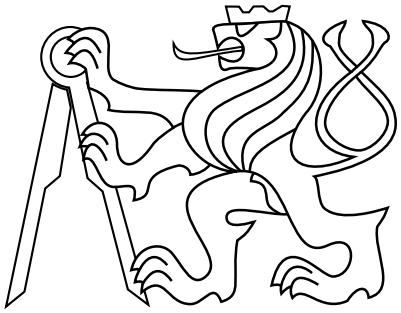
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics
Multi-robot Systems



Learning High-Speed Flight of Unmanned Aerial Vehicle in Cluttered Environments

Bachelor's Thesis

Vít Knobloch

Prague, May 2023

Study programme: Open Informatics
Specialization: Artificial Intelligence and Computer Science

Supervisor: Ing. Robert Pěnička, Ph.D.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Knobloch Vít** Personal ID number: **498947**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Learning High-Speed Flight of Unmanned Aerial Vehicle in Cluttered Environments

Bachelor’s thesis title in Czech:

Učení vysokorychlostního letu autonomního vzdušného robotu v prostředí s překážkami

Guidelines:

1) Analyze existing research in reinforcement learning and other significant approaches for high-speed flight of unmanned aerial vehicles in cluttered environments.
2) Implement simple simulation of quadrotor unmanned aerial vehicle. Connect the simulator with suitable reinforcement learning library to teach autonomous flight.
3) Create a set of training environments cluttered with obstacles to teach the reinforcement learning flight policy.
4) Learn perception aware flight in known training environment and experiment with reward and observation.
5) Compare the proposed method with existing approaches.
6) Try to learn a generalizing policy and evaluate the policy’s performance in both the training environments and in an unknown environments.

Bibliography / sources:

[1] Yunlong Song, Kexin Shi, Robert Penicka, and Davide Scaramuzza. Learning perception-aware agile flight in cluttered environments, arXiv, https://doi.org/10.48550/arxiv.2210.01841, 2022.
[2] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, Davide Scaramuzza, Learning High-Speed Flight in the Wild, AAAS, 2021.
[3] Yunlong Song, Mats Steinweg, Elia Kaufmann, Davide Scaramuzza, Autonomous Drone Racing with Deep Reinforcement Learning, IEEE IROS, 2021.
[4] Robert Penicka, Yunlong Song, Elia Kaufmann Davide Scaramuzza, Learning minimum-time flight in cluttered environments. IEEE Robotics and Automation Letters, 2022.

Name and workplace of bachelor’s thesis supervisor:

Ing. Robert Pěnička, Ph.D. Multi-robot Systems FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **24.01.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Robert Pěnička, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student's signature

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Vít Knobloch

May 24, 2023 in Prague

Acknowledgments

Firstly, I would like to thank my supervisor for his guidance, helpfulness, and kindness with which he led my work throughout the last year. Secondly, I would like to thank my parents and my family, who has supported me in everything my whole life and as well through my university studies. I would also like to thank my classmates who have made all the studying a bit easier and a lot more fun for me. The access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" is also gratefully acknowledged.

Abstract

Unmanned Aerial Vehicles (UAVs) and particularly quadrotors have become increasingly popular in recent years. Their high versatility allows their deployment in many tasks such as infrastructure inspection, farmland monitoring, delivery, and reconnaissance. Consequently, their autonomous capabilities are being researched to decrease the need for human supervision. Many of the tasks where quadrotors are used require high-speed flight such as search and rescue, cinematography, protected airspace guarding, and more. This thesis describes a method using policy trained with deep reinforcement learning to perform perception-aware high-speed flight of a quadrotor in simulated environments cluttered with obstacles. A computationally efficient learning environment is presented and paired with an existing reinforcement learning library to train the policy. The approach is tested in several environments on a number of tracks and the resulting behavior is described. The policy is able to control the quadrotor flight through multiple waypoints in a cluttered environment while using depth sensor information to avoid collisions.

Keywords Unmanned aerial vehicles, Automatic control, Simultaneous planning and control, Deep reinforcement learning, Quadrotor, Learning environment, Simulator, High-speed flight, Perception-aware flight

Abstrakt

Bezpilotní vzdušné prostředky a zejména kvadrokoptéry se v posledních letech začaly hojně využívat, jejich obratnost a všestranost z nich dělají ideální prostředky pro výkon nejrůznějších úkolů jako inspekce, mapování a prohledávání terénu, monitorování různých prostor, hlídání střeženého vzdušného prostoru a mnoho dalších. V návaznosti na to je prováděn rozsáhlý výzkum v oblasti jejich autonomních schopností, aby bylo možné tyto úkoly automatizovat. Pro mnoho z těchto úloh je nezbytná schopnost dronu pohybovat se rychle ve složitých prostředích. Tato práce popisuje metodu využívající hluboké posilované učení k naučení agenta schopného ovládat vysokorychlostní let kvadrotoru v simulovaném prostředí s překážkami. Bylo vytvořeno trénovací prostředí a napojeno na existující knihovnu s implementacemi algoritmů hlubokého posilovaného učení. Chování naučeného agenta je otestováno v několika prostředích a na různých trasách. Naučený agent je schopný ovládat dron při letu vysokou rychlostí ve složitých prostředích skrze několik zadaných cílů a využívat data ze simulovaného hloubkového senzoru, aby se vyhnul překážkám.

Klíčová slova Bezpilotní prostředky, Automatické řízení, Současné plánování a řízení, Hluboké posilované učení, Kvadrokoptéra, Trénovací prostředí, Simulátor, Vysokorychlostní let, Let s vědomým vnímáním

Abbreviations

FOV field of view

GPS global positioning system

LiDAR light detection and ranging

UAV unmanned aerial vehicle

PPO proximal policy optimization

TRPO trust region policy optimization

KL Kullback–Leibler divergence

MDP Markov decision process

POMDP partially observable Markov decision process

RL reinforcement learning

DRL deep reinforcement learning

CNN convolutional neural network

ReLU rectified linear unit

FW fixed-winged

RW rotary-winged

VTOL vertical takeoff and landing

SDF signed distance field

LV linear velocity and yaw rate

CTBR collective thrust and bodyrate

SRT single-rotor thrust

RCI Research Center for Informatics

Contents

Chapter 1

Introduction

Unmanned aerial vehicles (UAVs) became very popular over the course of the twenty-first century with an ever-increasing amount of research into their hardware and autonomous capabilities. Conventionally UAVs are telemetry monitored and operated by a pilot on the ground, either nearby or far away [2]. Lately, more focus is put on the development of autonomous control and planning algorithms to limit the need for human supervision. First UAVs have been large aircraft with military applications. But with the advancement of motors, GPS receivers, telemetry sensors, and microcontrollers, new smaller UAVs have been created and found applications in many areas including photography and filmmaking, infrastructure inspection, recreation, search-and-rescue, express delivery, and surveillance [2].

There are two main types of UAVs: fixed-winged (FW) and rotary-winged (RW). Fixed-winged design is simpler and more energy efficient and generally capable of carrying larger payloads. However, FW UAVs are incapable of vertical takeoff and landing (VTOL) and have to move forward in order to retain lift, this makes them impractical or unusable for many use cases. Rotary-winged UAVs have rotors that provide thrust, they are more agile, able to take off and land vertically, and hover in one place. Even though RW UAVs are less energy efficient, their advantages have made them the most popular design [2].

The thesis is about the use of reinforcement learning for simultaneous planning and control of perception-aware high-speed flight of an autonomous quadrotor with an onboard depth sensor. We say the agent is perception-aware when it actively uses sensor data about the surrounding environment to gain situational awareness. An introductory video showing some key components of our approach is available in Appendix A and on YouTube[1]. A quadrotor is a rotary-winged UAV with four rotors. Opposing pairs of rotors rotate in the same direction while neighboring rotors rotate in opposite directions. This allows for easy attitude control as yaw, pitch, and roll can be controlled by just changing the rotational speeds of individual rotors. Quadrotors are widely used and probably the most popular and researched design of UAVs. Planning and control for high-speed flight of autonomous quadrotors in unknown environments remains an open problem despite a lot of effort put into the research. Physical models have limited accuracy, and so do sensors onboard the quadrotor. The computational resources available online onboard are also limited and more computation means less battery life and as a result less flight time. Cluttered environments and fast flight pose another difficulty as such navigation requires highly agile maneuvering.

Existing solutions, that are efficient enough to compute onboard in flight, mostly use polynomial trajectory planning [11] or imitation learning [1], [7]. Polynomial trajectories are suboptimal for minimum-time flight because of the inherent smoothness of polynomials. Imitation learning has shown promising results when a sampling-based teacher was used in [7] to train a neural policy network to fly in unknown environments at speeds unreachable by polynomial trajectories. Most of the existing methods output a trajectory to be followed by

[1]Introductory video on YouTube: `https://youtu.be/7mR8IvWtQWU`

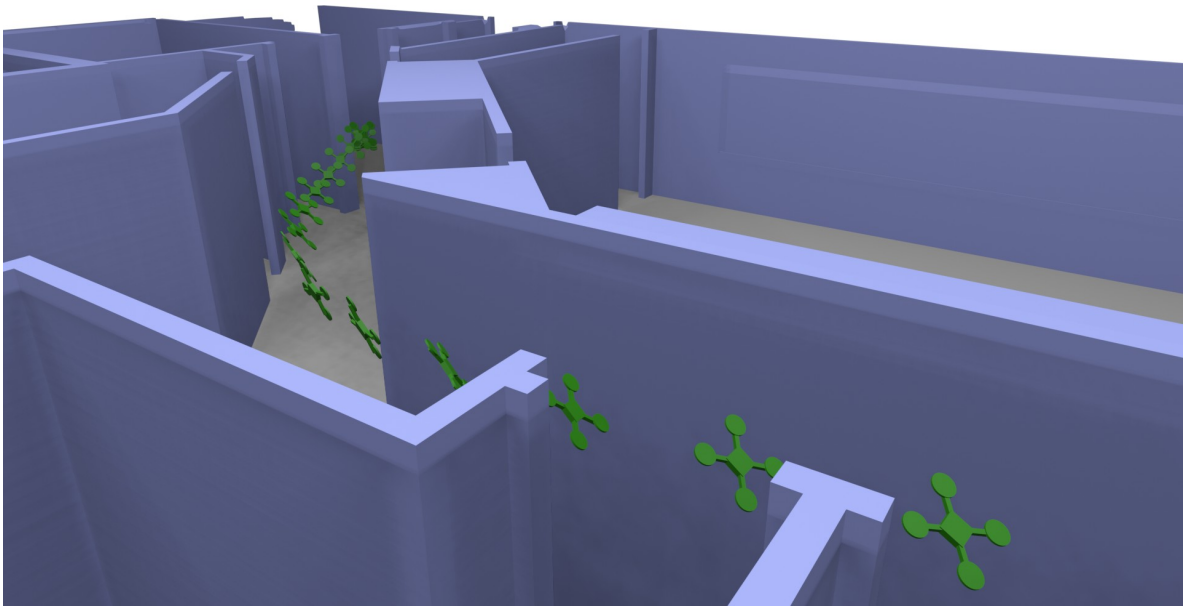


Figure 1.1: Render of a flight of a quadrotor controlled by our policy through a narrow hallway.

a high-level controller (such as MPC). The reinforcement learning approach presented in this thesis uses strictly reinforcement learning to train a policy that produces collective thrust and body rate commands [3] to perform the same task, needing only a low-level controller (such as PID) to follow the commands. High-level controllers need a full state estimation to work and are prone to in-flight disturbances and model mismatches [19].

Conducting trajectory planning and control simultaneously leads to better robustness against disturbances and mismatches, which might be the key to advancing the field of high-speed drone flight as these problems become more significant with increasing velocity. A new learning environment is presented in this thesis. The development of the learning environment allowed for the use of deep reinforcement learning. The experiments show the ability of RL to plan and control the drone through various environments if it was trained in those environments. A render of the trajectory from one of the experiments is shown in Figure 1.1. Furthermore, the results describe the performance of the policy in unknown environments. With further advancement of the learning environments and policy network structures, it is possible that reinforcement learning will replace imitation learning as the state-of-the-art method for high-speed perception-aware flight of quadrotor UAVs through environments with obstacles in the future.

Chapter 2

Related works

This thesis builds upon earlier works that suggest using deep reinforcement learning [5] or supervised learning [1], [7] algorithms for the problem of high-speed UAV flight. The authors compare their approach with other approaches commonly used for the problem. These include sampling-based methods and methods using smooth trajectory representation (polynomial or B-spline). The comparison shows that learned policy can outperform these methods in terms of flight execution and computational complexity and is a promising area of research.

