Master's thesis

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

Ludvig Løken Sundøen

Path Following and Collision Avoidance for Quadcopters using Deep Reinforcement Learning

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
Co-supervisor: Thomas Nakken Larsen
June 2022

NTNU
Norwegian University of
Science and Technology

Ludvig Løken Sundøen

Path Following and Collision Avoidance for Quadcopters using Deep Reinforcement Learning

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
Co-supervisor: Thomas Nakken Larsen
June 2022

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

NTNU
Kunnskap for en bedre verden

Abstract

Classical control methods depend on accurate models. Such models may not exist for complex systems, and control is limited to simple low-level tasks. Model-free Reinforcement Learning (RL) methods can find near-optimal controllers from data and experience without the need for such models and learn complex control policies with an arbitrary input abstraction.

Research in RL for continuous control has only kicked off in the last few years, especially with the rise in popularity and applicability of deep neural networks as universal function approximators. Proximal Policy Optimization (PPO) is widely accepted as the preferred RL algorithm for control problems in control applications.

This thesis applied the PPO algorithm to solve simultaneous path following and collision avoidance in a synthetic quadcopter environment, using guidance-theoretic navigation features for proprioception and spherical LIDAR for exteroception. A curriculum learning framework is implemented to stage the learning in incremental steps; the autonomous agent learns to stabilize the system, follow arbitrary paths in 3D, and avoid obstacles, incrementally. The resulting RL agent largely succeeds in path following but has difficulties avoiding obstacles. Furthermore, the resulting agent successfully generalizes to scenarios not encountered during the training phase. Future research directions are suggested to improve the agent's collision avoidance performance.

Sammendrag

Klassiske kontrollmetoder avhenger av nøyaktige modeller. Slike modeller eksisterer ikke alltid for komplekse systemer, og kontroll er begrenset til enkle oppgaver på lavt nivå.

Modellfrie Reinforcement Learning (RL) metoder kan finne nesten optimale kontrollere fra data og erfaring uten behov for slike modeller og lære komplekse kontrollpolicyer med en vilkårlig nivå av abstraksjon på inndataen.

Forskning på RL i kontinuerlig kontroll har bare startet de siste årene, spesielt med økningen i popularitet og anvendelighet av dype nevrale nettverk som universelle funksjonstilnærere. Proximal Policy Optimization (PPO) er allment akseptert som den foretrukne RL-algoritmen for kontrollproblemer i kontrollapplikasjoner.

Denne oppgaven brukte PPO-algoritmen for å løse baneføIging og kollisjonsunngåelse i en simulert kvadrorotor-applikasjon, ved å bruke styringsteoretiske navigasjonsfunksjoner for propriosepsjon og sfærisk LIDAR for eksterosepsjon. Et læringsrammeverk er implementert for å iscenesette læringen i trinnvist; den autonome agenten lærer å stabilisere systemet, følge vilkårlige veier i 3D og unngå hindringer, trinnvis. Den resulterende RL-agenten lykkes stort sett med å følge veien, men har vanskeligheter med å unngå hindringer. Videre generaliserer den resulterende agenten vellykket til scenarier som ikke oppstår under treningsfasen. Fremtidige forskningsretninger foreslås for å forbedre agentens ytelse for å unngå kollisjoner.

Preface

The research presented in this thesis was done at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology, with Professor Adil Rasheed as supervisor and PhD-candidate Thomas Nakken Larsen as co-supervisor.

This thesis is a continuation of my project thesis, "Path Following for a Quadcopter using Deep Reinforcement Learning", written in the fall of 2021. The project thesis and this master's thesis build upon the DRL framework developed in Havenstrøm et al. (2021). The project thesis resulted in the paper "Towards Enabling an Autonomous Digital Twin of a Quadcopter using Deep Reinforcement Learning", with some modifications. Parts of this thesis's Introduction, Theory, and Methodology stem from the project thesis and the resulting paper.

Acknowledgments
Adil was a great supervisor. He introduced me to research and baffled me with his big ideas. His passion continues to inspire me. Thank you Adil.
Thank you Thomas. He has helped me in every way, from debugging my code to brainstorming new ideas and proofreading. This thesis would not have been possible without him.
Thank you to my mom and dad for your ever-lasting support. I am grateful that you always have believed in me.
Thank you Nadia for being a cool girlfriend. You never cease to amaze me.

Trondheim, June 2022
Ludvig Løken Sundøen

Contents

1. Introduction

The main contribution of this thesis is a Deep Reinforcement Learning (DRL) controller for path following and obstacle avoidance for a quadcopter. A DRL environment with an autonomous underwater vehicle from a prior work is modified to a quadcopter. A spherical LIDAR is simulated, and a convolutional neural network is employed for feature extraction from the LIDAR data. The results are validated in unseen environments.

1.1. Motivation and Background

Before autonomous quadcopters can be used in applications like home delivery of medical equipment (EUCHI (2021)) and in retail (Haque et al. (2014)), and inspection of dangerous and hard-to-access environments, such as power lines, oil tanks, and nuclear waste (Clark et al. (2017)), they should at least be able to follow a prescribed path and avoid collisions with obstacles. The contribution of this paper is to present a model-free approach to path following and collision avoidance.

Many use cases of quadcopters today require an operator for control. By making quadcopters autonomous, their applications may become more effective, safer, and non-reliant on human operators. Traditionally, model predictive control, sliding mode control, adaptive control, and similar methods have been applied to control autonomous vehicles (Rubi et al. (2020)). However, the performance of these methods is highly correlated with the accuracy of the manually developed model. By deploying reinforcement learning in control, we may free ourselves of manual modeling and potentially increase performance.

1.2. Classical Path Following

We define *path following* as an algorithm that controls an agent to traverse a given continuous path without any prespecified time specification. Path following differs from trajectory tracking, as trajectory tracking also requires the agent to track the path within a prespecified timeframe. The given continuous path is typically constructed by some path planning algorithm, such as Dijkstra's algorithm (Dijkstra (1959)) or A* search (Hart et al. (1968)). There has been thorough research on the path following methods in classical control literature. This section elaborates on the established techniques utilized in path following studies. One element common to all of them is the reliance on the mathematical modeling of the dynamics.

1.3. Line of Sight

Line of Sight (LoS) path following is often a component of more sophisticated path following methods. The method seeks the vehicle to approach the path at a set lookahead distance Δ down the path, shown in Figure 1.1. Fossen et al. (2003) used LoS to control

an underactuated marine craft with two lower-level controllers derived using nonlinear control theory. They achieved excellent performance in a case study on a model boat.

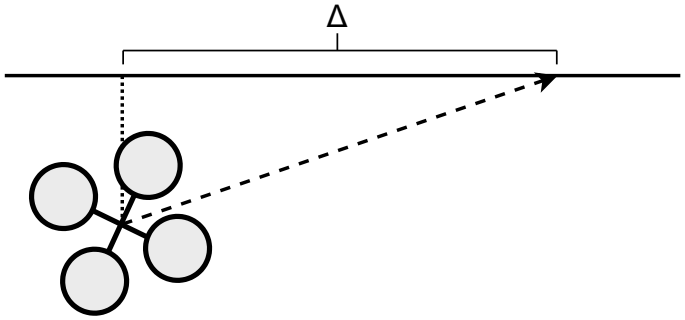


Figure 1.1.: Visualization of the LoS guidance method. The thick black line is the path, while the pointed dotted line is the proposed heading. The algorithms seek the vehicle to approach the path at a set lookahead distance Δ down the path.

Model Predictive Control

Model Predictive Control (MPC) traces back to the 1960s (Morari and H. Lee (1999)). The MPC approach is to transform a control problem into an optimization problem. MPCs calculate the optimal control sequence for several timesteps ahead, repeated at every timestep, leading to a high computational cost. The advantages of MPCs are that they plan for the coming timesteps and that energy usage can be minimized. However, as in the case with other classical methods, the controller is dependent on the modeling of the dynamics. MPCs often have to solve a multi-objective optimization problem with non-arbitrary weights. Faulwasser (2013) introduced MPC for path following problems. Yu et al. (2012) demonstrated a nonlinear MPC path following method that outperformed a nonlinear controller in their simulations. It guaranteed convergence to a reference path, even with input disturbances.

Sliding Mode Control

Sliding Mode Control (SMC) attains the control objectives by constraining the system dynamics to a pre-defined surface by employing a discontinuous control law. This discontinuity leads to an unwanted chattering effect, often cited as the method's main drawback. Nevertheless, there exist methods for dealing with this (Young and Drakunov (1992)). Dagci et al. (2003) made a path following controller design using SMC for a wheeled mobile robot and experimentally showed off the algorithm's effectiveness.

Feedback linearization

Feedback linearization (Jaulin (2015)) is an approach that involves finding a transformation of the nonlinear system through a change of variables and a suitable control input into a linear system. Then linear control theory is applied to the system. The advantages of feedback linearization compared to other methods, e.g., model predictive control and sliding mode control, are the simplicity of the control structure and proof of error convergence under certain conditions. A drawback is that mathematical knowledge of the dynamics is needed. The approach is widely used in the literature and has seen use

in control applications (Roza and Maggiore (2012)). Others, such as Moe et al. (2016), reduce the path following problem to the horizontal plane. Moe et al. then applied the LoS guidance law and an adaptive feedback linearizing controller to control the vessel with only absolute velocity sensors.

Vector Field Control

Vector field control is based upon a set of vectors virtually placed around the path such that if the vehicle follows the direction, it will converge to it. One of the main drawbacks is the difficulty of implementation. Nelson et al. (2007) achieved excellent performance on the path following scenarios when the path consisted of a combination of straight lines and circles. Sujit et al. (2014) showed that vector field control could achieve lower cross-track error than other, classical, path following algorithms.

Each of the aforementioned classical control methods often used in path following carries different drawbacks, whereas the main collective challenge may be their reliance on mathematical modeling. The correctness of the modeling often correlates highly with the performance of the resulting controller. If we free ourselves from modeling, we may obtain even better-performing controllers.

1.4. Classical Collision Avoidance

We define *collision avoidance* as an algorithm that controls an agent to avoid obstacles that are locally discovered. Therefore, Collision avoidance is often referred to as local path planning. As with path following, classical collision avoidance has seen much research. In this section, classical collision avoidance methods are presented.

Model Predictive Control

In addition, to be employed in path following algorithms, MPCs are also employed for collision avoidance. Alrifaee et al. (2016) used an MPC for collision avoidance using a Lagrangian relaxation approach. They achieved excellent performance in a realistic simulated environment. Eriksen et al. (2019) used an MPC for short-term collision avoidance for a maritime vessel while adhering to the International Regulations for Preventing Collisions at Sea (COLREGS). They achieved good performance both in simulations and in full-scale experiments at sea.

Vector Field Control

Khatib (1986) introduced vector field control for collision avoidance. In contrast to the path following application, the vector field is time-varying. The method was tested in simulations of a robotic manipulator. The work of Khatib (1986) laid the groundwork for more recent research on collision avoidance in mobile robots. Borenstein and Koren (1989) expanded upon the ideas of Khatib (1986) and introduced the "virtual-force" approach. The main idea is that obstacles in the vector field exert a virtual force, pushing the agent away from the obstacle. They successfully tested the method experimentally on a mobile robot with ultrasonic sensors. A drawback of vector field control is that the agent may never reach the target if the obstacle is close enough, and an assumption of infinite control energy.

Geometric Methods

Another subfield of collision avoidance algorithms is *geometric methods*. A notable feature of these methods is that they often have a low computational cost, which makes them able to react to obstacles rapidly (Huang et al., 2019). Collision cone collision avoidance is such a method. The collision cone method is based on creating an artificial cone formed between the agent and the obstacle, as seen in Figure 1.2. Lalish and Morgansen (2009) introduces a collision cone method that seeks to steer the heading of the agent outside the cone. Fiorini and Shiller (1998) takes into mind the obstacles' velocity and defines a collision cone such that the trajectory of the agent and the trajectory of the obstacle will not collide.

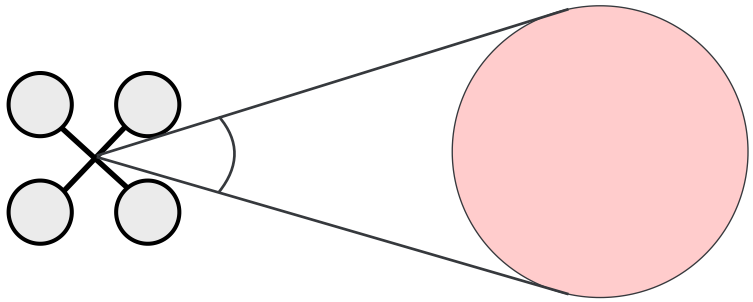


Figure 1.2.: Visualization of the collision cone collision avoidance method. The agent is to follow a heading that is not within the cone.

Dynamic Window

The *dynamic window* approach is a popular method (Fox et al., 1997). The approach reduces the set of translational and rotations velocities to the currently feasible velocities. The set of velocities is further reduced to the set that allows the vehicle to stop within a set time interval. The resulting set forms the "dynamic window". The approach searches in the dynamic window to find a set of feasible circular trajectories in a short time interval. The method chooses the trajectory that maximizes a defined reward function. Fox et al. test the method experimentally on synchronous drive mobile robot showing robust results in the AAAI'94 Mobile Robot Competition arena.

1.5. Deep Reinforcement Learning for Control

Reinforcement Learning (RL) is a general framework in which an agent aims to learn the optimal decision-making policy through sequential or episodic interaction with its environment. Unlike traditional control methods, model-free RL can be used to solve control problems without needing a dynamics model. Therefore, RL is often preferable when specifying a reward function to maximize is simpler than hand-crafting a control law. Difficulties in hand-crafting a control law could be due to complex, undefined, or high-dimensional dynamics. We will go deeper into RL in Section 2.5. RL algorithms have recently achieved superhuman performance in challenging games such as Chess, Go, and StarCraft II (Silver et al. (2017), Silver et al. (2016), AlphaStar (2019)). The driving force behind these accomplishments is the integration of deep neural networks (DNNs) for parameterizing the decision-making policy, enabling the DRL controller to make continuous or discrete actions in response to high-dimensional continuous or discrete observations. DNNs are widely used in computer vision and natural language processing.

In the later years, DRL is emerging as an attractive model-free control method for complex systems.

Q-learning

Watkins (1989) introduced Q-learning: a tabular model-free RL algorithm estimating the value of an action given a particular state. Mnih et al. (2015) introduced the deep Q-network (DQN), combining the Q-learning algorithm with powerful DNN function approximators. DQNs can play multiple Atari 2600 games at an above-average human level, even surpassing the top-level players in a few of them. They also demonstrated that the resulting algorithm could use raw pixel values as inputs to solve the tasks. Lillicrap et al. (2016) adapted deep Q-learning to the continuous action domain, establishing the Deep Deterministic Policy Gradient (DDPG) algorithm. They solved 20 simulated physics tasks, most of which were typical control tasks.

Proximal Policy Optimization

Bøhn et al. (2019) used Proximal Policy Optimization (PPO) to control a fixed-wing unmanned aerial vehicle's attitude. Then they experimented with the observation vector, adding the states of the previous timesteps. The result was a proof-of-concept controller that performed well in a simulated environment. It is yet to be seen how transferable the PPO agent's control strategies learned in simulations are to real-world conditions.

