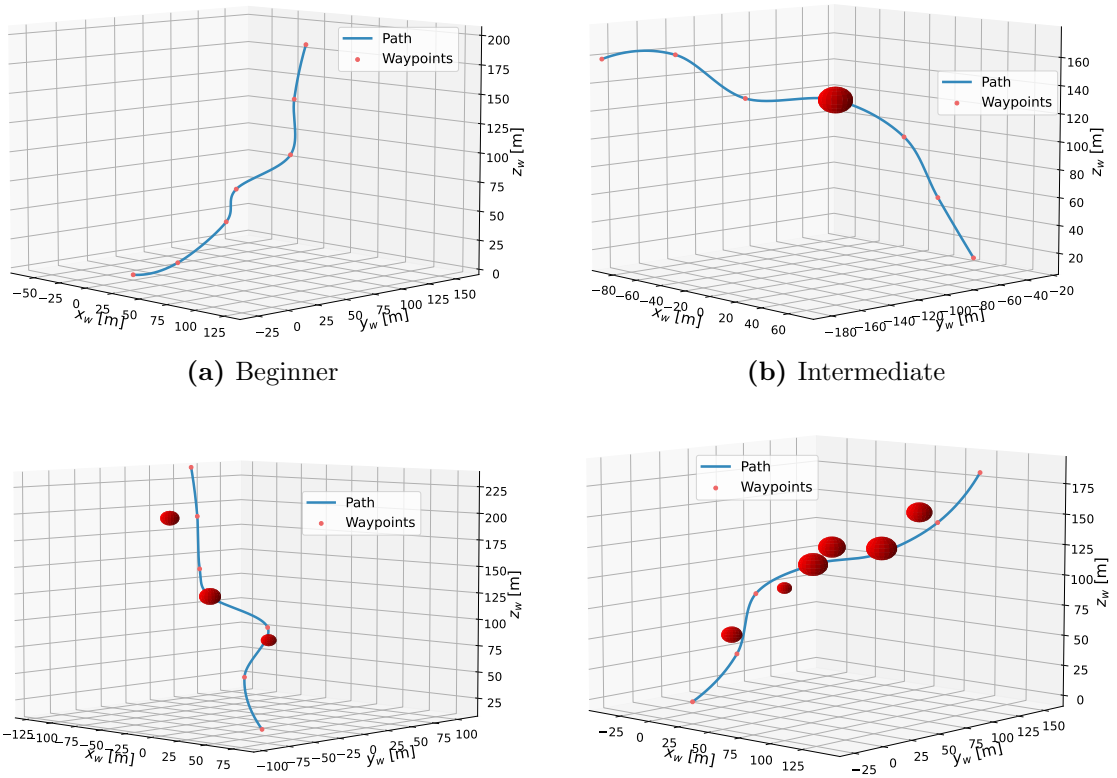
where N is the number of episodes. This happens if the quadcopter collides, the episode reward is below the minimum reward, or the quadcopter has used more than the maximum number of time steps. Likewise, the success rate is defined as

$$SR = \frac{\#\ of\ succeeded\ episodes}{N}, \tag{3.39}$$

and is the rate at which the quadcopter reaches the final waypoint. To differentiate between the quadcopter colliding and simply failing because it uses too much time, there is defined a collision rate as

$$CR = \frac{\#\ of\ collisions}{N}. \tag{3.40}$$

Finally, it is interesting to consider the progression of the quadcopter along the path. The progression is defined as how far along the path the quadcopter had reached when the episode ended. This helps differentiate if the quadcopter crashes into the first obstacle on the path or if it is able to avoid collisions for a while.



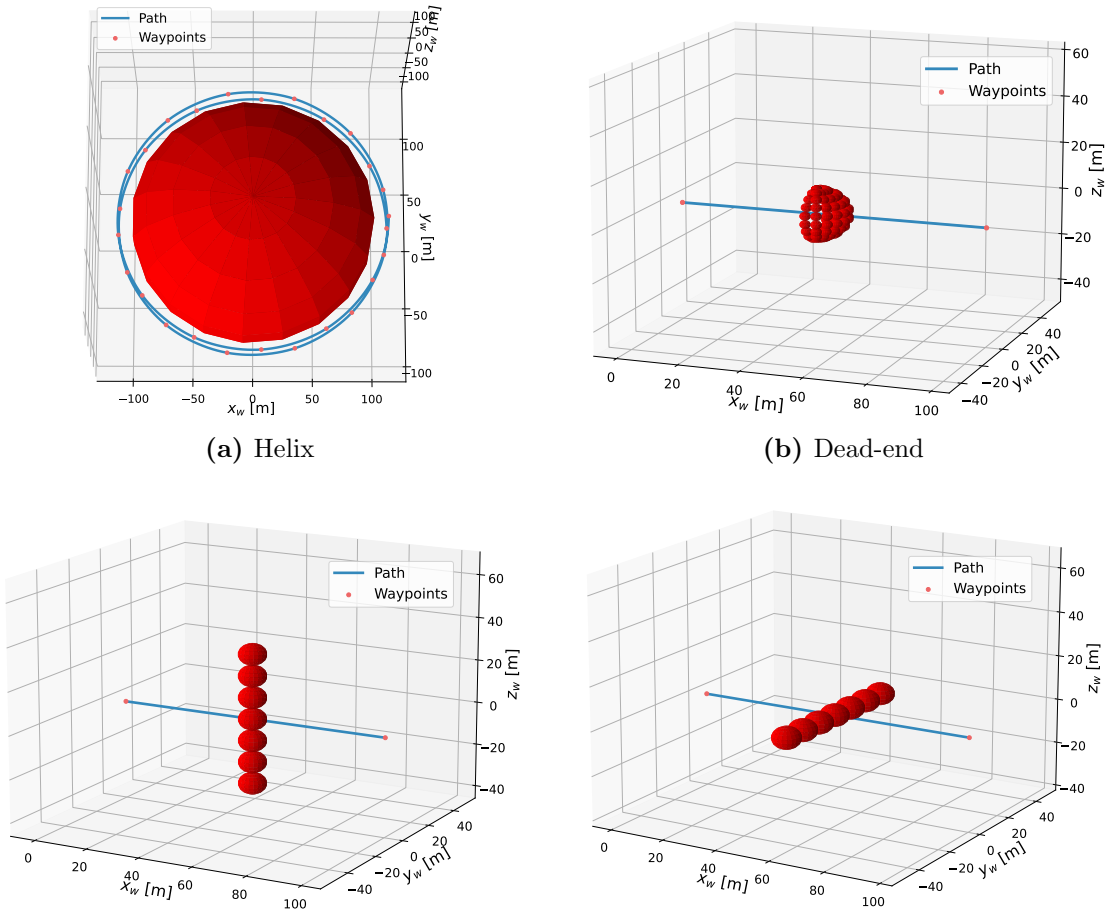
(a) Beginner

(b) Intermediate

(c) Proficient

(d) Expert

Figure 3.8: Example of test scenarios with increasing complexity, ranging from no obstacles to six obstacles.