Standard (non-learning) solutions break the problem down to perception, planning, and control [10]. This creates an easily interpretable pipeline but increases perception-to-control latency and compounds error in between the pipeline's stages [7]. Both of these factors get more significant with increasing velocity, which makes safe navigation progressively harder. Perception is commonly performed by using a depth camera and can be done either by interpreting a single depth image [11] or by building a 3D map from consequent depth images [10]. Using a single-depth image constrains the planned trajectories to the depth sensor's field of view (FOV) or treats the unseen space as safe, which can lead to fatal crashes, especially at high speeds. Building a 3D map of the environment can leverage knowledge from previous observations, but comes at a high computational cost which usually makes it unusable for high-speed flight.

Planning a trajectory using B-splines or polynomials is computationally effective as it exploits the differential flatness of quadrotor systems [21]. However, polynomial curves are inherently smooth, and change in control outputs is limited by their order. This property results in sub-optimal trajectories which don't use the full actuation potential of the quadrotor [6]. Sampling-based methods use the discretization of time-space and graph search algorithms to find near-optimal trajectories. They can use the actuators to their limits but are computationally demanding and cannot be carried out onboard in flight [6]. The controllers used to carry out the trajectories planned by standard methods are prone to error due to disregarding model mismatch [19]. If they can correct for the mismatch [19] they are still unaware of obstacles, which can lead to potential crashes when adjusting control outputs to get back on the planned trajectory. Several fast replanning algorithms have been developed to plan a new trajectory in time for the controller's tracking error to not be fatal. RAPPIDS [11] efficiently represents free space using pyramid shapes which allows for fast time-limited local replanning. RAPTOR [10] combines online topological path planning with several heuristically driven optimization techniques to replan a trajectory in a way that ensures unseen obstacles are observed in time for the quadrotor to avoid a fatal crash. Both of these methods limit the maximal velocity of the quadrotor due to the substantial system latency.

The learning approach addresses the issue of model mismatch by using a trained policy that combines perception, planning, and control. This allows for a very fast control loop period in order of milliseconds and flexible correction of errors introduced by model mismatch [5]. The RL based approach in [5] outperforms both of the standard approaches in terms of calculation

time and can exploit the full actuation potential. The downside to using deep learning is the bad interpretability of the solution and the sample-complex training. The model can only be trained in simulation as doing it on real hardware would be too slow and would endanger the hardware. Pěnička *et al.* [5] show that policy trained in simulation can accurately control real hardware. Song's *et al.* [1] policy flights real hardware with simulated perception. Both [5] and [1] are overfitted to the training environment, but Loquercio *et al.* [7] use policy trained entirely in simulation to safely navigate previously unknown real-world environments.

Pěnička *et al.* [5] combine DRL policy with topological pathfinding. A set of topologically distinct collision-free paths is found in the environment. Then a suitable guiding path is selected and a DRL model is trained to find and execute a minimum-time trajectory along it. While the model is overfitted to the environment and uses privileged information (the agent's absolute position) in observation, the results show the ability of a policy to outperform standard methods in terms of reliability and flight speed thanks to combining trajectory planning with flight control.

Song *et al.* [1] use imitation learning with teacher policy trained using principles introduced in [5] to train a perception-aware student policy to plan and execute a minimum-time trajectory in a known environment. The learned student policy tightly couples perception using a depth camera with control. The depth camera image is first encoded with a pre-trained neural encoder to reduce its inherently high dimension, then it is combined with quadrotor state information to form the observation. The results show that the student policy is reliably executing a high-speed trajectory, matching the performance of the teacher policy. It is shown using hardware-in-the-loop simulation that the trained model can be executed on real hardware and deal with in-flight disturbances and model mismatch errors.

Loquercio *et al.* [7] trained a perception-aware student policy by imitating a sampling-based teacher. The policy outputs a set of promising trajectories along with an estimate of collision risk, and the one with the least risk is selected to be followed by model predictive control (MPC). The policy uses an in-training learned convolutional neural network (CNN) to extract information from a noisy depth camera, partial state information available from onboard sensors, and a global reference path that need not be collision-free. The three best trajectories from the teacher are used in the training to acknowledge the possibility of multiple distinct feasible and near-optimal trajectories, for example, going left or right around an obstacle. Although the training is performed solely in simulation, their simulation of sensor noise, motion blur, and other disturbances is shown to be similar enough to enable zero-shot transfer from simulation to the real world. The results show the deep learning approach to be capable of outperforming standard methods in simulated and real unknown environments in control latency, reliability, and flight speed.

Chapter 3

Preliminaries

In this chapter, several topics related to the problem are introduced. The dynamic model of a quadrotor as used in the simulation is defined. A general reinforcement learning problem is defined as a partially observable Markov decision process and the concept of reinforcement learning and deep reinforcement learning is introduced. Proximal policy optimization is described as it was used in the training of the policies in this thesis. Lastly, the time-optimal collision-free trajectory is defined because the goal of the thesis is to train policies capable of planning and controlling quadrotor movement in a near-time-optimal manner through cluttered environments while avoiding collisions. The policies are evaluated based on the trajectory they generate flying through an environment.

3.1 Quadrotor's physics model

The physics model is described with differential equations, the state of the quadrotor, commanded rotational speeds of the rotors, and constants describing the physical properties of the quadrotor. The input to the dynamic model is the commanded rotational speeds of individual rotors. Articles [1], [5], [6] use a similar dynamic model.

The quadrotor is modeled with its state $x = [\mathbf{p}, \mathbf{q}, \mathbf{v}, \omega, \mathbf{\Omega}]^T$ which consists of position $\mathbf{p} \in \mathbb{R}^3$, velocity $\mathbf{v} \in \mathbb{R}^3$, unit quaternion rotation $\mathbf{q} \in \mathbb{SO}(3)$, body rates $\omega \in \mathbb{R}^3$, and the rotor rotational speed $\mathbf{\Omega} \in \mathbb{R}^4$. Some of the components are illustrated in Figure 3.1.

The dynamic equations are

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{3.1}$$

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \odot \begin{bmatrix} 0 \\ \omega \end{bmatrix}, \tag{3.2}$$

$$\dot{\mathbf{v}} = \frac{R(\mathbf{q})(\mathbf{f_T} + \mathbf{f_D})}{m} + g, \tag{3.3}$$

$$\dot{\omega} = J^{-1}(\tau - \omega \times J\omega), \tag{3.4}$$

$$\dot{\mathbf{\Omega}} = \frac{1}{k_{mot}}(\mathbf{\Omega_c} - \mathbf{\Omega}), \tag{3.5}$$

where $\odot$ denotes the quaternion multiplication, $R(\mathbf{q})$ is the rotation matrix given by quaternion $\mathbf{q}$, $\mathbf{f_T}$ is the thrust force, $\mathbf{f_D}$ is the drag force, m is the quadrotor's mass, g is gravitational acceleration, J is diagonal inertia matrix, τ is torque, k_{mot} is the time constant, and $\mathbf{\Omega_c}$ is the commanded speed.

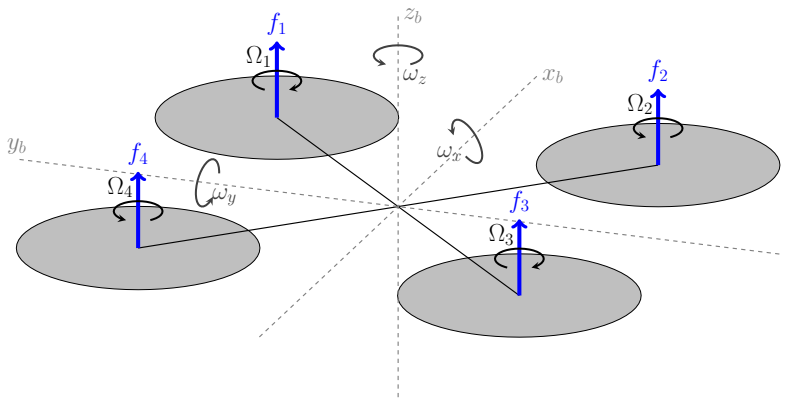


Figure 3.1: Illustration of several components of quadrotor's dynamic state. Quadrotor's body frame axes are x_b, y_b, and z_b. f_i are individual rotor thrusts, Ω_i are angular velocities of individual rotors. Body rates are ω_x, ω_y, and ω_z.

The collective thrust $\mathbf{f_T}$ is calculated as

$$\mathbf{f_T} = \begin{bmatrix} 0 \\ 0 \\ \sum f_i \end{bmatrix}, \tag{3.6}$$

where f_i are individual rotor thrusts, which are functions of rotor speeds Ω_i clamped to the allowed thrust range $[f_{min}, f_{max}]$:

$$f_i(\Omega_i) = c_f \cdot \Omega_i^2, \tag{3.7}$$

where c_f is the thrust coefficient.

The drag force induced by the aerodynamic drag is a linear function of velocity in the quadrotor's axes:

$$\mathbf{f_D} = -\begin{bmatrix} c_{Dx} & 0 & 0 \\ 0 & c_{Dy} & 0 \\ 0 & 0 & c_{Dz} \end{bmatrix} \cdot R(\mathbf{q})^T \cdot \mathbf{v}, \tag{3.8}$$

where c_{Dx}, c_{Dy}, and c_{Dz} are drag coefficients in respective quadrotor axes and $\mathbf{v}$ is the linear speed in world frame ($R(\mathbf{q})^T \cdot \mathbf{v}$ is the linear velocity in body frame).

The torque is calculated as

$$\tau = \begin{bmatrix} \frac{l}{\sqrt{2}}(f_1 - f_2 - f_3 + f_4) \\ \frac{l}{\sqrt{2}}(-f_1 - f_2 + f_3 + f_4) \\ \kappa(f_1 - f_2 + f_3 - f_4) \end{bmatrix}, \tag{3.9}$$

where l is the arm length and κ is the torque constant.

3.2 Partially observable Markov decision processes

Markov decision process (MDP) is a sequential decision process, which is characterized by a quadruple (S, A, P, R). S is a set of states called state space. A is a set of actions called action space. P is a transition probability function where $P(s'|s, a)$ is the probability that the

new state is s' given the action a was performed in state s. R is the reward function where $R(s', s, a)$ is the reward for transitioning from state s to state s' after performing action a. An agent is an algorithm that interacts with the environment in discrete steps. At each timestep, the agent receives the current state and chooses an action. Then the agent receives a reward and a new state. The current state and action taken influence the reward and next state as the next state is chosen from a probability distribution determined by the current state and action taken and the reward is a function of the two subsequent states and the action. The agent's goal is to maximize the received reward by choosing suitable actions. The policy is a function that maps the states to actions. Furthermore, in MDPs the history of states is irrelevant, meaning the transition probability distribution only depends on the current state and not on previous states [22].

In MDPs we assume the agent has full knowledge about the current state, partially observable Markov decision processes (POMDPs) are a generalization of MDPs where the agent doesn't receive the actual state, but only an observation determined by the state which can be identical for different states. The set of observations Ω, observation probability function O, and discount factor γ are added to the MDP quadruple. $O(o|s')$ is the probability of observing observation $o \in \Omega$ in state s'. The discount factor $\gamma \in [0, 1)$ determines how much future rewards influence the current decision. To solve POMDPs optimally the agent has to keep a belief about the current state which is updated at each timestep and depends on previous observations, this drastically increases the problem's complexity and quickly becomes computationally unsolvable for large problems [22].

3.3 Reinforcement learning

Reinforcement learning is an approach for solving POMDPs using learning by doing. The RL agent is learning how to map situations to actions in a way that maximizes the reward. The agent is not told which actions are good, it must discover which actions yield the most reward by trying them. Actions influence the immediate reward but also the next state and subsequently the future rewards. Trial-and-error search and delayed reward are the most important features of reinforcement learning [13]. RL can often lead to suboptimal solutions but it works even for large problems using limited resources.

