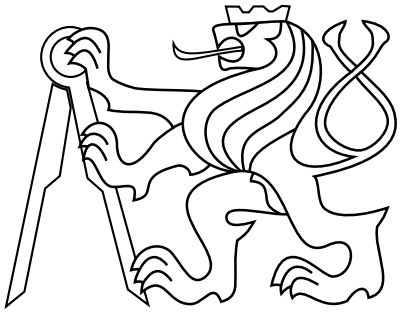
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics
Multi-robot Systems



Reinforcement Learning for Swarm Control of Unmanned Aerial Vehicles

Bachelor's Thesis

Karel Poncar

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: **Poncar Karel** Personal ID number: **498833**

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:

Reinforcement Learning for Swarm Control of Unmanned Aerial Vehicles

Bachelor’s thesis title in Czech:

Posilované učení pro řízení letu roje bezpilotních vzdušných robotů

Guidelines:

1. Analyze existing approaches for swarm control of unmanned aerial vehicles and the use of reinforcement learning for single UAV flight control.
2. Create a simulator of an unmanned aerial vehicle swarm. Connect the simulator with the selected reinforcement learning library and use a suitable learning method to teach swarming policy in known environments cluttered with obstacles using known robots’ states.
3. Provide the agents in simulation with observations that simulate real-world sensors, like 2D LiDAR and relative localization, and try to learn policy with such vision-based observation.
4. Evaluate the success rate of policies with known robots’ states and with the vision-based observation. Compare the proposed reinforcement learning method with existing methods for swarming.
5. (Optional) Try to learn a generalizing policy that would enable a swarm of quadrotors to fly without collisions through an unknown virtual environment.

Bibliography / sources:

[1] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, N. Dormann. "Stable baselines3." 2019.
[2] 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, July 2022
[3] Zhou, Xin, Jiangchao Zhu, Hongyu Zhou, Chao Xu, and Fei Gao. "Ego-swarm: A fully autonomous and decentralized quadrotor swarm system in cluttered environments." IEEE international conference on robotics and automation, 2021.
[4] Zhou, Xin, Xiangyong Wen, Zhepei Wang, Yuman Gao, Haojia Li, Qianhao Wang, Tiankai Yang et al. "Swarm of micro flying robots in the wild." Science Robotics 7, no. 66 (2022): eabm5954.

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: **27.01.2023** 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.

Karel Poncar

May 26, 2023 in Prague

Acknowledgments

I would like to express my gratitude to my bachelor thesis supervisor, Ing. Robert Pěnička, Ph.D., for his support and enthusiasm and for dedicating vast amounts of time to guide me in the development of this thesis. I would also like to thank Vít Knobloch for his cooperation in the development of the quadrotor simulator. And last but not least, I would like to thank my family for supporting me.

Abstract

This thesis focuses on the application of reinforcement learning (RL) techniques for the flight control of Unmanned Aerial Vehicle (UAV) swarms. The objective is to learn policies that provide control inputs separately to each individual UAV in the swarm to fly through multiple waypoints in a cluttered environment while keeping the swarm coherent. For this reason, we create a quadrotor simulator, that has the ability to emulate the dynamics of quadrotors and the environment with obstacles. This simulator is integrated with a standardized interface for RL, allowing interaction with an RL library. A suitable RL algorithm is chosen to train policy on individual UAVs in the swarm to navigate through the cluttered environments. To simulate difficult scenarios, the quadrotors in the swarm are deprived of their absolute position in the environment and they are provided with observations that simulate 2D Light Detection and Ranging (LiDAR). The success rate of the trained policies is evaluated in a scenario where the RL algorithm is provided with known robots' states (including the absolute position) and where the algorithm is provided with vision-based observations. Then, the performance of the proposed RL method is compared with another existing method for swarm control. Metrics such as success rate, the velocity of individual UAVs, and swarm cohesion are used to measure the effectiveness of the trained policies. Overall, this thesis contributes to the field of control of UAV swarms by exploring the RL techniques for this task. It provides insights into the challenges and opportunities of this approach for this problem.

Keywords Unmanned Aerial Vehicles, Reinforcement Learning, Swarming

Abstrakt

Tato práce se zaměřuje na aplikaci technik posilového učení (RL) pro řízení letu skupiny bezpilotních leteckých prostředků (UAV). Jejím cílem je naučit se strategii, která umožní každému UAV ve skupině samostatně ovládat svůj let, proletět více cílových bodů v prostředí s překážkami a současně udržovat skupinu pohromadě. Pro tento účel byl vytvořen simulátor kvadrokoptér, který dokáže emulovat prostředí s překážkami a dynamikou těchto robotů. Simulátor je vybaven s standardizovaným rozhraním pro RL, které umožňuje komunikaci s RL knihovnami. V této bakalářské práci zkoušíme učit strategie i na obtížnějších scénářích, kde jsou kvadrokoptéry zbaveny informace o své absolutní pozici v prostředí a jsou jim poskytována pozorování simulující 2D LiDAR. Úspěšnost naučených strategií je potom vyhodnocena ve scénáři, kdy kvadrokoptéry znají svoji absolutní pozici, a ve scénáři, kdy kvadrokoptéry mají místo polohy informace z LiDARu. Jsou zde také porovnávány naučené strategie s jiným projektem pro řízení skupiny UAV. K porovnávání jsou použity metriky jako rychlost kvadrokoptér a soudržnost skupiny. Celkově tato práce přispívá do oblasti řízení skupiny UAV pomocí RL a nabízí poznatky o problémech a příležitostech RL pro řešení tohoto problému.

Klíčová slova Bezpilotní Prostředky, Posilované Učení, Rojové Chování

Abbreviations

GNSS Global Navigation Satellite System

LiDAR Light Detection and Ranging

RL Reinforcement Learning

UAV Unmanned Aerial Vehicle

MDP Markov Decision Process

PPO Proximal Policy Optimization

POMDP Partially Observable Markov Decision Process

Dec-POMDP Decentralized Partially Observable Markov Decision Process

ESDF Euclidean Signed Distance Field

SB3 Stable Baselines 3

Contents

Chapter 1

Introduction

Unmanned Aerial Vehicles (UAVs) have become increasingly popular and important in recent years, due to their versatility and potential for use in a wide range of applications. They are used for a variety of purposes, including search and rescue [20], product delivery [14], agriculture [4], surveillance [23], mapping [24], military [11], and more. They have the ability to reach areas that are difficult or dangerous for humans to access and can be used to collect data and perform various tasks more efficiently than traditional methods.

In particular, Unmanned Aerial Vehicle (UAV) swarms (groups of coordinated UAVs working together to achieve a common goal) have been gaining more and more attention in recent years. UAV swarms can provide many advantages over individual UAVs, such as providing multiple views of a single target from different perspectives and covering larger areas than a single drone. Also, the presence of multiple UAVs allows the swarm to continue its mission even if one or more UAVs crash or malfunction.

With the increasing popularity of UAVs, there has been more and more focus on developing autonomous drones. To achieve autonomy, UAVs need extra capabilities in vision, self-localization, environment mapping, path planning, control, and more. One algorithm that can be used for flight control is Reinforcement Learning (RL).

RL is a type of machine learning that allows agents to learn how to behave in an environment by executing actions and receiving rewards. In the context of UAVs, RL can be used to control the flight of the UAVs in order to achieve the goal of a specific task.

Recent works (such as [16], or [3]) show, that the RL can be used for tackling flight control task of an individual UAV. Inspired by the recent works in the field, this thesis tries to extend the scope of RL use to control the flight of multiple UAVs. The objective is to train a policy, that would provide each agent in the swarm the necessary control inputs based on the observations from the surrounding environment. Moreover, this policy needs to control the agents in a way that they would not collide with each other or with the surrounding obstacles, and would fly in a coherent swarm through multiple waypoints in the environment. For this reason, we develop a high-performance simulator of 3D environments with stationary obstacles and we provide it with OpenAI Gym [18] interface for interaction with RL libraries. This simulator is used together with algorithm Proximal Policy Optimization (PPO) [17] from RL library Stable Baselines 3 (SB3) [6] to train policies suitable for our problem. In this thesis, we show that policies created by RL techniques are capable of controlling the flight of multiple UAVs through a cluttered environment.

The implementation part of the thesis is focused primarily on quadrotors. A quadrotor is a type UAV that is propelled by four propellers. It falls into the cathegory of rotary-wing UAVs. This cathegory also includes other types, such as tricopters or hexacopters. Outside of the rotary-wing cathegory, there are also fixed-wing UAVs [25] and hybrid UAVs. However, the RL methods for these types of vehicles is beyond the scope of this thesis and we focus on the quadrotors only.

Chapter 2

Related works

There is not much research done on the use of RL for controlling a swarm of UAVs. However, there are a few closely related topics: traditional methods for UAV swarming, the use of RL for flight control of individual UAV, and the use of RL for the cooperation of multiple agents. This chapter mentions the most relevant work on both of these topics.

2.1 Traditional methods for UAV swarming

For UAVs, the state-of-the-art methods generally break down autonomous navigation into two parts: trajectory planning and control [3]. First, a computation for a collision-free trajectory is needed and then a highly accurate controller is important so that the UAV will safely fly through the environment according to the planned trajectory. This approach is widely used for flight control of both individual UAV and UAV swarms.

Focusing on UAV swarms, the state of the art is primarily classified by the structural architecture of the swarm. According to this attribute, swarm architectures break down into two categories: centralized and decentralized.

The centralized-control approach of drone swarming involves using a central computation unit to compute the trajectories of individual UAVs in the swarm in order to achieve a specific formation. This approach is often used in large-scale aerial shows, where drones are used for artistic displays or entertainment purposes [8]. In these light shows, each drone typically follows a pre-computed trajectory and relies on Global Navigation Satellite System (GNSS) for localization.

On the contrary, the decentralized-control approach of drone swarming means that each drone in the swarm makes trajectory planning autonomously, according to the sensory inputs and information shared among agents. Decentralized swarming is generally more difficult to design since each individual robot has typically limited information about its surroundings. However, this system tends to be more robust to failures and communication disruptions.

Focusing on decentralized swarming, one of the state-of-art approaches is described in [5]. The authors of this article develop fully autonomous miniature quadrotors and provide them with a trajectory planner that can function adequately based on the limited information from onboard sensors. The planner satisfies various task requirements, including flight efficiency, obstacle avoidance, and swarm coordination. In this article, the authors also compare their planner with a few other state-of-art planners – MADER [1] and their previous work EGO-Swarm planner [7] – showing that the approach proposed in [5] outperforms other two planners in several benchmark comparisons. On top of that, they show the success of their approaches in various real-world tests, like multi-drone tracking with target occlusion or a flight through a dense forest [1].

[1] `https://youtu.be/L0fJ0EHHfOA` (accessed 16.5.2023)

Another successful work in the field of decentralized swarming is PACNav [2]. It presents an approach for navigating a swarm of UAVs to a desired goal as a compact group. Unlike other methods, PACNav does not require communication among the members or a global localization system. It relies only on observations from local sensors and the relative position of the other quadrotors, making it beneficial in demanding real-world conditions where global localization is unavailable or has high uncertainty and the swarm size makes communication unfeasible. The [2] is important for our work since we compare the achieved results with it.

2.2 RL for flight control of individual UAVs

In learning-based flight control, the drone is treated as an agent that interacts with its environment. In this scenario, the agent executes actions and receives observations from the environment and rewards. The goal is to find a control policy that maximizes the expected reward. Using this policy, the agent can predict the control commands directly from environment observation. In recent years, the learning-based approach has been experimentally used on UAVs and there are works that demonstrate effectiveness in using RL for the flight control task of an individual UAV.

The article [16] focuses on the use of RL for the flight control of an individual quadrotor. The authors trained a policy in the form of a neural network that was able to successfully perform point tracking with the quadrotor and quadrotor stabilization. The article also proposes a new RL algorithm, which had for their task better results than other state-of-art RL algorithms, that are aimed for general use. This RL algorithm was not used in our work, however, the rest of this article provides valuable insights into the use of RL techniques for controlling aerial vehicles like quadrotors.