(a) Helix **(b)** Dead-end

(c) Vertical **(d)** Horizontal

Figure 3.9: Unseen test scenarios to evaluate the DRL agent's ability to generalize to new environments.

Chapter 4

Results and Discussions

This section covers the performance of both the path following controller and DRL agent for collision avoidance. The path following controller was only tested in the *beginner* scenario, as it does not consider obstacles, and the results are presented in Section 4.1. The DRL agent for collision avoidance was tested in all four test scenarios, similar to what is encountered during training, and in the four unseen test scenarios to evaluate generalization abilities. The results are investigated in Section 4.2.

4.1 Path Following Controller

The path following controller, derived in Section 3.2, was tested in the *beginner* scenario, which is an environment without obstacles. The controller must be able to make the quadcopter follow the desired path and redirect to the path in case of deviations. Thus, there were conducted tests where the quadcopter started on the path and where the quadcopter's initial positions had random deviations from the path's starting position. When starting off the path, the initial position was given by

$$x_0^w \sim U(-10, 10), \tag{4.1a}$$
$$y_0^w \sim U(-10, 10), \tag{4.1b}$$
$$z_0^w \sim U(-10, 10), \tag{4.1c}$$

where $\sim U(-10, 10)$ represents the uniform distribution between -10 m and 10 m. The quadcopter was simulated for 100 episodes in both scenarios.

4.1.1 Quantitative Results

The quantitative results are summarized in Table 4.1.

The quadcopter was able to reach the target, both when starting on and off the path. The APE, defined as the Euclidean distance from the quadcopter to the closest point on the path, was naturally slightly higher when the quadcopter started away from the path, and this is because it will take some time to redirect to the path. However, an average APE of 5 cm when starting on the path must be considered a good performance.

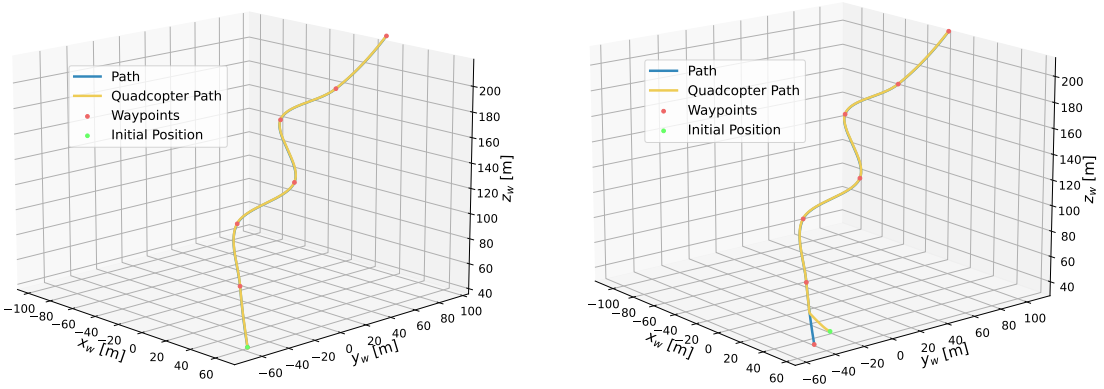
Tuning the controller gains was a trade-off between reducing the APE, reducing the time spent traversing the path, and avoiding instability. E.g., increasing $\boldsymbol{K}_{p,a}$ or $\boldsymbol{K}_{i,a}$ would improve performance concerning APE. However, the quadcopter would take longer to traverse the path and potentially become unstable. Likewise, $\boldsymbol{K}_{d,a}$ could have been increased to reduce the risk of instability, but the APE would increase. Thus, the exact performance metrics of the path following controller are not the most interesting results. The most important result is that the quadcopter stays stable in various scenarios and always stays relatively close to the path.

Table 4.1: Performance of path following controller from Section 3.2, tested over 100 episodes in the *beginner* level. In the first scenario, the quadcopter's initial position was located at the initial waypoint of the path. In the second scenario, the quadcopter started off the path to evaluate the controller's ability to handle deviations from the path.

Scenario	Success Rate [%]	Avg. Progression [%]	Avg. APE [m]
Beginner - on path	100	100	0.05
Beginner - off path	100	100	0.18

4.1.2 Qualitative Results

A visual inspection of the performance is shown in Figure 4.1, where a run in the same environment is shown with the quadcopter starting both on and off the path. When starting on the path, the quadcopter was always able to stay close and never experienced any fluctuations from the path. When the initial position of the quadcopter was not on the path, it was still able to move towards the path in the same direction as the path is heading. When coming close enough to the path, the quadcopter followed it without any problems.



(a) Quadcopter started on the path. **(b)** Quadcopter started off the path.

Figure 4.1: Sample tests of the path following controller in the *beginner* scenario, staring both on and off the path.

4.2 Deep Reinforcement Learning Agent

To evaluate the performance concerning collision avoidance, the DRL agent was tested over 100 episodes in each of the four testing scenarios described in Section 3.5: *beginner*, *intermediate*, *proficient* and *expert*. The agent was also simulated 100 times in each unseen test scenario to evaluate generalization abilities.

4.2.1 Quantitative Results

The quantitative results are summarized in Table 4.2.

It can be seen that the APE in the *beginner* scenario was significantly higher than when only the path following controller was used. A bit higher APE is expected, as the agent calculates an alternative path based on a lookahead point 5 m ahead. This means that the new path may not perfectly align with the original path between the current position of the quadcopter and the lookahead point. However, as the actual path does not contain many aggressive turns, an increase of 3 m is too high for this to be the only explanation. Thus, the DRL agent has not thoroughly learned that the action vector in such a case should be close to $\boldsymbol{a} = [0, 0]$ to stay near the path. There could be several reasons for this, e.g., that the agent is never trained in an environment entirely without obstacles. Thus, it might be that it was not exposed to enough training without obstacles to learn perfect behavior. Another reason might come from the path following reward, as it is a dead zone 10 cm away from the path, where no larger reward is given. However, this dead zone of 10 cm should not lead to an APE of 3.05 m. The reasons for this high APE will be further investigated in the qualitative analysis.

In each scenario, the success rate and collision rate explain about 90% of the episodes. This means that around 10% of the episodes were ended by the quadcopter deviating from the path and using too much time. Thus, it seems this is happening independently of the obstacles and is a weakness of the path following abilities of the controller. This supports the hypothesis that the DRL agent has a potential for improvement with respect to path following in the absence of obstacles.