There are three main components to RL: a policy, a reward signal, and a value function. The policy maps situations (observations/states) to actions, it chooses the best action to take in a certain situation in order to maximize the expected discounted reward. The expected discounted reward is

$$r_{exp} = \mathbb{E}[\sum_{t=0}^{\inf} \gamma^t r_t], \tag{3.10}$$

where r_t is the reward at timestep t and γ is the discount factor. The policy can be deterministic (a single action for the situation) or stochastic (a probability distribution over actions available in the situation). The policy is denoted π and $\pi(a|s)$ is the probability of taking action a in situation s under the policy π. The learned policy is sufficient to determine behavior, meaning the reward signal and value functions are not needed to map a situation to an action. The reward signal defines the goal of the RL problem by rewarding the agent more for better behavior. It is critical to choose an appropriate reward function as optimization of the wrong reward results in the wrong behavior. A state-action pair results in a reward signal. The agent's objective is to maximize the received reward over the long run. The value

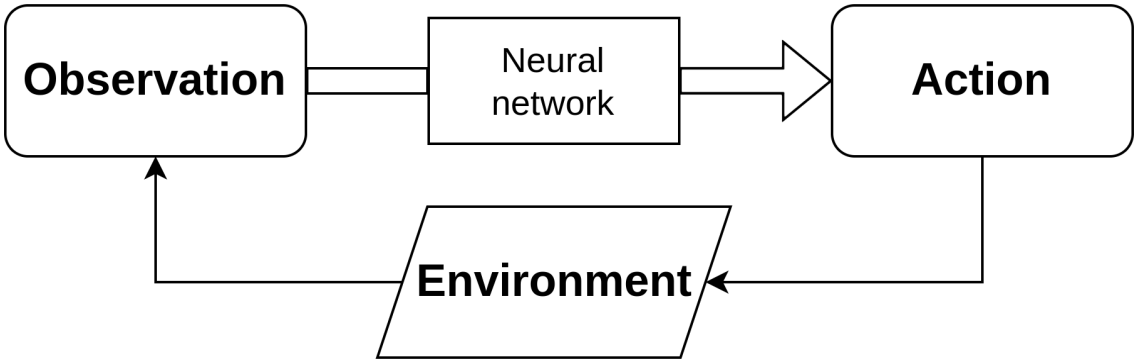


Figure 3.2: Observation-action loop of a DRL policy

function specifies what is good in the long run. The value of a state $V(s)$ is the total reward the agent can expect to accumulate in the future starting from the state [13]:

$$V(s) = \int_{a \in A} \pi(a|s) \int_{s' \in S} P(s'|s,a)[R(s',s,a) + \gamma V(s')]. \tag{3.11}$$

The agent learns iteratively to make better decisions by combining exploration and exploitation. Exploration is taking a random action to determine what is the expected reward when taking that action in that situation. Exploitation is taking currently the best action in the situation. One of the ways to determine the amount of exploration conducted is the learning rate (a higher learning rate means more exploration and less exploitation). When the problem is stationary, meaning the reward function doesn't change, the learning rate is usually lowered throughout the training. This leads to a better exploration of the state space at the beginning of the training and a more refined (possibly optimal) solution at the end of the training. In training the agent observes the state of the environment and chooses an action, then receives a reward and a new observation. Based on the reward the value function is updated and based on the value function the policy is updated. The value and policy update can be done after each step, or after collecting a batch of samples of observation, action, and reward.

When the problem has a large number of possible state-action pairs (or continuous state and/or action space), it is not possible to learn the value of each state or even visit each state in training. In that case, the policy and value functions are usually represented by deep neural networks. The advantage is that neural networks can estimate the values/actions even for unvisited regions of the state space. This subfield of RL is called deep reinforcement learning (DRL). The function of a trained DRL policy is illustrated in Figure 3.2.

3.4 Proximal policy optimization

Proximal policy optimization (PPO) is a type of gradient method to optimize a DRL policy. It is specifically designed to perform well for continuous control problems, such as quadrotor control. It collects a batch of samples by interacting with the environment and performs several epochs of stochastic gradient ascend on the surrogate objective function. The goal is to optimize the expected reward, but a different (surrogate) function is optimized

instead. This can be done because an improvement of the surrogate objective function also improves the actual objective function.

It is similar to trust region policy optimization (TRPO) [18]. TRPO introduces a surrogate loss function to guarantee policy improvement with non-trivial step sizes. The update is subjected to a maximum average Kullback–Leibler (KL) divergence between the policies in sampled states. The KL divergence of policies π_Θ and $\pi_{\Theta_{old}}$ in state s is

$$D_{KL}(\pi_\Theta||\pi_{\Theta_{old}}) = \int_{a\in A} \pi_\Theta(a|s)\log\frac{\pi_\Theta(a|s)}{\pi_{\Theta_{old}}(a|s)}. \tag{3.12}$$

The disadvantages of TRPO are the difficult implementation using approximations of the objective and the constraint in order to perform the update using the conjugate gradient algorithm [14].

PPO limits the divergence of the old and the new policies by introducing a clip on the probability ratio r_t:

$$r_t = \frac{\pi_\Theta(a_t|s_t)}{\pi_{\Theta_{old}}(a_t|s_t)}, \tag{3.13}$$

where π_Θ is the new policy, $\pi_{\Theta_{old}}$ is the policy before update, s_t is the state in timestep t, and a_t is the action taken by the policy in timestep t. The surrogate objective function of PPO is:

$$L^{CLIP}(\Theta) = \hat{\mathbb{E}}_t[\min(r_t(\Theta)\hat{A}_t, \text{clip}(r_t(\Theta), 1-\epsilon, 1+\epsilon)\hat{A}_t], \tag{3.14}$$

where Θ is the vector of policy parameters, $\hat{A}_t$ is the estimation of advantage in timestep t, ϵ is a hyperparameter (usually $\epsilon = 0.2$), and r_t is the probability ratio. The advantage estimation $\hat{A}_t$ with samples collected up to timestep T is calculated

$$\hat{A}_t = -V(s_t) + r'_t + \gamma r'_{t+1} + \cdots + \gamma^{T-t-1} r'_{T-1} + \gamma^{T-t} V(s_T), \tag{3.15}$$

where r'_t is the reward received in timestep t, $V(s)$ is the value of state s, and γ is the discount factor [14]. The advantage A_t is the difference between the expected and the received reward.

The min and clip combination limits the probability ratio when the objective improves, but doesn't limit it when the sample makes the objective worse, this gives more weight to samples in which the policy deteriorated compared to the ones in which it improved. PPO also allows for multiple updates on the same data sample, leading to improved sample efficiency. In comparison with TRPO, the update of PPO is not subject to any constraints and the stochastic gradient ascend algorithm is sufficient for the calculation of the update [14]. The authors of PPO compared their algorithm with other algorithms on several benchmark continuous control problems and show that PPO outperforms the other algorithms on almost all of the benchmarks [14]. Proximal policy optimization remains the state-of-the-art implementation of DRL for continuous control tasks.

3.5 Minimum-time collision-free trajectory

The goal of this thesis is to train a policy using DRL to plan and control quadrotor flight through a multi-waypoint track in a cluttered environment. The quadrotor's movement under the policy generates a trajectory, the aim is to train policy in a way that the generated

trajectory is the minimum-time collision-free trajectory. In this section, the formal continuous and simplified discretized minimum-time collision-free trajectory are defined.

Let's have an environment in three-dimensional space $W = \mathbb{R}^3$ and obstacles taking a part of the environment $W_{obst} \subset W$ [5]. The agent is also a part of the environment. Let's define the space taken by the agent. $A \subset \mathbb{R}^3$ is the set of points that make the agent's body when its position is $[0, 0, 0]^T$ and its rotation is identity. Then the space taken by the agent can be described as $W_a = A(\mathbf{q}, \mathbf{p}) = \{\mathbf{x}' \in W | \mathbf{p} + \mathbf{q} \odot \mathbf{x} = \mathbf{x}', \mathbf{x} \in A\}$, where $\mathbf{q} \in \mathbb{SO}(3)$ is the agent's rotation, $\mathbf{p} \in \mathbb{R}^3$ is its position, and $\odot$ denotes the quaternion rotation. The trajectory of the agent is a continuous function:

$$f : [t_{start}, t_{end}] \to \mathbb{SO}(3) \times \mathbb{R}^3, \tag{3.16}$$

that for each time gives a rotation $\mathbf{q}$ and a position $\mathbf{p}$. The trajectory (or more precisely its derivative) is subject to constraints given by the agent's physics model introduced in Section 3.1. A trajectory is collision-free if it holds:

$$t \in [t_{start}, t_{end}] \implies A(f(t)) \cap W_{obst} = \emptyset. \tag{3.17}$$

The agent in this thesis must reach a certain goal position and pass waypoints in predefined order along the way. Let's add waypoints $G \in W$ to the definition and let's name them $G_1, G_2, \ldots, G_k$. The trajectory is acceptable trajectory if there exist $t_{start} \leq t_1 \leq t_2 \leq \cdots \leq t_k = t_{end}$, for which holds:

$$\mathbf{p_i} \in G_i, f(t_i) = [\mathbf{q_i}, \mathbf{p_i}], \forall i \in \{1, \ldots, k\}. \tag{3.18}$$

The goal of the thesis is to train agents that can fly as fast as possible, in the best case to generate a minimum-time trajectory, which means from all the acceptable trajectories, we want one with the smallest flight time given by $T = t_{end} - t_{start}$.

In practice in the discretized computer simulation, the trajectory is also discretized in the following way. The agent's body is represented as a sphere with radius r moved to the agent's position. We discretize the continuous time into equal-length time steps. A trajectory is then a finite sequence of the agent's states $(s_0, s_1, \ldots, s_n)$. A collision-free trajectory is then one for which holds:

$$\delta(\mathbf{o}, \mathbf{p_i}) > r, \forall i \in \{1, \ldots, n\}, \forall \mathbf{o} \in W_{obst}, \tag{3.19}$$

where $\mathbf{p_i}$ is position in state s_i and $\delta(\mathbf{x}, \mathbf{y})$ is the euclidean distance between points $\mathbf{x}$ and $\mathbf{y}$. The trajectory is still subject to conditions of the physics model described in Section 3.1. The waypoints are circles in W, and passing through a waypoint G is a situation where the traversal between two consequent agent's positions intersects the waypoint in a point $\mathbf{g} \in G$:

$$\mathbf{g} = (1 - u) \cdot \mathbf{p_i} + u \cdot \mathbf{p_{i+1}}, \mathbf{g} \in G, u \in [0, 1], i \in \{1, \ldots, n - 1\}. \tag{3.20}$$

Chapter 4

Learning minimum-time flight

This chapter contains a description of the implemented learning environment (how are environments modeled, how are observations and actions represented, how is the reward calculated, and more), the properties of the reinforcement learning algorithm, a description of the training process for the fitted and generalizing policy, and pictures and descriptions of the environments.

4.1 Learning environment

A learning environment performs the simulation and defines the observation space, the action space, and the reward function. The RL model learns to take actions based on observations that lead to the highest expected reward. The learning environment receives actions, simulates the dynamics of the agent and the environment, and provides observations and rewards to the RL model. The quality of the learning environment directly impacts the quality of the trained policy - a wrong reward function leads to undesirable behavior, unnecessarily complex observation or action space negatively impacts training complexity, and imprecise physical simulation leads to policies unusable outside the simulation.

In cooperation with Karel Poncar who has a related assignment regarding RL for swarms of quadrotors, we developed a framework for learning environments for various reinforcement learning tasks in 3D environments with stationary obstacles. Furthermore, we added a quadrotor model for RL tasks with quadrotors. The topic of this thesis was to use the framework to develop a learning environment for single-quadrotor perception-aware agile flight in cluttered environments. The learning environment is developed in C++ and compiled as a Python package. The C++ part includes dynamics simulation, collision detection, reward calculation, and observation calculation. It provides the standardized OpenAI Gym [15] interface. This interface is supported by several reinforcement learning libraries from which Stable-baselines3 [8] is used in this thesis. The learning environment takes inspiration from the Flightmare simulator [9] (used in [1], [5], [7]) and uses parts of its code related to quadrotor dynamics simulation, which is open-source and available from GitHub[1]. The following subsections describe individual components of the learning environment.

4.1.1 Obstacle representation

The environments cluttered with obstacles are modeled using a signed distance field (SDF). We can take any cluttered environment represented as triangulate meshes saved to a Wavefront OBJ[2] file, e.g. exported from Blender [12], and convert it to SDF. Converting a

[1]Flightmare GitHub repository: `https://github.com/uzh-rpg/flightmare`

[2]Wikipedia article about OBJ file format: `https://en.wikipedia.org/wiki/Wavefront_.obj_file`

mesh to SDF is a complex operation that takes a long time, it is done once ahead of time and the resulting SDF is saved to a file, which is then loaded by the learning environment. The drawback of this approach is that we can't model dynamic obstacles and we can't add or remove obstacles in runtime either. The benefit is that we can check for collisions in constant time. We could check for collisions using mesh representations directly, but it is computationally demanding and becomes harder with the increasing count and geometrical complexity of obstacles. The SDF collision detection increased the simulation speed of our learning environment around tenfold. Since the policies used in this thesis were trained on hundreds of millions of simulation steps, the constant time collision detection shortened training time from days to hours.