One of the more recent works [3] proposes a successful approach to the minimum flight control task of a quadrotor through a sequence of waypoints in the presence of obstacles. This approach combines deep RL and topological path planning to train neural network controllers for minimum-time quadrotor flight in cluttered environments. The article [3] shows that in terms of speed and success rate, the proposed method outperforms existing state-of-the-art approaches in a majority of test cases. Moreover, their method is validated in a real-world experiment, showing, that the method is applicable to real hardware as well. This paper is closely related since in our work the same quadrotor dynamics model is used (the quadrotor dynamics is described in Section 3.3). We have also used one of the testing environments (the *random columns* environment, Fig. 4.1a) in our work.

2.3 The use of RL in multi-agent tasks

Although RL has focused primarily on single agents, it can be extended straightforwardly to multiple independent agents. In such cases, the environment can be set in a way that invokes competition (typically when a positive reward for one agent means a negative reward for another agent) or cooperation (typically when the collective reward is maximized).

The first approach is widely used in two-player board games where the goal is to win against the opponent. Here, the state-of-the-art approaches typically train a policy in the form of a neural network by supervised learning from human expert games and then the policy is further improved by RL from games of self-play. Using this approach, learning-based policies were able to achieve superhuman performance in games like Go or chess [13].

The other approach (cooperation of the agents) is also documented in the literature. For example, in the work [27] the author experiments with RL for the cooperation of two agents, that are trying to capture a prey and compares their success rate with a single agent trying to accomplish the same task. This work is related to ours since we also use learning-based methods to achieve the cooperation of the agents in order to complete a certain task.

Inspired by the success of works in learning-based flight control and learning-based cooperation, we apply RL techniques to achieve a coherent swarm of quadrotors that flies through multiple waypoints in an environment with the presence of obstacles.

Chapter 3

Problem Description

The project aims to use RL methods for controlling a swarm of UAVs in cluttered environments. Since there are many types of UAVs and all these types have different dynamics, we worked in the implementation part with quadrotors as representatives of UAVs. From now on, this work will focus on quadrotors only (instead of UAVs in general).

The first section briefly describes the concepts of RL necessary for explaining the RL algorithm we used. The second section defines the flight control task of a swarm of quadrotors as a problem suitable for RL methods. After that, the work will focus on the dynamics of quadrotors. The last section of this chapter explains the goal we are trying to achieve with the proposed method.

To ensure clarity, the mathematical notation is summarized in Table 3.1. This notation will be used in this chapter as well as in the **Methodology** and **Results** chapters.

Table 3.1: Mathematical notation, nomenclature, and notable symbols.

$\mathbf{x}$, $\boldsymbol{\alpha}$	vector, pseudo-vector, or tuple
$x = \mathbf{a}^\intercal \mathbf{b}$	inner product of $\mathbf{a}$, $\mathbf{b} \in \mathbb{R}^3$
$\mathbf{x} = \mathbf{a} \times \mathbf{b}$	cross product of $\mathbf{a}$, $\mathbf{b} \in \mathbb{R}^3$
$\mathbf{x} = \mathbf{a} \odot \mathbf{b}$	quaternion multiplication of $\mathbf{a}$, $\mathbf{b} \in \mathbb{R}^4$
$\dot{x}$	1^{st} time derivative of x
$SO(3)$	3D special orthogonal group of rotations

3.1 Description of a general task for RL

This section is focused on the characteristics of the tasks, on which the RL methods could be applied. This will give us the necessary foundation to describe the algorithm (PPO, 4.2.1) that we used to tackle this problem. Firstly, the Markov Decision Process (MDP) is explained, and then how the concept of MDP is used in RL.

3.1.1 Markov decision process

MDP is a mathematical concept used to model decision-making in processes that are working in *discrete* steps. In every step $k \in \mathbb{N}$, the process is in a definite state s_k. The state of the process can be changed by applying action a_k, which will move the process to the next state s_{k+1} determined by the transition probability function. Moreover, an immediate reward is generated by transition $s_k \to s_{k+1}$ while taking action a_k. Specifically, MDP is a tuple $(S, A, p(\cdot), r(\cdot))$, where S is a set of states, A is a set of all actions, $p : (s'|s, a) \to [0, 1]$ is the transition probability function of moving to state $s' \in S$ when taking action $a \in A$ in state

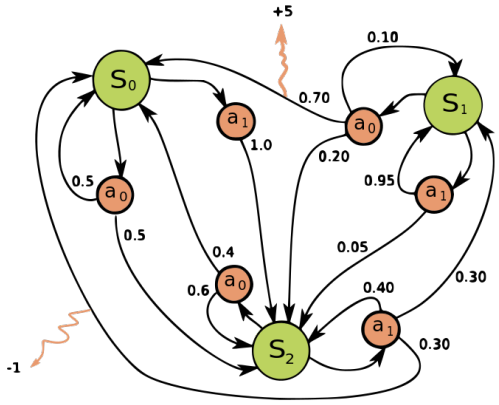


Figure 3.1: An example of an MDP with 3 states (denoted in green), 2 actions (in red color), and 2 rewards (red arrows). Black arrows describe transitions and the numbers next to them are transition probabilities. [1]

$s \in S$, $r : (s, a, s') \to \mathbb{R}$ is the reward function, which determines the immediate reward for moving from s to s' while taking action a. An example of MDP is shown in Fig. 3.1.

In MDP, the transition probability of moving from state s_k to state s_{k+1} is dependent only on the state s_k and on the action a_k. The reward is also definite for each tuple (s, a, s'). In other words, MDP does not "remember" the sequence of previous states, it only considers the current state and the action taken in the current state.

Note, that the reward $r(s_k, a_k, s_{k+1})$ is possible to know at step k only if the transition probability function p and the reward function r are known. If these functions are unknown, we find out the reward for the transition and the action *after* the action was made, at the step $k + 1$. In this case, we will use notation r_{k+1}.

A *policy* determines which actions to consider in a given state. More specifically, a policy is a function $\pi : (S, A) \to [0, 1]$ taking a state $s \in S$ and an action $a \in A$ and returning the probability of taking the action a in the state s.

For a given MDP $(S, A, p(\cdot), r(\cdot))$ and a given policy π we are often interested in the reward the MDP generates when we take actions given by the policy π. For this reason, we define *value function* $v_\pi : S \to \mathbb{R}$, which assigns each state s in the MDP a value representing the expected reward.

The expected reward could be the cumulative sum of all rewards obtained from the MDP. However, for an infinite process, the cumulative sum of the rewards could converge to infinity (or negative infinity) and its maximization would not be possible. In such a scenario, we use *discounted reward*. The discounted reward in step t is calculated as:

$$G_t = \sum_{k=0}^{T} \gamma^k \cdot r_{t+k+1}, \tag{3.1}$$

where T represents the remaining length of the process from step t and $\gamma \in [0, 1]$ is a discount rate. If $\gamma \neq 1$ the discounted reward could be approximated even for infinite processes $T = \infty$ since the sum converges to a certain value.

[1] Image source: `https://en.wikipedia.org/wiki/Markov_decision_process` (accessed 5.5.2023)

The MDP defines the standard model for RL problems. The formulation of the problem as an MDP allows using the above-mentioned concepts and building algorithms to improve the policy. It is worth mentioning, that there are also other important concepts that MDP provides, such as *Q-function*. However, this section primarily focuses on the concepts that will be used later in when describing PPO (Section 4.2.1), which is the RL algorithm we use.

3.1.2 Connection between MDP and RL

RL focuses on learning optimal decision-making policies by trial and error through interactions with an environment. It includes algorithms for learning the optimal policy, such as policy iteration or policy gradient methods.

The structure of RL problems always contains an agent and an environment. The agent refers to the decision maker – it chooses an action based on the information it gets from its surroundings. The agent's task is to learn from past experience to create a policy maximizing the expected reward. The environment is the world that the agent interacts with. Based on the given action and the given state of the agent, it provides a reward and an observation about the new state. The agent's goal is to get the most reward in the long run. This interaction is depicted in Figure 3.2.

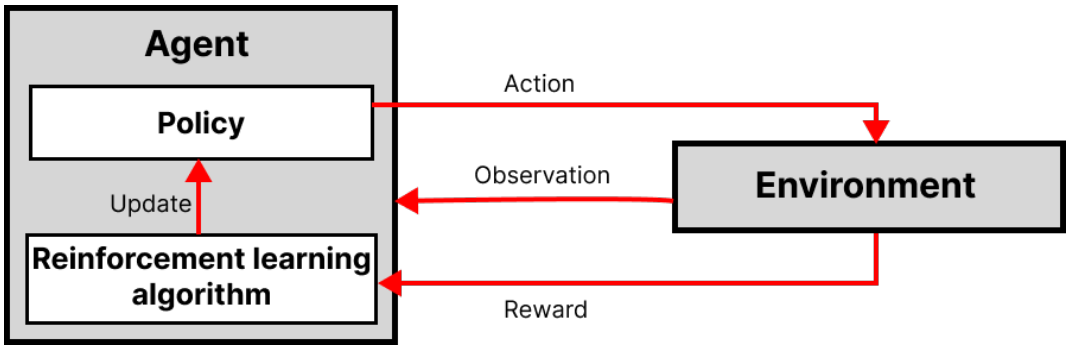


Figure 3.2: Interaction of agent and environment. The agent interacts with the environment in discrete time steps by executing actions and receiving observations and rewards. The agent behaves according to a policy, that is updated over time by a RL algorithm.

As mentioned in Section 3.1.1, MDP represents a standard model for RL problems. Therefore, it is common to consider the interaction between the agent and the environment in discrete time steps. There is some research on interaction in continuous time [22], however, most of the research in RL is focused on the methods applicable in discrete time steps only.

To proceed further, we will describe terms, that are not commonly used in MDPs. However, these terms will be useful to us in the future.

A *sequence of interactions* between the agent and the environment of length k, where k denotes the number of time steps, is a sequence:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, ..., s_k; \tag{3.2}$$

where $s_i \in S$, $a_j \in A$, $r_l \in R$ for each $i \in \{0, \ldots, k\}$, $j \in \{0, \ldots, k-1\}$, $l \in \{1, \ldots, k\}$.

The set of *terminal states* $T \subset S$ refers to a subset of all states where the agent can no longer take any action. For example, in a game where the objective is to reach a certain score, a terminal state would be reached when the score is achieved or when the game is lost.

In the presence of terminal states, the interactions with the environment naturally break to *episodes*. The episode could be described as a sequence of interactions between the agent and the environment where the last state s_k is terminal. If the task naturally beaks to episodes, it is called *episodic task*. Otherwise, we call it *continuous task*.

These terms will make it easier for us to define the flight control task of a quadrotor in the next section.

3.2 Defining flight control task as a problem suitable for RL

In the context of the flight control task problem of a swarm of quadrotors, the agent is a quadrotor. The environment is a three-dimensional space $W \subseteq \mathbb{R}^3$ with a set of agents, a set of obstacles, and a set of waypoints in it. The obstacle geometry $O \subset W$ is a subspace in W. Quadrotor's geometry is a function $A(\boldsymbol{x}) \subset W$ taking the state $\boldsymbol{x}$ of the agent and returning the space that makes the agent's body in a given state (the exact representation of the agent's state is described later in the Section 3.3).

C is configuration space (space of all agent's states), $C_{obs} = \{\boldsymbol{x} \in C | A(\boldsymbol{x}) \cap O \neq \emptyset\}$ is a set of configurations where the agent is in a collision and $C_{free} = \{\boldsymbol{x} \in C | A(\boldsymbol{x}) \cap O = \emptyset\}$ as a set of states where the agent is not in collision.

The goal of the agent is to pass a sequence of waypoints in the environment. A waypoint $G \subset C_{free}$ is a circle. It is described with its position $\boldsymbol{p}_{wp} \in W$, orientation $\boldsymbol{q}_{wp} \in SO(3)$ and radius (tolerance) $\alpha_{wp} \in \mathbb{R}^+$. We define passing of the waypoint G as a situation when the waypoint is between two consequent agent's positions:

$$\boldsymbol{g} = (1 - u) \cdot \boldsymbol{p}_i + u \cdot \boldsymbol{p}_{i+1} \text{ for any } \boldsymbol{g} \in G, u \in [0, 1]. \tag{3.3}$$

Now that the terms C_{obs} and waypoint passing have been described, we can define the set of terminal states. For a single agent, the state of the agent $\boldsymbol{x}$ is terminal if $\boldsymbol{x} \in C_{obs}$ or if the agent passed the given sequence of waypoints in the correct order.