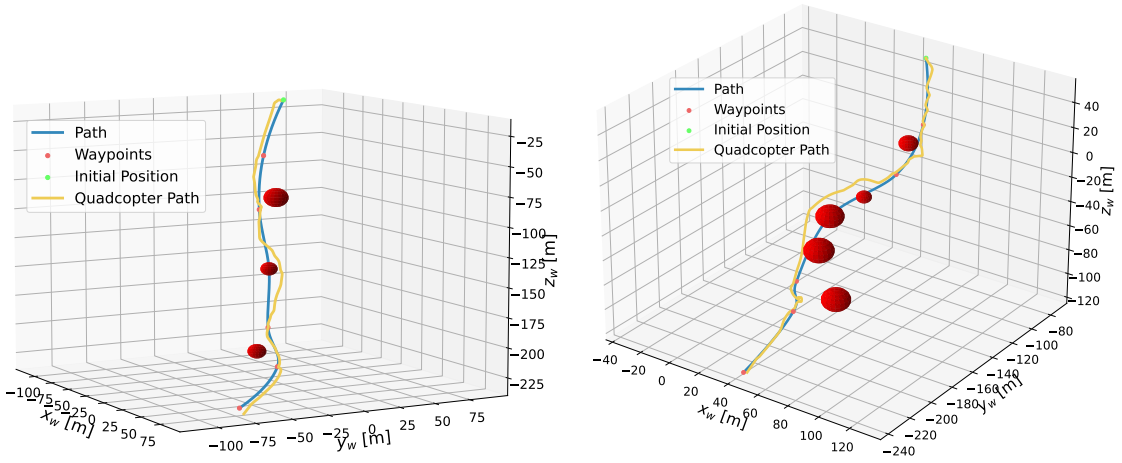
The collision rate increased as the complexity level of the scenarios increased, as expected. The quadcopter performed quite well with a collision rate of only 18% in the *intermediate* level. These results show the potential of combining classical control with data-driven control and are significantly better than the performance of the end-to-end trained agent, developed by the authors of [71], using the same gym-quad framework. However, as the obstacles came closer, it had more difficulties avoiding collisions. The collision rate increased to 44% and 57% for the *proficient* level and *expert* level, respectively. Thus, it does imply that basic collision avoidance was learned, but the agent experienced difficulties when encountering several obstacles.

Table 4.2: Performance of DRL agent on test scenarios, each simulated for 100 episodes.

Scenario	Success Rate [%]	Collision Rate [%]	Avg. Progression [%]	Avg. APE [m]
Beginner	91	N/A	94	3.05
Intermediate	72	18	83	3.57
Proficient	44	44	66	3.03
Expert	31	57	59	3.70

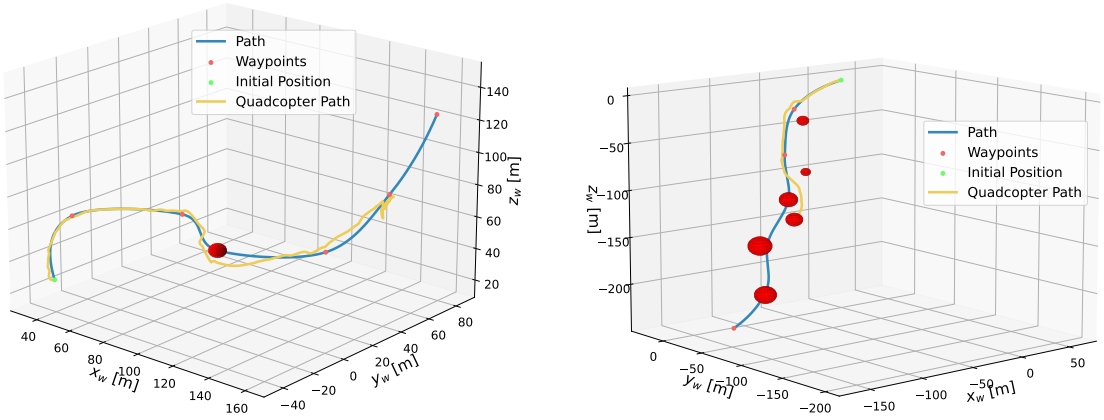
4.2.2 Qualitative Results

To get a deeper insight into the performance of the DRL agent, this section presents plots illustrating its behavior in various scenarios.



(a) Successful example from *proficient* scenario. **(b)** Successful example from *expert* scenario.

Figure 4.2: Successful runs of the DRL agent in the *proficient* scenario and *expert* scenario.



(a) Failed example from *intermediate* scenario. **(b)** Failed example from *expert* scenario.

Figure 4.3: Failed runs of the DRL agent in the *intermediate* scenario and *expert* scenario.

Figure 4.2 depicts two arbitrary runs where the agent reached the final waypoint. In the *proficient* scenario, two obstacles were placed off the path and one on the path. The quadcopter stayed close to the path in the absence of obstacles and when passing the two obstacles next to the path. When encountering the obstacle on the path, it performed an evasive maneuver to avoid colliding. This scenario showcases a desired behavior. The quadcopter was also able to navigate the *expert* scenario well, avoiding the three obstacles that are obstructing the path. After passing the obstacles, the quadcopter's trajectory converges to the path.

In Figure 4.3, there are displayed failed runs in the *intermediate* and *expert* scenarios. Figure 4.3a shows how the quadcopter passed the obstacle before it failed around the 6th waypoint. The agent appears to become unstable, causing the quadcopter to move uncontrollably until the time limit was met. This illustrates what is happening in about 10% of the episodes, regardless of the presence of obstacles. It is unclear why the DRL agent suddenly gets problems following the path. One reason might be that training on path following was not comprehensive enough and that training scenarios without obstacles should have been included. The collision in Figure 4.3b shows the agent's challenge when the density of obstacles increases. Trying to avoid one obstacle, it crashed into another in the middle of the maneuver. This indicates that the agent struggles with handling LiDAR inputs from several obstacles in the CNN.

In all these scenarios, it is observed that, even in the absence of obstacles, the quadcopter could not follow the path perfectly. The trajectory usually had some slight deviations from the path, even though it was pretty close. This was also reflected through the quantitative results, where the APE was 3.05 m in the *beginner* scenario. A possible solution could be to reduce the lookahead distance from 5 m, to allow the generated path to coincide better with the original path. Another solution could have been to have a steeper decrease in reward when the quadcopter deviates from the path. Such a change would also influence the tuning of the collision avoidance reward.

4.2.3 Unseen Scenarios

The DRL agent was also tested for 100 episodes in each of the four unseen scenarios to evaluate if the agent could generalize to new scenarios. To generate statistically significant results, the quadcopter's initial positions had random deviations from the path's starting position. The deviations from the path's initial position were given by

$$x \sim U(-5, 5), \tag{4.2a}$$
$$y \sim U(-5, 5), \tag{4.2b}$$
$$z \sim U(-5, 5), \tag{4.2c}$$

where $\sim U(-5, 5)$ represents the uniform distribution between -5 m and 5 m. It is important to stress that these scenarios deviate significantly from those encountered during training. In the *helix* scenario, the obstacle was larger than the agent had seen before. In the *dead-end*, *vertical*, and *horizontal* scenarios, the obstacles were placed more closely than ever in training and with new structures. The quantitative results from the simulations are summarized in Table 4.3 and plots illustrating a typical behavior in each environment are given in Figure 4.4.