Kaufmann et al. (2022) benchmarked control policies with PPO for agile quadcopter flight. The control policies differed in their action spaces. The most high-level policy used linear velocity commands for control, while the most low-level used, as in this paper, commanded forces to each of the quadcopter's rotors. However, the middle-level policy of commanding collective thrust and body rates seemed to outperform the others. Additionally, the authors randomized the domain and tested the resulting agents in real-life settings, achieving agile flight with speed beyond $45[km/h]$.

Degrave et al. (2022) used the Maximum a Posteriori Policy Optimisation (MPO) algorithm for magnetic control of plasma in a nuclear fusion reactor. Training in a highly accurate simulation made it possible to achieve high performance and robustness to uncertain operating conditions, even on real hardware.

1.6. Deep Reinforcement Learning for Path Following

Hwangbo et al. (2017) used DRL to guide a quadcopter in a waypoint tracking problem, in contrast to path following, where the agent would follow a continuous path. They introduced a deterministic, on-policy method that is more conservative but more stable than similar methods, and it directly maps the observation vector to motor thrust. Moreover, they showed that their controller is transferable to the real world. Pi et al. (2021) considered a disturbance observer to estimate the external forces exerted on the quadcopter, which is added to the observation vector, resulting in a system more robust to external forces.

Martinsen and Lekkas (2018a) employed the DDPG method for straight path following for an underactuated marine vessel. They also added the influence of an unknown ocean current. Martinsen and Lekkas (2018b) used transfer learning for curved path following. The agent attained faster training than the one directly trained on curved paths, achieving indistinguishable performance on three vessel models.

1.7. Deep Reinforcement Learning for Path Following and Collision Avoidance

Path following and collision avoidance are heavily coupled, and the controller needs to weigh the goal of following the path against the risk of hitting obstacles. Therefore, there have been several attempts to do path following and collision avoidance conjoined.

Meyer et al. (2020) used PPO for path following and collision avoidance for an underactuated marine surface vessel. Obstacles were detected using rangefinder sensors, and the processed ranges were a part of the observation space. The results demonstrated that RL is a viable approach to the dual objective problem of path following and collision avoidance. Larsen et al. (2021) used the framework of Meyer et al. (2020) to compare how PPO performs up against other DRL methods, Soft Actor-Critic, DDPG, and Twin Delayed DDPG. They concluded that PPO seemed to have the highest performance in the environment. Havenstrøm et al. (2021) built upon Meyer et al. (2020), extending it to 3D for an autonomous underwater vessel. The agent showed great potential, having a tracking error of less than $0.5m$, even in scenarios with currents.

Batra et al. (2021) demonstrated the use of a multi-agent PPO learning quadcopter swarm control with collision avoidance. As a requirement for swarm control, the quadcopters were rewarded for avoiding each other and reaching their goal waypoint. They trained in a simulated environment incorporating both sensor noise and collision effects, resulting in a swarm controller capable of flight in a real-world controlled environment.

1.8. Research Objectives

Path following and collision avoidance for a quadcopter using DRL is a complex task. However, if solved, it shows the potential for DRL in control and could pave the way for further research. The primary research objective of this paper is to develop a path following and collision avoidance method for quadcopters using DRL. A secondary research objective is to investigate whether CNNs could be used for dimensionality reduction of LIDAR observations. Another secondary research objective is to make the controller generalize to an unseen environment, validating the results.

1.9. Research Questions

To achieve our research objectives, we formulate the three research questions:

- Will the resulting DRL controller with the PPO hyperparameters from Havenstrøm et al. (2021) and Sundøen et al. (2021) to achieve path following and collision avoidance?
- Can we exploit the dimensionality-reducing property in convolutional neural networks for encoding LIDAR observations?
- Will the resulting DRL controller be able to perform in unseen environments?

1.10. Thesis Structure

This paper is divided into five main sections. In the current section, Section 1, we introduced the path following and collision avoidance problem, along with state-of-the-art methods. Section 2 introduces the quadcopter model, the path planning method, the defined guidance law, and neural networks. We also explain the basics of RL and go deeper into PPO. In Section 3, we describe the environment setup and implementation details. In Section 4, we show and discuss our results, while in Section 5, we conclude our findings and discuss further works.

2. Theory

Although we consider a model-free approach, the current DRL algorithms are generally not suitable for training directly on hardware due to their poor sample efficiency and unpredictable black-box properties. Therefore, this section establishes the reference frames, transformations, dynamics models, and path generation tools necessary for setting up a framework suitable for training a path following and collision avoidance DRL controller in a simulation.

2.1. The Quadcopter Model

2.1.1. Reference Frames

The state variables are given in two reference frames, the North East Down (NED) frame and the body frame. Both frames are drawn in Figure 2.1, where the NED frame is denoted n and the body frame denoted b. The NED frame is a tangent plane defined relative to the earth's reference ellipsoid, where X_n and Y_n point towards true north and true east, respectively. Thus, Z_n is orthogonal to X_n and Y_n and points towards the earth's core. Navigation in the NED frame is commonly referred to as "flat earth navigation". The NED frame is considered suitable for navigation within a small deviation in latitude and longitude, whereas vehicles crossing over larger distances require other reference frames. Quadcopters usually move within a small geographic area, thus making the NED frame a suitable choice. We also consider the "body" frame, which moves with the quadcopter. For quadcopters, the origin is generally chosen as the center of mass because it simplifies the equations of motion and makes control more intuitive. The unit vector X_b points longitudinal to the vehicle, while Y_b points transversely. Z_b is orthogonal to X_b and Y_b. Thus, we have established the necessary reference frames to control a vehicle within a small geographic region.

Table 2.1.: Notation of velocities, angular rates, positions and angles in NED- and body-frame.

Degree of Freedom	Velocity and Angular Rate	Position and Angle
1 Motion in the X-direction (Surge)	u	X_n
2 Motion in the Y-direction (Sway)	v	Y_n
3 Motion in the Z-direction (Heave)	w	Z_n
4 Rotation about the X-direction (Roll)	p	ϕ
5 Rotation about the Y-direction (Pitch)	q	θ
6 Rotation about the Z-direction (Yaw)	r	ψ

2.1.2. Kinematic Equations

Some elements, such as the propeller forces, operate relative to the body frame, while the path is given in the NED frame. Therefore, there is a need to find a transformation between the frames. An overview of the state variables can be found in Table 2.1,

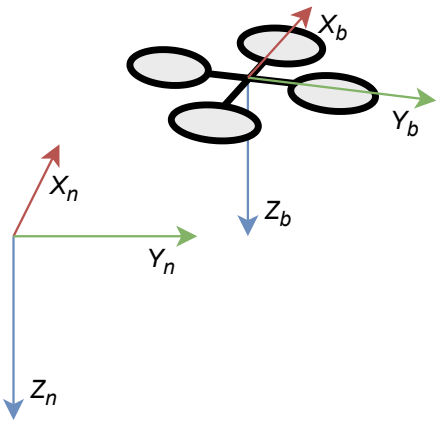


Figure 2.1.: The reference frames NED and body. NED is a tangent plane defined relative to the earth's reference ellipsoid, where X_n and Y_n are axis pointing towards true north and true east, respectively, while Z_n is orthogonal to X_n and Y_n and pointing down towards the earth. Body-frame is a frame fixed in the center of mass of the quadcopter, where the axis X_b and Y_b points longitudinal and transversely, respectively. Z_b is orthogonal to X_b and Y_b.

following SNAME-notation SNAME (1950). The quadrotor's kinematics state vector is represented in $\eta = [p_n, \Theta_{nb}]^T = [x_n, y_n, z_n, \phi_{nb}, \theta_{nb}, \psi_{nb}]^T$, where x_n, y_n, and z_n are the positions in NED-frame, while ϕ_{nb},θ_{nb},ψ_{nb} are the yaw, pitch, and roll angles of the body-frame relative to NED-frame. p_n is the positional coordinate vector, $p_n = [x_n, y_n, z_n]^T$, while v_b is the translational velocity vector, $v_b = [u, v, w]$, where u, v, and w are the velocities along the body-frame in X_b, Y_b, and Z_b respectively. $\omega^w_{b/n} = [p, q, r]^T$, where p, q, and r are the rotational velocities around X_b, Y_b, and Z_b. The velocity vector is defined as $\nu = [v_b, \omega^w_{b/n}]^T$. The rotation matrix between the linear velocities in NED-frame and body-frame is then given by Equation 2.1.

$$R^n_b(\Theta_{nb}) = \begin{bmatrix} c\psi c\theta & -s\psi c\phi + c\psi s\theta s\phi & s\psi s\phi + c\psi c\phi s\theta \\ s\psi c\theta & c\psi c\theta + s\phi s\theta s\psi & -c\psi s\phi + s\theta s\psi c\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix}. \tag{2.1}$$

The inverted rotation matrix $R^b_n(\Theta_{nb})$, is defined as

$$R^b_n(\Theta_{nb}) = (R^n_b(\Theta_{nb}))^T. \tag{2.2}$$

The transformation of the rotational velocities is

$$T_{\Theta_{nb}}(\Theta_{nb}) = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix}, \tag{2.3}$$

where s is sine, c is cosine, and t is the tangent. The matrix contains singularities at pitch angles $\theta = \pm\frac{\pi}{2}$. However, in most operating modes, the quadcopter will not reach these pitch angles.

The governing differential equation describing the first-order differential equation between the velocity in the body and NED frame is

$$\dot{\eta} = J_{\Theta_{nb}}(\eta)\nu,$$

where $J_\theta(\eta)$ is

$$J_{\Theta_{nb}}(\eta) = \begin{bmatrix} R_n^b(\Theta_{nb}) & 0 \\ 0 & T_{\Theta_{nb}}(\Theta_{nb}) \end{bmatrix}.$$

2.1.3. Equations of Motion

The equations of motion describe how a vehicle behaves given its physical characteristics and the applied forces, τ. These are essential for simulating the quadcopter. We base the model setup on the work of Kim et al. (2010) while we employ the model parameters used in Mamo (2021). To simplify the control problem, we make some assumptions for the model. We assume that:

- The effect of moving air mass around the quadcopter is negligible, $M_a = 0$.
- There exists a function mapping the thrust of a propeller, F_i, directly to the needed power, W_i, of the propeller: $P_i{=}f(F_i)$. This function often exists, and it is easier to explain the behavior of the quadcopter with this simplification.
- The quadcopter is axis-symmetrical, resulting in I being a diagonal matrix.
- There is no wind, $\tau_{wind} = 0$, nor any other external forces. This assumption implies that the quadcopter is flown indoors or in calm weather.

These assumptions will likely render the DRL controller unusable in real-world applications. However, this paper only focuses on demonstrating the potential of a DRL-based controller for path following.

The equations of motions are shown in the following equation

$$M\dot{\nu} + C(\nu) + g(\eta) = Bu$$

where M is

$$M = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_x & 0 & 0 \\ 0 & 0 & 0 & 0 & I_y & 0 \\ 0 & 0 & 0 & 0 & 0 & I_z \end{bmatrix} \tag{2.4}$$

The Coriolis forces are given as

$$C(\nu) = \begin{bmatrix} 0 & 0 & 0 & (I_z - I_y)qr & (I_x - I_z)pr & 0 \end{bmatrix}^T$$

In this case, the restoring force is simply the gravitational force, as given in Equation 2.5. The gravitational force acts along Z_n, but our dynamics are modeled in body-frame, so we need to transform from NED-frame to body-frame. The gravitational force vector $g(\eta)$ is

$$g(\eta) = \begin{bmatrix} mgs\theta & -mgc\theta s\phi & mgc\theta c\phi & 0 & 0 & 0 \end{bmatrix}^T, \tag{2.5}$$

where m is the mass of the quadcopter and g is the gravitational acceleration defined in Table 2.2. The control inputs are the four rotors of the quadcopter. The inputs are

Table 2.2.: Model parameters from Kim et al. (2010)

Parameter	Value
l	$0.5m$
m	$0.5kg$
g	$9.81m/s^2$
I_x	$0.005kgm^2$
I_y	$0.005kgm^2$
I_z	$0.01kgm^2$
λ	0.08

simplified to one force F_i for each rotor. The control force τ is

$$\tau = Bu$$
$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & -l & 0 & l \\ -l & 0 & l & 0 \\ -\lambda & \lambda & -\lambda & \lambda \end{bmatrix},$$

where λ is the inflow ratio, l is the length from the origin of body to each of the rotors, and u is the vector

$$u = [F_1, F_2, F_3, F_4]^T,$$

where F_1, F_2, F_3 and F_4 are the forces exerted from the rotors. All model parameter values can be found in Table 2.2.

2.2. Continuous 3D Path Generation

In this section, we introduce a solution to 3D path generation. In real scenarios, one would instead employ a global path planning algorithm such as A* Hart et al. (1968), or Dijkstra Dijkstra (1959). We have set up the agent to detect obstacles while running rather than online while running. Therefore, the path planning consists only of finding a viable path through some randomly placed waypoints. We define the first waypoint,w_1, in the origin. The subsequent waypoints, $w_i = [x_i, y_i, z_i]$, are then defined by

$$x_i = x_{i-1} + dcos(\alpha)cos(\beta),$$
$$y_i = y_{i-1} + dsin(\alpha)cos(\beta),$$
$$z_i = z_{i-1} - dsin(\beta)$$

where $i \in [2, n]$, d is the distance between the waypoints, and α and β are instances from two uniform distributions. Parameter values for n, d, α, and β can be found in Table 3.3. The setting of the parameters α and β will determine the curvature of the path. We design a path that goes through all the waypoints while also being physically feasible to traverse.