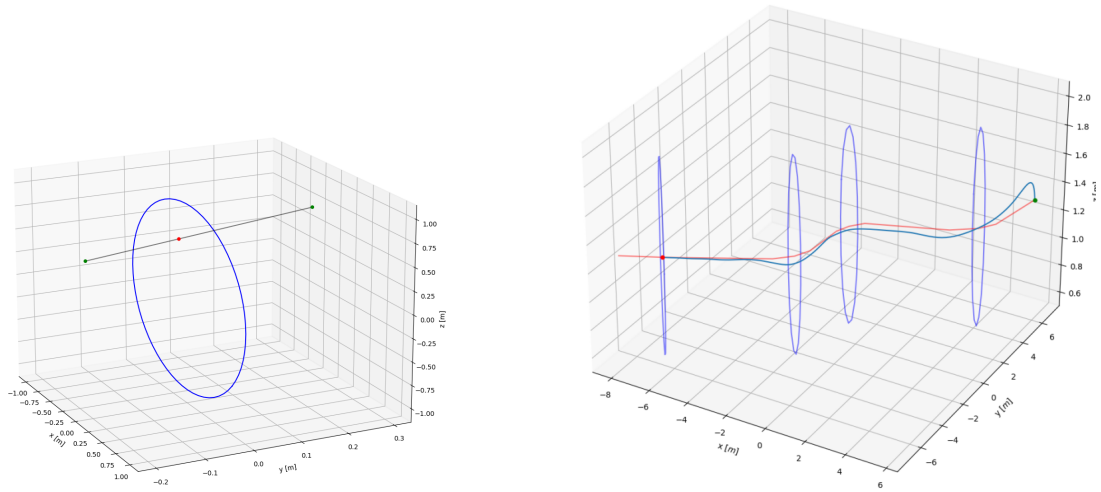
4.1.2 Track representation

The tracks consist of one or more waypoints that must be passed in a predefined order and a guiding path connecting them. The agent must fly through tracks in training, its performance is evaluated and rewarded by the reward function. The waypoints are given by their position, direction, and tolerance. The position is the center point of the waypoint. The direction is the desired direction of waypoint passing. The position and direction determine the waypoint plane, i.e. the plane that is perpendicular to the direction and contains the position. The tolerance is the maximal distance from the waypoint's position at which the agent can pass the waypoint plane for the flythrough to be accepted. Effectively the waypoints are circles in the environment that the quadrotor must pass. In the discretized timesteps it is checked whether two consequent positions of the agent are on opposite sides of the waypoint plane and the point at which the traversal between the subsequent positions passes through the waypoint plane is within the tolerance from the waypoint's position. An illustration of an accepted waypoint flythrough is displayed in Figure 4.1a. Figure 4.1b shows a plot of the trajectory of the agent while passing multiple waypoints.

The guiding path is a low-fidelity collision-free path connecting the waypoints. It is used to provide information about the desired direction of flight to the agent. For the tracks used in this thesis the guiding paths have been manually created, but there are algorithms for finding such paths in an environment by constructing a graph representation of the environment and using graph search, e.g. the ones described in [23], [24]. The guiding path is supposed to be provided by a high-level planner in the future, while the policy can conduct the control loop with a very short period in order of milliseconds, the guiding will only need to be recomputed in substantially longer intervals. The guiding path isn't subject to the quadrotor's physics model and doesn't have to be time-optimal even though the aim is for the resulting policy to navigate the track optimally in terms of flight time. A render of an environment with a guiding path is shown in Figure 4.2.

4.1.3 Action space

There are more representations of control commands traditionally used in solutions to quadrotor control problems. Since the term actions is more common in RL theory the control commands will be referred to as actions. These include single-rotor thrust (SRT), collective thrust and bodyrate (CTBR), and linear velocity and yaw rate (LV). SRT actions directly specify desired thrusts of individual rotors with no need for an additional controller. CTBR uses a low-level controller to achieve the desired collective thrust and body rates (angular velocities along the quadrotor's body frame axis). LV specifies the desired linear velocity



(a) Illustration of discretized waypoint passing.

(b) Plot of trajectory of the agent.

Figure 4.1: Illustration of waypoint passing. (4.1a) The waypoint is displayed as a blue circle. The subsequent agent's positions are shown in green, their traversal in gray. The red point is the intersection of the traversal with the waypoint plane. (4.1b) The blue line is the agent's trajectory. The red line is the guiding path. The blue circles are the waypoints. The green point is the initial position. The red point is the terminal position.

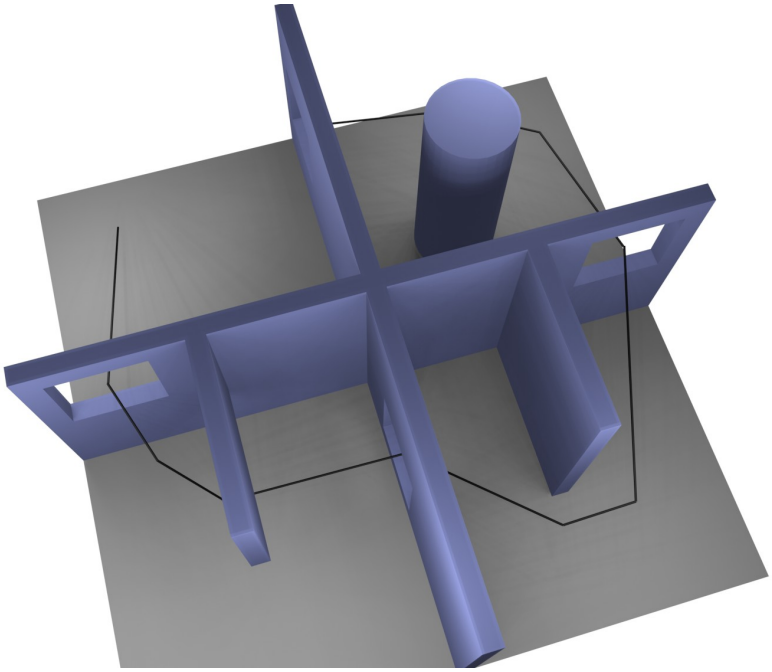
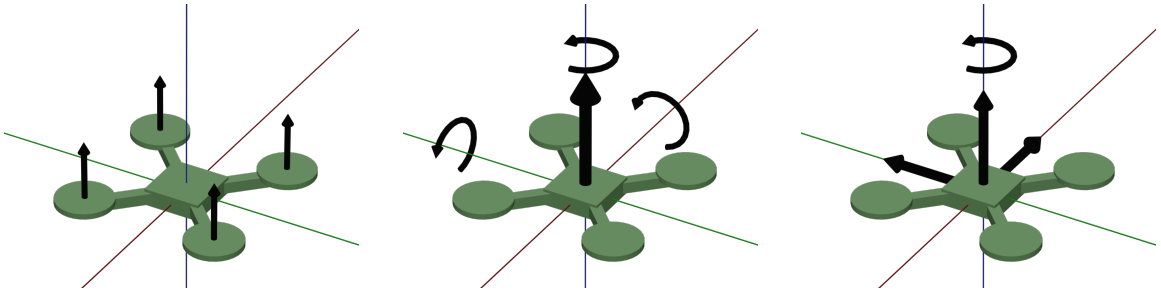


Figure 4.2: Example of a complex environment with a guiding path. Obstacles are shown in light blue. The black line is the guiding path.



(a) single-rotor thrust (SRT) (b) collective thrust and bodyrate (CTBR) (c) linear velocity and yaw rate (LV)

Figure 4.3: Comparison of different action spaces.

and yaw rate of the quadrotor and requires a high-level controller with access to full state estimation to track the action. The results of the comparison of these action spaces conducted in [3] show that single-rotor thrust is not robust against dynamics mismatch as it is too low-level. Linear velocity and yaw rate is the most robust and can be deployed on different platforms as the commands don't rely so closely on the properties of the platform. However, LV actions don't exploit the platform's full dynamic capabilities leading to limited agility. Collective thrust and bodyrate is more robust than SRT and outperforms LV in agile flight scenarios. An illustration of discussed action types' components is shown in Figure 4.3.

After initial experiments with single-rotor thrust actions, which proved to be hard to learn for the model and very unstable throughout training, it was decided to use desired collective thrust and body rates as the aim is to train for highly agile maneuvering of the quadrotor. The actions are in the form of $\mathbf{a} = [f_t, \omega]^T \in \mathbb{R}^4$. The action consists of desired collective thrust $f_t \in \mathbb{R}$ and desired body rates $\omega \in \mathbb{R}^3$. Each of the components is bounded to a limited interval determined by the properties of the simulated quadrotor platform.

4.1.4 Agent simulation

The agent is a simulated quadrotor along with an algorithm for its control and planning. It keeps the information about its state including position $\mathbf{p}$, attitude $\mathbf{q}$, linear velocity $\mathbf{v}$, body rates ω, and rotational speeds of individual rotors $\mathbf{\Omega}$. The agent also includes information about the physical properties of the simulated platform, these include its mass m, arm length l, inertia matrix J, drag coefficients $\mathbf{c_D}$, thrust coefficient c_f, torque constant κ, and minimal and maximal rotational speed of rotors Ω_{min} and Ω_{max}. Minimal and maximal rotor rotational speeds determine the minimal and maximal individual rotor thrust as $f_{min} = c_f \cdot \Omega_{min}^2$ and $f_{max} = c_f \cdot \Omega_{max}^2$. The symbols reference the notation in the quadrotor's physics model described in Section 3.1.

In every simulation step, the RL model passes an action in the form of $\mathbf{a} = [f_t, \omega]^T$ (Subsection 4.1.3) to the learning environment. This action is then tracked by a simple controller with a limited rate of change of rotor rotational speeds. An update to the agent's state is then computed according to the equations in the physics model (Section 3.1), the computation is conducted using the 4th-order Runge-Kutta method [20], which is a numerical method to approximately solve ordinary differential equations.

For collision detection, the agent is a sphere with the center at the agent's position. The radius r of the sphere is chosen such that the sphere encapsulates the whole quadrotor

with a small added safety margin. This representation is used to speed up collision detection following the SDF obstacle representation described in Subsection 4.1.1. The simulator checks if the agent's position after the update is closer to an obstacle than r, in which case it detects a collision. While this simplified representation leads to some falsely detected collisions when the agent is close to an obstacle, the benefit of simulation speed-up outweighs the negatives as some margin between the obstacles and the quadrotor is desired anyways.

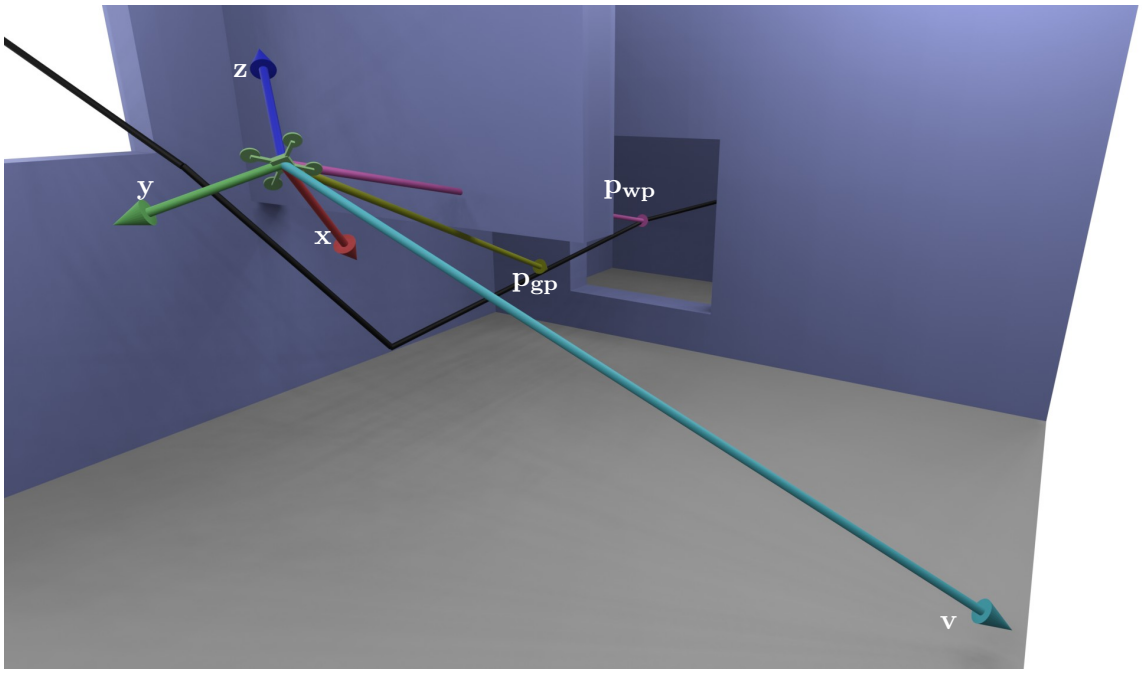
4.1.5 Observation

Observation is partial or complete information about the state of the agent and environment and/or sensor information about the state. It is the data the policy uses to predict actions. An RL model with more general observation with components easily accessible from onboard sensors has a better chance for successful generalization and real-world deployment as some information may be inaccessible from in-flight data or its estimation may introduce significant error. While providing more data in the observation might allow the model to create a more accurate representation of the state, the dimension of observation shouldn't be too high. As the model doesn't have access to full state information, the dimension of observation practically becomes the dimension of state space. With too large dimensions of observation, the model faces the curse of dimensionality. The curse of dimensionality is a phenomenon occurring with sampling from high dimensional spaces, where the sampled states are sparser, which leads to the need for more samples to achieve satisfactory results.