Table 4.3: Performance of DRL agent on the unseen test scenarios, each simulated for 100 episodes.

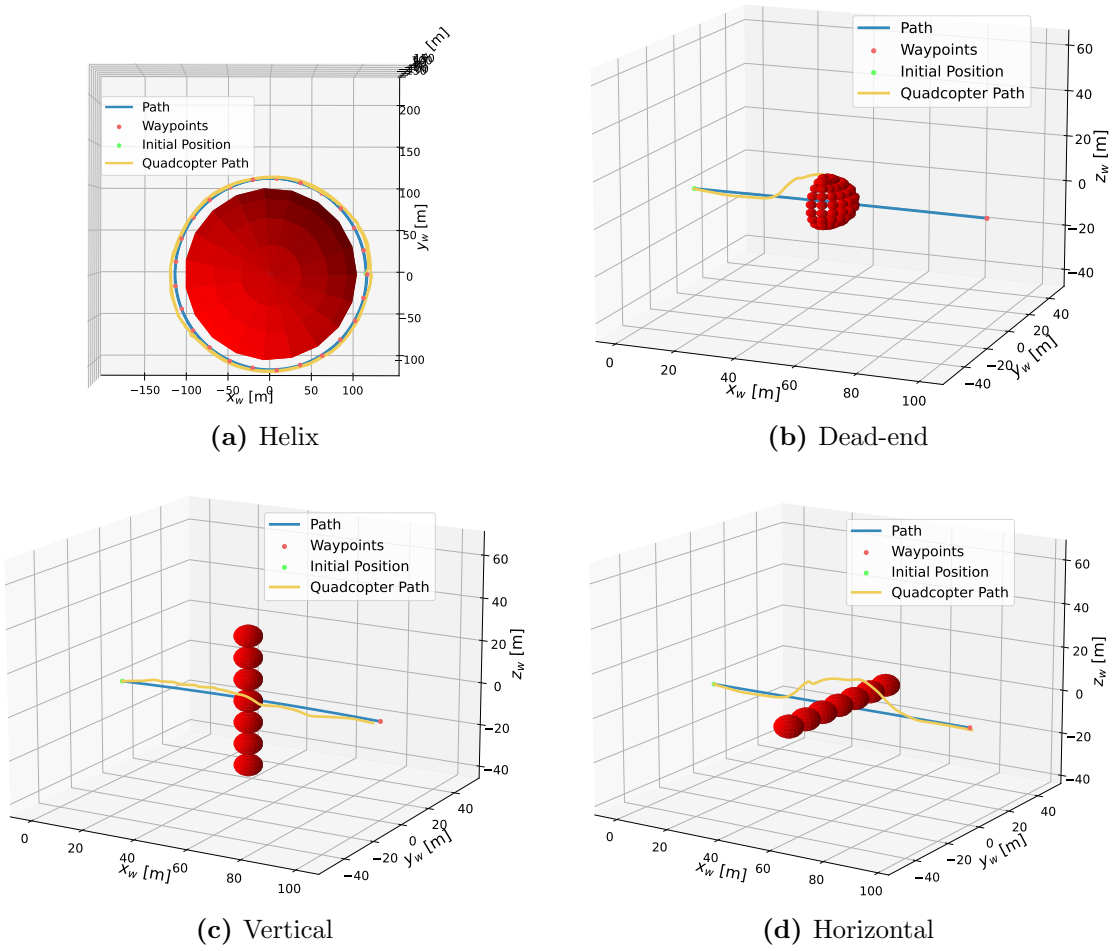
Scenario	Success Rate [%]	Collision Rate [%]	Avg. Progression [%]	Avg. APE [m]
Helix	100	0	100	6.81
Dead-end	20	80	55	4.30
Vertical	88	0	88	4.82
Horizontal	100	0	100	4.40

In the *helix* scenario, the quadcopter reached the final waypoint in every simulation. The sample test in Figure 4.4a shows that the quadcopter followed the path quite easily and never got too close to the obstacle. The waypoints are placed approximately 10 m from the obstacle, meaning the LiDAR with a range of 25 m continuously detects the obstacle to its left side. This illustrates that the agent is able to differentiate between where the obstacle is located relative to its heading. This is consistent with the collision avoidance reward, which penalizes obstacle detection based on the LiDAR-data's quadcopter-relative direction.

Being trapped in local minima is a problem often seen in reactive algorithms, which are methods processing real-time sensor data to make decisions [18]. In practice, this means being trapped in a dead-end. The purpose of the *dead-end* scenario was to investigate if the agent had acquired the intelligence to solve such a challenge. This scenario proved the most difficult for the agent, only succeeding in 20% of the episodes. Seen in Figure 4.4b, the quadcopter did initially stay close to the path, and when approaching the obstacles, it started the evasive maneuver. However, in 80% of the episodes it crashed into one of the obstacles. It did clearly detect the obstacles, but the unseen structure might have been why it could not succeed. Another reason why it crashed so often, could be that all obstacles encountered during training had a maximum radius of 10 m, meaning the quadcopter should rarely be further away from the path than 10 m. As the half-sphere has a radius of 15 m, it might have been too difficult for the agent to avoid collision as it was not trained to stay so far away from the path. It is, however, worth noticing that the quadcopter did not get trapped in most episodes, and tried to perform an evasive maneuver.

Testing in the *vertical* and *horizontal* scenarios evaluated the agent's ability to choose a suitable way around the obstacles based on their pose. The extreme cases have all the obstacles placed horizontally or vertically, and the agent should not try to take the long way around either. The agent performed well, never colliding in either scenario, meaning that the agent was able to generalize. In both Figure 4.4c and Figure 4.4d, the quadcopter passed the obstacles on the lateral side of the stacking direction, which is how an intelligent controller should behave. After passing the obstacles, the quadcopter converged to the path in both scenarios. It is worth noticing that the agent used too long time in 12% of the episodes in the *vertical* scenario, which is consistent with the behavior in the other test scenarios.

In general, it does appear that the agent has been able to generalize to unseen environments. Although the quadcopter did often crash in the *dead-end* scenario, it was not trapped as a reactive algorithm easily could have been. The other three scenarios were all navigated intelligently. As with the other test scenarios, it can be observed that the quadcopter has potential for improvement concerning path following in the absence of obstacles since the APE always was of significance.



(a) Helix **(b)** Dead-end

(c) Vertical **(d)** Horizontal

Figure 4.4: Typical behavior of the quadcopter in the unseen test scenarios. The agent successfully navigated the *helix*, *vertical*, and *horizontal* scenarios in most of the episodes, but did often collide in the *dead-end* scenario.