Aerial vehicles are unable to obtain momentum from the ground. Therefore, the path generated can not have any sharp angular movements or instantaneous stops, as this

would require an infinite amount of force from the rotors (Chang and Huh (2015)). These conditions imply that the path should not have vertexes and that the velocity and acceleration should be continuous. Further implicating that the movements of the quadcopter should not have vertices and that velocity and acceleration should be continuous. We classify paths in terms of continuity, G^n, where $n \in \mathbb{N}$. A G^n continuous path is a continuity of the n^{th}-order differential values. A G^2 continuous path has continuous acceleration and, therefore, continuous velocity. Chang and Huh (2015) expand the G^2-continous quadratic polynomial interpolation (QPMI) (Huh and Chang (2014) Chang and Huh (2014)), used for smoothing a 2D path to make continuous paths in 3D and prove G^2 continuity. In the 3D version of QPMI, we use a parametric function

$$p(s) : (x(s), y(s), z(s)),$$

where s is the length along the path, which can be calculated using the piece-wise distances. $p(s)$ is a path defined for each triplet of subsequent waypoints. The three functions $x(s)$, $y(s)$, and $z(s)$ are defined as

$$\begin{aligned} x(s) &= a_{x_m} s^2 + b_{x_m} s + c_{x_m}, \\ y(s) &= a_{y_m} s^2 + b_{y_m} s + c_{y_m}, \\ z(s) &= a_{z_m} s^2 + b_{z_m} s + c_{z_m}, \\ m &= 1...n_w \end{aligned}$$

where the coefficients can be found by solving

$$\begin{bmatrix} a_{x_m} \\ b_{x_m} \\ c_{x_m} \end{bmatrix} = \begin{bmatrix} s_{m-1}^2 & s_{m-1} & 1 \\ s_m^2 & s_m & 1 \\ s_{m+1}^2 & s_{m+1} & 1 \end{bmatrix}^{-1} \begin{bmatrix} x(s_{m-1}) \\ x(s_m) \\ x(s_{m+1}) \end{bmatrix},$$

$$\begin{bmatrix} a_{y_m} \\ b_{y_m} \\ c_{y_m} \end{bmatrix} = \begin{bmatrix} s_{m-1}^2 & s_{m-1} & 1 \\ s_m^2 & s_m & 1 \\ s_{m+1}^2 & s_{m+1} & 1 \end{bmatrix}^{-1} \begin{bmatrix} y(s_{m-1}) \\ y(s_m) \\ y(s_{m+1}) \end{bmatrix},$$

$$\begin{bmatrix} a_{z_m} \\ b_{z_m} \\ c_{z_m} \end{bmatrix} = \begin{bmatrix} s_{m-1}^2 & s_{m-1} & 1 \\ s_m^2 & s_m & 1 \\ s_{m+1}^2 & s_{m+1} & 1 \end{bmatrix}^{-1} \begin{bmatrix} z(s_{m-1}) \\ z(s_m) \\ z(s_{m+1}) \end{bmatrix}$$

The n_w waypoints we consider requires $3(n_w - 2)$ polynomials. We then make one polynomial for every three waypoints, then a smoothing function between the polynomials is applied. We can define the whole path as $P(s) : (X(s), Y(s), Z(s))$ where $X(s)$, $Y(s)$, and $Z(s)$ are

$$X(s) = \begin{Bmatrix} x_s(s) & s_1 \leq s \leq s_2 \\ \mu_{r,m}(s) x_{m+1}(s) + \mu_{f,m} x_m(s), (2 \leq m \leq n_w - 1) & s_2 \leq s \leq s_{n_w - 1} \\ x_{n_w - 1}(s) & s_{n_w - 1} \leq s \leq s_{n_w} \end{Bmatrix},$$

$$Y(s) = \begin{Bmatrix} y_s(s) & s_1 \leq s \leq s_2 \\ \mu_{r,m}(s) y_{m+1}(s) + \mu_{f,m} y_m(s), (2 \leq m \leq n_w - 1) & s_2 \leq s \leq s_{n_w - 1} \\ y_{n_w - 1}(s) & s_{n_w - 1} \leq s \leq s_{n_w} \end{Bmatrix},$$

$$Z(s) = \left\{ \begin{array}{cc} z_s(s) & s_1 \leq s \leq s_2 \\ \mu_{r,m}(s) z_{m+1}(s) + \mu_{f,m} z_m(s), (2 \leq m \leq n_w - 1) & s_2 \leq s \leq s_{n_w - 1} \\ z_{n_w - 1}(s) & s_{n_w - 1} \leq s \leq s_{n_w} \end{array} \right\},$$

where $\mu_{r,m}$ and $\mu_{f,m}$ are member functions, defined by

$$\mu_{r,m}(s) = \frac{s - s_m}{s_{m+1} - s_m},$$

$$\mu_{f,m}(s) = \frac{s_{m+1} - s}{s_{m+1} - s_m},$$

$$m = 2, 3, ..., n_w - 1$$

The resulting path is then G^2-continuous, making it continuous in both velocity and acceleration. A path generated through some waypoints can be seen in Figure 2.2.

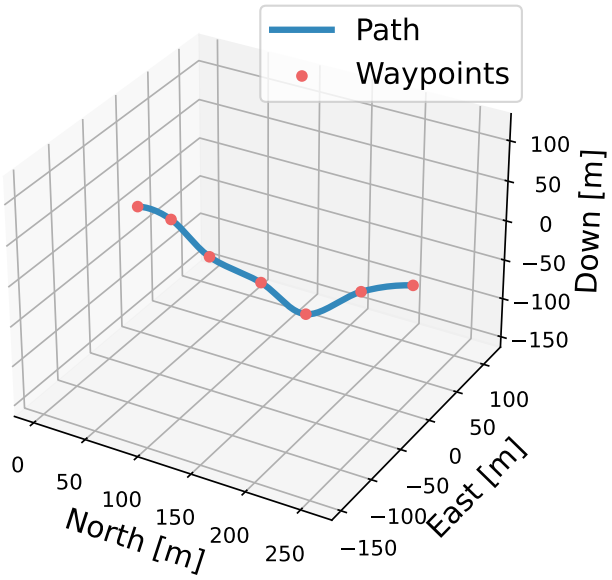


Figure 2.2.: An arbitrary generated 3D QPMI G^2-continuous path.

2.3. Guidance Laws for 3D Path Following

In this section, we introduce the guidance laws used for path following. The theory in this part is used to define the reward function and the error vector, ϵ, which we later use to evaluate the performance of the DRL controller. First, we define a frame in which we define the errors. Then we use the errors to define a guidance law that outputs a desirable direction to move the quadcopter.

The Frenet-Serret frame, introduced by Micaelli and Samson (1994), describes the geometry of a curve. Although the frame is defined along the whole path, this work considers only the Frenet-Serret frame that is currently the closest to the quadcopter. The Frenet-Serret frame is defined from the three axes X_{sf}, Y_{sf}, and Z_{sf}, where X_{sf} is the tangent vector along the path, Y_{sf} is the normal unit vector of the path and Z_{sf} is the cross product of X_{sf} and Y_{sf}. This frame is shown in Figure 2.3.

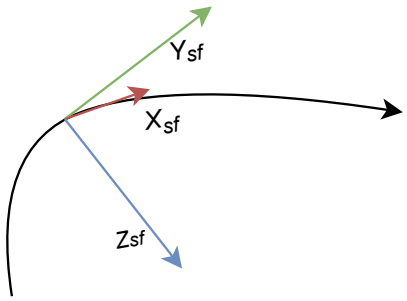


Figure 2.3.: The Frenet-Serret frame with origin at some arbitrary point along the path. X_{sf} is the tangent vector along the path, Y_{sf} is the normal unit vector of the path and Z_{sf} is the cross product of X_{sf} and Y_{sf}.

It is on this frame that the along-path errors are defined. We define the error vector, ϵ_t, as

$$\epsilon_t = [\bar{s}_t, e_t, h_t], \tag{2.8}$$

where $\bar{s}_t$, e_t, and h_t is the quadcopter's position relative to the Frenet-Serret frame. The entries describe the along-track, cross-track, and vertical-track errors, respectively. Since we define the errors at the frame closest to the quadcopter, the along-track error, $\bar{s}_t$, is always zero and thus disregarded.

To accomplish path following, the controller needs to align the quadcopter's velocity vector, V^n, with the tangential vector of the path. The quadcopter is not seeking to go to the point closest to the path; instead, it aims at a point given by the look-ahead distance, Δ. Δ is the distance down along the path from the origin of the Frenet-Serret frame.

We consider a path parameterization of the net variable, $\bar{\omega}$, where the path is represented by $p_p(\bar{\omega}) \in \mathbb{R}$. The error vector ϵ can then be calculated using

$$\epsilon(t) \triangleq R_n^{sf}(p(t) - p_p(\bar{\omega}))$$

where $p(t)$ is the position of the quadcopter, $p_p(\bar{\omega})$ is the position closest on the path closest to the quadcopter, and R_n^{sf} is the rotation matrix between the NED-frame and the Frenet-Serret frame in the same form as in Equation 2.1 (Breivik and Fossen (2009)).

According to Havenstrøm et al. (2021), the desired azimuth, $\chi_d(e)$, and elevation angle, $v_d(h)$, can then be determined as

$$\chi_d(e) = \chi_p + \chi_r(e),$$
$$v_d(h) = v_p + v_r(h),$$

where χ_r and v_r are defined as

$$\chi_r(e) = arctan(\frac{e}{\Delta}),$$

$$v_r(h) = arctan(\frac{h}{\sqrt{e^2 + \Delta^2}}),$$

and Δ is the look-ahead distance defined in Table 3.3. The errors, χ_e and v_e, are defined as

$$\chi_e = \chi_d - \chi, \tag{2.9a}$$

$$v_e = v_d - v, \tag{2.9b}$$

respectively, where χ and v are the rotations around NED-frame, defined as

$$\chi = arctan2(\dot{y}, \dot{x})$$

and

$$v = arctan2(-\dot{z}, \sqrt{\dot{x}^2 + \dot{y}^2})$$

χ_e and v are used in the reward function in Section 3.5. When the quadcopter approaches the path, the error terms e and h will go towards zero, which will result in the correction angles χ_r and v_r going towards zero. The resulting velocity vector V^n will then go towards the corresponding tangent of the path.

2.4. Neural Networks

Machine learning methods approximates a function $y = f(x)$ where x is the observed data and $f(x)$ is the function that is approximated. The methods range from linear regression to more advanced methods such as support vector machines and neural networks.

Neural networks is a group of machine learning methods inspired by the human brain. Neurons are the fundamental units of neural networks, and they share the functionality of their brain namesake. Neurons are often expressed mathematically, as seen in Figure 2.4. I is the neural input, while W is the associated weights, and $A(\cdot)$ is the activation function. The activation function is often *tanh*, the sigmoid function or the Rectified Linear Unit function (ReLU). The simplest type of neural net is the perceptron, which

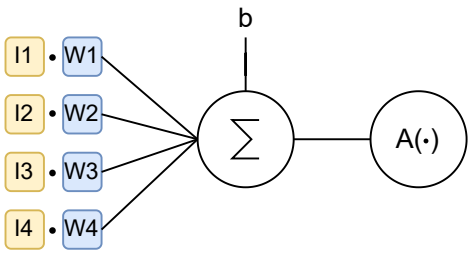


Figure 2.4.: A neuron in a neural network. I is the neural input, W is the associated weights, b is the bias and $A(\cdot)$ is the activation function.

consists of a single neuron. Commonly the neurons are stacked in multiple layers, as in Figure 2.5. The computational layers between the input and the output layer are often referred to as the hidden layers. Neural networks with many hidden layers are often called "deep". In RL, "deep" is typically used if parts of or the whole policy structure include one or more artificial neural networks.

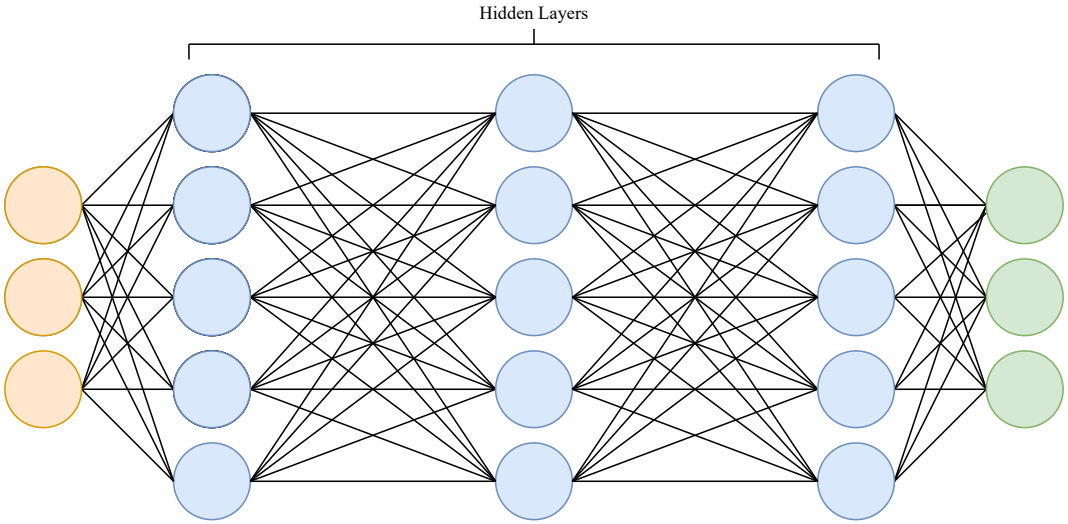


Figure 2.5.: An arbitrary neural network, where the orange nodes are the inputs, the blue nodes are neurons and the green nodes are the ouputs.

CNNs are neural networks that incorporate the convolution operator. In convolution layers, a filter slides across the height and width of the input. The dot product of each element of the filter and the input is calculated at each spatial position, as seen in Figure 2.6. In this way, they manage to capture the spatial information in the data and enable encoding from high dimensional space to latent lower-dimensional space. CNNs are often employed in computer vision problems where spatial information may be relevant, such as image classification, super-resolution, and video processing. The curse of dimensionality describes the explosive nature of increasing data dimensions and its exponential increase in computational efforts required for processing Karanam (2021). By reducing the feature space into more latent lower-dimensional space, CNNs are often used to avoid the curse of dimensionality.

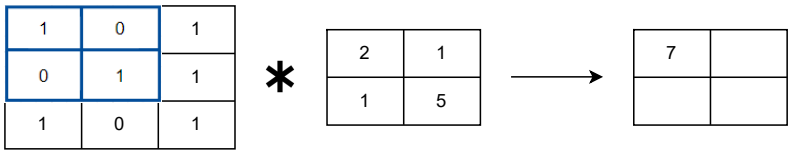


Figure 2.6.: An example of the use of an convolution operator with a kernel size of 2.

2.5. Reinforcement Learning

RL is a framework in which an agent learns a decision-making policy through trial and error. Given an observation, o_t, from the environment, the policy performs an action, a_t, in the environment. This notation corresponds to x and u, commonly used in control theory. In our case, the environment consists of the components described in Section 2.1. The environment can be denoted as a transition model between states $p(s_{t+1} \mid s_t, a_t)$.

In RL, we often differentiate between observations and states. While the state is a true description of the world, observations may contain partial information about the world. If an agent can observe the full state of the environment, such as in the game of chess, it is fully observed. However, this is seldom true for real-world applications.

The set of valid actions is called the action space. Figure 2.7 shows the flow of information between the agent and the environment. The environment generates the observation and the accompanying reward, and the agent then chooses an action that affects the environment.

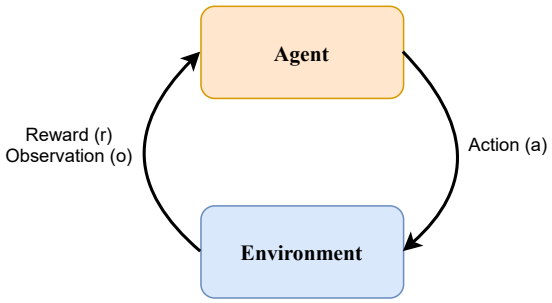


Figure 2.7.: Interaction between the agent and the environment in an RL framework. The environment generates the observation o, with an accompanying reward r. The agent then chooses an action a that affects the environment.

In essence, every model-free RL algorithms attempts to find the policy, $\pi(a \mid o)$, that maximizes the expected total reward, $\mathbb{E}\left[\sum_t r_t\right]$. Unlike in classical control theory, RL enables finding a controller in cases where the environment is too hard to model, e.g., in high dimensional spaces or noisy environments.

2.5.1. Deep Reinforcement Learning

Deep learning exploits the powerful universal approximation properties of NNs by employing them to approximate the policy function $\pi(a \mid o)$. Given enough data, it is often possible to learn better representations with deep neural networks than with shallower networks (Alom et al. (2019)). Although DNNs are notorious for needing vast amounts of data to train, they fit well into the RL framework due to the RL agent producing its own training data. In DRL algorithms, the decision-making policy is parameterized by a neural network. This allows the learner to observe continuous data input and output continuous actions, removing the need to discretize both.