In general, a swarm of quadrotors is a group of quadrotors that works together in a coordinated manner to accomplish a specific task or objective. Here, the objective is that each quadrotor must pass a (non-empty) sequence of waypoints in the environment while keeping the swarm coherent. Generally, the sequence of waypoints can be different, but in our work, it is the same for each quadrotor in the swarm. We do not impose hard limits on the swarm cohesion, but individual quadrotors are encouraged with rewards to stay close to the rest of the group.

Note, that we defined the RL task as a *single* agent interacting with the environment, and yet here multiple agents interacting with the same environment have been introduced. One way to tackle this problem is to have a fully centralized method to learn and act in a centralized fashion. This can be considered a big single-agent problem – the policy takes the observations from all agents as one joint observation and returns the joint action for all agents in the environment. The other approach to this problem is by applying single-agent RL techniques for each agent and treating other agents as part of the environment so that each agent learns its own policy. The third approach is in the middle of the two: a single policy is trained by the experience of all agents, but the policy takes a single agent's observation and returns that agent's action. This approach is used in our work.

For multiple agents, the set of terminal states needs to be redefined, since the bodies of other agents in the environment must be taken into account. Let's have a swarm of n quadrotors and for a given time step $k \in \mathbb{N}$, let's denote $\boldsymbol{x}_i$ as the state of the i-th quadrotor in the environment. This state $\boldsymbol{x}_i$ in the time step k is terminal if any of the following three conditions are satisfied:

- agent is in a collision with an obstacle: $\boldsymbol{x}_i \in C_{obs}$,
- agent is in a collision with another agent:

$$A(\boldsymbol{x}_i) \cap (\bigcup_{j,j\neq i} A(\boldsymbol{x}_j)) \neq \emptyset, \tag{3.4}$$

- all agents passed the given sequence of waypoints,
- any other agent has been terminated.

The last condition (termination of one agent results in termination of all agents) is not necessary. Generally, this condition is unwanted, since termination of one quadrotor should not automatically result in termination of all quadrotors. However, we impose this condition, because deem the mission to be successful only if all agents passed the given sequence of waypoints. Moreover, the RL algorithm is built in a way that it iteratively improves a policy to maximize the reward. It would not satisfy us if the algorithm converged to a local reward maximum with a policy, that would consistently crash one or more agents during the flight.

It might be worth mentioning, that the environment, as described here, is well beyond what can be modeled with MDP. First of all, the policy is based on observations of the agents, not the full states of the agents. Second of all, multiple agents are interacting with the same environment. According to the related literature (such as [21]), this type of problem can be modeled with a Decentralized Partially Observable Markov Decision Process (Dec-POMDP), which is an extension of MDP and Partially Observable Markov Decision Process (POMDP). An n-agent Dec-POMDP is a tuple $(I, S, \{A_i\}_i, P, R, \{\Omega_i\}_i, O, T, b_0)$, where I is a finite set of agents, S is a finite set of states, A_i is a finite set of actions for agent i, $P : (s, a) \to s'$ is the transition probability function, $R : (s, a) \to \mathbb{R}$ is reward function, Ω_i defines a finite set of observations available for agent i, $O(o|s, a) \in [0, 1]$ is a function of observation probabilities, T is a finite horizon and b_0 is initial belief. However, this work is not focused on the theoretical background and methods in Dec-POMDP, and our approach to the task is via RL with neural network function approximators, so Dec-POMDP will not be further discussed.

The observations from the environment are typically information collected by the sensors of the agent. The actions that the agent performs, are the control inputs. The control inputs are typically single rotor thrusts or other physical properties from which the rotor thrusts can be calculated or controlled. Since we work with discrete time steps, the action (eg. the single rotor thrusts) is executed by the agent for the whole duration of the time step. The way that the actions affect the state of the quadrotor in the environment is described by quadrotor dynamics.

3.3 Quadrotor dynamics

Before we dive into details about the quadrotor dynamics, we first need to clarify the terms *world frame* and UAV's *body frame*, since these terms are important for this section. The world frame is a fixed reference coordinate system that is stationary relative to a specific reference point. It is independent of the motion of the UAV. The position and orientation of

objects are described relative to a set of fixed axes. The axes of the world frame are represented as the x-, y-, and z-axes, corresponding to forward, leftward, and upward directions, respectively. In the real world, the world frame typically corresponds to GNSS. On the other hand, the body frame is a frame fixed to the UAV itself, so it moves and rotates with the motion of the UAV. The axes of the body frame are also represented as the front (x-axis), left (y-axis), and upward (z-axis) directions of the UAV.

Now we will focus on quadrotor dynamics itself. We use the same dynamics model as in the research paper [3]. The state (also referred to as configuration) of the quadrotor is defined as $\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{q}, \boldsymbol{\omega}, \boldsymbol{\Omega}]^T$, where $\boldsymbol{p} \in W$ is the position of the quadrotor, $\boldsymbol{v} \in \mathbb{R}^3$ is quadrotor's velocity, $\boldsymbol{q} \in SO(3)$ is quadrotor's rotation (around pivot point $\boldsymbol{p}$) represented in quaternions, $\boldsymbol{\omega} \in \mathbb{R}^3$ is quadrotor's angular velocity in quadrotor's body frame, and $\boldsymbol{\Omega} \in \mathbb{R}^4$ is the rotational speed of each rotor. The dynamics equations are:

$$\dot{\boldsymbol{p}} = \boldsymbol{v}, \tag{3.5}$$

$$\dot{\boldsymbol{v}} = \frac{R(\boldsymbol{q})(\boldsymbol{f}_T + \boldsymbol{f}_D)}{m} + \boldsymbol{g}, \tag{3.6}$$

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \odot \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}, \tag{3.7}$$

$$\dot{\boldsymbol{\omega}} = \boldsymbol{J}^{-1}(\boldsymbol{\tau} - \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega}), \tag{3.8}$$

$$\dot{\boldsymbol{\Omega}} = \frac{\boldsymbol{\Omega}_c - \boldsymbol{\Omega}}{k_{mot}}, \tag{3.9}$$

where:

- $R(\boldsymbol{q}) \in SO(3)$ is the rotation matrix given by quaternion $\boldsymbol{q}$,
- $\boldsymbol{f}_T \in \mathbb{R}^3$ is the collective thrust,
- $\boldsymbol{f}_D \in \mathbb{R}^3$ is the drag force,
- $m \in \mathbb{R}^+$ is the mass of the quadrotor,
- $\boldsymbol{J} \in \mathbb{R}^{3\times3}$ is diagonal inertia matrix,
- $\boldsymbol{g} = \begin{bmatrix} 0 & 0 & -9.81 \end{bmatrix}^T \in \mathbb{R}^3$ denotes Earth's gravity,
- $\boldsymbol{\Omega}$ is the speed of the propellers, $\boldsymbol{\Omega}_c$ is the commanded speed of the propellers and k_{mot} is the time constant.

The individual motor thrusts are calculated as functions of rotors' angular velocities:

$$f_i(\Omega_i) = [c_f \cdot \Omega_i^2] \text{ for } i \in \{1, \ldots, 4\}, \tag{3.10}$$

where c_f is a thrust coefficient. The thrust of individual rotors is then limited to range $[f_{min}, f_{max}]$. Collective thrust is calculated as:

$$\boldsymbol{f}_T = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} f_i \end{bmatrix}, \tag{3.11}$$

and the torque $\boldsymbol{\tau}$ is calculated as:

$$\boldsymbol{\tau} = \begin{bmatrix} \frac{l}{\sqrt{2}}(f_1 - f_2 - f_3 + f_4) \\ \frac{l}{\sqrt{2}}(-f_1 - f_2 + f_3 + f_4) \\ \kappa(f_1 - f_2 + f_3 - f_4) \end{bmatrix}, \tag{3.12}$$

where l is arm length of the quadrotor and κ is the torque constant. The velocity in the body frame $\boldsymbol{v}_B$ [15] is calculated as:

$$\boldsymbol{v}_B = R(\boldsymbol{q})^{-1} \cdot \boldsymbol{v}. \tag{3.13}$$

The drag force $\boldsymbol{f}_D$ is modelled as linear function of velocity in body frame $\boldsymbol{v}_B$ [15] with drag coefficients (k_x, k_y, k_z):

$$\boldsymbol{f}_D = - \begin{bmatrix} k_x v_{B,x} \\ k_y v_{B,y} \\ k_z v_{B,z} \end{bmatrix}. \tag{3.14}$$

The exact values of the coefficients are mentioned in Chapter 5.

3.4 Formulation of the problem

Suppose we have an environment as described in 3.2. We denote the set of obstacles O, a non-empty set of quadrotors A, and a sequence of waypoints $(G_1, \ldots, G_n)$. Let's suppose that all quadrotors behave according to a policy π. The goal for this environment is to find a specific policy π, according to which all quadrotors will have a non-collisional flight in the environment and all quadrotors would pass all waypoints in the given sequence $(G_1, \ldots, G_n)$, while keeping the swarm coherent. According to this policy, each quadrotor behaves in a way to create a specific sequence of interactions: $s_0, a_0, r_1, s_1, a_1, r_2, s_2, ..., s_k$ where $a_0, \ldots, a_{k-1}$ are the actions of the quadrotor, $s_0, \ldots, s_{k-1}$ are non-terminal states and s_k is the state where all quadrotors passed the given sequence of waypoints, and $r_1, \ldots, r_{k-1}$ are the rewards from the environment.

Chapter 4

Methodology

This chapter describes our approach to solving the flight control task of a swarm of quadrotors. The first section describes the simulator, environment modeling, and detection of collisions in the simulator. The second section describes communication with chosen RL library. Note, that in the implementation part, we worked with quadrotors only as the representatives of the UAVs.

4.1 Simulator

In order to solve our problem, I in cooperation with Vít Knobloch have developed a quadrotor mechanics simulator (written primarily in C++) and integrated it with a standardized OpenAI Gym [18] interface for interaction with existing RL libraries. We both used this simulator for experiments: while Vít focused on using RL for agile single quadrotor flight control, I focused on quadrotor swarming.

The simulator has the ability to simulate multiple environments simultaneously and also multiple quadrotors in a single environment. Thanks to the multi-environment simulation ability, the learning could be parallelized across multiple threads (the OpenMP library [1] has been used for this purpose). It was important to ensure that the simulator runs at high speeds to enable fast training. To achieve this, the rendering was not part of the simulator and instead, we used Blender [2] for the final flight visualizations.

4.1.1 Environment

This section focuses on obstacle and quadrotor modeling in the environment. The quadrotor-obstacle or quadrotor-quadrotor collision detection are also discussed here.

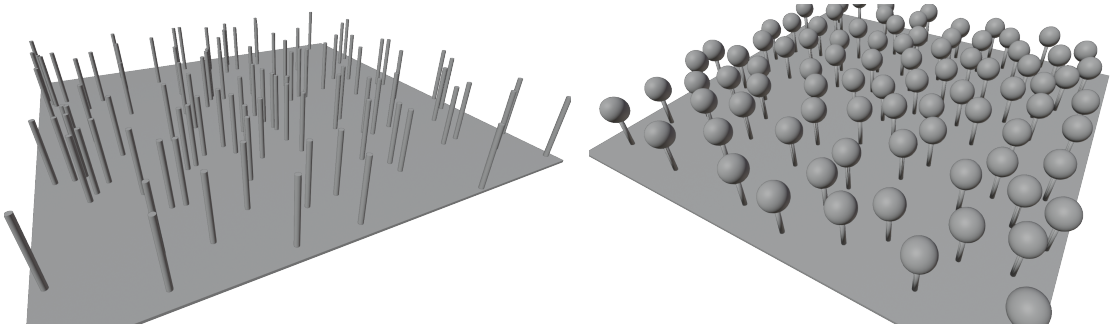
In the environment, all obstacles are represented by triangulated mesh, which enabled us to model complex environments. Examples of complex environments are depicted in Fig. 4.1. The quadrotors were also modeled by triangulated mesh, but for the implementation simplicity and computation speed, they were replaced with spheres (Fig. 4.2) in the simulator.