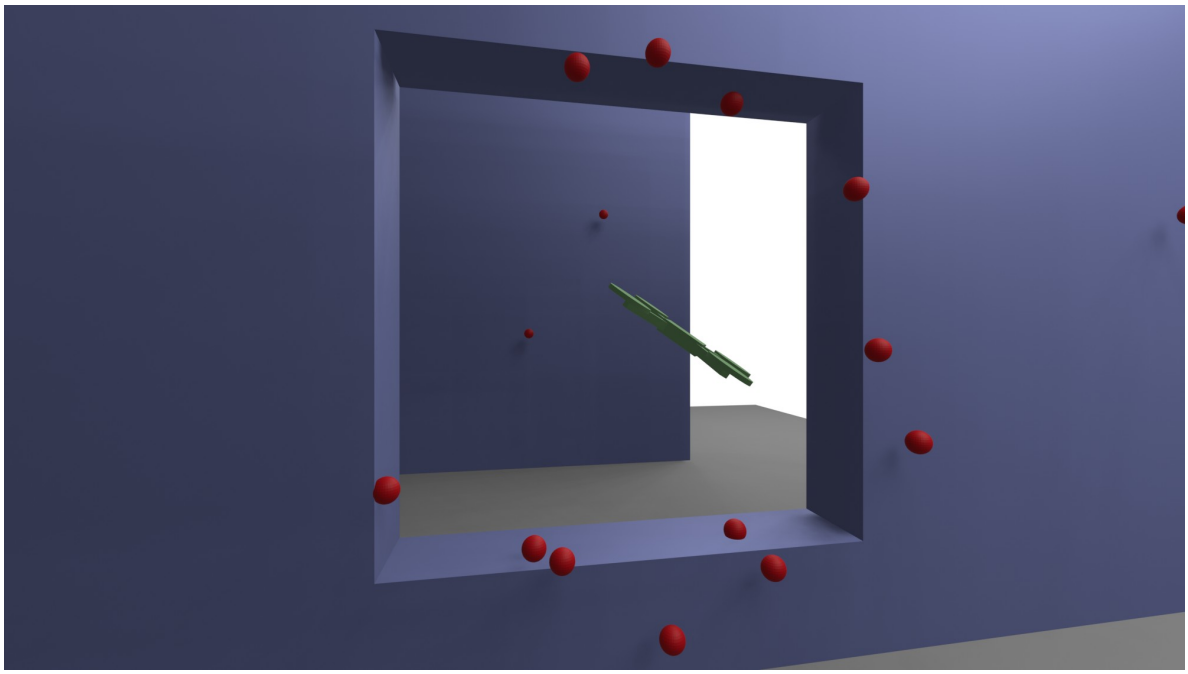
The observation used in this thesis has the form $O = [\mathbf{v}, R(\mathbf{q}), \mathbf{p_{wp}}, \mathbf{p_{gp}}, \mathbf{d_{rc}}]^T \in \mathbb{R}^{34}$, where $\mathbf{v} \in \mathbb{R}^3$ is the linear velocity, $R(\mathbf{q}) \in \mathbb{R}^9$ is the rotation matrix induced by the quadrotor's attitude $\mathbf{q}$, $\mathbf{p_{wp}} \in \mathbb{R}^3$ is the relative position of the next waypoint, $\mathbf{p_{gp}} \in \mathbb{R}^3$ is the relative position of the furthest directly visible point on the guiding path, and $\mathbf{d_{rc}} \in \mathbb{R}^{16}$ are depth measurements from the simulated depth sensor. Figure 4.4 provides a graphical explanation of the meaning of individual observation components.

Sensors for the perception of the surroundings, such as RGB cameras, depth cameras, or LiDARs, are used with perception-aware policies. A depth camera is closest to the simulated sensor used in this thesis. The observation provided to the model in this thesis doesn't include data from a high-resolution depth camera because the learning environment doesn't yet have the ability to produce realistic high-resolution depth images. Instead, the provided depth data come from a simpler simulated depth sensor that casts 144 rays from the position of the quadrotor in different directions centered around the direction of the x-axis of the quadrotor's body frame (in a regular 12×12 grid with predefined horizontal and vertical FOV) and returns the distances at which the simulated ray hits an obstacle or a maximal distance if no obstacle is closer than that distance. These 144 measurements are then subsampled from each 3×3 region to just the minimal value in that region, leaving 16 (4×4) depth measurements to the $\mathbf{d_{rc}}$ part of the observation. A depth image adds a lot of dimensions to the observation with every pixel being a unique measurement. Convolutional neural networks (CNNs) are often used to extract a low number of features from the depth image [1], [7]. CNNs are not used in the presented RL model because it is not needed with the relatively low complexity of the provided depth data. Also by not using CNNs training complexity is lower and the time for the inference of actions from observations is lower.

The relative position of the next waypoint $\mathbf{p_{wp}}$ is simply the difference between the agent's position and the position of the next waypoint to be passed. The relative position of the furthest directly visible point on the guiding path $\mathbf{p_{gp}}$ is the difference between the



(a) Observation components without depth perception.



(b) The depth perception part of the observation.

Figure 4.4: Illustration of observation components. (4.4a) The red, green, and blue arrows show the axes of the quadrotor's body frame (x, y, and z respectively) encoded in the rotation matrix. The yellow arrow is the relative position of the furthest directly visible point on the guiding path. The purple arrow is the relative position of the next waypoint. The light-blue arrow is the linear velocity of the quadrotor. The black line is the guiding path. (4.4b) Each of the red spheres represents one of the subsampled depth sensor measurements (the minimal out of the 3×3 region). The sphere is located at the distance provided in the observation in the direction from the agent in which it was measured.

agent's position and the point on the guiding path found by following the guiding path from the point closest to the agent's position until there's an obstacle in between the agent and the point on the guiding path.

4.1.6 Reward function

The reward function is a key component of reinforcement learning since the model learns by maximizing the expected reward. It is crucial to have a well-constructed reward as a bad reward leads to undesirable behavior. The reward awarded to the agent in this thesis consists of many components each of which is responsible for an aspect of the desired behavior. The weights of these components are finetuned so that the final policy demonstrates the desired behavior. The components of the reward are:

- Progress
- Traveled distance
- Waypoint reached
- Orientation
- Angular velocity
- Velocity
- Obstacle avoidance
- Collision

The **progress reward** is the most significant part of the reward. It is proportional to the distance traveled along the guiding path since the last time step, it is scaled down with increasing distance from the guiding path to drive the policy to follow the safe collision-free path more closely. The progress reward is computed by finding the closest point on the guiding path and comparing its distance along the guiding path with the one from the previous time step, this is illustrated in Figure 4.5.

The **traveled distance reward** is proportional to the agent's distance along the guiding path (from the start). It helps to overcome local minima introduced by other components of the reward because thanks to this reward actions taken closer to the finish always produce higher rewards. It is normalized by the length of the guiding path and the coefficient of the progress reward such that it is always smaller. Traveled progress reward is more significant early in the training and becomes less significant throughout the training as the model already learned to pass the situations producing local minima. These local minima often occur in sections with sharp turns and situations where the quadrotor needs to slow down to avoid crashes.

The **waypoint reached reward** is just a high positive reward granted upon passing a waypoint, it is scaled down with increasing distance from the waypoint's position which motivates the policy to aim for the center of the waypoint. The **orientation reward** is a penalization for not facing forward, facing forward means aligning the x-axis (of body frame) of the quadrotor to point in the direction of the furthest directly visible point on the guiding path. This behavior is desired because then the depth sensor provides more relevant measurements and the agent's situational awareness is better. In practice the quadrotor's x-axis and the relative position of the furthest directly visible point on the guiding path are projected to the xy plane. The projection is included because the roll and pitch need to be changed for maneuvering and we mostly care about the yaw angle. These projections are normalized and the distance between the normalized projections is calculated. The penalization is proportional to the distance.

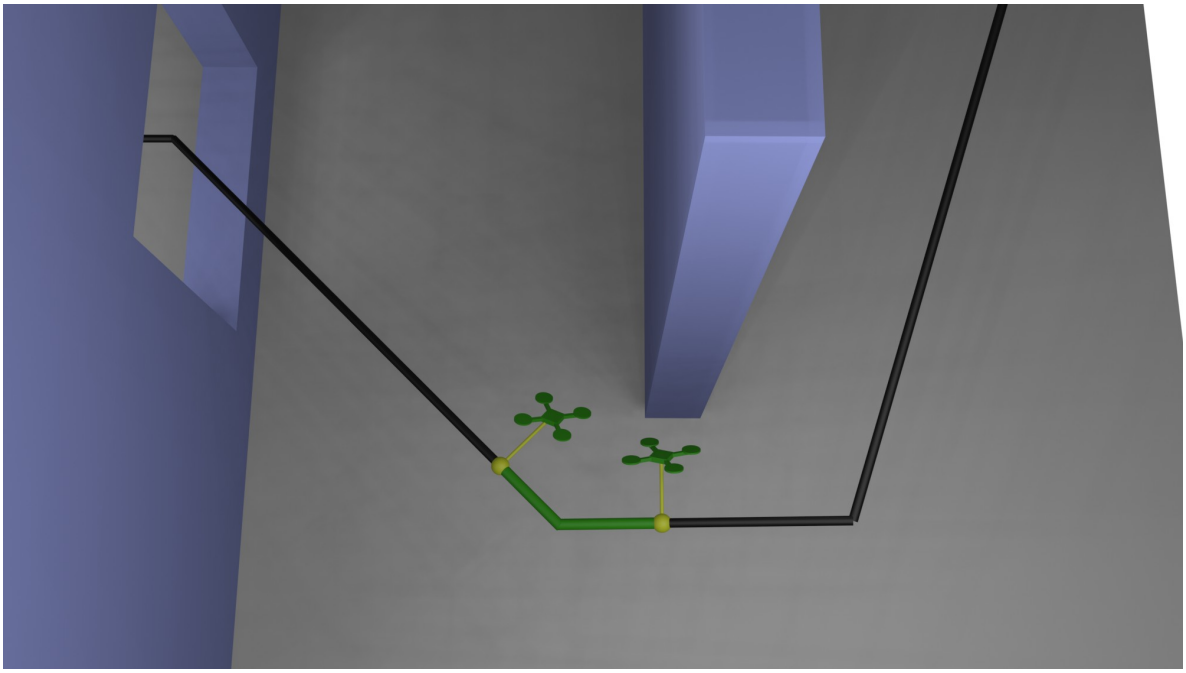


Figure 4.5: Calculation of progress reward. The quadrotors represent the states in two consequent time steps. The yellow points are the closest points on the guiding path. The length of the green line is the progress. The lengths of the yellow lines are the distances from the guiding path.