2.5.2. Reward Function

All model-free RL methods aim to maximize the expected reward. The expected reward is the sum of the rewards in all future time steps, often discounted with a factor $\gamma \in (0, 1]$. A small γ leads to a greedy agent, primarily concerning what happens in the immediate future. A high γ will lead to a similar valuation of long-term and short-term rewards. This factor is added to account for the uncertainty of future time steps, as we often can be more certain about what happens in the near future.

We usually differentiate between sparse and dense rewards. *Sparse rewards* are rewards are given at the end of each episode, e.g., +1 when the quadcopter reaches the goal, 0 otherwise. Using sparse rewards may lead to the learner never converging or being slow to learn in complex environments, such as environments with long time horizons or challenging exploration. In the quadcopter example, it will likely take long before the quadcopter reaches the final waypoint, and even then, it will likely take long before it accomplishes it again.

Dense rewards are given at every time step, e.g., rewarding the agent proportionally to its distance from the path. This reward scheme will often increase the learning rate and

thus be more likely to converge. Both dense and sparse rewards involve the manual design of the reward function, although dense rewards often require more domain knowledge. In the quadcopter example, it is trivial to implement a reward function that rewards the quadcopter when reaching the goal waypoint, but finding the distance to the path is not trivial. The designer will then infuse the design of the reward function with their inherent bias towards a way they think the agent should act, which likely is suboptimal. In contrast, sparse rewards let the agent discover the optimal policy by itself.

2.5.3. Proximal Policy Optimization

In DRL, there is a wide array of algorithms, among which we have chosen to use Proximal Policy Optimization (Schulman et al. (2017)). Larsen et al. (2021) found PPO to exhibit superior robustness to changes in the environment complexity and reward function and to generalize well to unseen environments with a domain gap compared to other state-of-the-art DRL methods. In the paper, the authors tested the algorithms in scenarios similar to the one we are employing.

PPO is an improved and simplified version of the relatively complicated Trust Region Policy Optimization method (TRPO). It attains much of the data efficiency and reliable performance of TRPO while only using first-order approximations.

A problem in DRL is determining the step size of the neural net update. Whereas an arbitrary large step size may lead to instability, a small step size may lead to inefficient use of data. PPO deals with this dynamically. The version of PPO we apply uses the clipped objective function, $L^{CLIP}(\theta)$, defined as

$$L^{CLIP}(\theta) = \hat{E}_t[min(r(\theta)\hat{A}_t, clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)],$$

where θ is the policy parameters, $\hat{E}_t$ denotes the empirical expectation over timesteps, ϵ is a scalar clipping parameter, and $r(\theta)$ is defined in Equation 2.10, where $\pi_\theta(s,a)$ is the current policy, and $\pi_{old}(s,a)$ is the old policy. Finally, $\hat{A}$ is the advantage function seen in Equation 2.11,

$$r(\theta) = \frac{\pi_\theta(s,a)}{\pi_{old}(s,a)} \tag{2.10}$$

$$\hat{A} = \pi(s,a) - V(s), \tag{2.11}$$

where a is the action vector, s is the state vector, and $V(s)$ is the value function of the state. Thus, PPO restricts the step length such that appropriate step lengths are applied. The algorithm itself can be seen in Algorithm 1.

Algorithm 1: Proximal Policy Optimization

```
for i... do
    for actor ← 1 to N do
        Run policy π_θold in environment for T timesteps
        Compute advantage estimates Â_1, ..., Â_T
    Optimize L with respect to θ, with K epochs and minibatch size M ≤ NT
    θ_old ⟵ θ
```

3. Method and Implementation

In this section, we look at how the theory from Section 2 is used to set up the study. First, we define the environment and explain our use of curriculum learning in Section 3.1. We then describe our LIDAR simulation set up in Section 3.2. Later we define the observation space and action space of the agent in Section 3.3 and Section 3.4. In Section 3.2 we set up an simulation of a LIDAR sensor. We design a suitable reward function in Section 3.5 and a policy network in Section 3.6. Lastly, we go through the training details in Section 3.7 and performance evaluation in Section 3.8.

3.1. Environment and Curriculum Learning

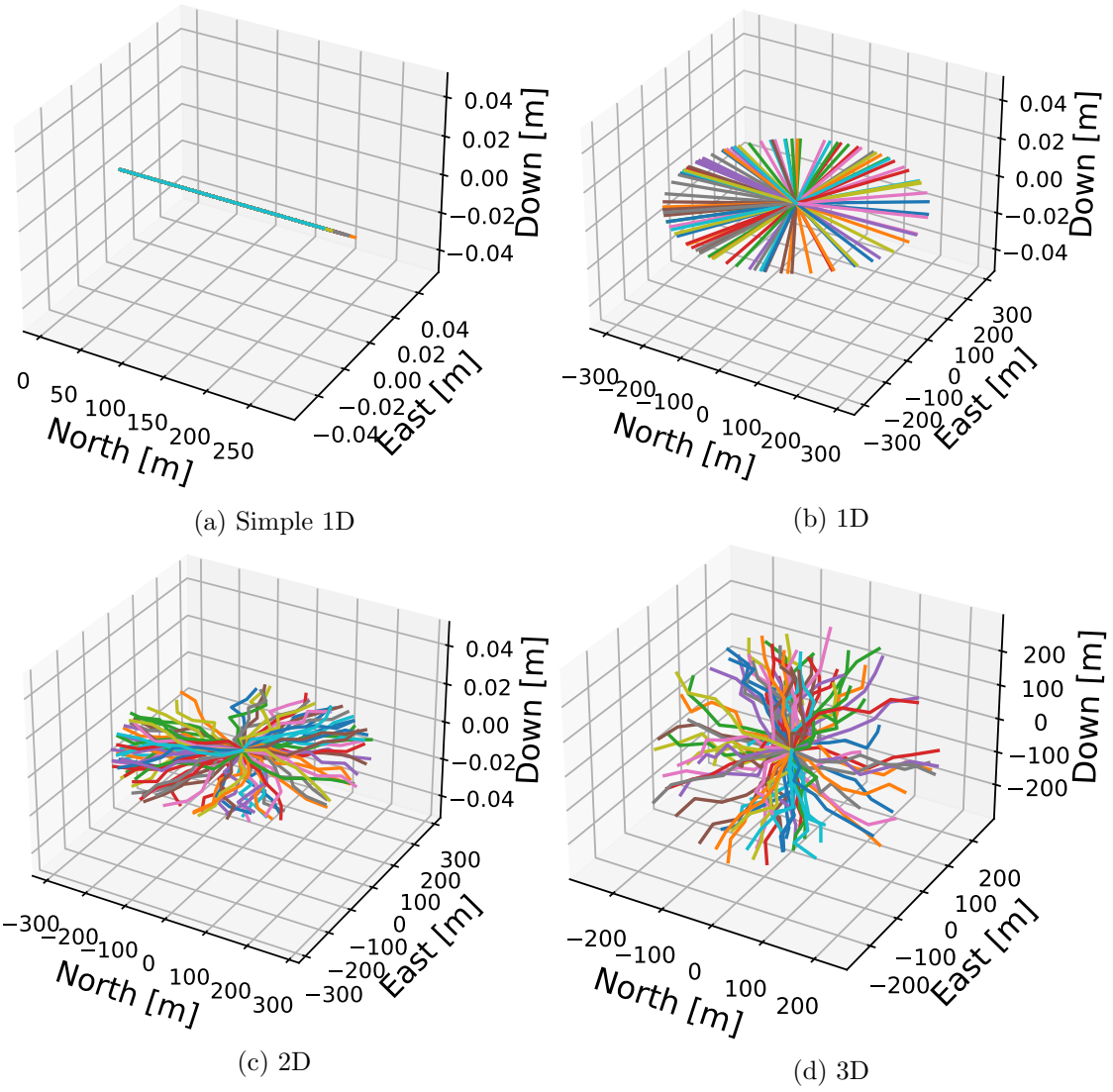
The environment consists of a quadcopter with its dynamics from Section 2.1, a path defined from Section 2.2, and zero to two obstacles. The path is generated from the randomly initiated waypoints and differs from episode to episode. We utilize curriculum learning, which is the technique of gradually increasing the complexity of the scenarios introduced when training an RL agent (Bengio et al. (2009)). The idea is that an agent learns the challenging scenarios more easily by first learning the basics, similar to how humans learn. We set up five learning scenarios, defined in Table 3.2. An essential factor to note is that the observation vector contains the orientation in relation to NED. In the first scenario, Simple 1D path, the agent is made to follow a path in a single direction. An effort is made to make the agent work regardless of orientation with the randomized starting direction of the paths in the other scenarios. A visualization of the scenarios without obstacles can be found in Figure 3.1d.

The obstacles are initiated as in Havenstrøm et al. (2021), between $\frac{1}{3}$ and $\frac{2}{3}$ of the length of the path. The obstacles are spheres with radii uniformly distributed between 4 and $10m$ with centers on the path.

The hyperparameters for the training environments can be found in Table 3.3. An episode is terminated if any of the three conditions are met: (a) $r_t < r_{min}$, (b) $t > t_{max}$, or (c) the quadcopter reaches the final waypoint.

Table 3.1.: Proximal policy optimization parameters

Parameter	Value
n_steps	1024
learning_rate	$2.5 \cdot 10^{-4}$
batch_size	64
gae_lambda	0.95
gamma	0.99
n_epocs	4
clip_range	0.2
ent_coef	0.001



(a) Simple 1D

(b) 1D

(c) 2D

(d) 3D

Figure 3.1.: 100 runs each of the four training scenarios without obstacles.

3.2. Simulating a LIDAR for Obstacle Detection

For the agent to avoid obstacles, it needs to be able to detect them. In Havenstrøm et al. (2021) the authors set up a virtual sensor with a configuration corresponding to a sonar. Sonars are not usually used in quadcopter sensor suites. On the other hand, LIDARs are widely used. Therefore, we will apply the same setup as in Havenstrøm et al. (2021) while making minor modifications to suit our use case better. We employ the same setup, except expanding the LIDAR from only detecting in a small area in front of the vessel to being a 3D LIDAR. To simulate the LIDAR, we make two simplifications:

- There is no sensor noise.
- The LIDAR samples in a all directions simultaneously.
- There are no blind spots. In a real environment, there would be blind spots due to the occlusions originating from the quadcopter itself.

When simulating the sensor, a realistic yet low-cost operation is needed. The sensor

Table 3.2.: The five scenarios without obstacles used to train the DRL agent

Scenario	Description
Simple 1D path	The Y_n and Z_n coordinates of the waypoints are set to zero.
1D path	The waypoints are on a line with an randomly chosen initial direction.
2D path	The waypoints are on a plane with a randomly chosen initial direction.
3D path	The waypoints are in three dimensional space with an randomly chosen initial direction.
3D path with obstacles	The waypoints are in three dimensional space with an randomly chosen initial direction with two obstacles placed between 1/3 and 2/3 of the length of the path.

Table 3.3.: Parameters used in the reward function, guidance law and modelling of the environment

Parameter	Value	Unit
c_{pf}	-6	N/A
Δ	3	$[m]$
r_{min}	-2000	N/A
t_{max}	10000	N/A
F_{min}	-6	$[N]$
F_{max}	6	$[N]$
$total\_steps$	$150 \cdot 10^6$	N/A
$step\_size$	0.01	$[s]$
d	50	$[m]$
α	$U_{[-\pi/4,\pi/4]}$	N/A
β	$U_{[-\pi/4,\pi/4]}$	N/A
n	7	N/A

is simulated as a 15 by 15 grid of rangefinder sensors. A 3D LIDAR with a 360-degree field of view is employed. Each of the rangefinders is simulated as lines pointing out of the LIDAR as seen in Figure 3.2. The rangefinders are placed equally spaced in angle space. However, most LIDAR samples are denser on their horizontal axis than on the vertical axis Velodyne (2022).

A line search is done to find the obstacles. The line search returns the obstacle closest to the sensor up to the max range of the sensor of $25m$. In addition, each reading is transformed to what Havenstrøm et al. (2021) coined "closeness". This transformation can be found in Equation 3.2. The result is a 15 times 15 array of sensor measurements. This range is lower than one could find on LIDARs, but it is limited due to performance considerations.

A 15 times 15 array directly input into the observation vector could lead to the well-known curse of dimensionality. Therefore, reducing the feature space is desirable. We call the resulting vector $\boldsymbol{s}$.

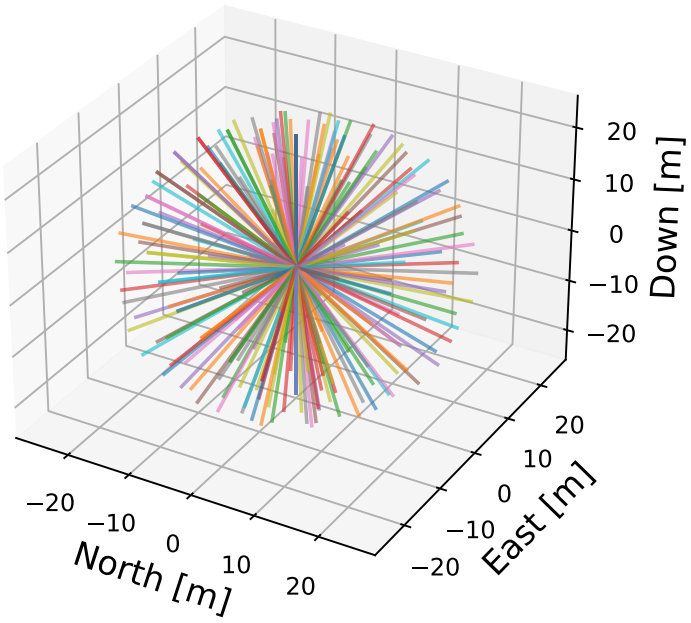


Figure 3.2.: A vizualisation of the LIDAR beams emitting from the origin.

3.3. Observation Space

In this thesis, we divide the observation space into the navigational and LIDAR space. The navigational space provides the agent with information about the state of the quadcopter and the quadcopter's direction in relation to the path. On the other hand, the LIDAR space is the LIDAR data stream. The navigational vector is implemented as fully observed, providing true knowledge about the agent's state, thus rendering a higher possibility of convergence towards an optimal policy. In real-world scenarios, the navigational vector variables would consist of estimated variables.

First we define the path error vector $\boldsymbol{e}_{path}$ as

$$\boldsymbol{e}_{path} = \begin{bmatrix} \chi_{e,1} & \upsilon_{e,1} & \chi_{e,3} & \upsilon_{e,3} & \chi_{e,5} & \upsilon_{e,5} & e & h \end{bmatrix},$$

where $\upsilon_{e,1-5}$ and $\chi_{e,1-5}$ are the directional errors with a lookahead of 1, 3, and 5m respectively, defined in Equation 2.9. υ and χ makes it possible for the agent to acquire knowledge of the future curvature of the path. The full navigational vector o_n is defined as

$$o_n = \begin{bmatrix} \boldsymbol{\nu}^T & \boldsymbol{\Theta}_{\boldsymbol{nb}}^T & \boldsymbol{e}_{path} \end{bmatrix}^T,$$

, where $\boldsymbol{\nu}$ and $\boldsymbol{\Theta}_{\boldsymbol{nb}}$ are defined in Section 2.1.2.