Because of the spherical representation of quadrotors, the collision of any two quadrotors in the swarm can be checked by:

$$\|\boldsymbol{p}_i - \boldsymbol{p}_j\| < 2 \cdot r \text{ for any } i, j \in \{1, \ldots, N\}, i \neq j, \tag{4.1}$$

where $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathbb{R}^3$ are positions of i-th and j-th quadrotor in the swarm, N is the number of quadrotors in the swarm and r is the radius of the quadrotor's sphere (Fig. 4.2b).

[1] `https://www.openmp.org/` (accessed 1.5.2023)
[2] `https://www.blender.org/` (accessed 1.5.2023)



(a) Random columns (b) Replica of forest presented in [2]

Figure 4.1: Examples of complex environments used in this work.

(a) Quadrotor mesh (b) Sphere representation of the mesh in the simulator

Figure 4.2: Quadrotor mesh and the simulator representation of the quadrotor as the smallest sphere that envelopes the whole quadrotor mesh.

For quadrotor-obstacle collision detection, we used Euclidean Signed Distance Field (ESDF) [19] . ESDF represents the environment as a 3D matrix of scalar values. Each element in the matrix represents a point in the environment and the value represents the distance from the point to the nearest obstacle. The sign of the scalar value indicates whether the point is inside or outside of an obstacle. Since creating the ESDF from the environment with meshes is a computationally demanding action, the ESDF is precomputed before the learning. During the flight, the ESDF serves as a look-up table representing the distance from the closest obstacle.

One of the main advantages is that ESDF allows for very fast collision detection, as it only requires a simple look-up in the matrix. Hence, it detects a collision in constant time and no extra computation is required. On the other hand, the limitation of this approach is that the method requires pre-computing the ESDF matrix. That makes it impractical for simulations where the environment changes over time since dynamic obstacles are not taken into account during the pre-computation. For that reason, we lose the ability to simulate environments with moving obstacles. However, we need fast simulations because the policy learning process takes typically several hunderds milions of time steps. It is therefore a suitable method for us.

4.2 Interaction with reinforcement learning library

For interaction with the RL library, our simulator has been provided with a standardized OpenAI Gym [18] interface. The Stable Baselines 3 (SB3) [6] was used as the RL library. The SB3 is an open-source library of RL algorithms built on top of the machine-learning platform PyTorch [12]. It is designed to use state-of-the-art RL algorithms by providing a common interface and implementation for RL algorithms. From the variety of RL algorithms that SB3 provides, we chose the Proximal policy optimization (PPO) [17] method.

4.2.1 Proximal Policy Optimization

PPO [17] is a type of RL algorithm that uses neural network function approximators. PPO is a policy gradient method. In policy gradient methods, the policy π_θ is a neural network (sometimes referred to as the *actor*) parametrized by a set of weights θ. The algorithms use gradient ascent to iteratively update the policy parameters to improve the policy. Instead of using a buffer to store all past experiences, the neural network is updated based on the last batch of experience only. The batch of experience is discarded after the policy gradient update.

In PPO, there is also a second neural network (called the *critic*). This neural network works as a value function. It tries to predict the discounted reward from the current state onward. During the learning, this neural network is also frequently updated using the experience that the agent collects in the environment.

The PPO algorithm consists of the following steps:

1. Collect experience: the agent interacts with the environment by executing the current policy.
2. Compute advantages $\hat{A}_t = r_t - b(s_t)$ for each run. For a given run, the advantage $\hat{A}_t$ at a given time step t is the difference between the discounted sum of rewards r_t and the value function estimate $b(s_t)$ (sometimes, the value function estimate is called the

baseline estimate, that is why we use the symbol b for that function). Basically, if the advantage $\hat{A}_t > 0$, then the action the agent took in the state s_t was better than expected, and if $\hat{A}_t < 0$, then the action was worse.

3. Compute clipped surrogate objective (explained later).
4. Optimize policy: the policy parameters are updated by taking gradient steps to maximize the surrogate objective.
5. Repeat steps 1-4 until convergence.

In PPO, the clipped surrogate objective is used to update the policy parameters. It is designed to prevent the policy from changing too much in a single update step, which in some cases has negative effects on the stability of the learning process. The clipped surrogate objective is computed as follows: first, the ratio of probabilities $r_{so} : \theta \to \mathbb{R}_0^+$ of taking an action under the new policy and the old policy is calculated:

$$r_{so}(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}. \tag{4.2}$$

Here, $\pi_\theta(a_t|s_t)$ is the probability of taking action a_t in state s_t under the new policy parameterized by θ, and $\pi_{\theta_{old}}(a_t|s_t)$ is the probability of taking the same action under the old policy parameterized by θ_{old}.

Next, the clipped surrogate objective is computed:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_{so}(\theta)\hat{A}_t, \text{clip}(r_{so}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right], \tag{4.3}$$

where the $\hat{\mathbb{E}}_t$ indicates the empirical average over a batch of samples, and the parameter $\epsilon \in [0, 1]$ is a hyperparameter that controls the amount of clipping. The $r_{so}(\theta)\hat{A}_t$ is the default objective for policy gradient methods. It "pushes" the policy towards the actions, that have a positive advantage $\hat{A}_t$ over the baseline. The term $\text{clip}(r_{so}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ is the clipped version of the default objective. In this way, the algorithm prevents the updates of the policy from being too large. The objective function is then maximized using gradient ascent to update the policy parameters θ.

The whole algorithm, as proposed in [17] is described in pseudocode here:

Algorithm 1 PPO, Actor-Critic Style

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

The algorithm PPO requires a couple of hyperparameters. For most of these hyperparameters, SB3 provides a default value. To better suit our task, some of the hyperparameters have been tweaked from their default values provided by SB3.

We set the neural network of both the actor and the critic to consist of 3 fully connected hidden layers, where the first and the second hidden layer has 512 nodes and the third hidden layer has 256 nodes. Initially, we tried using *tanh* activation function, but this activation

function worked poorly in some cases, arguably due to the vanishing gradient, so $ReLU$ was used as the activation function instead. We set the upper limit of the episode length to 1500 time steps (where each time step is set to $\Delta t = 0.02s$ in the default setting), although even the longer episodes typically did not surpass the length of 600 time steps (note: the maximal episode length and the length of the time step are not hyperparameters of PPO). To prevent the agents from overfitting to just a section of a track, the batch size for the neural network is set to 25000, which corresponds to multiple episodes of samples. The discount factor is set to $\gamma = 0.98$ and the learning rate is set to linearly decrease throughout the learning process from $3 \cdot 10^{-4}$ to $3 \cdot 10^{-5}$. The other hyperparameters were left at their default values.

To know more about the training progress, we use periodical plots of the quadrotors' paths (Fig. 4.3). We also used Tensorboard [3], which is a web-based tool used to track and visualize important metrics during learning (such as mean episode length, mean episode reward, and particularly the reward components, described later in Section 4.2.4).

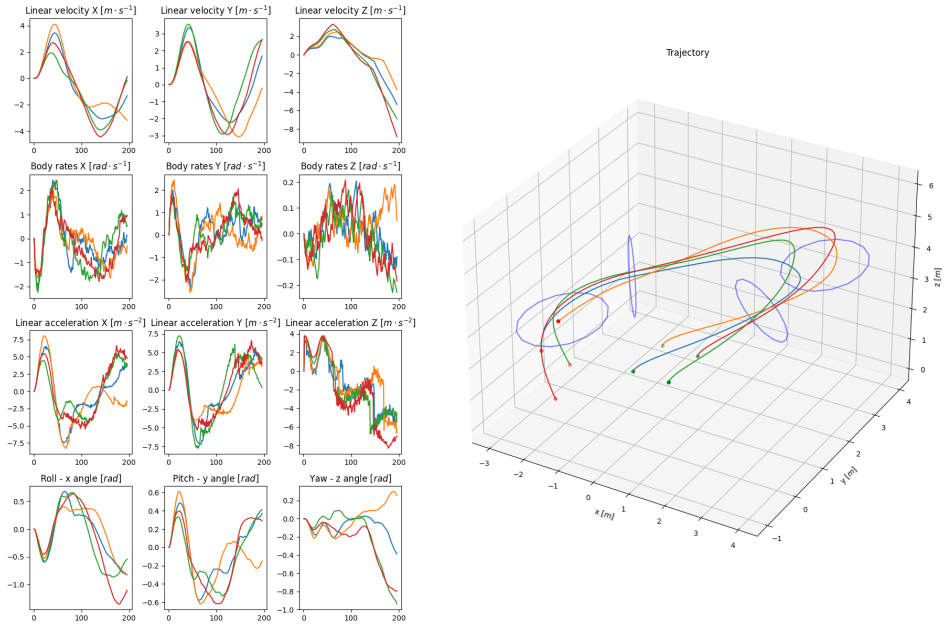


Figure 4.3: An example of a periodical plot, where four quadrotors fly through four waypoints.

4.2.2 Action space

In the context of RL, action refers to the decision made by the agent's policy at each time step. In our model, the action space $S_A = [f, \boldsymbol{\omega}_c]^T \subset \mathbb{R}^4$, representing total thrust f, and commanded body rates $\boldsymbol{\omega}_c$. Note, that $\boldsymbol{f}_T$ from Section 3.3 directly corresponds to the total thrust from the policy's action: $\boldsymbol{f}_T = \begin{bmatrix} 0 & 0 & f \end{bmatrix}^T$. We use body rate PID to produce the single rotor thrusts from the commanded $\boldsymbol{\omega}_c$. The 4th order Runge-Kutta scheme [26] is used for the forward integration of the dynamics 3.5 – 3.14. The action is performed for a fixed time step (our simulation used a default time step of $\Delta t = 0.02s$).

[3] `https://www.tensorflow.org/tensorboard/` (accessed 12.5.2023)

4.2.3 Observation space

Observation refers to the information that the agent receives about its state in the environment. Two approaches have been tried based on the different structures of the observation. The success rate of these two approaches is then compared on the random columns environment (Fig. 4.1a).

The observation in one approach contains all the necessary information about the agent's state as well as the perfect knowledge about surrounding obstacles. It has the following form: $S_O = [\boldsymbol{p}, \boldsymbol{v}, R(\boldsymbol{q}), \boldsymbol{p}_{wp}, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{n-1}, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_m]^T \subset \mathbb{R}^{18+3\cdot(n-1)+2\cdot m}$, where

- $\boldsymbol{p} \in \mathbb{R}^3$ is the position of the quadrotor,
- $\boldsymbol{v} \in \mathbb{R}^3$ is quadrotor's velocity,
- $R(\boldsymbol{q}) \in SO(3)$ is quadrotor's rotation in form of rotation matrix,
- $\boldsymbol{p}_{wp}$ is the relative position of the waypoint,
- $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{n-1} \in \mathbb{R}^3$ denotes relative position of other quadrotors in the swarm (n denotes number of quadrotors in the swarm),
- $\boldsymbol{r}_j \in \mathbb{R}^2$ denotes the relative position of the j-th nearest column to the quadrotor. We considered only relative position in the x-axis and y-axis of the world frame because the columns in this environment are modeled to be vertical. We worked with $m = 5$ nearest columns in the observation.

Note, that in Section 3.3, we defined the angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$ and the rotational speeds of each rotor $\boldsymbol{\Omega} \in \mathbb{R}^4$ as a part of the agent's state. However, $\boldsymbol{\omega}$ is a part of the action of the quadrotor (as described in Section 4.2.2). The $\boldsymbol{\Omega}$ is not a part of the action, but it is controlled by PID body rate controller using the supplied policy action. Therefore, we deemed $\boldsymbol{\omega}$, and $\boldsymbol{\Omega}$ to be unnecessary for the observation of the agent.