The **angular velocity reward** is a penalization proportional to the norm of the quadrotor's angular velocity (body rates). It helps to promote smooth movement without very aggressive maneuvers, these very aggressive maneuvers aren't easily reproducible by real hardware and would enlarge the gap between simulation and real-world deployment as unmodeled physical properties like arm bending and sensor inaccuracies are more significant with larger body rates. The **velocity reward** is a penalization for too slow or too fast flight speed. It is set to zero while the agent stays within the defined speed range and the penalization rises (the reward declines) exponentially with increasing distance from the defined speed range.

The **obstacle avoidance reward** is a penalization for moving in the direction of perceived obstacles, its value is determined in the following way. For each depth measurement in the observation, the dot product of the velocity vector $\mathbf{v}$ and the unit direction of the measurement $\mathbf{d}$ is calculated. The dot product is larger in measurement directions that are closer to the direction of the velocity. The reward for one depth measurement is then set to the value of $\min(0, -\mathbf{v}^T\mathbf{d}/(1+m^2))$ where m is the measured distance to an obstacle. The value is clamped to negative numbers because the dot product is negative when the direction of measurement faces the other way than the velocity vector. That would reward the agent for facing backward which is not desired. Figure 4.6 shows the graph of the reward value for $\mathbf{v}^T\mathbf{d} = 1$. The value of the collision avoidance reward is the average of the reward for individual measurements. The purpose of this reward is to provide the model with a more direct consequence of the depth measurements and to lead the model to give more importance to the depth measurements in the direction of the quadrotor's movement. It promotes a safer behavior of the agent: slowing down in front of obstacles and passing them with a larger margin.

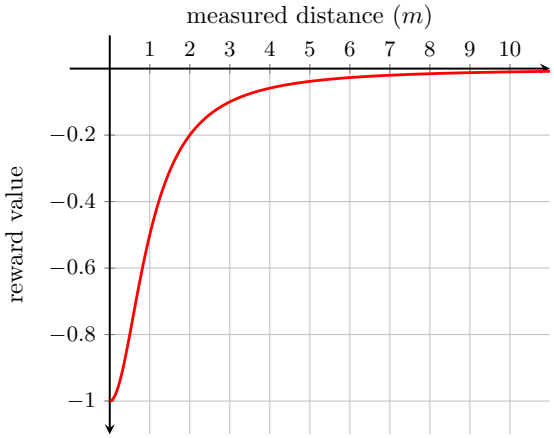


Figure 4.6: Graph of the value of collision avoidance reward for a single depth measurement in dependence on the measured distance. The graph is for the dot product of velocity and measurement direction $\mathbf{v}^T\mathbf{d} = 1$.

The **collision reward** is just a large negative reward granted upon collision with an obstacle or for reaching a position that is too far from the guiding path. In either case, the episode is terminated. Coefficients of the reward components, their relative sizes, and inference of the used components and their coefficients are provided in the results (Chapter 5). Equations 4.1 to 4.12 describe the complete calculation of the reward. The mathematical notation used in these equations is listed below in Table 4.1.

$$R = \begin{bmatrix} c_P \\ c_{TD} \\ c_{WR} \\ c_O \\ c_{AV} \\ c_V \\ c_{OA} \\ c_C \end{bmatrix}^T \begin{bmatrix} r_P \\ r_{TD} \\ r_{WR} \\ r_O \\ r_{AV} \\ r_V \\ r_{OA} \\ r_C \end{bmatrix}, \tag{4.1}$$

$$r_P = p \cdot s_{sp} \cdot \exp(-d_o) \cdot (1 - \frac{d_{gp}}{d_{gp\_max}}), \tag{4.2}$$

$$s_{sp} = \begin{cases} 10^{v_{max}-||\mathbf{v}||}, & ||\mathbf{v}|| > v_{max} \\ 20^{||\mathbf{v}||-v_{min}}, & ||\mathbf{v}|| < v_{min} \\ 1, & v_{min} \leq ||\mathbf{v}|| \leq v_{max} \end{cases}, \tag{4.3}$$

$$d_o = ||\frac{\mathbf{x}'}{||\mathbf{x}'||} - \frac{\mathbf{p}'_{\mathbf{gp}}}{||\mathbf{p}'_{\mathbf{gp}}||}||, \tag{4.4}$$

$$\mathbf{x}' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{x_b}, \mathbf{p}'_{\mathbf{gp}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{p_{gp}}, \tag{4.5}$$

$$r_{TD} = d_t \cdot s_{sp}, \tag{4.6}$$

$$r_{WR} = \begin{cases} 1 - \frac{d_{WP}}{t_{WP}}, & \text{if waypoint was passed in this timestep} \\ 0, & \text{otherwise} \end{cases}, \tag{4.7}$$

$$r_O = \exp(d_o) - 1, \tag{4.8}$$

$$r_{AV} = ||\omega||, \tag{4.9}$$

$$r_V = \begin{cases} 10^{||\mathbf{v}|| - v_{max}}, & ||\mathbf{v}|| > v_{max} \\ 20^{v_{min} - ||\mathbf{v}||}, & ||\mathbf{v}|| < v_{min} \\ 0, & \text{otherwise} \end{cases}, \tag{4.10}$$

$$r_{OA} = \frac{1}{|M_s|} \sum_{(m,\mathbf{d}) \in M_s} \min(0, \frac{-\mathbf{v}^T \mathbf{d}}{1 + m^2}), \tag{4.11}$$

$$r_C = \begin{cases} 1, & \text{collision detected} \\ 0, & \text{otherwise} \end{cases}. \tag{4.12}$$

Table 4.1: Mathematical notation for reward equations.

$r_P, r_{TD}, r_{WR}, r_O, r_{AV}, r_V, r_{OA}, r_C$	progress, traveled distance, waypoint reached, orientation, angular velocity, velocity, obstacle avoidance, and collision reward components
$c_P, c_{TD}, c_{WR}, c_O, c_{AV}, c_V, c_{OA}, c_C$	progress, traveled distance, waypoint reached, orientation, angular velocity, velocity, obstacle avoidance, and collision reward coefficient
p	progress along guiding path since the last timestep
s_{sp}	speed scale
d_o	orientation distance
d_{gp}, d_{gp_max}	distance and maximal allowed distance to the guiding path
$\mathbf{v}$	quadrotor's velocity vector
v_{min}, v_{max}	minimal and maximal allowed velocity
$\mathbf{x_b}$	direction of body frame x axis
$\mathbf{p_{gp}}$	relative position of furthest directly visible point on the guiding path
d_t	distance traveled along the guiding path
d_{WP}	distance to the passed waypoint
t_{WP}	tolerance of the passed waypoint
ω	vector of body rates
M_s	set of subsampled depth measurements
m, $\mathbf{d}$	measured distance and direction of a depth measurment

4.2 Reinforcement learning model set-up

This thesis uses the StableBaselines3 [8] implementation of the proximal policy optimization (PPO) algorithm. The principle of PPO is described in Section 3.4. The policy network is a multi-layer perceptron with three hidden layers of 256, 256, and 128 nodes, respectively. The value function estimating network is also a multi-layer perceptron and has three hidden layers of sizes 512, 512, and 256. Both of the neural networks use the ReLU activation function. The policy network is quite small, which is typical for learning-based quadrotor control methods [1], [4], [7]. The reasons are that the inference has to run quickly even with limited computational resources onboard and generalization is promoted this way since a smaller neural network is not able to overfit as much.

Hyperparameters for the training that are changed from their default value [8], [14] are the discount factor, the batch size, and the learning rate. The discount factor determines how much the value of reward from one action propagates to previous steps, a higher discount factor means the future rewards will be considered more when choosing the current action.

The value used for the discount factor is 0.99 which is a high value for most applications, but in the problem of quadrotor navigation (and other continuous control tasks) actions influence the state long in the future. The batch size is set to 25000, batch size is the number of samples collected by interacting with the environment before a PPO update step is performed. Higher batch size improves training convergence but slows down policy improvement. The average episode length in training is approximately 250 steps for the included training scenarios, so the batch size used counts for around a hundred episodes. Multiple episodes in one batch are desired as forming a small batch would result in overfitting to just a section of the training tracks. Overfitting is a phenomenon in learning-based methods where the learned model behaves well on a set of samples, but poorly on different samples. Furthermore, the samples are collected parallelly from 100 identical learning environments, in every learning environment the episode is in a different stage further promoting sampling from different parts of the state space. For the same reason the aerodynamic drag coefficients (denoted c_{D_x}, c_{D_y}, and c_{D_z} in physics model in Section 3.1) are randomized for each episode. The learning rate in Stable-Baselines3 [8] implementation of PPO determines the step size of the stochastic gradient descent algorithm on the value function error. A higher learning rate leads to faster improvement of the state value estimate but also limits convergence which shows in occasional deterioration of the agent's behavior. The used model is set to decrease the learning rate linearly throughout training from $5 \cdot 10^{-4}$ to $3 \cdot 10^{-5}$ to accommodate both fast improvement in the beginning and good convergence in the end. Inference of the used parameters is provided in the results (Chapter 5).

4.3 Training and tuning instruments

The policies trained for this thesis results section were trained for 600 million steps in various environments. The training was conducted on a high-end PC with AMD Ryzen 7 5800X 8-Core Processor and NVidia RTX 3090 graphics card or on the Research Center for Informatics (RCI) cluster where 16 cores of the Intel Xeon Scalable Gold 6150 36-core processor and an NVidia Tesla V100 graphics card were used. With either hardware configuration, the training simulation speed reached around 25000 simulation steps per second. While the simulation speed is high thanks to using SDF collision detection and not simulating high-resolution depth camera images, the training of a single policy still takes 6-7 hours to complete.

Several instruments were developed and included to help with evaluating the policy's performance throughout training and help tune reward component weights, RL model hyperparameters, training length, and other constants discussed further in this section such as checkpoint count and distancing, and the time of lifting the speed limitation. These instruments include customized TensorBoard [17] logging to visualize per-episode stats, such as reward size, reward component sizes, and average episode length throughout training. Furthermore, plotting of per-step reward component values, telemetry data, and trajectory for one episode every 5 million timesteps was developed and used, all of this data is also logged to CSV files. A Blender [12] script was developed to keyframe animations from the episode data logs, these animations provide intuitive insight into the agent's behavior by displaying it in the 3D model of the environment in a way natural to humans.

Checkpoints were introduced to promote more balanced sampling from different sections of the track. A checkpoint was placed at every 1 meter of the guiding path length. At the start of the training, the checkpoints are filled with hover states, and each time the agent passes by

a checkpoint there is a 10% chance that the agent's current state is saved as the checkpoint state. Every time the episode terminates and is reset, there is a 50% chance that the initial state is set to a random checkpoint state instead of the default initial state. The checkpoints ensure a better exploration of further sections of the track, limit overfitting to the beginning of the track, and speed up the policy's convergence.

The flight speed of the quadrotor is limited to 5 $m \cdot s^{-1}$ for the first half of the training and the limit is increased in several steps throughout the second half of the training. While the 5 $m \cdot s^{-1}$ flight velocity is not too slow compared with existing solutions, it is far below the platform's capabilities. The speed limit is soft and enforced by the speed reward (Subsection 4.1.6). The velocity is capped to discourage high-risk maneuvers while the model is still learning the basics of quadrotor control. The model is also able to learn to complete the whole track within this part of the training which is important as passing narrow passages is hard in high velocities. The coefficient for the traveled distance reward (Subsection 4.1.6) is set so that the size of the reward for the episode is proportional to the total progress reward throughout the episode if the agent was flying along the guiding path at the upper bound of the speed range, as the limit is lifted the traveled distance reward coefficient is not changed which decreases the significance of the traveled distance reward.

4.3.1 Training generalizing policy

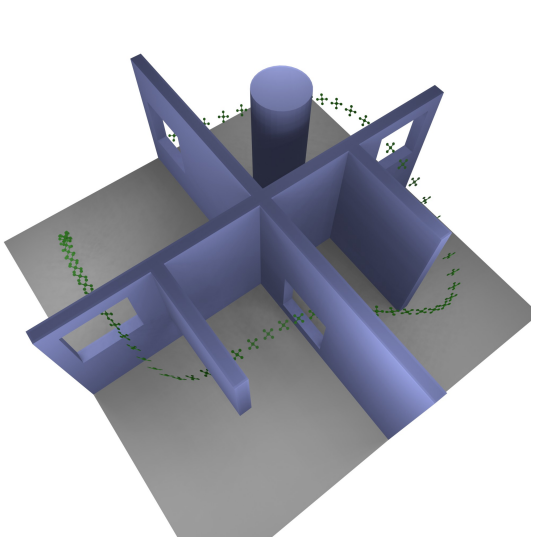
The generalizing policy is trained in the same way as the policy fitted to a single track which is described above. However, there are more tracks, each of which has its own initial position, checkpoints, waypoints, and guiding path. Each track is configured with options that decide whether the track will be used for training and whether it will be used for evaluation. Every time the episode terminates and the environment is reset the track which will now be used is selected from all the training-enabled tracks or (in case the learning environment is set to evaluation configuration) from all the evaluation-enabled tracks. The evaluation configuration also sets the probability of resetting to a state saved in a checkpoint to zero. The generalizing policies were trained on 1 billion simulation steps. For the generalization examples in this thesis, there were 7 different tracks in the forest environment, 5 of which have been used for training and the remaining 2 for evaluation.