The LIDAR space, o_l, is the full 15x15 grid as defined in section 3.2. It is important to note that the rangefinders are equally spaced in angle space, not 3D cartesian space. The definition is done to make a data format suitable to be the input of the neural network.

The full observation vector, o, is defined as

$$o = \begin{bmatrix} o_n, o_l \end{bmatrix}$$

3.4. Action Space

In our setup, the action vector is the output of the policy network. Actions can both be discrete or, as in our case, continuous. The action vector, a, is defined as

$$a = u = \begin{bmatrix} F_1 & F_2 & F_3 & F_4 \end{bmatrix}^T,$$

where F_1, F_2, F_3, and F_4 are the forces exerted by the four propellers. These range from F_{min} to F_{max} as seen in Table 3.3. Using a low abstraction level action space entails that the policy network needs to learn the complex relationship between the observation space of navigational and LIDAR variables and the force on each of the propellers. The action space coincides with the lowest level of abstraction in Kaufmann et al. (2022).

3.5. Reward Function

The main objective of any model-free RL algorithm is to find a control policy that maximizes the reward function. To produce a path following and collision avoidance policy, we design additive reward terms that encourage path adherence and discourage collisions and unstable behaviors. The path following reward term, r_t^{pf}, is given by

$$r_t^{pf} = c_{pf}(\chi_{e,3}^2 + \upsilon_{e,3}^2),$$

where $\chi_e = |\chi_d - \chi|$ and $\upsilon_e = |\upsilon_d - \upsilon|$. The definitions of χ_d, χ, υ_e, and υ can be found in Section 2.3. c_{pf} is a manually set tuning parameter, whose value can be found in Table 3.3. Stability is crucial for successful path following. Moreover, high torques and vibrations can damage the sensors, motors, and other hardware onboard. Therefore, a reward term guiding the quadcopter towards stability was implemented as

$$r_t^s = \sum_{i \in [0,1,2]} c_{s,i} \cdot \omega_{b/n,i}^{\omega}, \tag{3.1}$$

where $c_{s,i}$ is a manually tuned parameter that is set to -1 where $|\omega_{b/n,i}^{\omega}| < 4\pi$ and 0 elsewhere, trying to avoid the tuning problem seen in Sundøen et al. (2021), where too much emphasis on r_t^s may have led to a worse path following performance. Making the reward zero within this range should allow the quadcopter to perform path following without sacrificing the reward in steadiness. r_t^s will penalize the agent for being in states with higher rotational velocities, urging it to keep the quadcopter stable in flight. Setting the hyperparameters c_{pf} and c_s is not trivial. Setting c_s too low will make the quadcopter unable to maneuver, whereas setting c_s close to zero will make it hard to stabilize. Therefore, tuning these parameters is a journey of trial and error, in which the authors spent much time. The value c_{pf} is shown in Table 3.3.

Dense rewards are favorable in continuous control as the policy should be generally smooth. A dense closeness penalty provides a richer feedback signal to the RL agent, such that more of its experience is usable for learning collision avoidance. A reward applied when the quadcopter collides would be a sparse reward. To make a dense reward, we need to rely on sensory data. We define obstacle closeness $c(d_{i,j})$ as

$$c(d_{i,j}) = clip(1 - \frac{d_{i,j}}{d_{max}}, 0, 1), \tag{3.2}$$

where $d_{i,j}$ is the i'th and j'th sensor grid position and d_{max} is the maximal range of the sensor. As long as there are no obstacles, $c(d_{i,j})$ will render the reward zero. The

collision avoidance reward r_t^{ca} is defined in Equation 3.3.

$$r_t^{ca} = -\frac{\sum_{i \in I} \sum_{j \in J} \beta_{ca}(\theta_j, \psi_i)(\gamma_c max((1 - c(d_{i,j}))^2, \epsilon_c))^{-1}}{\sum_{i \in I} \sum_{j \in J} \beta_{ca}(\theta_j, \psi_i)} \tag{3.3}$$

, i and j denotes the sensor sector, ϵ_c is small scalar made to hinder singularities where $c(d_{i,j}) = 0$, β_{ca} is a scaling factor making the rewards relative to the direction of the quadcopter. β_{ca} is defined as

$$\beta_{ca}(\theta_j, \psi_i) = (1 - (2|\psi_j|/\gamma_a)) + \epsilon_{ca}$$

, where ϵ_{ca} is a small scalar made to penalize obstacles at the edge, while θ_j and ψ_i are the vehicle-relative LIDAR directions. A visualization of the collision avoidance reward function can be found in Figure 3.3, Note that the resolution is set way higher than in the training.

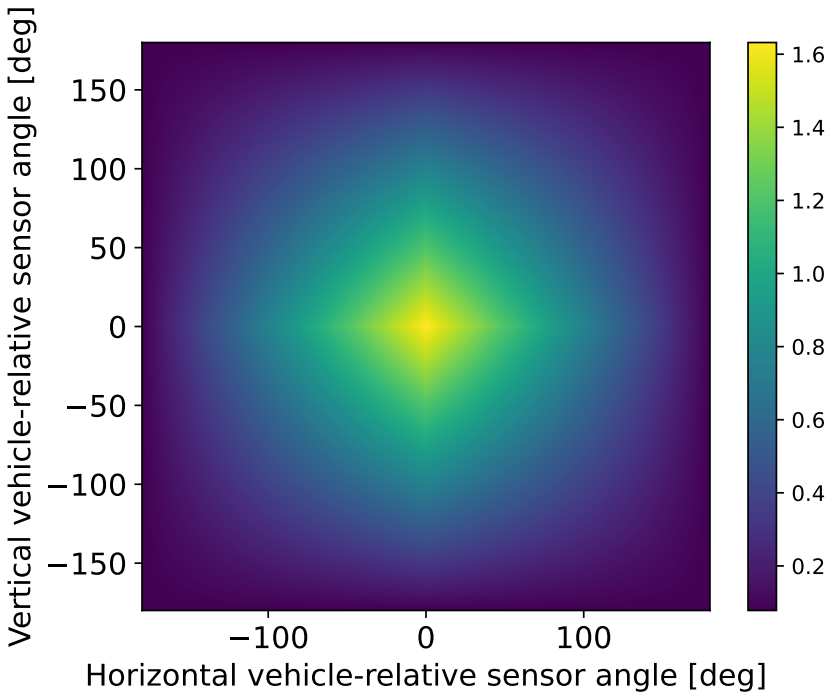


Figure 3.3.: The collision avoidance reward function visualized as from Havenstrøm et al. (2021)

r_t^{ca} seeks to maximize the distance to objects rather than directly penalizing collisions. The implications are important to notice. This implies that a too negatively tuned r_t^{ca} will make the agent go far off the path to avoid obstacles, while a too high r_t^{ca} can make going off-path to avoid an obstacle not worth it.

The resulting reward function is thus given by:

$$r_t = r_t^s + r_t^{pf} + r_t^{ca} \tag{3.4}$$

We expect that in the early stages of the training process, the rotational velocities will be high due to the exploratory nature of DRL. Unlike r_t^{pf}, r_t^s has no upper limit. p, q, and r are squared, likely leading to r_t^s dominating r_t^{pf} in the early stages, i.e., $r_t \approx r_t^s$, guiding the agent to learn to be stable before trying to follow the path.

Table 3.4.: CNN feature extraction parameters

Parameter	Value
padding	24
kernel_rate	5
stride	2
in_channels	1

3.6. Policy Network

The neural network architecture can be divided into two parts: CNN feature extraction and "Mlppolicy". The setup can be seen in Figure 3.4.

As stated in the theory section, CNNs excel at extracting spatial information out of data. Therefore, we use a CNN to extract features out of the LIDAR data. The CNN feature extractor is set up as seen in Figure 3.5 with alternating Conv2d and ReLU layers and parameters found in Table 3.4. Additionally, Torch's circular padding is employed, solving the problem of discontinuity between 0 rad and 2π rad. The last node flattens the data, making it suitable for the dense MlpPolicy.

On the other hand, the navigational states are not preprocessed. "Mlppolicy" is a deep neural network with ordinary dense layers native to SB3. The navigational states and the LIDAR data features are input to Mlppolicy. We employ similar nets in the value network and policy network. Each of them got three hidden layers, where the first layer has 128 neurons, the second layer has 64 neurons, and the last layer has 32 neurons. The net is deeper than in Sundøen et al. (2021) to account for the increased complexity of collision avoidance.

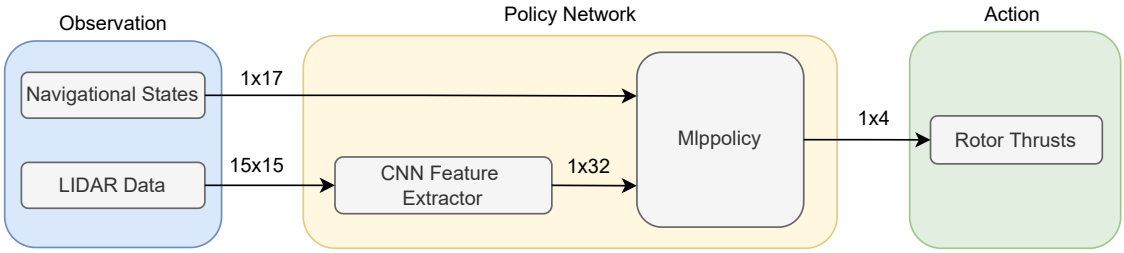


Figure 3.4.: The neural network architecture. A CNN was used to extract features from the high dimensional LIDAR data. The policy network map to the rotor thrusts.

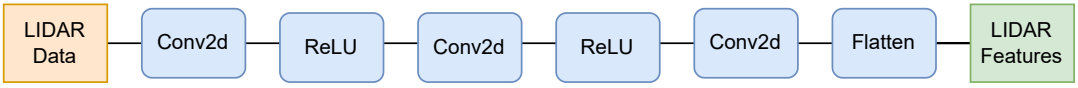


Figure 3.5.: The feature extractor of the LIDAR Data set up in Torch.

3.7. Training Details

The repository used for training can be found on GitHub (Sundøen et al. (2021)). StableBaselines3 (SB3) is a set of reliable implementations of RL algorithms in PyTorch (Raffin et al. (2021)). We use SB3's PPO implementation with the parameters stated in Table 3.1.We follow the curriculum learning strategy described in Table 3.2 to train the DRL agent. Thus, the agent starts training in the trivial Simple 1D path scenario, primarily stabilizing the quadcopter and maintaining strict altitude control. The difficulty increases after training for $30 \cdot 10^6$ time steps as the training proceeds to the 1D scenario, increasing one degree of freedom for the path generation. Thus, in a curriculum learning fashion, the environment's difficulty increases over time until the agent has trained for a total of $150 \cdot 10^6$ time steps. The number of timesteps and parameters for training are found in Table 3.3.

We run the training with 16 agents in parallel, drastically reducing the training wall-time. All simulations were conducted on IDUN, a high-performance computing cluster owned by the Norwegian University of Science and Technology Idun (2022). Approximately 15 hours of wall-time was required to reach $150e6$ time steps for this hardware.

3.8. Performance Evaluation

Evaluating the resulting controller can be troublesome because there are multiple objectives. Therefore, we have a diverse toolset of performance metrics. Firstly we employ visual inspection. The desired path, the quadcopter path, and the obstacles are plotted and visually inspected for deviations. Albeit not an accurate tool, visual inspection is great for getting a general impression of the method's performance. There is also a possibility of discussing errors by observing deviations from the path. Due to the scaling of the axes in the 3D plots, the errors might be troublesome to observe. Especially in 3D plots, an error may be observed, but its magnitude is not necessarily trivial to see. Therefore, we define an error metric $Err_{t,n}$ as

$$Err_{t,n} = \sqrt{e_{t,n}^2 + h_{t,n}^2},$$

where n is the episode number, $Err_{t,n}$ represents the absolute error to the closest point on the path. To get an accurate view over multiple episodes, we define the *Absolute Path Error*, APE_t, as

$$APE_t = \frac{1}{N} \sum_{n=1}^{N} Err_{t,n},$$

where N is the total number of episodes. In other words, APE_t is the episode-averaged path error. Additionally, we employ the median APE to get a single number error measure. A median is employed since it is less susceptible to outliers than a mean.

Although the APE_t will give an overview of the controller's path following performance, it does not consider the obstacles. The agent will deviate when approaching objects, resulting in higher APE_t. Therefore, two additional metrics are defined. Failure Rate, FR, is defined as

$$FR = \frac{\text{Number of failed episodes}}{N}, \tag{3.5}$$

where episodes the agent traverses less than 90 % of the path by the time the episode is terminated for any of the reasons mentioned in Section 3.1, are considered a failures.

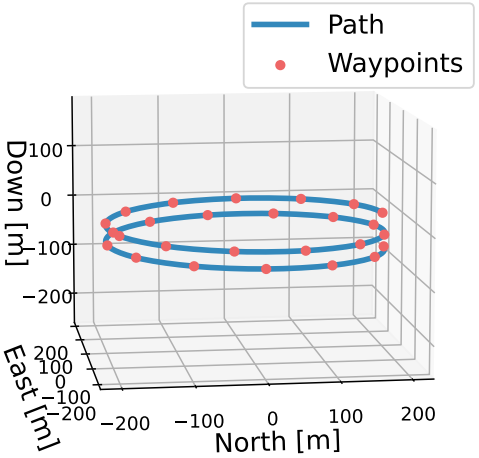
Table 3.5.: The four scenarios without obstacles used to train the DRL agent

Scenario	Description
Sphere	A spherical obstacle within a helical path
Dead-end	A half sphere surface with a path going directly through
Vertical	Obstacles placed vertically with a path going directly through
Horizontal	Obstacles placed horizontally with a path going directly through

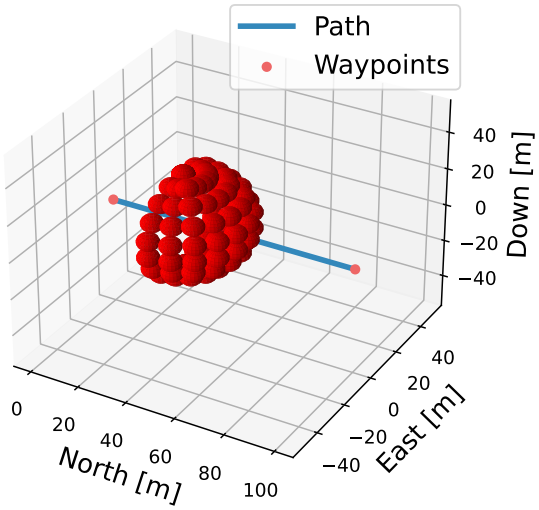
Collision Rate, CR, is defined as

$$CR = \frac{\text{Number of episodes the agent crashes into an obstacle}}{N} \tag{3.6}$$

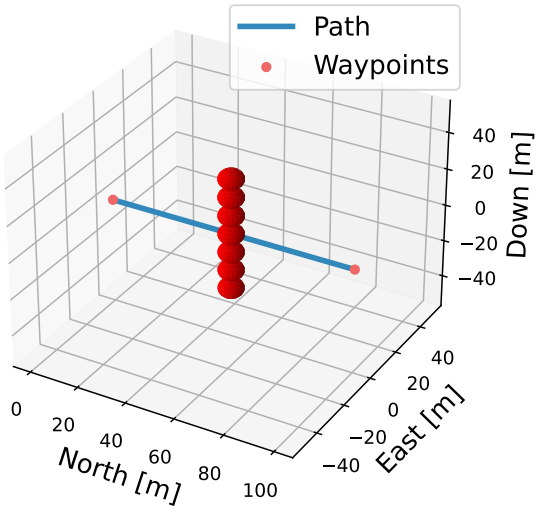
We also define a diverse set of scenarios unseen by the controller. The description of the scenarios can be found in Table 3.5. The main goal of testing the resulting controller on unseen scenarios is to determine whether it has generalized.