The observation in the other approach is similar to the one mentioned above, however, it lacks the absolute position of the quadrotor and it contains vision-based information from 2D Light Detection and Ranging (LiDAR) instead of the relative position of the columns: $S_O = [\boldsymbol{v}, R(\boldsymbol{q}), \boldsymbol{p}_{wp}, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{n-1}, d_1, \ldots, d_m]^T \subset \mathbb{R}^{18+3\cdot(n-1)+m}$, where $d_j \in \mathbb{R}^+$ denotes the distance of the closest point in the j-th partition of point cloud recorded by the LiDAR (partitioning of LiDAR's point cloud is explained later).

For implementation simplicity, the LiDAR was set to make a full revolution in every time step, resulting in a spinning frequency of 50 Hz (default time step length in our simulation was $\Delta t = 0.02s$). We experimented with the number of steps per revolution – LiDAR was the most computationally demanding part of the simulation, so too many steps would slow down simulations significantly, and too few steps per revolution might result in a quadrotor "overseeing" a close obstacle. In the end, 60 steps per revolution proved to be sufficient, which resulted in a sampling frequency of 3000 samples/second.

Neural networks often learn better with smaller observation dimensions because having fewer input features reduces the complexity of the learning problem. We deemed 60 points from LiDAR too much information for the neural network, from which most of the data is irrelevant. For this problem, only the closest points in the recorded point cloud are relevant. Therefore, the number of points passed to observation from the point cloud has been reduced in the following way: the space around the quadrotor has been partitioned into m partitions (as depicted on Fig. 4.4, we used default value $m = 6$) and only the closest point recorded by the LiDAR in each partition has been considered. Then, the distance of the closest point was passed into the observation. This allowed us to reduce the dimension of the observation space significantly.

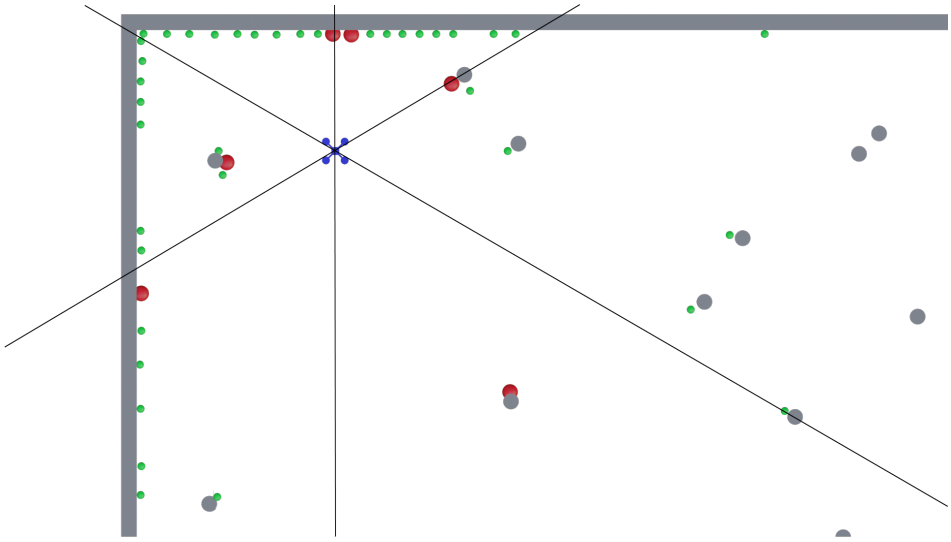


Figure 4.4: Visualization of observation from LiDAR. The quadrotor is depicted in blue, the obstacles are dark gray, the point cloud from LiDAR is green and the reduced observation set is red. Black lines denote the partitioning of the surrounding environment into 6 partitions. [4]

Note, that we used the matrix from ESDF to cast the LiDAR rays. Because of this approach, LiDAR was not implemented to detect other quadrotors. Instead, the agent was provided with the relative position of other agents in the swarm as part of the observation.

4.2.4 Reward components

The reward is an important part of RL as it guides the agent's learning process by providing a measure of how well the agent is performing its task. The goal of the agent is to learn a policy that maximizes the cumulative reward over time. To achieve desired behavior of the policy, the reward consists of several components: *pass goal*, *progress*, *collide*, *obstacle avoidance*, *velocity*, *angular velocity*, *separation*, *cohesion*, and *velocity alignment*.

The **pass goal** reward is a reward for passing each waypoint in the course. This reward component is dependent on the distance of the passing point from the center of the waypoint and on the tolerance of the waypoint (radius of the circle that represents the waypoint):

$$r_g = r_{\psi g} \cdot \exp \frac{-\|\boldsymbol{p} - \boldsymbol{p}_g\|}{\alpha}, \tag{4.4}$$

where $r_{\psi g}$ is a reward hyperparameter for passing the goal, $\boldsymbol{p} \in \mathbb{R}^3$ is the position of the quadrotor, $\boldsymbol{p}_g \in \mathbb{R}^3$ is the position of the center of the waypoint, and $\alpha \in \mathbb{R}^+$ is tolerance of the waypoint.

The **progress** reward encourages the agent to fly towards its next waypoint. It is positive if the agent moves toward the waypoint and negative if the agent moves away:

$$r_{\Delta dist} = (\|\boldsymbol{p}_1 - \boldsymbol{p}_g\| - \|\boldsymbol{p}_2 - \boldsymbol{p}_g\|) \cdot r_{\psi \Delta dist}, \tag{4.5}$$

where $\boldsymbol{p}_1 \in \mathbb{R}^3$ is the position of the agent in the previous step, $\boldsymbol{p}_2 \in \mathbb{R}^3$ is the position of the agent in the current step, $\boldsymbol{p}_g \in \mathbb{R}^3$ is the position of the next waypoint to fly through, and $r_{\psi \Delta dist} \in \mathbb{R}^+$ is a reward hyperparameter for progress.

[4] A render of a flight of a single quadrotor with visualization of the point cloud from LiDAR is available on YouTube: `https://youtu.be/gmHftUmAdtE`.

The **collide** reward is a reward to penalize collision with an obstacle or with another quadrotor:

$$r_c = r_{\psi c}, \tag{4.6}$$

where $r_{\psi c} \in \mathbb{R}^-$ is a reward hyperparameter for collision.

The **obstacle avoidance** reward component is a negative reward for flying too close to obstacles. The formula for this reward varied based on what type of observation was used. When using observation with the agent's position and the relative position of the nearest columns, the reward was calculated as follows:

$$r_{oa} = max\left(0, d_{obst\_sep} - d_{obst}\right) \cdot r_{\psi oa}, \tag{4.7}$$

where $d_{obst} \in \mathbb{R}^+$ is the distance to the nearest obstacle (this value is obtained from the ESDF matrix), $d_{obst\_sep} \in \mathbb{R}^+$ is a hyperparameter, that determines the maximum distance from an obstacle when the negative reward for obstacle avoidance starts to apply, and $r_{\psi oa} \in \mathbb{R}^-$ is reward hyperparameter for obstacle avoidance. When using vision-based observation, the reward for obstacle avoidance is:

$$r_{oa} = \sum_{i=1}^{m} max\left(0, \frac{\boldsymbol{p}_i^T \cdot \boldsymbol{v}}{1 + \|\boldsymbol{p}_i\|^2}\right) \cdot r_{\psi oa}, \tag{4.8}$$

where $\boldsymbol{p}_i \in \mathbb{R}^3$ is the relative position of the closest point in the i-th partition of the point cloud recorded by the LiDAR, $\boldsymbol{v} \in \mathbb{R}^3$ is the velocity of the quadrotor, and $r_{\psi oa} \in \mathbb{R}^-$ is reward hyperparameter for obstacle avoidance.

The **velocity** reward is a negative reward for flying either too fast or too slow. If the lower end of the speed is not awarded negatively, the agent can become stuck in a local minimum. That would result in the agent hovering in one position, which is an undesired behavior. By punishing high speeds with negative rewards, we encourage the agent to stay within a specific speed range, which reduces the search space for the RL algorithm and therefore makes the learning problem easier. The *velocity* reward component is calculated as follows:

$$r_v = \begin{cases} r_{\psi v} \cdot 20^{v_{min} - \|\boldsymbol{v}\|} - 1 & \text{for } \|\boldsymbol{v}\| < v_{min}, \\ 0 & \text{for } v_{min} \leq \|\boldsymbol{v}\| \leq v_{max}, \\ r_{\psi v} \cdot 10^{\|\boldsymbol{v}\| - v_{max}} - 1 & \text{otherwise}, \end{cases} \tag{4.9}$$

where $v_{min}, v_{max} \in \mathbb{R}^+$ are hyperparameters that denote the minimal and maximal desired speed of the agent (we used $v_{min} = 0.2m \cdot s^{-1}$, $v_{max} = 1.5m \cdot s^{-1}$), $\boldsymbol{v} \in \mathbb{R}^3$ is the agent's speed, and $r_{\psi v} \in \mathbb{R}^-$ represents *velocity* reward hyperparameter.

The **angular velocity** reward is a negative reward that discourages quick changes in the quadrotor's orientation. If angular velocity is not awarded negatively, it might lead to the agent exploiting the simulated quadrotor dynamics by changing orientation very quickly. This behavior is not reproducible in the real world. The angular velocity reward is:

$$r_w = r_{\psi w} \cdot \|\boldsymbol{w}\|, \tag{4.10}$$

where $\boldsymbol{w} \in \mathbb{R}^3$ represents angular velocity of the agent and $r_{\psi w} \in \mathbb{R}^-$ is angular velocity reward hyperparameter.

To explain reward components *separation*, *cohesion*, and *velocity alignment* we first describe the Boids model.

Boids model

The Boids model [28] is a simplified model of flocking behavior used to simulate the coordinated movement of a group of animals (like birds or fish). The model is based on three simple rules: separation, velocity alignment, and cohesion. These rules describe how individual agents in a swarm should interact with the environment and with each other to achieve coordinated behavior. In our work, the agents were encouraged to maintain these rules with reward.

The **separation** (Fig. 4.5a) rule states that each agent should avoid colliding with other agents by maintaining a minimum distance from them. This helps to prevent congestion and ensures that each agent has enough space to move freely. In our work, the *separation* reward is negative. For j-th agent in the swarm, the separation reward is calculated as follows:

$$r_s = \sum_{i=1}^{n} \begin{cases} (d(\boldsymbol{p}_i - \boldsymbol{p}_j) - d_{\psi s}) \cdot \frac{r_{\psi s}}{c - d_{\psi s}} & \text{for } i \neq j,\ d(\boldsymbol{p}_i - \boldsymbol{p}_j) < d_{\psi s}, \\ 0 & \text{otherwise,} \end{cases} \tag{4.11}$$

where $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathbb{R}^3$ denote the position of i-th and j-th quadrotor in the swarm, $r_{\psi s} \in \mathbb{R}^-$ is the reward hyperparameter for separation, $c \in \mathbb{R}^+$ is the clearance radius of the quadrotor (depicted in Fig. 4.2b), $d_{\psi s} > c$ is hyperparameter representing the maximum distance when the negative reward starts to apply and $d(\boldsymbol{p}_i - \boldsymbol{p}_j) = \sqrt{\begin{bmatrix} 1 & 1 & \frac{1}{4} \end{bmatrix} \cdot (\boldsymbol{p}_i - \boldsymbol{p}_j)}$ denotes a norm partially neglected in the z-axis of the world frame.

Note the obscure norm we use for the calculation of the distance between i-th and j-th quadrotor. When running the experiments, we noticed that the quadrotors exploited the fact that they could fly above one another. In the real world, this behavior might result in a fatal crash, as the wind from the rotors of the upper agent could cause instability of the lower one. The simulator is not implemented to consider this wind effect. For that reason, we adjust the separation reward calculation by partially disregarding the distance between the quadrotors in the z-axis, which incentivizes the agents not to fly above one another.