Chapter 5

Conclusion and Further Work

This chapter provides a conclusion on the research in terms of how the research questions are answered, hence realizing the research objectives. In addition, suggestions for further work are provided.

5.1 Conclusion

The primary research objective of this thesis was to obtain a fitting abstraction between low-level classical control and high-level Reinforcement Learning (RL) to solve the dual objective of path following and collision avoidance for a quadcopter. To achieve this, a geometrical path following controller was derived and implemented. Using Deep Reinforcement Learning (DRL), an agent was trained to intelligently perform local navigation around obstacles discovered with a simulated LiDAR. A Convolutional Neural Network (CNN) with a spherical suite was used for dimensionality reduction of the LiDAR measurements. To encourage the correct behavior of the quadcopter, a reward function was developed to penalize obstacle closeness and collisions while rewarding path closeness. The DRL agent was trained in synthetic and stochastically generated environments using the Proximal Policy Optimization (PPO) algorithm, before it was ultimately tested in scenarios of increasing complexity and in scenarios never encountered before.

The main conclusions of the thesis are:

- The path following controller performs excellently standalone and has proven to function as a control abstraction for the DRL agent. By abstracting away the path following problem and having the DRL agent only decide the next waypoint, the quadcopter stays relatively close to the desired path and is able to follow the path in 91% of the episodes in the *beginner* scenario. These results show the potential of combining classical control with data-driven control and are significantly better than the performance of the end-to-end trained agent, developed by the authors of [71], using the same gym-quad framework.
- By succeeding in 72% of episodes in the *intermediate* scenario, the DRL agent showed it has learned to perform evasive maneuvers for collision avoidance. This is supported by plots, such as in Figure 4.2. The quadcopter stays close to the

path until it approaches an obstacle, where it sacrifices path adherence to avoid collision before converging to the path again.

- The agent proves generalization abilities by performing well in unseen test scenarios with a considerable domain gap to the training scenarios, both in terms of path shape and obstacle configuration. Intelligently avoiding collision in three of the scenarios, the agent does not rely on predefined rules and heuristics. The *dead-end* scenario proved the most difficult, probably due to a lack of training scenarios requiring the quadcopter to move far away from the path.

5.2 Further Work

The DRL agent relies on a CNN-based perception, assuming the LiDAR measurements are a 2D grid. This is probably an unrealistic assumption, as the detected points in reality are 3D points. By considering the measurements as a point cloud, deep net architectures designed for 3D geometric data could be utilized and possibly improve collision avoidance performance. An example of such a network is the PointNet [56], which has shown excellent performance within segmentation and classification of point clouds.

The action space of the DRL agent restricts it to always place the next waypoint a certain lookahead distance down the path. This implies that the quadcopter cannot turn back and fly in the opposite direction of the path or hover still over the ground. Such behavior could be desirable in urban environments. It might be necessary to turn around if the quadcopter tries to navigate through a corridor that turns out to be blocked or hovering over the ground as the quadcopter waits for another vehicle to pass. To solve this problem, the action space could be modified to use sphere coordinates to determine the next waypoint. The dimensionality of the action space would have been increased to three, as two angles and the radial distance are required. Note that this would demand a change in reward function to stimulate the quadcopter to move along the path and not only to the path's closest point. Such reward functions have already been designed in similar control problems [28, 48, 71].

Although the DRL approach has shown notable benefits, a disadvantage is that the agents act as black-box models offering little insight into the learned mappings. Safety guarantees will probably become a prerequisite if DRL-based controllers for quadcopters are to be used in urban environments. Thus, measures should be taken to understand the underlying reasoning better. One approach could be using symbolic regression to find value-functions based on state transitions, as the authors of [38] did.

Even if great results were obtained from training in simulation, there would have been required training directly in the real world. This is called transfer learning and is used to apply the learned behavior in a related task, such as operations in the physical world. As RL is used for high-level model-free control, the need for transfer learning is probably reduced compared to an end-to-end trained agent. This is because the low-level controller can account for physical effects such as wind gusts. However, other effects, such as noisy LiDAR measurements, could still cause problems for the agent. Thus, some transfer learning should be applied. This could cause some challenges, for example, the possibility of collisions and breaking the quadcopter. The training process will also be slow if only one quadcopter is used. Therefore, performing most of the training in synthetic environments is desirable.

Bibliography

[1] Alrifaee, B., Kostyszyn, K., and Abel, D. Model predictive control for collision avoidance of networked vehicles using lagrangian relaxation. *14th IFAC Symposium on Control in Transportation SystemsCTS 2016 49*, 3 (Aug 2016), 430–435. https://www.sciencedirect.com/science/article/pii/S2405896316302683.

[2] Ambrosino, G., Ariola, M., Ciniglio, U., Corraro, F., De Lellis, E., and Pironti, A. Path generation and tracking in 3-d for uavs. *IEEE Transactions on Control Systems Technology 17*, 4 (May 2009), 980–988. https://ieeexplore.ieee.org/document/4908914.

[3] Amidi, O., and Thorpe, C. E. Integrated mobile robot control. In *Mobile Robots V* (Mar 1991), W. H. Chun and W. J. Wolfe, Eds., vol. 1388 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pp. 504–523. https://ui.adsabs.harvard.edu/abs/1991SPIE.1388..504A.

[4] Andersson, O., Wzorek, M., Rudol, P., and Doherty, P. Model-predictive control with stochastic collision avoidance using bayesian policy optimization. In *2016 IEEE International Conference on Robotics and Automation (ICRA)* (May 2016), pp. 4597–4604. https://ieeexplore.ieee.org/document/7487661.

[5] Balchen, J. G., Andresen, T., and Foss, B. A. *Reguleringsteknikk*, sixth ed. Institutt for teknisk kybernetikk, NTNU, Trondheim, Norway, 2016. https://folk.ntnu.no/tronda/regtek-kurs/bok-reguleringsteknikk.pdf.

[6] Bohn, C., and Atherton, D. An analysis package comparing pid anti-windup strategies. *IEEE Control Systems Magazine 15*, 2 (Apr 1995), 34–40. https://ieeexplore.ieee.org/document/375281.

[7] Boivin, E., Desbiens, A., and Gagnon, E. Uav collision avoidance using cooperative predictive control. In *2008 16th Mediterranean Conference on Control and Automation* (Jun 2008), pp. 682–688. https://ieeexplore.ieee.org/document/4602109.

[8] Byrnes, C., and Isidori, A. Asymptotic stabilization of minimum phase nonlinear systems. *IEEE Transactions on Automatic Control 36*, 10 (Oct 1991), 1122–1137. https://ieeexplore.ieee.org/document/90226.