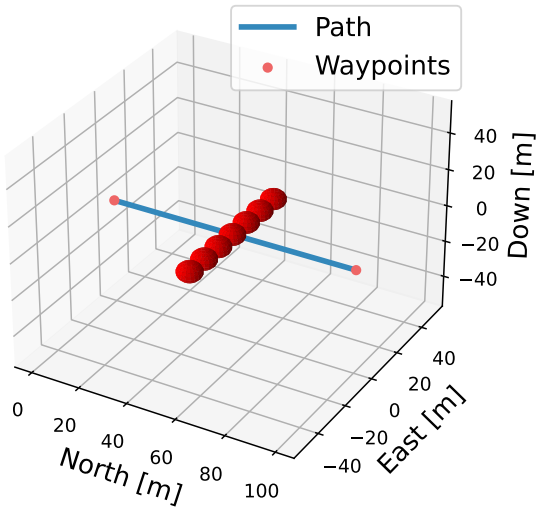
(a) Sphere scenario without the obstacle. Spherical obstacle with radius of 185 and centered in the origin was added. The obstacle was not visualized due to a bug in matplotlib.



(b) Dead-end scenario.



(c) Vertical scenario.



(d) Horizontal scenario.

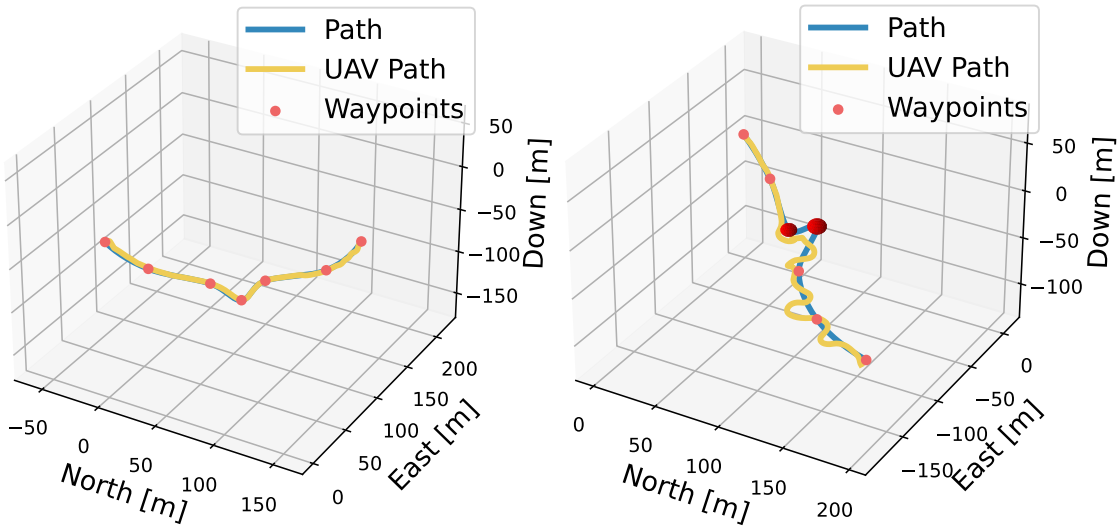
Figure 3.6.: Overview of the testing scenarios. These are unseen for the agent and are used in the evaluation to find whether the controller has generalized.

4. Results and Discussions

In this section, we will cover the results of the DRL controller. The focus will be on the two most challenging training scenarios, the 3D scenario without and with obstacles. Section 4.1 shows the resulting controller's performance employing visual inspection of single runs. Then, in Section 4.2, we look at quantitative measurements from running the algorithm over 100 simulations. Lastly, Section 4.3 investigates the performance in unseen scenarios.

4.1. Visual Inspection

This section presents four 3D plots of episodes to get a general idea of the controller's performance. Figure 4.1 and Figure 4.2 depicts four arbitrary runs where the agent succeeds in reaching the goal in Figure 4.1 and fails in Figure 4.2. The path of each of the episodes starts out at the origin.



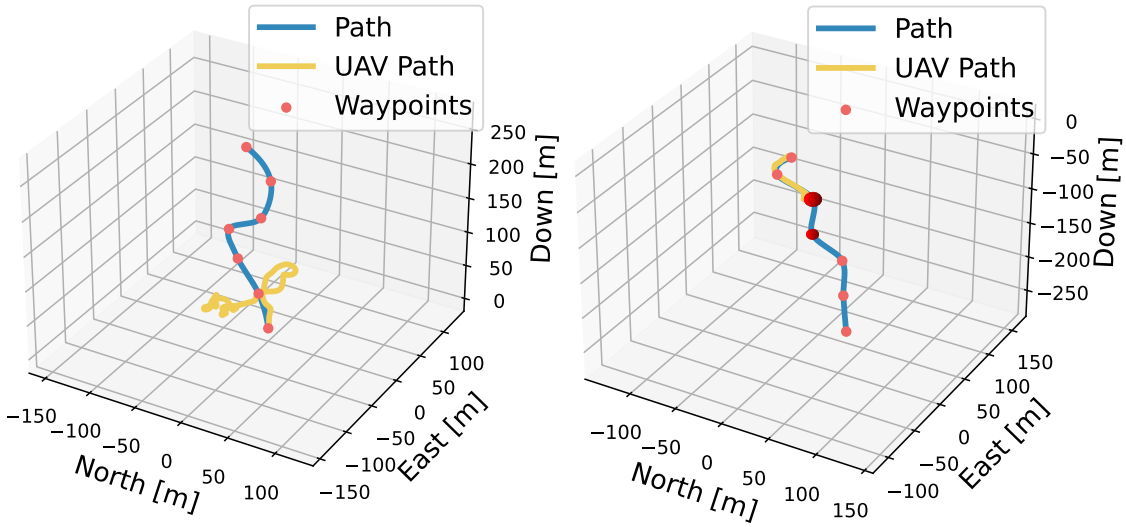
(a) 3D scenario without obstacles

(b) 3D scenario with obstacles

Figure 4.1.: Successful runs of the resulting agent on the 3D scenario, with and without obstacles.

In Figure 4.1a the agent follows the path closely, just as it should when there are no obstacles. There is a slight deviation between the third and fourth waypoint, where the curvature of the path is high, similar to deviations seen in Sundøen et al. (2021). In Figure 4.1b, the agent follows the path for the first two waypoints. The agent goes off the path with the first obstacle coming into the LIDAR range, and it stays off the path until it has surpassed the second obstacle. It then seems to try to recover. However, it overshoots several times. Although not unstable behavior, the agent uses a long time before finally reaching an acceptable path following close to the last waypoint. The

successful dodging of obstacles indicates that the CNN feature extractor seems to work, at least to an extent.



(a) 3D scenario without obstacles

(b) 3D scenario with obstacles

Figure 4.2.: Unsuccessful runs of the resulting agent on the 3D scenario, with and without obstacles.

In Figure 4.2a the agent follows the path until the second waypoint, then it becomes unstable and goes off the path until the end of the episode. It is unclear why this happens. A possible reason could be that the path following performance degraded since the last training batch was done on only scenarios with obstacles. In Figure 4.2b the agent crashes right into the first obstacle. It can be observed that the agent swerves around in the area before the obstacle ultimately crashes into it.

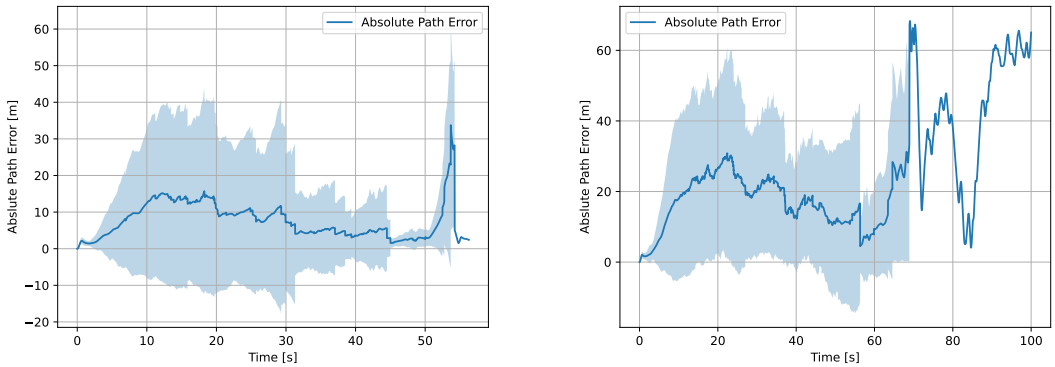
Both runs in Figure 4.2 severely fail. In both cases, the agent moves in an unstable manner resulting in one of the agents crashing and the other continuing to go out of control.

Even when the agent succeeds, it has trouble dodging obstacles. It does manage to get back on track but shows oscillations before establishing a standard path following. The same behavior was commonly seen over multiple runs. However, it is hard to state what is going wrong. Due to the nature of neural networks, DRL is a black-box method, one can not find out the specific reasoning behind its actions. The lack of explainability makes it particularly hard to find out how to solve failures. One can only look at the performance and theorize where the issues originate.

There could be multiple reasons. The agent needs to go far off the path to avoid the obstacles. It could be that the agent in the scenarios without obstacles has learned to follow the path only when close to the path itself, therefore not handling necessary deviations well. The problem could be solved by additional training in the scenario with obstacles, by introducing a larger offset to the initial position of the agent, or by simply adding environmental disturbances such as wind to force the agent off the path in training.

4.2. Quantitative Analysis on Training Scenarios

In this section, the quantitative results are presented and discussed. Figure 4.3 shows the mean absolute path error at each time step in the 3D scenario without and with obstacles over 100 episodes each. The shadowed blue area is the range of the standard deviation of the error at the given time step. A table containing failure rate, collision rate, and median APE is also presented. The same plots and tables for the other training scenarios can be found in Appendix A.



(a) 3D scenario without obstacles

(b) 3D scenario with obstacles

Figure 4.3.: The absolute path error over 100 episodes each in the 3D scenario without and with obstacles. At the start, the average is of all the 100 starting agents. However, as time passes, episodes start to terminate. The shadowed blue area is the standard deviation of the error. Areas where the standard deviation is zero, are likely due to a single agent surviving that long.

Table 4.1.: Failure rate and collision rate in the 3D scenario without and with obstacles over 100 episodes.

Scenario	Failure Rate	Collision Rate	Median APE
3D without obstacles	0.26	N/A	$5.78m$
3D with obstacles	0.65	0.30	$21.08m$

In Figure 4.3a the APE is low in the first time steps before steadily increasing up to an error of around $15m$ at 12 seconds. At the same time, the standard deviations show a similar increase. The behavior is as expected since the quadcopter initializes close to the path. After peaking, the error slowly converges back onto the path. The error spikes at around 53 seconds could be a result of similar behavior as in the single run in Figure 4.2a.

In the first 10 seconds of Figure 4.3b, the agent shows error characteristics similar to Figure 4.3a, as expected since the first part of the paths is identical in the two scenarios. Afterward, the error in the 3D scenario with obstacles rises. Most agents have observed an obstacle and are in an evasive maneuver at by 10 seconds. Between 22 seconds and around 55 seconds, the error declines while oscillatory. These oscillations could be the results of behavior similar to that of the single-agent run in Figure 4.1b. By 55 seconds, the error reaches its lowest point. It could be that the agents either have become back on

the path or have ended their episodes. As discussed in section 4.1, this could be caused by the agent not having correctly learned how to get back on the path and potentially solved by employing a higher offset to the initial position of the agent.

Table 4.1 shows the failure rate, collision rate, and median absolute path error of the two formerly examined scenarios. The table serves as an easy-to-read overview of the agent's performance. We read from the table that the failure rate severely increases when adding obstacles. Most of this increase can be attributed to the collision rate of 0.30, but not all. The remaining failures may stem from the agent evading the obstacle too aggressively, such that it never recovers and results in a failure. The median APE increases almost four-fold between the two scenarios. A slight increase is necessary and expected to dodge obstacles. However, a median APE of $21.08m$ seems excessive when the maximum obstacle radius is $10m$.

This section gave a detailed overview of the controller's performance in the training environment. The controller was able to dodge some obstacles but still crashed often. The controller had a lower failure rate and median APE in the scenario without obstacles, implying that it had retained its path following ability.

4.3. Performance in Unseen Scenarios

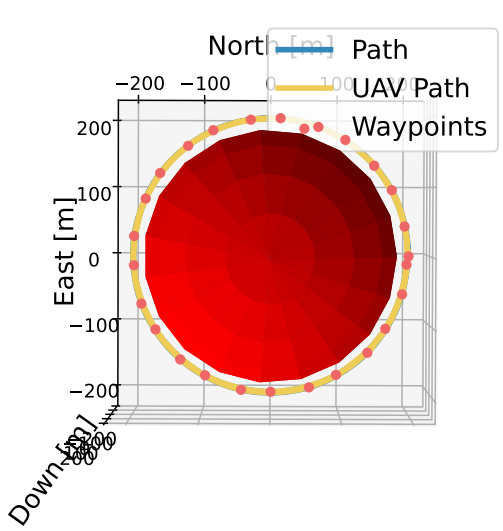
In this section, we present the behavior of the trained agent in a series of test scenarios never observed by the agent. Visual inspection of single runs and the failure rate, collision rate, and median APE over ten runs is presented. The reduction from 100 runs in section 4.2 to 10 runs in this section is due to the higher runtime that is caused by employing multiple obstacles. The agent was initialized with an offset uniformly distributed between 0 and 5. If this step had not been taken, that agent would follow the same path in each run. The single runs can be found in Figure 4.4.

It is important to stress that the agent had never seen scenarios with an obstacle as large as in Figure 4.4a. However, despite that, it can be seen in Figure 4.4a that the agent could follow the helical path with deviations lower than the 3D scenario without obstacles. The agent stays on the path even though the LIDAR detects the obstacle to its side, showing that the agent has become aware of which direction it is heading compared to the direction towards the obstacle it observes. Potentially, when compared to Figure 4.3b, the agent might instead ignore adjacent obstacles and focus on following the path. The behavior is consistent with the desired behavior from the collision avoidance reward term, which penalizes obstacles in front of the quadcopter significantly more than elsewhere, as seen in Figure 3.3. However, in Table 4.2 a Failure Rate of 0.20 can be observed. While not perfect, the Failure Rate is still low.