The **cohesion** (Fig. 4.5b) rule states that each agent should move towards the center of mass of its neighbors. This helps to keep the group together and ensures that the agents maintain a consistent spatial configuration. The formula for cohesion reward of j-th agent in the swarm is:

$$r_c = \left\| \boldsymbol{p}_j - \frac{1}{n-1} \sum_{i=1}^{n} \begin{cases} \boldsymbol{p}_i & \text{for } i \neq j, \\ 0 & \text{otherwise,} \end{cases} \right\| \cdot r_{\psi coh}, \tag{4.12}$$

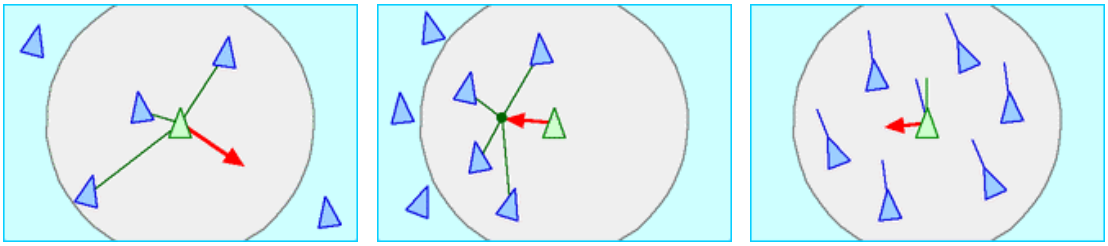
where $\boldsymbol{p}_j, \boldsymbol{p}_i \in \mathbb{R}^3$ are positions of the j-th and i-th agent in the swarm, $n \in \mathbb{N}$ is the number of the agents in the swarm, and $r_{\psi coh} \in \mathbb{R}^-$ is the cohesion reward hyperparameter.

The **velocity alignment** (Fig. 4.5c) rule states that each agent should align its velocity with the average velocity of its neighbors. This helps to ensure that the agents move in roughly the same direction and maintain a consistent velocity. The velocity alignment for j-th agent is calculated using the equation:

$$r_{va} = \left\| \boldsymbol{v}_j - \frac{1}{n-1} \sum_{i=1}^{n} \begin{cases} \boldsymbol{v}_i & \text{for } i \neq j, \\ 0 & \text{otherwise,} \end{cases} \right\| \cdot r_{\psi va}, \tag{4.13}$$

where $\boldsymbol{v}_j, \boldsymbol{v}_i \in \mathbb{R}^3$ are velocities of j-th and i-th agents in the swarm, $n \in \mathbb{N}$ is the number of agents in the swarm and $r_{\psi va} \in \mathbb{R}^-$ is reward hyperparameter for velocity alignment.

We used Tensorboard [10] for monitoring the training progress and visualization of the reward components (Fig. 4.6).



(a) Separation (b) Cohesion (c) Alignment

Figure 4.5: The rules in the Boids model. [5]

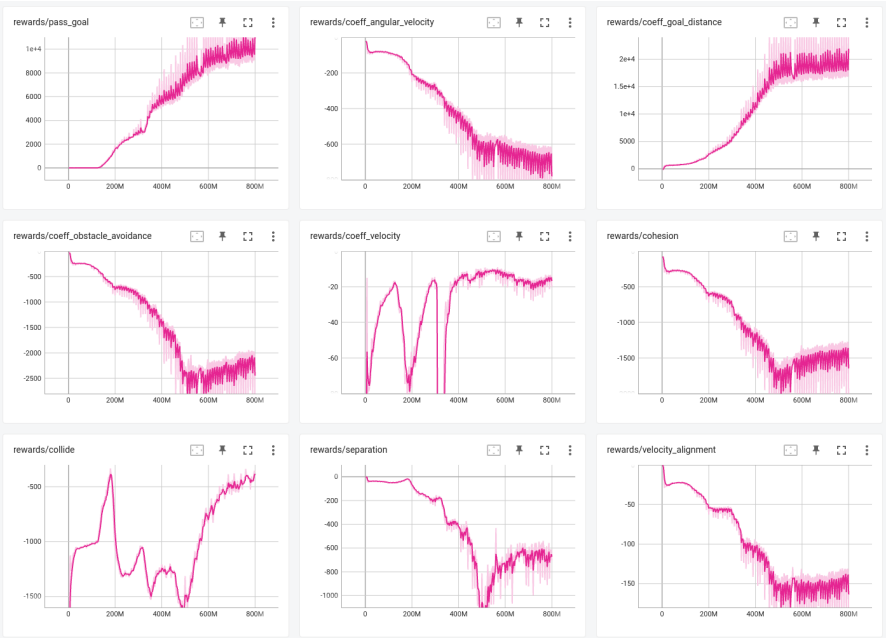


Figure 4.6: Tracking of rewards components throughout the learning using Tensorboard. The reward named *coeff_goal_distance* in the Tensorboard refers to the *progress* reward.

4.2.5 Checkpoints

To speed up the learning process, we used *checkpoints*. Whenever any quadrotor in the swarm passes a waypoint, the *swarm state* (the full state of each agent in the swarm) is saved with a given probability $p_{save} \in [0, 1]$ (this probability is a hyperparameter, we set $p_{save} = 0.2$). We call these saved swarm states *checkpoints*. Each waypoint on the track holds an array of checkpoints. Whenever the agents are terminated and a new episode begins, the simulator either sets the state of all agents to their initial state or to a random checkpoint with a given probability p_{reset} (we used $p_{reset} = 0.5$). In this way, the agents have a higher probability to encounter the situations in the later stages of the track, which gives them more experience to learn from these situations. For that reason, the checkpoints speed up the learning process.

It is also worth mentioning, that each waypoint could hold only a limited amount of checkpoints (in our simulator, this amount was set to 8). When the array of checkpoints for a given waypoint is full and a new swarm state needs to be saved, it just replaces the oldest record.

[5] Image source: `https://www.red3d.com/cwr/boids/` (accessed 13.4.2023)

Chapter 5

Results

This chapter contains two sections. In the first section, we evaluate the success rates of two types of policies based on different observations (as described in 4.2.3). In the second section, the vision-based policies are compared with PACNav [2], which is a method for UAV swarming.

The hyperparameters of the reward components (described in Section 4.2.4) were tuned in a way that collective rewards from the individual components were ordered by the particular priority of the component: $r_{\text{progress}} \gg -r_{\text{collide}} \gg r_{\text{pass\_goal}} \approx -r_{\text{obstacle\_avoidance}} \approx -r_{\text{separation}} \approx -r_{\text{cohesion}} \gg -r_{\text{velocity}} \approx -r_{\text{velocity\_alignment}} \approx -r_{\text{coeff\_angular\_velocity}}$. We deemed the 5 reward components (progress, collide, pass goal, obstacle avoidance, and separation) to be the most important because that is the main objective of the swarm - to fly through multiple goals and not collide with obstacles or with each other. The other 4 reward components (cohesion, velocity, velocity alignment, and angular velocity) were not as important to us, because they were not directly linked to the main objective. Instead, these components reduced of the search space for the RL algorithm, because they discouraged an undesired behavior of the agents (like flying too far apart or flying too fast).

The parameters of quadrotor dynamics (Section 3.3) are described in Table 5.1. The linear drag coefficients (k_{vx}, k_{vy}, k_{vz}) are randomized with normal distributions $N(0, k_{vx}), N(0, k_{vy}), N(0, k_{vz})$ after each restart of the episode.

Table 5.1: Parameters of the quadrotor.

	Variable	Value	Variable	Value
Quadrotor	$m\ [kg]$	0.73	$l\ [m]$	0.34
	$f_{min}\ [N]$	0	$f_{max}\ [N]$	3.4
	$diag(J)\ [Nm^2]$	[0.008, 0.008, 0.012]	$\kappa\ [-]$	0.016
	$\omega_{max}\ [rad \cdot s^{-1}]$	1000.0	$c_f\ [-]$	1.56×10^{-6}
	$k_{vx}\ [N \cdot s \cdot m^{-1}]$	0.13	$k_{vy}\ [N \cdot s \cdot m^{-1}]$	0.14
	$k_{vz}\ [N \cdot s \cdot m^{-1}]$	0.21		

5.1 Comparison of vision-based policies and policies with known state

In our work, two approaches have been tried based on the structure of the observation (as described in Section 4.2.3). Both of these approaches have been tested on 4 different tracks [1] in the random columns environment (Fig. 4.1a): one track with a single waypoint, two tracks

[1] The flight renders of the agents with vision-based observations through these tracks are available on YouTube: `https://youtu.be/Lr4l4pmz-zk`.

with two waypoints, and one track with three waypoints.

These two approaches are compared in the following aspects: success rate, distance traveled toward the goal, and learning time. Since the upper limit on the velocity of the quadrotors has been imposed (as discussed in Section 4.2.4), we do not compare these two approaches in terms of the speed of individual quadrotors. Similarly, most of the hyperparameters of the reward components are the same for both types of policies, it is, therefore, irrelevant to compare these two approaches in any metrics that are directly dependent on reward hyperparameter settings (like the distance between the quadrotors). These properties are compared later in Section 5.2.

The swarm was trained in a maximum of 1 billion steps. A powerful PC with a graphics card NVIDIA GeForce RTX 3090 and with a processor AMD Ryzen 7 5800X 8-Core with a core frequency of 3.8 GHz was used for the training. The speed of our simulator varied depending on the observation type. Typically, it had around 65000 fps on the learning with the observation, which contained information about the nearest columns and the position of the agent. With this observation, the learning took up to 4 hours 30 minutes. The learning with observations that simulate real-world sensors was significantly slower since the simulation of LiDAR was computationally expensive. With this type of observation, the simulation achieved speeds of around 42000 fps, which resulted in learning time of up to 7 hours.

We consider success to be when *all* four quadrotors fly from the starting position through all waypoints (meaning that a crash of just a single quadrotor in the swarm would result in a failure). In Table 5.2, we show the comparison of the success rates of both types of policies.

Table 5.2: Comparison of policies in success rate

id	no. of waypoints	length	success rate full state	success rate LiDAR
1	1	19.47 m	79%	86%
2	2	13.06 m	100%	100%
3	2	20.94 m	100%	95%
4	3	22.51 m	99%	97%

To find out more about the learning, the success rates and the average distance traveled toward the goal have been monitored throughout the learning. The results are shown in Fig. 5.1 – Fig. 5.4. From these figures, we can notice that the vision-based policy was typically more difficult to learn. In the first 200 million time steps, the policy that had the information about the agent's position and about the relative position of the nearest columns always outperformed the vision-based policy in the matter of distance traveled toward the goal. However, at the end of the learning, both policies were able to control the swarm throughout the environment with a relatively high success rate.

To fully describe the figures, we also need to address the instability of the learning approach, which can be seen in every graph. In some of our experiments, the success rate of the resulting policy was low, although the policy had a high success rate in some stages of the learning process. We tried to eliminate this instability by tuning the reward hyperparameters, increasing the batch size, and tuning the learning rate of the PPO algorithm. However, none of these adjustments fixed the stability of the learning algorithm and we were not able to fix this issue. The problem might be in the nature of our approach. When controlling multiple quadrotors, even a slight change of policy might result in a fatal crash and termination of the whole swarm. As mentioned in Section 4.2.1, PPO does not remember all its previous

experiences. Instead, it discards the batch of experience after the policy update. The gradient ascend step in PPO should improve the policy, however, sometimes, this update might lead to a fatal crash and termination of all agents in the swarm.