[9] Cabecinhas, D., Cunha, R., and Silvestre, C. A globally stabilizing path following controller for rotorcraft with wind disturbance rejection. *IEEE Transactions*

on Control Systems Technology 23, 2 (Jun 2015), 708–714. https://ieeexplore.ieee.org/document/6837464.

[10] Carlsen, Ø. Introducing tracking vessel obstacles when training autonomous surface vehicles. *Unpublished*, 2022.

[11] Carlsen, Ø. Path following controller and drl-based collision avoidance for quadcopter. https://github.com/orjanic/gym_quad, 2023.

[12] Chakravarthy, A., and Ghose, D. Obstacle avoidance in a dynamic environment: a collision cone approach. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 28*, 5 (Sep 1998), 562–574. https://ieeexplore.ieee.org/document/709600.

[13] Chang, S.-R., and Huh, U.-Y. A collision-free g2 continuous path-smoothing algorithm using quadratic polynomial interpolation. *International Journal of Advanced Robotic Systems 11*, 12 (Dec 2014), 194. https://doi.org/10.5772/59463.

[14] Chang, S.-R., and Huh, U.-Y. Curvature-continuous 3d path-planning using qpmi method. *International Journal of Advanced Robotic Systems 12*, 6 (Jan 2015), 76. https://doi.org/10.5772/60718.

[15] Choi, D., Lee, K., and Kim, D. Enhanced potential field-based collision avoidance for unmanned aerial vehicles in a dynamic environment. *AIAA Scitech 2020 Forum* (Jan 2020). https://arc.aiaa.org/doi/10.2514/6.2020-0487.

[16] Clark, R. A., Punzo, G., MacLeod, C. N., Dobie, G., Summan, R., Bolton, G., Pierce, S. G., and Macdonald, M. Autonomous and scalable control for remote inspection with multiple aerial vehicles. *Robotics and Autonomous Systems 87* (Jan 2017), 258–268. https://www.sciencedirect.com/science/article/pii/S0921889016301579.

[17] Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. Openai baselines. https://github.com/openai/baselines, 2017.

[18] Eriksen, B.-O. H., Breivik, M., Pettersen, K. Y., and Wiig, M. S. A modified dynamic window algorithm for horizontal collision avoidance for auvs. In *2016 IEEE Conference on Control Applications (CCA)* (Sep 2016), pp. 499–506.

[19] Euchi, J. Do drones have a realistic place in a pandemic fight for delivering medical supplies in healthcare systems problems? *Chinese Journal of Aeronautics 34*, 2 (Feb 2021), 182–190. https://www.sciencedirect.com/science/article/pii/S100093612030279X.

[20] Faessler, M., Franchi, A., and Scaramuzza, D. Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories. *IEEE Robotics and Automation Letters 3*, 2 (Apr 2018), 620–626. https://doi.org/10.1109%2Flra.2017.2776353.

[21] Fossen, T. I. *Handbook of Marine Craft Hydrodynamics and Motion Control*, second ed. John Wiley & Sons, Trondheim, Norway, 2021. https://www.wiley.com/en-us/Handbook+of+Marine+Craft+Hydrodynamics+and+Motion+Control%2C+2nd+Edition-p-9781119575054.

[22] François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., and Pineau, J. *An Introduction to Deep Reinforcement Learning*, vol. 11. Now Publishers, 2018. https://doi.org/10.1561%2F2200000071.

[23] Ghosh, A., Sufian, A., Sultana, F., Chakrabarti, A., and De, D. *Fundamental Concepts of Convolutional Neural Network*. Springer International Publishing, Cham, 2020, pp. 519–567. https://doi.org/10.1007/978-3-030-32644-9_36.

[24] Goss, J., Rajvanshi, R., and Subbarao, K. Aircraft conflict detection and resolution using mixed geometric and collision cone approaches. *AIAA Guidance, Navigation, and Control Conference and Exhibit* (Aug 2004). https://arc.aiaa.org/doi/10.2514/6.2004-4879.

[25] Guan, Y., Li, S. E., Duan, J., Li, J., Ren, Y. Sun, Q., and Cheng, B. Direct and indirect reinforcement learning. https://doi.org/10.48550/arXiv.1912.10600, 2021.

[26] Ha, L. N. N. T., Bui, D. H. P., and Hong, S. K. Nonlinear control for autonomous trajectory tracking while considering collision avoidance of uavs based on geometric relations. *Energies 12*, 8 (Apr 2019). https://www.mdpi.com/1996-1073/12/8/1551.

[27] Haque, M., Muhammad, M., Swarnaker, D., and Arifuzzaman, M. Autonomous quadcopter for product home delivery. In *2014 International Conference on Electrical Engineering and Information & Communication Technology* (Apr 2014), IEEE, pp. 1–5. https://ieeexplore.ieee.org/abstract/document/6919154.

[28] Havenstrøm, S. T., Rasheed, A., and San, O. Deep reinforcement learning controller for 3d path following and collision avoidance by autonomous underwater vehicles. *Frontiers in Robotics and AI 7* (Jan 2021). https://www.frontiersin.org/articles/10.3389/frobt.2020.566037.

[29] Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[30] Hsiao, F.-I., Chiang, C.-M., and Hou, A. Continuous control with deep reinforcement learning. https://aa228.stanford.edu/old-projects/, 2019.

[31] Huh, U.-Y., and Chang, S.-R. A g2 continuous path-smoothing algorithm using modified quadratic polynomial interpolation. *International Journal of Advanced Robotic Systems 11*, 2 (Jan 2014), 25. https://doi.org/10.5772/57340.

[32] Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M. Control of a quadrotor with reinforcement learning. *CoRR abs/1707.05110* (Jul 2017). http://arxiv.org/abs/1707.05110.

[33] Kaufmann, E., Bauersfeld, L., and Scaramuzza, D. A benchmark comparison of learned control policies for agile quadrotor flight, 2022. https://arxiv.org/abs/2202.10796.

[34] Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation* (Mar 1985), vol. 2, pp. 500–505. https://ieeexplore.ieee.org/document/1087247.

[35] Kim, J., Kang, M.-S., and Park, S. *Accurate Modeling and Robust Hovering Control for a Quad-rotor VTOL Aircraft*. Springer Netherlands, Dordrecht, 2010, pp. 9–26. https://doi.org/10.1007/978-90-481-8764-5_2.

[36] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations* (Dec 2014). https://arxiv.org/abs/1412.6980.

[37] Koch, W., Mancuso, R., West, R., and Bestavros, A. Reinforcement learning for uav attitude control. https://arxiv.org/abs/1804.04154, 2018.

[38] Kubalík, J., Zegklitz, J., Derner, E., and Babuska, R. Symbolic regression methods for reinforcement learning. *CoRR abs/1903.09688* (Mar 2019). https://arxiv.org/abs/1903.09688.