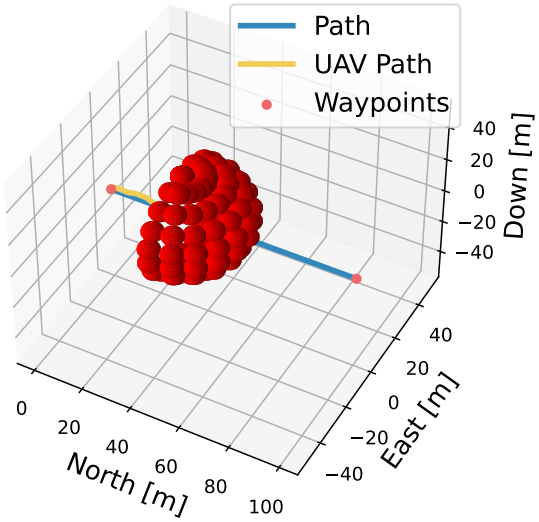
In the dead-end scenario in Figure 4.4b, the agent encountered a severely tricky scenario, as it could not follow the usual paths around the object. The agent swerved around in the middle of the half-sphere before it ultimately crashed. We recorded a 1.00 collision rate in Table 4.2, concluding that the agent failed in the dead-end scenario.

In Figure 4.4c, the possibility of flying above the obstacle was removed. The agent did not struggle with this scenario, successfully completing it. The zero failure rate in Table 4.2 further strengthens the belief that the controller can handle the scenario. A slightly lower median APE than in the 3D scenario with obstacles in Table 4.1 was observed.

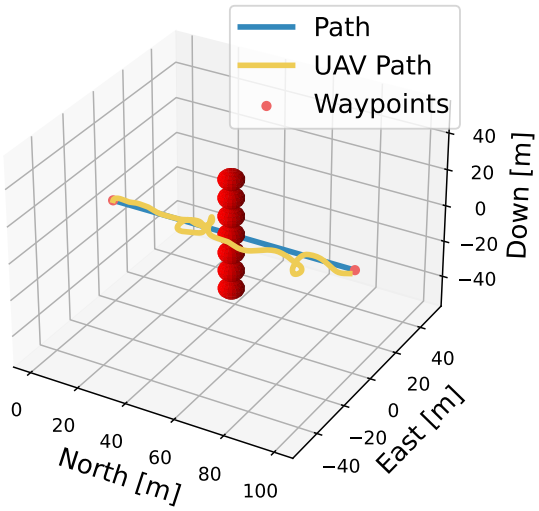
In Figure 4.4d the obstacles were placed horizontally. As in Figure 4.4c, the agent tries to dodge the obstacles vertically. After still observing obstacles, it instead goes over. During this time, a collision was recorded. The quadcopter was swerving, accidentally



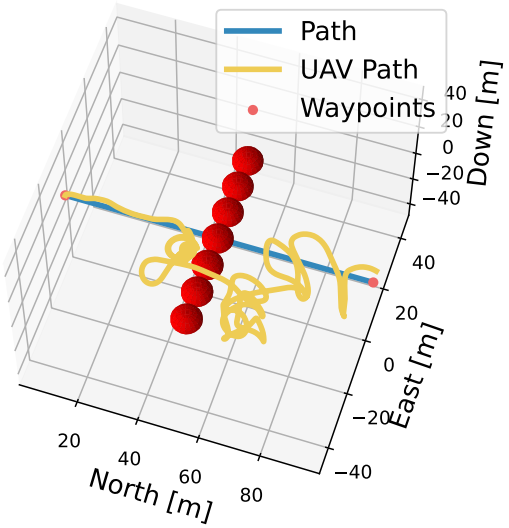
(a) A run in an environment with a spherical obstacle and a helical path surrounding it.



(b) A single run in an unseen environment. The environment consists of a spherical obstacle with the path surrounding it.



(c) A run in an environment with obstacles placed vertically next to each other and the path is a straight line.



(d) A run in an environment with obstacles placed horizontally next to each other and the path is a straight line.

Figure 4.4.: Single runs of the agent in various unseen scenarios. The viewing angles of the figures are adjusted to extract the most information from the runs.

hitting the obstacles. As seen in Table 4.2, it crashes every single time.

This section shows that the agent has successfully generalized its behavior into some of the unseen scenarios. The agent succeeded in most runs in the sphere scenario, showing a lower median APE than the 3D scenario without obstacles. Figure 4.4c and Figure 4.4d showed that the agent may have a preference on dodging obstacles in the horizontal plane. It was fascinating to observe that in Figure 4.4d, the agent first tried to go around the obstacles in the horizontal plane but then went over them instead. The agent may have overfitted to evade the obstacles in the horizontal plane. Employing a more diverse set of obstacle placement in the training scenarios could solve this problem. For example, objects could have a random offset of the path. The agent failed in the

Table 4.2.: Failure rate and collision rate in the unseen scenarios over 10 episodes.

Scenario	Failure Rate	Collision Rate	Median APE
Sphere	0.20	0.10	$0.58m$
Dead end	1.00	1.00	$5.48m$
Vertical	0.00	0.00	$20.30m$
Horizontal	1.00	1.00	$11.71m$

dead-end scenario, hovering in the middle of the half sphere.

4.4. Improving the Performance

The complexity of the problem could cause a lack of performance. As mentioned in Section 3, using DRL to solve the task gives rise to the dual objective of rewarding both path following and collision avoidance, and these objectives do not always coincide. Additionally, the system contains faster and different dynamics compared to Havenstrøm et al. (2021), possibly making the current setup unfeasible.

Countermeasures to the complexity were made but were probably not sufficient. The depth of the policy network was increased from Sundøen et al. (2021), yet it may still have been too shallow. Tuning the hyperparameters of PPO, seen in Table 3.1, could also be a possibility. However, tuning DRL agents is a time-consuming process. A higher-level action space could have been chosen. The action space could instead be the commanded collective thrust and body rate, as this is the best-performing control abstraction in Kaufmann et al. (2022). The higher abstraction level action space would require a classical control method to map between the collective thrust and body rate to the forces of each of the rotors. The complexity of the task may have been too high but could be solved by either a deeper policy network or by having a higher abstraction level action space.

A solution to the performance problem could be in the training setup. As can be seen in Table 3.2, the first four scenarios do not have any obstacles. The discrete curriculum learning might have introduced obstacles too late and too sudden in training, such that the policy is predominantly path following with a slight consideration for collision avoidance. Solving the problem could be done by introducing continuous curriculum learning. Instead of the obstacles being discretely introduced, we could ease in the obstacles more carefully from earlier on. There could be few to no obstacles in the early time steps, but as time passes, more obstacles can be introduced according to a random distribution. A similar approach could also be applied to the path, making the path increasingly have more curvature and expand through the spatial dimensions. Continuous curriculum learning can increase performance, especially in collision avoidance, and remove the domain gap between the different scenarios.

5. Conclusion and Further Work

5.1. Conclusion

The primary research objective of this paper was to develop a path following and collision avoidance method for quadcopters using Deep Reinforcement Learning (DRL). For this purpose, the reward function was composed of guidance-theoretic features for path following, obstacle closeness was penalized to induce collision avoidance, and high rotational velocities were penalized to induce stability. A spherical LIDAR suite was implemented for external sensing of obstacles, and a CNN served the purpose of dimensionality reduction through feature extraction to a latent-space encoding. The DRL agent was trained in a synthetic and stochastic environment using the PPO algorithm and a curriculum learning approach. Ultimately, the method resulted in an autonomous DRL agent with high path following performance and room for improvement in collision avoidance.

Will the resulting DRL controller with the PPO hyperparameters from Havenstrøm et al. (2021) and Sundøen et al. (2021) achieve path following and collision avoidance?

The resulting DRL controller was successful in scenarios without obstacles but struggled in scenarios with obstacles. In Section 4.1, we showed the resulting agent's behavior, both when successful and unsuccessful at reaching its goal. In Section 4.2, we saw a high failure rate where approximately half of the failures were crashes. The hyperparameters from Havenstrøm et al. (2021) and Sundøen et al. (2021) were at least a good starting point that has translated well into this new problem formulation.

Can we exploit the dimensionality-reducing property in convolutional neural networks for encoding LIDAR observations?

Introducing CNNs should allow for feature reduction with minimal loss of information. Our results show that CNNs seem to be a viable solution. Nevertheless, some changes are proposed to increase the performance. We observed that the agent was able to react to obstacles when they were in the path of the quadcopter while ignoring them when they were out of the way.

The lack of collision avoidance performance could be a tuning problem. The feature extractor was set up with a kernel size of 2. The tiny kernel size may have led to the features not fitting inside the kernels, decreasing the feature extractor performance. Increasing the kernel and padding size could lead to an improvement.

A more sophisticated CNN architecture could be applied. A limitation of the feature map output of convolutional layers is that they record the precise position of obstacle features in the input; small movements in the position of an obstacle feature in the input LIDAR data will result in a different feature map. Commonly, the problem is solved by downsampling using pooling layers. Pooling have seen use in many similar applications (He et al. (2019), Zhang et al. (2020)).

Will the resulting DRL controller generalize into unseen scenarios?

The unseen scenarios serve as the test set. Using the unseen scenarios, we can observe whether the agent has overfitted onto the training scenarios or it has generalized. The unseen test scenarios were generated with a considerable domain gap to the training scenarios in both path and obstacles, while the dynamics stayed the same. The generalization is good in the scenarios where the agent could employ similar evasion strategies that work for the training scenarios, but the agent does not generalize otherwise. The failure to generalize is most apparent in the dead-end scenario. The helical one is surprising and combined with dead-end might suggest that the agent overly favors path following. Additionally, the failure rate in the scenario with vertically stacked obstacles was $0,00$, whereas, in the training scenario, it was 0.65. When the obstacles were stacked horizontally, the agent crashed in every episode. It seems like the agent learned to dodge obstacles, but only by moving in the horizontal plane.

The solution could be to implement a more diverse set of training scenarios. In the training, the obstacles were only placed on the path. We propose to randomize the placement of obstacles within the vicinity of the path,

5.2. Further Work

Despite the demonstrated advantages of the DRL approach, we should stress that the methodology relies on a black-box Deep Neural Network (DNN) to learn the optimal policy. If DRL-based controllers are to used for quadcopters, safety guarantees will become a prerequisite. Implementing and analyzing a similar setup as in this paper using symbolic regression Kubalík et al. (2019) could give a better understanding of the underlying behavior and reasoning of the controller.

The collision avoidance performance of the DRL controller can be improved. A solution could be to employ a higher-level action space, as seen in Kaufmann et al. (2022) to increase the performance potential. Another solution could be to employ continuous curriculum learning to expose the controller to obstacles earlier in training with less domain gap. Additionally, the CNN kernel size could be increased or the architecture modified.

Furthermore, our motivation for arguing in favor of DRL was that it is model-free. However, we utilized a model in this study because the training was done in a purely synthetic environment. For the training to be genuinely model-free, we will have to train the agent in real settings. There are several problems with this. First of all, a back-of-the-envelope calculation in Appendix A reveals that it would require 17 days of continuous training if we only had one quadcopter in real settings. One possibility is to use transfer learning, where the model is first trained in a simulated environment, and then the training continues in the physical world. Another challenge with training in real settings is the possibility of the quadcopter crashing. Therefore, real-world conditions training can be unsafe and have considerable costs in continuously repairing the quadcopter. A solution could be to employ a predictive safety filter. The filter works by evaluating the proposed control input and then deciding if it is safe to apply.

While a true model-free approach is currently infeasible, the DRL framework is highly accommodating to combining existing model-based predictive models and low-level controllers with high-level model-free control. Going away from end-to-end may restrict the performance ceiling but accelerate learning and facilitate safe operation.

Bibliography

Md. Zahangir Alom, Tarek Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Nasrin, Mahmudul Hasan, Brian Essen, Abdul Awwal, and Vijayan Asari. A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8:292, 03 2019. doi: 10.3390/electronics8030292.

AlphaStar. Alphastar: Mastering the real-time strategy game starcraft ii. `https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii`, 2019. Accessed: 2021-12-15.

Bassam Alrifaee, Kevin Kostyszyn, and Dirk Abel. Model predictive control for collision avoidance of networked vehicles using lagrangian relaxation**this work is supported by mobilem, which is funded by deutsche forschungsgemeinschaft (dfg). *IFAC-PapersOnLine*, 49(3):430–435, 2016. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2016.07.072. URL `https://www.sciencedirect.com/science/article/pii/S2405896316302683`. 14th IFAC Symposium on Control in Transportation SystemsCTS 2016.

Sumeet Batra, Zhehui Huang, Aleksei Petrenko, Tushar Kumar, Artem Molchanov, and Gaurav S. Sukhatme. Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning, 2021. URL `https://arxiv.org/abs/2109.07735`.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL `https://doi.org/10.1145/1553374.1553380`.

Eivind Bøhn, Erlend M Coates, Signe Moe, and Tor Ame Johansen. Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 523–533. IEEE, 2019.

J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1187, 1989. doi: 10.1109/21.44033.

Morten Breivik and Thor Fossen. *Guidance Laws for Autonomous Underwater Vehicles*, pages 70–71. IntechOpen, 01 2009. ISBN 978-953-7619-49-7. doi: 10.5772/6696.

Seong-Ryong Chang and Uk-Youl Huh. A collision-free g2 continuous path-smoothing algorithm using quadratic polynomial interpolation. *International Journal of Advanced Robotic Systems*, 11(12):194, 2014. doi: 10.5772/59463. URL `https://doi.org/10.5772/59463`.

Seong-Ryong Chang and Uk-Youl Huh. Curvature-continuous 3d path-planning using qpmi method. *International Journal of Advanced Robotic Systems*, 12(6):76, 2015. doi: 10.5772/60718. URL `https://doi.org/10.5772/60718`.

Ruaridh A Clark, Giuliano Punzo, Charles N MacLeod, Gordon Dobie, Rahul Summan, Gary Bolton, Stephen G Pierce, and Malcolm Macdonald. Autonomous and scalable control for remote inspection with multiple aerial vehicles. *Robotics and Autonomous Systems*, 87:258–268, 2017.

Oguz H Dagci, Umit Y Ogras, and U Ozguner. Path following controller design using sliding mode control theory. In *Proceedings of the 2003 American Control Conference, 2003.*, volume 1, pages 903–908. IEEE, 2003.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602:414–419, 02 2022. doi: 10.1038/s41586-021-04301-9.

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Bjørn-Olav H Eriksen, Morten Breivik, Erik F Wilthil, Andreas L Flåten, and Edmund F Brekke. The branching-course model predictive control algorithm for maritime collision avoidance. *Journal of Field Robotics*, 36(7):1222–1249, 2019.

Jalel EUCHI. Do drones have a realistic place in a pandemic fight for delivering medical supplies in healthcare systems problems? *Chinese Journal of Aeronautics*, 34(2): 182–190, 2021. ISSN 1000-9361. doi: https://doi.org/10.1016/j.cja.2020.06.006. URL `https://www.sciencedirect.com/science/article/pii/S100093612030279X`.

Timm Faulwasser. *Optimization-based solutions to constrained trajectory-tracking and path-following problems*. "Shaker Verlag GmbH", 01 2013. ISBN 978-3844015942. doi: 10.2370/9783844015942.

Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998. doi: 10.1177/027836499801700706. URL `https://doi.org/10.1177/027836499801700706`.

Thor Fossen, Morten Breivik, and Roger Skjetne. Line-of-sight path following of underactuated marine craft. 09 2003. doi: 10.1016/S1474-6670(17)37809-6.