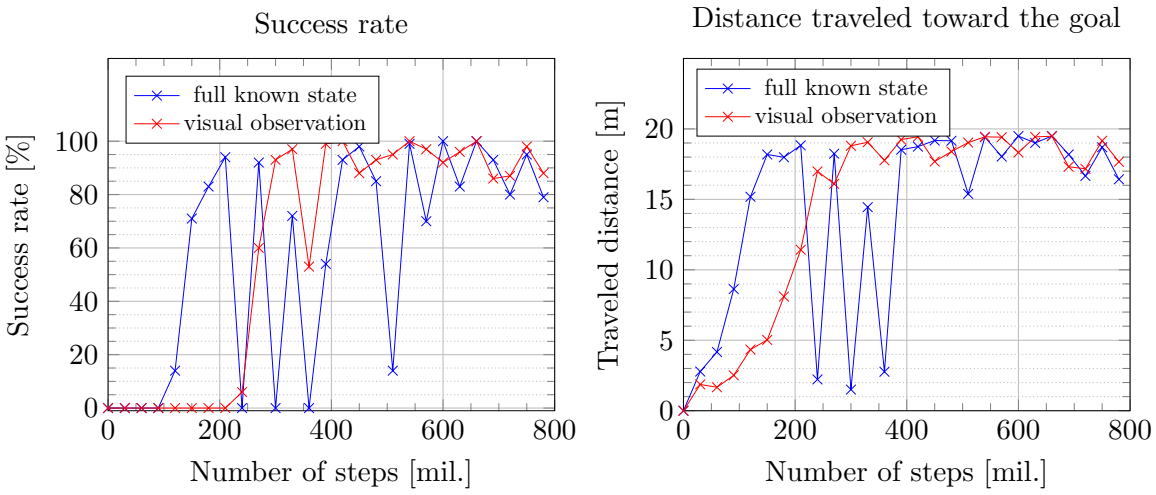


Figure 5.1: Track 1 (single waypoint)

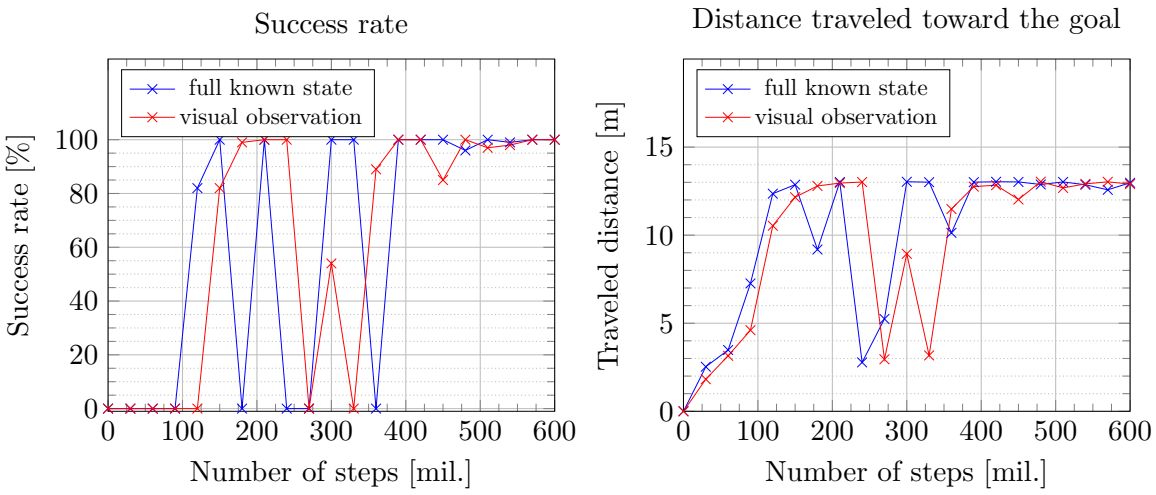


Figure 5.2: Track 2 (two waypoints)

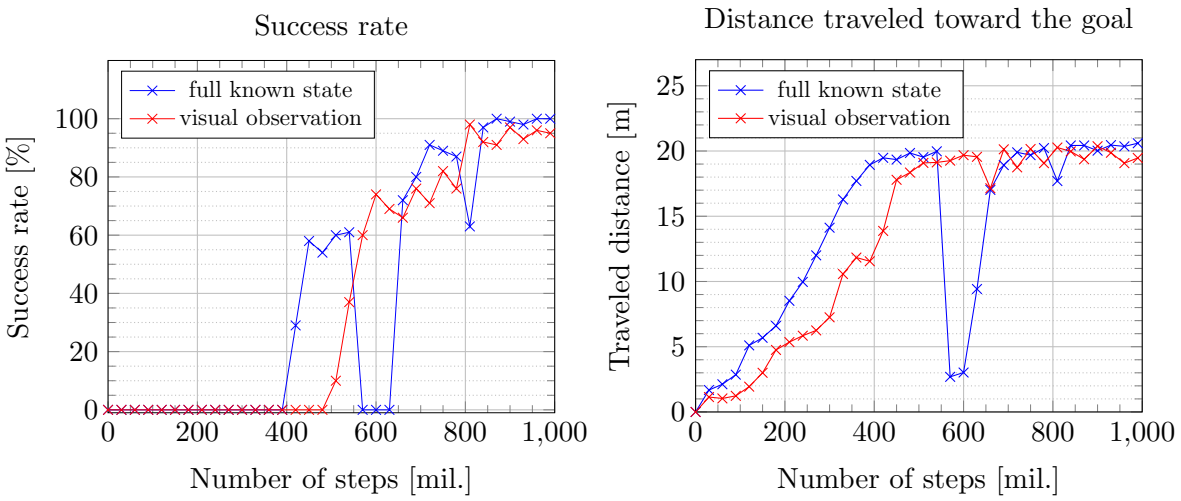


Figure 5.3: Track 3 (two waypoints)

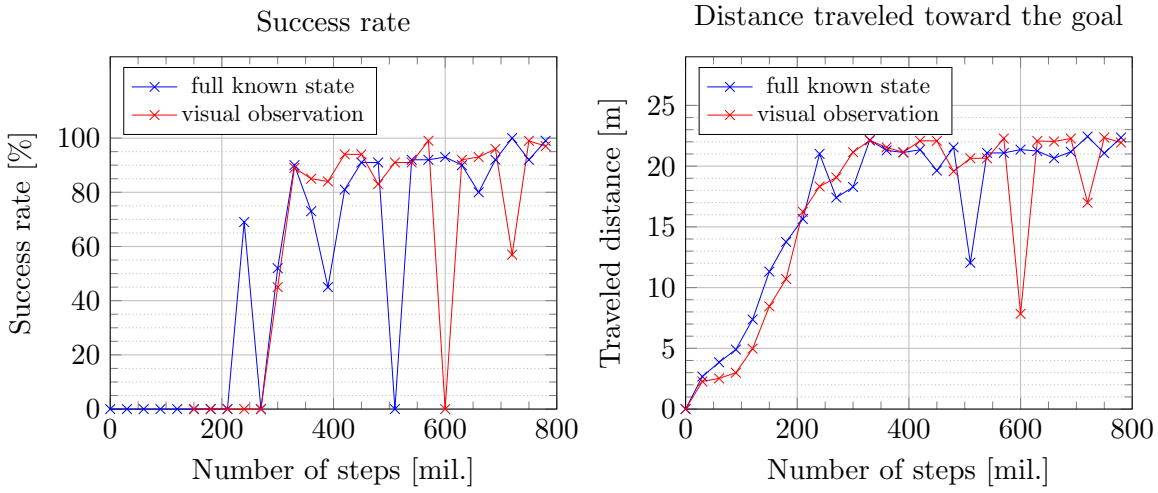


Figure 5.4: Track 4 (3 waypoints)

5.2 Comparison with PACNav

This section compares the proposed RL method with PACNav [2], which is a method for UAV swarming. We test our approach on 10 experiments with 3 UAVs [2] and then compare it to the results presented in [2]. The comparison has the following attributes: the time to fly through the track, the smallest, average, and largest distance between the quadrotors during the flight, and the order of the swarm.

The order metric [9] captures the correlation of agents' movements and gives an indication of how ordered the flock is. For a given time step k, it is expressed by

$$\Omega[k] = \frac{1}{N(N-1)} \sum_{i,j \neq i} \frac{\boldsymbol{v}_i[k]^T \cdot \boldsymbol{v}_j[k]}{\|\boldsymbol{v}_i[k]\| \|\boldsymbol{v}_j[k]\|} \in [-1, 1], \tag{5.1}$$

[2] A render of the flight on one of these tracks is available on YouTube: `https://youtu.be/zRP6rhlPVjo`.

where N is the number of UAVs in the swarm, $v_i[k], v_j[k] \in \mathbb{R}^3$ denote instantaneous velocities of i-th and j-th UAV in the swarm in the given time step k. In other words, if $\Omega = 1$, then all agents in the swarm are moving in the same direction, and $\Omega < 1$ implies a misalignment of the agents in the swarm. Fig. 5.5 shows the order metrics in a swarm of two agents.

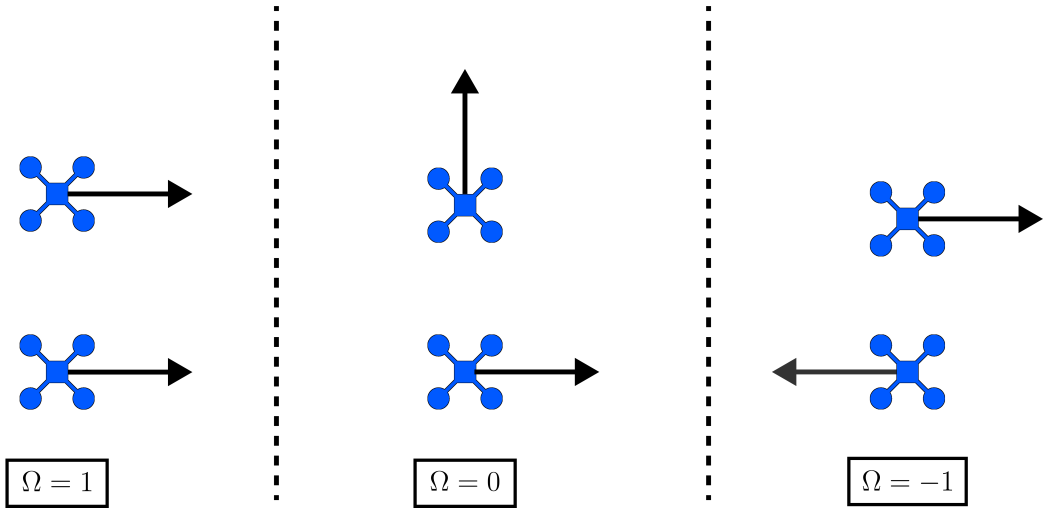


Figure 5.5: Three illustrative scenarios of the order metrics for two quadrotors. The arrows represent the movement directions of the quadrotors.

To compare our methods, we need to introduce the term *uninformed UAV*. This is a type of UAV that does not have information about the position of the goal. Instead, it follows other informed UAVs and tries to maintain a coherent swarm. The concept of the uninformed UAVs is used in every experiment in the PACNav paper and we compare our methods to their experiment consisting of a swarm of three UAVs, where one is uninformed.

In the article [2], the authors evaluate their approach using 10 forest environments with an area $50m \times 50m$. Unfortunately, their forests are randomly generated – the number of trees is fixed to 104 and the tree size is fixed too, however, the tree distribution in the environment is randomized. The paper includes an image of only one forest and the source code in their repository provides only the exact measurements of a tree model. To make the comparison fair, we create a similar replica of the forest shown in [2] in our simulator (4.1b). Since this environment is large, we run 10 independent experiments on this forest (instead of creating 10 different environments). These experiments involve varying the starting points of the UAVs and their goal points, allowing a comprehensive evaluation of the proposed approach. The distance between the start and the goal point in our experiments is $40m$, which is approximately the same as in [2].

Surprisingly, our methodology did not work in the forest environment, as the learning algorithm failed to converge entirely. One potential factor contributing to this failure is the fact that the forest from the article is sparse and large in comparison to the *random columns* environment (Fig. 4.1a), which we used in the previous section. The forest has 104 trees in an area of $2500m^2$ and the random columns environment has 100 columns in $400m^2$. For this reason, the search space for the agents is larger, so the agents spend a substantial amount of the time flying in places where they should not be. As a result, the RL algorithm is learning from these misguided locations.