[39] Kukreti, S., Kumar, M., and Cohen, K. Genetically tuned lqr based path following for uavs under wind disturbance. *2016 International Conference on Unmanned Aircraft Systems (ICUAS)* (Jun 2016), 267–274. https://ieeexplore.ieee.org/document/7502620.

[40] Larsen, T. N., Hansen, H., and Rasheed, A. Risk-based convolutional perception models for collision avoidance in autonomous marine surface vessels using deep reinforcement learning. *Unpublished*, 2023.

[41] Larsen, T. N., Teigen, H. Ø., Laache, T., Varagnolo, D., and Rasheed, A. Comparing deep reinforcement learning algorithms' ability to safely navigate challenging waters. *Frontiers in Robotics and AI 8* (Sep 2021). https://www.frontiersin.org/articles/10.3389/frobt.2021.738113.

[42] Levine, W. S. *The Control Handbook*, second ed. CRC Press, Boca Raton, USA, 1996. https://www.routledge.com/The-Control-Handbook-Control-System-Applications-Second-Edition/Levine/p/book/9781420073607.

[43] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *CoRR* (Sep 2015). https://arxiv.org/abs/1509.02971.

[44] Mamo, M. Trajectory control of quadcopter by designing second order smc controller. *Journal of Electrical Engineering 7* (Jan 2021), 10. https://jeeeccs.net/index.php/journal/article/view/175/156.

[45] Manjunath, A., Mehrok, P., Sharma, R., and Ratnoo, A. Application of virtual target based guidance laws to path following of a quadrotor uav. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)* (Jun 2016), IEEE, pp. 252–260. https://ieeexplore.ieee.org/document/7502565.

[46] Mellinger, D., and Kumar, V. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation* (May 2011), pp. 2520–2525. https://ieeexplore.ieee.org/document/5980409.

[47] Mellinger, D. W. Trajectory generation and control for quadrotors. *Publicly Accessible Penn Dissertations* (Jan 2012). https://repository.upenn.edu/edissertations/547/.

[48] Meyer, E., Robinson, H., Rasheed, A., and O., S. Taming an autonomous surface vehicle for path following and collision avoidance using deep reinforcement learning. *IEEE Access 8* (Feb 2020), 41466–41481. https://ieeexplore.ieee.org/document/9016254.

[49] Micaelli, A., and Samson, C. Trajectory tracking for two-steering-wheels mobile robots. *IFAC Proceedings Volumes 27*, 14 (Sep 1994), 249–256. Fourth IFAC Symposium on Robot Control, Capri, Italy, https://www.sciencedirect.com/science/article/pii/S1474667017473228.

[50] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. *CoRR abs/1602.01783* (Jun 2016). https://arxiv.org/abs/1602.01783.

[51] Niermeyer, P., Akkinapalli, V. S., Pak, M., Holzapfel, F., and Lohmann, B. Geometric path following control for multirotor vehicles using nonlinear model predictive control and 3d spline paths. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)* (Jun 2016), pp. 126–134. https://ieeexplore.ieee.org/document/7502541.

[52] Nugraha, A., and Agustinah, T. Quadcopter path following control design using output feedback with command generator tracker los based at square path. *Journal of Physics: Conference Series 947* (Jan 2018), 012074. https://iopscience.iop.org/article/10.1088/1742-6596/947/1/012074/pdf.

[53] Panomrattanarug, B., Higuchi, K., and Mora-Camino, F. Attitude control of a quadrotor aircraft using lqr state feedback controller with full order state observer. In *The SICE Annual Conference 2013* (Sep 2013), pp. 2041–2046. https://ieeexplore.ieee.org/document/6736320.

[54] Plaat, A. *Deep Reinforcement Learning*, first ed. Springer Nature Singapore, Leiden, The Netherlands, 2022. https://doi.org/10.1007%2F978-981-19-0638-1.

[55] Pérez-Carabaza, S., Scherer, J., Rinner, B., López-Orozco, J. A., and Besada-Portas, E. Uav trajectory optimization for minimum time search with communication constraints and collision avoidance. *Engineering Applications of Artificial Intelligence 85* (Oct 2019), 357–371. https://www.sciencedirect.com/science/article/pii/S0952197619301411.

[56] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR abs/1612.00593* (Dec 2016). https://arxiv.org/abs/1612.00593.

[57] Raffo, G. V., Ortega, M. G., and Rubio, F. R. Backstepping/nonlinear h∞ control for path tracking of a quadrotor unmanned aerial vehicle. In *2008 American Control Conference* (Jun 2008), pp. 3356–3361. https://ieeexplore.ieee.org/document/4587010.

[58] Ratnoo, A., Hayoun, S., Granot, A., and Shima, T. Path following using trajectory shaping guidance. *Journal of Guidance, Control, and Dynamics 38* (Aug 2013). https://arc.aiaa.org/doi/full/10.2514/1.G000300.

[59] Richards, C., and Turner, M. Combined static and dynamic anti-windup compensation for quadcopters experiencing large disturbances. *Journal of Guidance, Control, and Dynamics 43* (Feb 2020), 1–12. https://arc.aiaa.org/doi/10.2514/1.G004575.

[60] Roza, A., and Maggiore, M. Path following controller for a quadrotor helicopter. In *2012 American Control Conference (ACC)* (Jun 2012), pp. 4655–4660. https://ieeexplore.ieee.org/document/6315061.

[61] Rubí, B., Pérez, R., and Morcego, B. A survey of path following control strategies for uavs focused on quadrotors. *Journal of Intelligent & Robotic Systems 98* (May 2020), 241–265. https://doi.org/10.1007/s10846-019-01085-z.

[62] Sanchez, I., D'Jorge, A., Ferramosca, A., Raffo, G., and Gonzlez, A. H. Path following and trajectory tracking model predictive control using artificial variables for constrained vehicles. In *2019 XVIII Workshop on Information Processing and Control (RPIC)* (Sep 2019), IEEE, pp. 198–203. https://ieeexplore.ieee.org/document/8882189.

[63] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization. *CoRR abs/1502.05477* (Feb 2015). https://arxiv.org/abs/1502.05477.

[64] Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Jun 2016), Y. Bengio and Y. LeCun, Eds. https://arxiv.org/abs/1506.02438.

[65] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR abs/1707.06347* (Jul 2017). https://arxiv.org/abs/1707.06347.

[66] Society of Naval Architects and Marine Engineers (U.S.). Technical and Research Committee. Hydrodynamics Subcommittee. *Nomenclature for Treating the Motion of a Submerged Body Through a Fluid: Report of the American Towing Tank Conference*. Technical and research bulletin. Society of Naval Architects and Marine Engineers, 1950. https://books.google.no/books?id=VqNFGwAACAAJ.

[67] Stevšić, S., Nägeli, T., Alonso-Mora, J., and Hilliges, O. Sample efficient learning of path following and obstacle avoidance behavior for quadrotors. *IEEE Robotics and Automation Letters 3*, 4 (Jul 2018), 3852–3859. https://ieeexplore.ieee.org/document/8412596.