D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, 1997. doi: 10.1109/100.580977.

Md Haque, Muztahid Muhammad, Dipayan Swarnaker, and Md Arifuzzaman. Autonomous quadcopter for product home delivery. In *2014 International Conference on Electrical Engineering and Information Communication Technology*, pages 1–5. "IEEE", 04 2014. doi: 10.1109/ICEEICT.2014.6919154.

Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/tssc.1968.300136. URL https://doi.org/10.1109/tssc.1968.300136.

Simen Theie Havenstrøm, Adil Rasheed, and Omer San. Deep reinforcement learning controller for 3d path following and collision avoidance by autonomous underwater vehicles. *Frontiers in Robotics and AI*, 7:211, 2021. ISSN 2296-9144. doi: 10.3389/frobt.2020.566037. URL https://www.frontiersin.org/article/10.3389/frobt.2020.566037.

Xin He, Aili Wang, Pedram Ghamisi, Guoyu Li, and Yushi Chen. Lidar data classification using spatial transformation and cnn. *IEEE Geoscience and Remote Sensing Letters*, 16(1):125–129, 2019. doi: 10.1109/LGRS.2018.2868378.

Sunan Huang, Rodney Swee Huat Teo, and Kok Kiong Tan. Collision avoidance of multi unmanned aerial vehicles: A review. *Annual Reviews in Control*, 48:147–164, 2019.

Uk-Youl Huh and Seong-Ryong Chang. A g2 continuous path-smoothing algorithm using modified quadratic polynomial interpolation. *International Journal of Advanced Robotic Systems*, 11(2):25, 2014. doi: 10.5772/57340. URL https://doi.org/10.5772/57340.

Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *CoRR*, abs/1707.05110, 2017. URL http://arxiv.org/abs/1707.05110.

Idun, 2022. URL https://www.hpc.ntnu.no/.

Luc Jaulin. 2 - feedback linearization. In Luc Jaulin, editor, *Mobile Robotics*, pages 45–100. Elsevier, 2015. ISBN 978-1-78548-048-5. doi: https://doi.org/10.1016/B978-1-78548-048-5.50002-4. URL https://www.sciencedirect.com/science/article/pii/B9781785480485500024.

Shashmi Karanam. Curse of Dimensionality — A "Curse" to Machine Learning, 2021. URL https://towardsdatascience.com/curse-of-dimensionality-a-curse-to-machine-learning-c122ee33bfeb.

Elia Kaufmann, Leonard Bauersfeld, and Davide Scaramuzza. A benchmark comparison of learned control policies for agile quadrotor flight. *EEE International Conference on Robotics and Automation (ICRA)*, 2022. doi: 10.48550/ARXIV.2202.10796. URL https://arxiv.org/abs/2202.10796.

Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986. doi: 10.1177/027836498600500106. URL https://doi.org/10.1177/027836498600500106.

Jinhyun Kim, Min-Sung Kang, and Sangdeok Park. *Accurate Modeling and Robust Hovering Control for a Quad-rotor VTOL Aircraft*, pages 9–26. Springer Netherlands, Dordrecht, 2010. ISBN 978-90-481-8764-5. doi: 10.1007/978-90-481-8764-5_2. URL https://doi.org/10.1007/978-90-481-8764-5_2.

Jirí Kubalík, Jan Zegklitz, Erik Derner, and Robert Babuska. Symbolic regression methods for reinforcement learning. *CoRR*, abs/1903.09688, 2019. URL http://arxiv.org/abs/1903.09688.

Emmett Lalish and K Morgansen. *Distributed reactive collision avoidance for multivehicle systems*. PhD thesis, Citeseer, 2009.

Thomas Nakken Larsen, Halvor Ødegård Teigen, Torkel Laache, Damiano Varagnolo, and Adil Rasheed. Comparing deep reinforcement learning algorithms' ability to safely navigate challenging waters. *Frontiers in Robotics and AI*, 8:287, 2021. ISSN 2296-9144. doi: 10.3389/frobt.2021.738113. URL https://www.frontiersin.org/article/10.3389/frobt.2021.738113.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2016. URL http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15.

Mebaye Mamo. Trajectory control of quadcopter by designing second order smc controller. *Journal of Electrical Engineering*, 7:10, 01 2021.

Andreas B. Martinsen and Anastasios M. Lekkas. Straight-path following for underactuated marine vessels using deep reinforcement learning. *IFAC-PapersOnLine*, 51(29):329–334, 2018a. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2018.09.502. URL https://www.sciencedirect.com/science/article/pii/S2405896318321918. 11th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2018.

Andreas B. Martinsen and Anastasios M. Lekkas. Curved path following with deep reinforcement learning: Results from three vessel models. In *OCEANS 2018 MTS/IEEE Charleston*, pages 1–8, 2018b. doi: 10.1109/OCEANS.2018.8604829.

Eivind Meyer, Haakon Robinson, Adil Rasheed, and Omer San. Taming an autonomous surface vehicle for path following and collision avoidance using deep reinforcement learning. *IEEE Access*, 8:41466–41481, 2020. doi: 10.1109/ACCESS.2020.2976586.

A. Micaelli and C. Samson. Trajectory tracking for two-steering-wheels mobile robots. *IFAC Proceedings Volumes*, 27(14):249–256, 1994. ISSN 1474-6670. doi: https://doi.org/10.1016/S1474-6670(17)47322-8. URL https://www.sciencedirect.com/science/article/pii/S1474667017473228. Fourth IFAC Symposium on Robot Control, Capri, Italy, September 19-21, 1994.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Signe Moe, Kristin Y. Pettersen, Thor I. Fossen, and Jan T. Gravdahl. Line-of-sight curved path following for underactuated usvs and auvs in the horizontal plane under the influence of ocean currents. In *2016 24th Mediterranean Conference on Control and Automation (MED)*, pages 38–45, 2016. doi: 10.1109/MED.2016.7536018.

Manfred Morari and Jay H. Lee. Model predictive control: past, present and future. *Computers & Chemical Engineering*, 23(4):667–682, 1999. ISSN 0098-1354. doi: https://doi.org/10.1016/S0098-1354(98)00301-9. URL https://www.sciencedirect.com/science/article/pii/S0098135498003019.

Derek R. Nelson, D. Blake Barber, Timothy W. McLain, and Randal W. Beard. Vector field path following for miniature air vehicles. *IEEE Transactions on Robotics*, 23(3): 519–529, 2007. doi: 10.1109/TRO.2007.898976.

Chen-Huan Pi, Wei-Yuan Ye, and Stone Cheng. Robust quadrotor control through reinforcement learning with disturbance compensation. *Applied Sciences*, 11(7), 2021. ISSN 2076-3417. doi: 10.3390/app11073257. URL `https://www.mdpi.com/2076-3417/11/7/3257`.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

Ashton Roza and Manfredi Maggiore. Path following controller for a quadrotor helicopter. In *2012 American Control Conference (ACC)*, pages 4655–4660. IEEE, 2012.

Bartomeu Rubi, Ramon Pérez, and Bernardo Morcego. A survey of path following control strategies for uavs focused on quadrotors. *Journal of Intelligent & Robotic Systems*, 98(2):241–265, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016. doi: 10.1038/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

SNAME. *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. URL `https://books.google.no/books?id=VqNFGwAACAAJ`.

P.B. Sujit, Srikanth Saripalli, and Joao Borges Sousa. Unmanned aerial vehicle path following: A survey and analysis of algorithms for fixed-wing unmanned aerial vehicless. *IEEE Control Systems Magazine*, 34(1):42–59, 2014. doi: 10.1109/MCS.2013.2287568.

Ludvig Løken Sundøen, Adil Rasheed, and Thomas Nakken Larsen. Path following for quadcopters using deep reinforcement learning. *unpublished*, 2021.

Velodyne. Velodyne Puck Spec Sheet, 2022. URL `https://www.amtechs.co.jp/product/VLP-16-Puck.pdf`.

C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Oxford, 1989.

K David Young and Sergey V Drakunov. Sliding mode control with chattering reduction. In *1992 American Control Conference*, pages 1291–1292. IEEE, 1992.

Shuyou Yu, Xiang Li, Hong Chen, and Frank Allgöwer. Nonlinear model predictive control for path following problems. *IFAC Proceedings Volumes*, 45(17):145–150, 2012. ISSN 1474-6670. doi: https://doi.org/10.3182/20120823-5-NL-3013.00003. URL `https://www.sciencedirect.com/science/article/pii/S1474667016314410`. 4th IFAC Conference on Nonlinear Model Predictive Control.

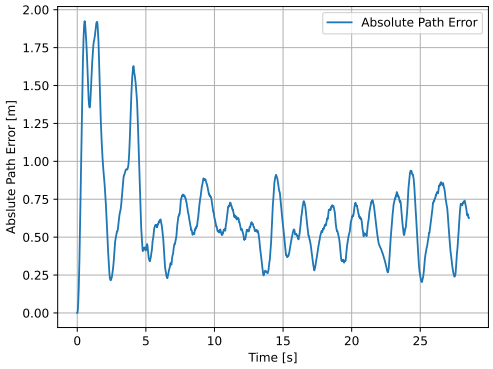
Mengmeng Zhang, Wei Li, Qian Du, Lianru Gao, and Bing Zhang. Feature extraction for classification of hyperspectral and lidar data using patch-to-patch cnn. *IEEE Transactions on Cybernetics*, 50(1):100–111, 2020. doi: 10.1109/TCYB.2018.2864670.

A. Appendix

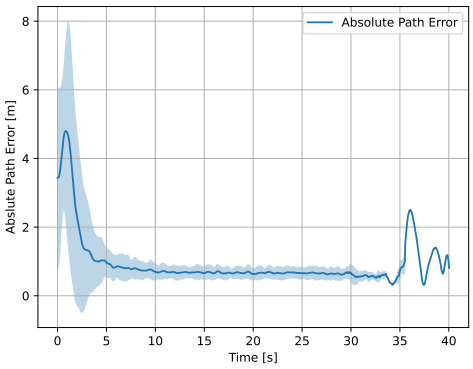
Table A.1.: Failure rate and collision rate in Simple 1D, 1D and 2D scenarios over 100 episodes.

Scenario	Failure Rate	Collision Rate	Median APE
Simple 1D	0.26	N/A	$0.58m$
1D	0.00	N/A	$0.68m$
2D	0.00	N/A	$0.87m$

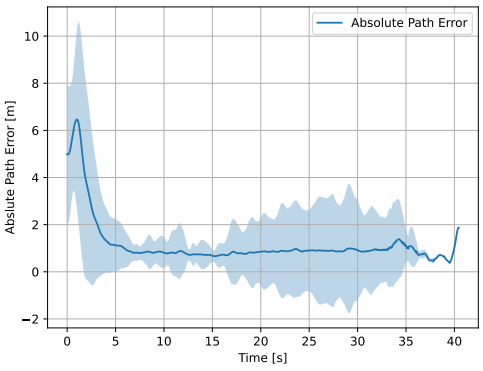
$$training\_days = \frac{timesteps\_total \cdot step\_size}{60^2 \cdot 24} = \frac{150 \cdot 10^6 \cdot 0.01}{60^2 \cdot 24} \approx 17 \text{ days} \tag{A.1}$$



(a) Simple 1D scenario

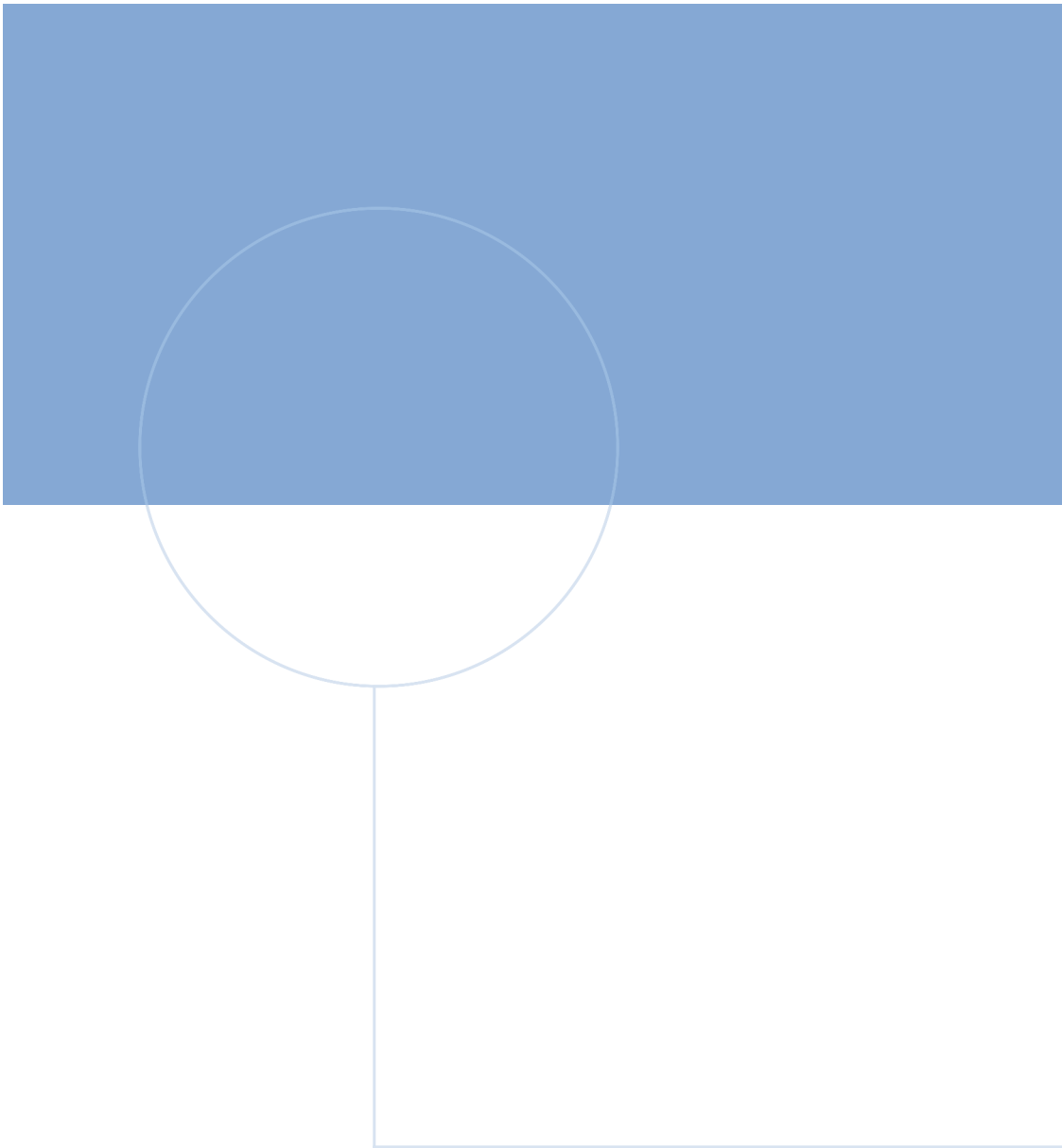


(b) 1D scenario



(c) 2D scenario

Figure A.1.: The absolute path error over 100 episodes each in the Simple 1D, 1D and 2D scenario. The blue shadowed area is the standard deviation of the error.



NTNU

Norwegian University of
Science and Technology