4.4 Used environments

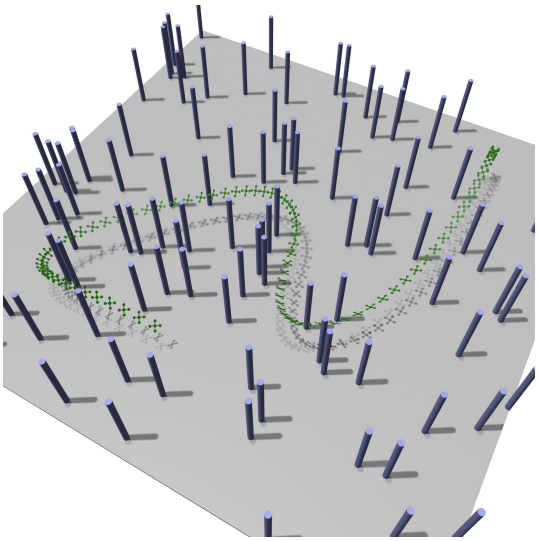
Four different environments were used in this thesis, renders are shown in Figure 4.7. Each of them is different in nature to test the policies in different conditions. Even though these environments are fundamentally different the hyperparameters of training, reward component weights, and other global settings of the learning environments have proven to be successful for training in all of them.

The obstacle course environment is a compact environment with sharp turns and narrow spaces. Highly agile maneuvers including quick changes in elevation are required in order to fly fast through the predefined track. The forest environment is filled with randomly placed poles that resemble tree trunks in a forest. The forest environment is larger with more sparsely spaced obstacles and offers many different ways to set up a track, this was used to test the generalization abilities of the trained policy. The generalizing policy was trained on five different tracks through the forest environment and tested on two other tracks, the mutual overlap of the tracks is very little or none. The office environment tests the ability of the model

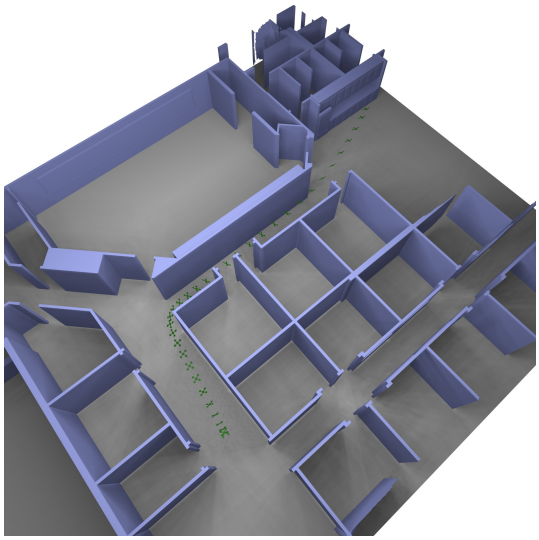
to learn to fly through narrow hallways as it offers a more realistic model of a human-made indoor environment. The drone racing environment represents a two-story racing track similar to the ones used in human drone racing competitions. It tests the agility of the drone and the ability to fly vertically through a gap which has shown to be quite difficult for the proposed perception-aware agent.



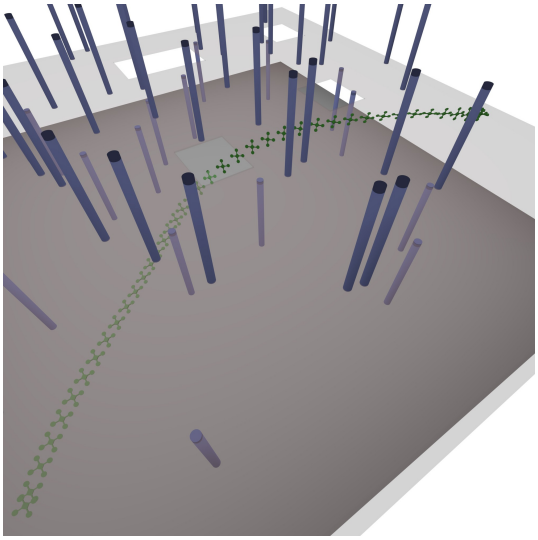
(a) Obstacle course environment



(b) Forest environment



(c) Office environment



(d) Drone racing environment

Figure 4.7: The used environments.

Chapter 5

Results

The method proposed in this thesis demonstrates the ability of an agent to learn perception-aware highly agile flight using only reinforcement learning while existing learning-based solutions [1], [7] rely on imitation learning. Learning with a progressively increasing upper limit on flight speed, reward for traveled distance, and reward induced directly from sensor information has shown to be the key ingredients for this achievement. The presented RL agent learned to track a topological guiding path in known environments and leverage information gained from an onboard depth sensor. The results are based on tests conducted with four different environments and a total of 21 tracks. The training for these tests was conducted on the hardware described in Section 4.3.

5.1 Reward function inference

Trial-and-error experiments with various reward components and the sizes of their coefficients were substantial in determining the final reward coefficients which are displayed in Table 5.1. The first reward components included were the progress, collision, waypoint reached, and angular velocity rewards. The orientation reward was added to constrain the yaw angle so that the depth sensor faces the direction of the agent's velocity. The policies trained before this component was introduced did change the side facing the flight direction in order to minimize flight time. The velocity reward was added to limit the minimal flight speed to 0.5 $m \cdot s^{-1}$ as without it the agent would learn to stop in difficult sections of the track and hover in place until the episode was terminated. Then it was decided to use the velocity reward to also limit the upper bound on flight speed in the first part of training which helped with passing of narrow passages and shortened the complete training time needed. Experiments with positive or negative reward for time spent alive were also conducted to achieve the effect of the speed lower bound, but it turned out to be ineffective.

The collision avoidance reward was introduced to add significance to the depth measurements as without this reward component the agent behaved the same way after the depth sensor was added as it did before. This observation suggested the policy does not understand the meaning of the depth measurements and disregards them instead. That would be a problem for generalization when the agent can't overfit to other observation components to navigate safely. It was resolved by adding the reward component which is directly influenced by the depth measurements. The last addition to the reward components was the traveled distance reward, it was added when the agent learned to overfit to the penalizations at the start of the track when it starts from a hover state and the progress reward is close to zero. This resulted in early deliberate crashes. We tried resolving it by lowering the penalizations in the first 30 steps of the episode. While it helped, it was not successful for all testing tracks. So on top of lowering the penalizations, the traveled distance reward was added as now the agent always benefits from getting further along the guiding path.

Table 5.1: Reward components, their coefficients, and the sum of the component's values over an episode of the trained policy on one of the testing tracks. l_{GP} is the length of the guiding path.

Reward component	Coefficient	Episode value
Progress	150	1998.0
Traveled distance	0.008 / l_{GP}	67.8
Waypoint reached	100	226.6
Orientation	-0.3	-1.6
Angular velocity	-0.5	-131.2
Velocity	-0.2	-0.1
Obstacle avoidance	0.1	-28.6
Collision	-150	0

The final reward is dominated by the progress reward which is by far the largest component. The other components form a relatively low portion of the final reward value. However, tests with higher coefficients for the penalizations (orientation, angular velocity, velocity, obstacle avoidance) resulted in crashes either at the start or at difficult sections of the track, such as narrow passages, vertical sections of the guiding path, and sharp turns. Tests without some of the reward components (orientation, angular velocity) resulted in flight with very different behavior, which is expected but shows that these penalizations have a significant effect on the resulting policy even though their value is small in comparison with the progress reward. Table 5.1 contains the per-episode sizes of the reward components for a trained policy. Note that these values depend heavily on the track and environment and that the penalizations are much higher throughout the training, but low in the end as the agent learns to avoid them. The equations for calculating the reward are located at the end of Subsection 4.1.6.

5.2 Hyperparameters inference

Hyperparameters of the RL algorithm were also determined by experimenting, their final values are listed in Table 5.2. The policy network was initially smaller with just two layers, but that was too small as the depth sensor and restrictions on yaw angle were added. The policy network was enlarged but kept as small as possible to keep the advantage of learning-based methods - the fast inference of actions. The value function network was enlarged to its final size for the same reason. The size of the value function network does not affect the action inference time so it doesn't have to be as small as possible. The hyperbolic tangent activation function was originally used. However, very slow policy improvements at the beginning of the training were experienced after the depth sensor measurements were included in observation and neural networks for policy and value functions were enlarged. This indicated the vanishing gradient problem and was resolved by changing the activation function to ReLU. The discount factor is set to 0.99. Tests with different values ranging from 0.9 to 0.999 showed worse or similar results. Other values were also tested for the batch size but all resulted in similar behavior. The reason for the used batch size of 25000 samples is that it accounts for around a hundred training episodes which is enough to stop overfitting to sections of the track. Since we train parallelly on 100 learning environments 25000 samples per batch is about one episode per learning environment. The learning rate is set to decrease linearly throughout training from $5 \cdot 10^{-4}$ to $3 \cdot 10^{-5}$. Various settings of the learning rate were repeatedly tested

Table 5.2: Values of hyperparameters for Stable-Baselines3 PPO implementation [8], [14].

Hyperparameter name	Value
Policy network architecture	[256, 256, 128]
Value function network arch.	[512, 512, 258]
Activation function	ReLU
Discount factor	0.99
Batch size	25000
Learning rate	$5 \cdot 10^{-4} \rightarrow 3 \cdot 10^{-5}$

and their performance varied depending on the track and environment. The final value was selected because it worked quite well for all of the tested tracks. However, the starting value of the learning rate is already quite small which meant the training had to be prolonged to achieve the same results. With the previous value of the learning rate fixed at $5 \cdot 10^{-4}$ about 300 million training steps were sufficient (600 million are used in this thesis), but the policy wouldn't converge for some tracks.

5.3 Overfitted policy

The quadrotor's behavior in test flights suggests the policy takes the depth sensor data into consideration. The comparison of the policy trained using principles from this thesis with a policy trained with a similar reward structure, but without the depth sensor information shows a difference in the agent's approach around obstacles. The policy trained without depth observation follows a more aggressive trajectory close to the obstacles while the policy trained with depth observation leaves more margin around obstacles and slows down more around sharp turns. This results in slower flight times, but helps generalization and promotes safer flight behavior overall. The policies presented in this thesis still depend on the guiding path to indicate the desired direction of movement. The policy trusts this information completely and follows the guiding path around corners at speeds too high to avoid a fatal crash in case an unobserved obstacle is present behind the corner. When the guiding path will be generated from sensor information in flight and unobserved obstacles on the guiding path will appear in training the agent will likely learn to fly more safely around corners.

Table 5.3 shows the comparison of flight times of different methods for 12 tracks. Our method is the only one that is perception-aware. This disadvantages the proposed method since it has to keep a suitable yaw angle (always face in the flight direction) for the depth sensor data to be relevant. The polynomial and sampling-based methods have full state information and complete information about obstacles. The minimum-time RL [4] method has privileged information about the quadrotor's absolute position. It also uses the guiding path similar to the method from this thesis, but instead of the furthest directly visible point on the guiding path it uses a slightly different point on the guiding path, one that can be connected to the agent with a straight line with a margin around the line large enough to fit the whole quadrotor. Both, our method and the minimum-time RL method, are overfitted to the individual tracks in this table. While the flight times of the presented method are longer than the times of the sampling-based and minimum-time RL methods, the results still show high agility given our method's perception awareness.

Table 5.3: Comparison of flight times of the presented perception-aware policy with other methods on 12 different tracks. Results of the other methods were taken from [5]. The other methods don't account for perception awareness, which disadvantages the presented method since it has to keep a suitable yaw angle. The listed times are the best out of 30 runs. Our method is overfitted to the tracks as well as the minimum-time RL method [5].