[68] Sultana, F., Sufian, A., and Dutta, P. Advancements in image classification using convolutional neural network. In *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)* (Nov 2018), IEEE. https://doi.org/10.1109%2Ficrcicn.2018.8718718.

[69] Sultana, F., Sufian, A., and Dutta, P. A review of object detection models based on convolutional neural network. In *Advances in Intelligent Systems and Computing*. Springer Singapore, 2020, pp. 1–16. https://doi.org/10.1007%2F978-981-15-4288-6_1.

[70] Sun, J., Tang, J., and Lao, S. Collision avoidance for cooperative uavs with optimized artificial potential field algorithm. *IEEE Access 5* (Aug 2017), 18382–18390. https://ieeexplore.ieee.org/abstract/document/8022685.

[71] Sundøen, L. L. Path following and collision avoidance for quadcopters using deep reinforcement learning. https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3022408, 2022.

[72] Sutton, R. S., and Barto, A. G. *Reinforcement Learning: An Introduction*, second ed. The MIT Press, London, England, 2018. https://mitpress.mit.edu/9780262039246/reinforcement-learning/.

[73] Sutton, R. S., Mcallester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12* (Feb 2000), vol. 12, MIT Press, pp. 1057–1063. https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

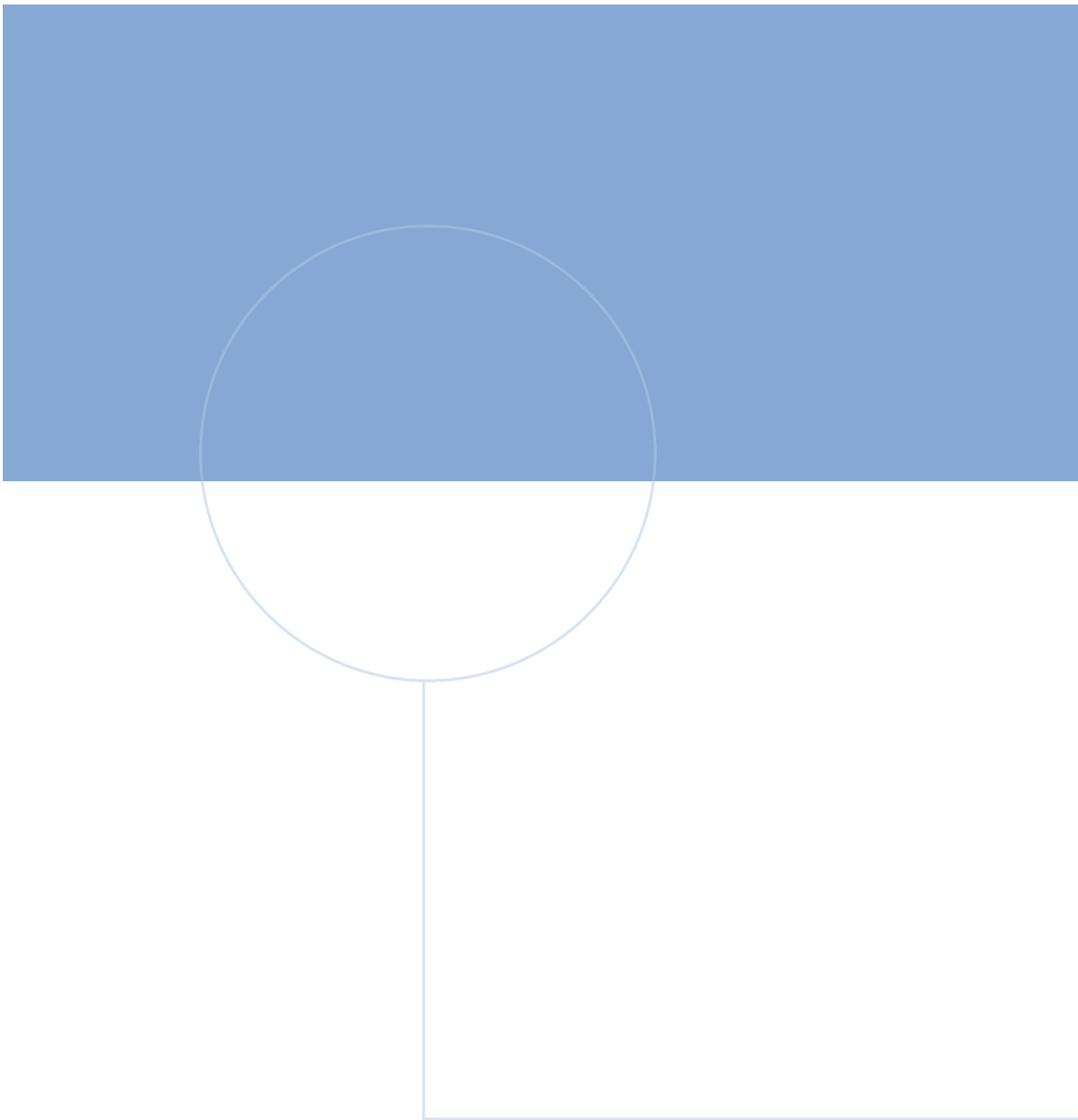
[74] Van Loock, W., Pipeleers, G., Diehl, M., De Schutter, J., and Swevers, J. Optimal path following for differentially flat robotic systems through a geometric problem formulation. *IEEE Transactions on Robotics 30*, 4 (Mar 2014), 980–985. https://ieeexplore.ieee.org/document/6757008.

[75] Xiang, X., Wang, Z., Mo, Z., Chen, G., Pham, K., and Blasch, E. Wind field estimation through autonomous quadcopter avionics. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)* (Sep 2016), pp. 1–6. https://ieeexplore.ieee.org/document/7778071.

[76] Yasin, J. N., Mohamed, S. A. S., Haghbayan, M.-H., Heikkonen, J., Tenhunen, H., and Plosila, J. Unmanned aerial vehicles (uavs): Collision avoidance systems and approaches. *IEEE Access 8* (Jun 2020), 105139–105155. https://ieeexplore.ieee.org/document/9108245.

[77] Zaitoun, N. M., and Aqel, M. J. Survey on image segmentation techniques. In *International Conference on Communications, management, and Information technology (ICCMIT'2015)* (Oct 2015), vol. 65, pp. 797–806. https://www.sciencedirect.com/science/article/pii/S1877050915028574.

[78] Zhaowei, M., Tianjiang, H., Lincheng, S., Weiwei, K., Boxin, Z., and Kaidi, Y. An iterative learning controller for quadrotor uav path following at a constant altitude. In *2015 34th Chinese Control Conference (CCC)* (Jul 2015), IEEE, pp. 4406–4411. https://ieeexplore.ieee.org/document/7260322.



NTNU

Norwegian University of
Science and Technology