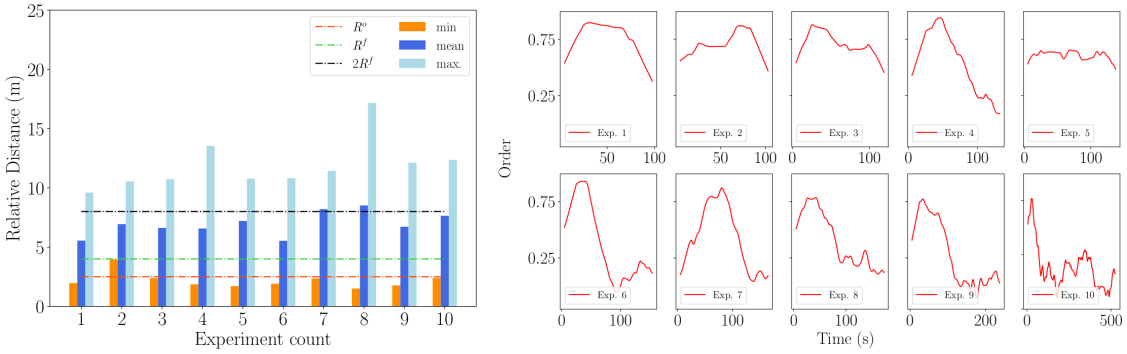
Another factor in the poor convergence rate of our methodology is that the episode always started from the beginning of the track and the track was long. For this reason, the

information present in the batch for updating the neural network predominantly consisted of data from the beginning of the track. Hence, the policy was unable to control the quadrotors in the later stages of the track at all.

To overcome these issues, we put 7 waypoints in between the starting position and the goal position. Each track in the experiment was approximately $40m$ long, which resulted in one waypoint every $5m$. This approach reduced the search space for the agents because of the structure of *progress* reward and also the positive reward for passing a waypoint incentivized the agents to stay on the course. Moreover, each time an agent flew through a waypoint, a checkpoint was saved and the agents had a certain probability to start a new episode in this saved state (as described in Section 4.2.5). For that reason, the experience of the agents was distributed more equally throughout the whole track. Also, to simplify the learning problem, we keep all quadrotors in the swarm informed, meaning, that each quadrotor has information about the relative position of the waypoint.

It can be seen from Fig. 5.6 and Fig. 5.7 that the quadrotors in our methodology tend to fly closer to each other than in [2]. In all test cases, the average distance between the quadrotors was between $1.59m$ and $2.05m$ in the proposed method, whereas according to Fig. 5.6a, the average distance between the UAVs was $5m$ or more in all their experiments. The trade-off for keeping our swarm tightly coherent was the fact that the order of the swarm in our methodology was higher than in the compared method, which also partially influenced the completion time. The average completion time of the proposed method was $26.6s$, whereas, in PACNav, it took the swarm anywhere between $100s$ and $500s$ to complete the track. Overall, our methodology is focused more on agile swarming, whereas the compared article values the safety of the quadrotors more.

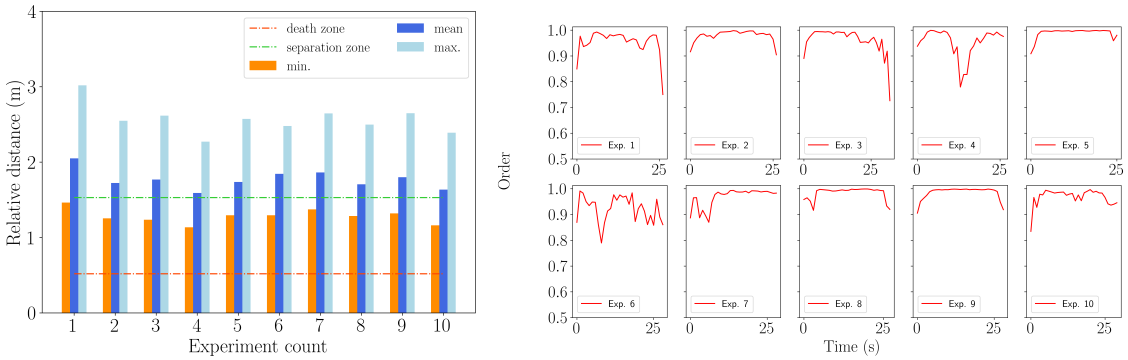
It is also important to mention, that in our approach, the policy that is controlling the quadrotors, learns by overfitting on a single track. For that reason, our methodology would probably fail if a different track would be used for testing purposes. Despite the overfitting, our work shows the potential of using the RL for flight control of a swarm of quadrotors.



(a) Relative distance between UAVs.

(b) Order of the swarm.

Figure 5.6: Results of a swarm of 3 quadrotors from [2]. The $R^0 = 2.5m$ in their work is a design parameter. If an obstacle (*e.g.*, tree, another UAV) is closer to the quadrotor than R^0, then it is considered an immediate threat and the UAV reacts to it according to the collision avoidance methods proposed in the paper. The $R^f = 4m$ is also a design parameter for the uninformed UAVs. Only UAVs farther than R^f are considered as potential targets by uninformed UAVs. The design parameters R^0, R^f are not used in our work.



(a) Relative distance between UAVs.

(b) Order of the swarm.

Figure 5.7: Results of a swarm of 3 quadrotors from the proposed method. For us, the important design parameter is the *separation zone*. Two quadrotors get a negative reward if the distance between them is smaller than this parameter. Another important value is the *death zone*. The situation, when two agents are this close to each other, means immediate termination of the agents, which results in the termination of the whole swarm.

Chapter 6

Conclusion

This thesis focused on utilizing reinforcement learning techniques for controlling the flight of a swarm of quadrotors. To achieve this goal, we developed a simulator of quadrotor physics and integrated it with an interface for reinforcement learning libraries. This simulator was used together with the reinforcement learning method proximal policy optimization provided by Stable Baselines 3 to train a policy that would control the flight of individual quadrotors in a swarm.

In our experiments, we compared two different approaches based on distinct observations. We compared these approaches on a swarm of four quadrotors in an environment with column obstacles. In one scenario, the observation contained all the important information about the agent's state as well as the exact relative position of the nearest columns from the agent. In the other scenario, the information about the absolute position of the agent was not available in the observation and instead of the relative position of the nearest columns, the agent was provided with the observation from 2D LiDAR. The success rate of these approaches was evaluated on four different tracks, showing, that policies provided with known robots' states learned slightly faster, however after the learning, both policies showed a high success rate.

The reinforcement learning approach to swarm control has been compared with another method of swarming, PACNav [2]. This comparison showed that the quadrotors in the proposed method tend to fly closer to each other and keep the swarm tightly coherent. Additionally, the proposed method showed higher agility and speed compared to the method presented in [2].

In this work, we also acknowledged some of the limitations of our approach, particularly the instability of our learning algorithm and overfitting of the reinforcement learning to one track. Despite these limitations, we have shown that reinforcement learning might be a promising approach to swarm control in the future and this work can serve as a basis for future research in the field of drone swarms controlled by reinforcement learning.

Chapter 7

References

[1] K. Kondo, R. Figueroa, J. Rached, J. Tordesillas, P. C. Lusk, and J. P. How, "Robust mader: Decentralized multiagent trajectory planner robust to communication delay in dynamic environments," *arXiv preprint arXiv:2303.06222*, 2023.

[2] A. Ahmad, D. Bonilla Licea, G. Silano, T. Baca, and M. Saska, *PACNav: A collective navigation approach for UAV swarms deprived of communication and external localization*, IOP Science, Oct. 2022. DOI: 10.1088/1748-3190/ac98e6.

[3] 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, 2022.

[4] K. Spanaki, E. Karafili, U. Sivarajah, S. Despoudi, and Z. Irani, "Artificial intelligence and food security: Swarm intelligence of agritech drones for smart agrifood operations," *Production Planning & Control*, vol. 33, no. 16, pp. 1498–1516, 2022.

[5] X. Zhou, X. Wen, Z. Wang, *et al.*, "Swarm of micro flying robots in the wild," *Science Robotics*, vol. 7, no. 66, eabm5954, 2022.

[6] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, 2021.

[7] X. Zhou, J. Zhu, H. Zhou, C. Xu, and F. Gao, "Ego-swarm: A fully autonomous and decentralized quadrotor swarm system in cluttered environments," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 4101–4107.

[8] M. Schranz, M. Umlauft, M. Sende, and W. Elmenreich, "Swarm robotic behaviors and current applications," *Frontiers in Robotics and AI*, vol. 7, p. 36, 2020.

[9] E. Soria, F. Schiano, and D. Floreano, "Swarmlab: A matlab drone swarm simulator," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 8005–8011.

[10] D. C. Vogelsang and B. J. Erickson, "Magician's corner: 6. tensorflow and tensorboard," *Radiology: Artificial Intelligence*, vol. 2, no. 3, 2020.

[11] M. Mozaffari, W. Saad, M. Bennis, Y.-H. Nam, and M. Debbah, "A tutorial on uavs for wireless networks: Applications, challenges, and open problems," *IEEE communications surveys & tutorials*, vol. 21, no. 3, pp. 2334–2360, 2019.

[12] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[13] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[14] B. D. Song, K. Park, and J. Kim, "Persistent uav delivery logistics: Milp formulation and efficient heuristic," *Computers & Industrial Engineering*, vol. 120, pp. 418–428, 2018.

[15] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2017.

[16] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017. DOI: 10.1109/LRA.2017.2720851.

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

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

[19] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart, "Signed distance fields: A natural representation for both mapping and planning," in *RSS 2016 Workshop: Geometry and Beyond-Representations, Physics, and Scene Understanding for Robotics*, University of Michigan, 2016.

[20] G. Bevacqua, J. Cacace, A. Finzi, and V. Lippiello, "Mixed-initiative planning and execution for multiple drones in search and rescue missions," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 25, 2015, pp. 315–323.

[21] J. S. Dibangoye, A.-I. Mouaddib, and B. Chai-draa, "Point-based incremental pruning heuristic for solving finite-horizon dec-pomdps," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 2009, pp. 569–576.

[22] F. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *Circuits and Systems Magazine, IEEE*, vol. 9, pp. 32 –50, Jan. 2009. DOI: 10.1109/MCAS.2009.933854.

[23] E. Semsch, M. Jakob, D. Pavlicek, and M. Pechoucek, "Autonomous uav surveillance in complex urban environments," in *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, IEEE, vol. 2, 2009, pp. 82–85.

[24] J. Everaerts *et al.*, "The use of unmanned aerial vehicles (uavs) for remote sensing and mapping," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 37, no. 2008, pp. 1187–1192, 2008.

[25] R. Beard, D. Kingston, M. Quigley, *et al.*, "Autonomous vehicle technologies for small fixed-wing uavs," *Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 1, pp. 92–108, 2005.

[26] J. C. Butcher, "A history of runge-kutta methods," *Applied numerical mathematics*, vol. 20, no. 3, pp. 247–260, 1996.

[27] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[28] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987, pp. 25–34.

Chapter A

Appendix

There are rendered videos and the source code of the project in the appended DVD. The DVD has the following structure:

```
/
├── videos
│   ├── track_1.mp4
│   ├── track_2.mp4
│   ├── track_3.mp4
│   ├── track_4.mp4
│   └── pacnav_comparison_track_example.mp4
└── flightsim
    ├── examples
    │   └── multi_quadrotor
    │       ├── config
    │       ├── include
    │       ├── learning
    │       │   ├── train.py
    │       │   └── test_policy.py
    │       ├── scripts
    │       └── src
    ├── include
    ├── lib
    ├── src
    ├── README.md
    └── ...
```

In the `videos` section there are renders of some of our experiments in the *random columns* environment (Fig. 4.1a) as well as a render of a chosen track from the comparison with PACNav [2], that was described in the Chapter 5 [1].

The other folder `flightsim` is the source code of our simulator. It is a copy of the content on our Git repository. The `flightsim/include` and `flightsim/src` directories contain necessary base classes for the `flightsim/examples`. The `flightsim/examples/multi_quadrotor` is the directory with the implementation of the learning environment for the swarm of quadrotors. In this directory, the `config` folder contains the meshes of the quadrotor and of the environments and configuration files for the simulator. The `learning/train.py` is the main script for the learning and it uses other Python scripts from the `scrits` directory. To find out more, check out `README.md`, which contains all the necessary information to set up the environment and start the learning.

[1] These renders are also available on YouTube: `https://youtu.be/Lr4l4pmz-zk` and `https://youtu.be/zRP6rhlPVjo`.