Scenario		**Polynomial [16]**	**Sampling-based [4]**	**Minimum-time RL [5]**	**Our**
Environment	Track	Time [s]	Time [s]	Time [s]	Time [s]
Forest	0	3.86	0.96	0.98	1.30
	1	3.43	0.96	1.00	1.36
	2	3.22	0.96	1.00	1.46
	3	5.25	1.30	1.28	2.04
Office	0	7.34	1.93	1.62	4.38
	1	6.49	1.69	1.64	2.82
	2	6.38	1.93	1.56	2.48
	3	5.14	1.58	1.40	2.30
Drone racing	0	5.79	1.34	1.20	2.20
	1	5.13	1.36	1.20	2.58
	2	4.94	1.37	1.38	3.80
	3	4.73	1.57	1.34	2.28

5.4 Generalizing policy

A set of seven tracks with 4-waypoints each was created to test the generalization. In the tests, the policy was trained on five tracks and the remaining two were left out in training for evaluation. The policy was not able to learn to fly through the testing tracks but was able to fly through the training tracks. Given the small size of the policy network, it is unlikely that the policy was able overfit to the combination of five different tracks of this length. This shows a tendency of the RL approach toward generalization.

On the tracks not used in training the quadrotor starts flying in the direction of the guiding path, the x axis of the body frame keeps aligned with the desired flight direction, and the starting altitude is kept. However, the quadrotor either deviates off course and crashes into an obstacle or misses the waypoint, or flies too fast and crashes at the first sharp turn. In none of the experiments was the agent able to pass more than two of the four waypoints on an unknown track.

The agent was able to finish at least three of five training tracks in all of the experiments. In this case, the agent was able to navigate the tracks at high velocities and avoid collisions. Figure 5.1 shows a comparison of the trajectory of the generalizing policy on a training track with the trajectory of a policy overfitted to the track.

5.5 Final assessment

The training convergence of the proposed method is quite slow, the training requires at least 2-3 hundred million simulation time steps to achieve satisfactory results in high-speed flight. Policy divergence is quite common when the agent can't get past difficult track sections, such as narrow passages and sharp turns. It is hard to limit the occurrence of policy divergence since the reasons behind it are highly environment and track dependent. The interpretability of learned solutions remains a problem as is the case with most learning-based approaches.

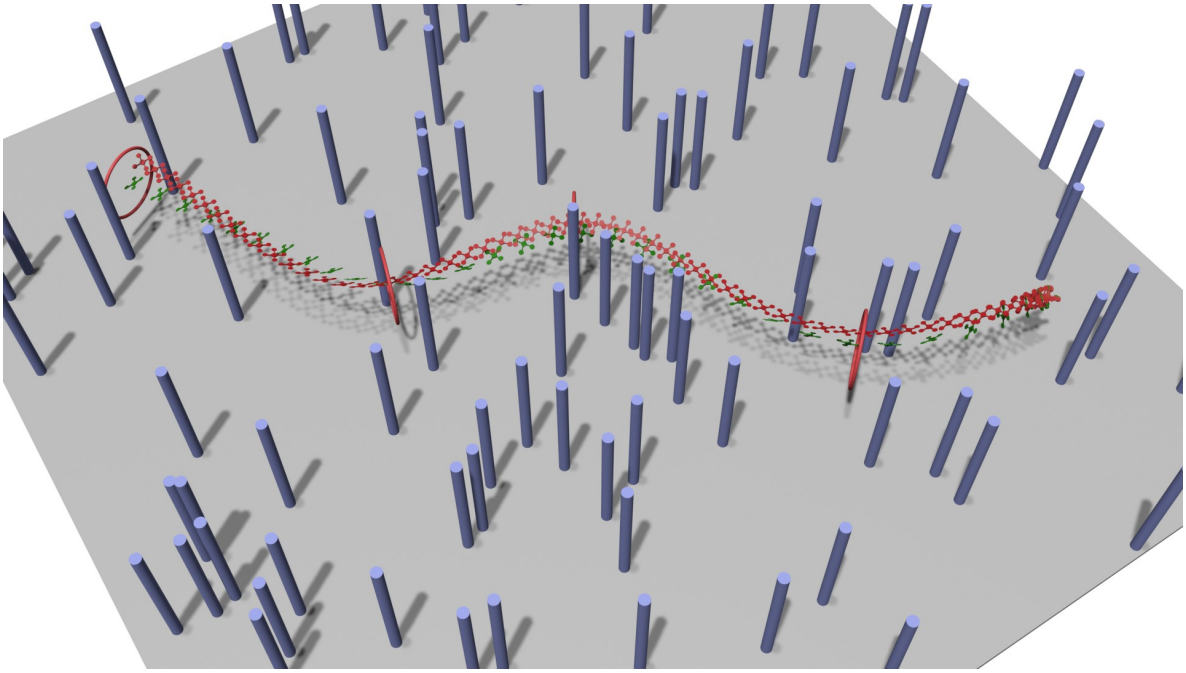


Figure 5.1: Comparison of a trajectory of the generalizing (red) and the overfitted (green) policy on a track in the forest environment. Both policies were trained on the track, the generalizing was also simultaneously trained on 4 other tracks.

The reliability of the learned policy (the percent of test flights successfully finished) is 70-100% depending on the track and environment. This is comparable to the results of the existing solutions. The reliability of non-learning-based solutions is under 50% for flight speed around 10 $m \cdot s^{-1}$ [10] which are commonly reached by the presented method. While the learning-based approach reaches around 90% reliability at these speeds [7]. The policy from [7] performs trajectory planning from observations and outputs local trajectories. On the other hand, the policy from this thesis compounds planning with control and outputs collective thrust and body rate commands.

The results provided reasoning for the used reward structure and hyperparameters, a description of the achieved flight behavior, a comparison with existing methods, and a comparison of the overfitted and the generalizing policy. Animations of flights provide an intuitive graphical representation of the described flight behavior. A video of animated flights of trained policies on various tracks is included in Appendix A and is also available on Youtube[1]. The flights in the video are displayed from an onboard camera and from a camera following the drone.

[1]Flight videos on YouTube: `https://youtu.be/HUE0Ezs8Tg4`

Chapter 6

Conclusion

We have developed a template for learning environments with OpenAI Gym [15] interface ready to be paired with existing reinforcement learning libraries. Using this template a learning environment with observation and reward function for high-speed perception-aware quadrotor flight in cluttered environments has been developed along with tools for parameter finetuning. This learning environment has been progressively upgraded to learn different high-speed policies. Starting from minimum-time quadrotor flight, then flight with yaw restrictions but without depth sensor observation, and finally perception-aware flight with depth information in observation. The learned policies were tested for the ability of collision avoidance in high-speed flight which required agile maneuvering. It was shown the fully reinforcement-learning-based approach has the ability to train perception-aware policies in known environments and the potential to train generalizing policies for flight in environments similar to the training environments. The advantages of the proposed method are the ability to perform agile maneuvers and the low computational complexity of calculating actions from observations. This allows for a very short control loop which results in high durability against disturbances and model mismatch errors.

The possibilities for future research are mainly in the area of generalization. Randomly generated environments and tracks would help with the training of the generalizing policies. Pairing the model with a high-level planner that can construct topological guiding paths online in flight is necessary to allow deployment in completely unknown environments. Further development of the learning environment is needed to model the dynamics, sensor noise, and disturbances accurately enough to enable zero-shot transfer to real-life scenarios. The learning environment will also have to generate high-resolution depth images to be presented in observation and a convolutional neural network will need to be added to the policy network to process these depth images. The information gained from the high-resolution depth images could be more useful than the information from the simple simulated depth sensor from this thesis allowing an improvement in the agent's situational awareness and consequently flight behavior. More research into the interpretation of the learned behavior could suggest possible optimizations in the training process.

Chapter 7

References

[1] Y. Song, K. Shi, R. Penicka, and D. Scaramuzza, *Learning perception-aware agile flight in cluttered environments*, 2023. arXiv: 2210.01841 [cs.RO].

[2] M. Idrissi, M. Salami, and F. Annaz, "A review of quadrotor unmanned aerial vehicles: Applications, architectural design and control algorithms," *Journal of Intelligent & Robotic Systems*, vol. 104, no. 2, p. 22, 2022.

[3] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A benchmark comparison of learned control policies for agile quadrotor flight," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 10 504–10 510. DOI: 10.1109/ICRA46639.2022.9811564.

[4] R. Penicka and D. Scaramuzza, "Minimum-time quadrotor waypoint flight in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5719–5726, 2022.

[5] R. Penicka, Y. Song, E. Kaufmann, and D. Scaramuzza, "Learning minimum-time flight in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7209–7216, Jul. 2022. DOI: 10.1109/lra.2022.3181755.

[6] P. Foehn, A. Romero, and D. Scaramuzza, "Time-optimal planning for quadrotor waypoint flight," *Science Robotics*, vol. 6, no. 56, eabh1221, 2021.

[7] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, eabg5810, 2021.

[8] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 12 348–12 355, 2021.

[9] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Conference on Robot Learning*, PMLR, 2021, pp. 1147–1157.

[10] B. Zhou, J. Pan, F. Gao, and S. Shen, "Raptor: Robust and perception-aware trajectory replanning for quadrotor fast flight," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1992–2009, 2021.

[11] N. Bucki, J. Lee, and M. W. Mueller, "Rectangular pyramid partitioning using integrated depth sensors (RAPPIDS): A fast planner for multicopter navigation," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4626–4633, Jul. 2020. DOI: 10.1109/lra.2020.3003277.

[12] B. O. Community, *Blender - a 3d modelling and rendering package*, Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. [Online]. Available: http://www.blender.org.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[15] G. Brockman, V. Cheung, L. Pettersson, *et al.*, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[16] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research: The 16th International Symposium ISRR*, Springer, 2016, pp. 649–666.

[17] Martin Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: http://tensorflow.org/.

[18] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, Jul. 2015, pp. 1889–1897. [Online]. Available: https://proceedings.mlr.press/v37/schulman15.html.

[19] W. Dong, G.-Y. Gu, X. Zhu, and H. Ding, "High-performance trajectory tracking control of a quadrotor with disturbance observer," *Sensors and Actuators A: Physical*, vol. 211, pp. 67–77, 2014.

[20] D. Tan and Z. Chen, "On a general formula of fourth order runge-kutta method," *Journal of Mathematical Science & Mathematics Education*, vol. 7, no. 2, pp. 1–10, 2012.

[21] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525. DOI: 10.1109/ICRA.2011.5980409.

[22] D. Braziunas, "Pomdp solution methods," *University of Toronto*, 2003.

[23] J. Hwang, J. Kim, S. Lim, and K. Park, "A fast path planning by path graph optimization," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 33, pp. 121–129, Feb. 2003. DOI: 10.1109/TSMCA.2003.812599.

[24] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

Chapter A

Contents of appended DVD

There are rendered videos and the project's source code on the appended DVD. The contents of the DVD have the following structure:

```
DVD root
├── videos
│   ├── intro_video.mp4
│   └── flight_videos.mp4
└── flightsim
    ├── examples
    │   ├── perception_aware
    │   │   ├── config
    │   │   │   ├── blender
    │   │   │   └── ...
    │   │   ├── include
    │   │   ├── src
    │   │   ├── learning
    │   │   │   ├── train.py
    │   │   │   └── test_policy.py
    │   │   └── scripts
    │   │       ├── simulator.py
    │   │       └── ...
    │   └── ...
    ├── include
    ├── src
    └── ...
```

The `intro_video.mp4` is a video explaining components of learning perception aware flight in the learning environment. It is also available on YouTube[1]. The `flight_videos.mp4` shows rendered flights from the drone's point of view and from a following camera in several scenarios included in the thesis. It is also available on YouTube[2]. In the `flightsim` folder is the content of the Git repository of the project. In the `include` and `src` subdirectories is the implementation of the general learning environment framework. The implementation of the learning environment for high-speed perception-aware flight presented in this thesis is inside the `examples/percetion_aware` folder. The main training script is `learning/train.py` and it imports other Python files from the `scripts` folder. The file `scripts/simulator.py` implements a Python wrapper for the learning environment and provides the OpenAI Gym

[1] Introductory video on YouTube: `https://youtu.be/7mR8IvWtQWU`

[2] Flight videos on YouTube: `https://youtu.be/HUE0Ezs8Tg4`

interface. The `src` and `include` subfolders contain implementation of the C++ part of the learning environment. The `config` folder contains configuration files defining the scenarios, waypoints, and quadrotor dynamics. Files with guiding path definitions and models of the cluttered environments are located in the `config/blender` subfolder.