TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Elektrotechnik und Informationstechnik
Bioinspired Communication Systems

Multi-Agent Collision Avoidance in Quadcopter Swarms via Deep Reinforcement Learning

Master-Thesis
Elektro- und Informationstechnik

Eingereicht von

Ramzi Ourari

am
03.11.2020

1. Gutachten: Prof. Dr. techn. Heinz Koeppl
2. Gutachten: M.Sc Kai Cui

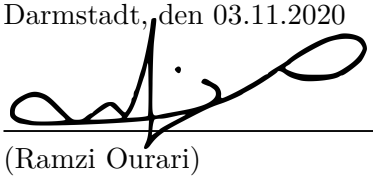
Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Ramzi Ourari, die vorliegende Arbeit gemäß §22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden. Bei der abgegebenen Arbeit stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

English translation for information purposes only:

Thesis statement pursuant to §22 paragraph 7 and §23 paragraph 7 of APB TU Darmstadt:
I herewith formally declare that I, Ramzi Ourari, have written the submitted thesis independently pursuant to §22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form. I am aware, that in case of an attempt at deception based on plagiarism (Âğ38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once. In the submitted thesis the written copies and the electronic version for archiving are pursuant to Âğ 23 paragraph 7 of APB identical in content.

Darmstadt, den 03.11.2020



(Ramzi Ourari)

Abstract

We humans practice collision avoidance on a daily basis when we go about our day. We might not always pay attention but, in many situations we quickly navigate, sometimes negotiate, our way through a crowd. We are able to *jointly* with other strangers, settle collision-free paths. This skill is important for autonomous agents to learn if they are to operate in real-world. This work seeks to learn decentralized decision making strategies that enable collision-free navigation in a fleet of self-interested agents. To that end, we investigate reinforcement learning (RL) techniques to recover policies that enable agents to safely navigate to their targets. To deal with arbitrary number of agents, we develop a new scalable observation model following a *topological* interaction rule. Our learned policies are tested in a simulation environment and subsequently transferred to real-world drones. Videos documenting the learning process and the hardware implementation can be found at `https://sites.google.com/view/dronecolony/startseite`

Contents

1 Introduction

Humans are tools creators. We created weapons to hunt, money to trade, internet to communicate. We even created spaceships to explore beyond our planet earth. We invent to make our life easier and overcome environmental challenges. Why then not create fleets of autonomous UAVs[1])? We will be able to use them for search and rescue tasks, or maybe to deliver an ad-hoc communication network to places damaged by natural catastrophes, or deliver vaccines in times (or places) humans can not, or maybe with less efficiency.

Ubiquitous applications of autonomous UAVs rely on safe operations which is not an easy thing to guarantee outside a laboratory. Real-world is noisy, unpredictable and notoriously hard to simulate. To navigate such environments, one needs decision-making capabilities under uncertainty. This skill is necessary for autonomous agents to have. Simply because *situations* are different, maybe infinite, but they can be categorized in smaller sets. So, instead of trying to cover all situations, we aim for agents that can connect situations with categories and accordingly, decide on a course of actions.

We humans practice collision avoidance on a daily basis when we go about our day. We might not always pay attention but, in many situations we quickly navigate, sometimes negotiate, our way through a crowd. We are able to *jointly* with other strangers, settle collision-free paths. This skill is important for autonomous agents to learn if they are to operate in real-world. The thesis at hand aims to build on the success of Reinforcement Learning (RL) [57], a sub-field of Machine Learning (ML) [51], to solve the task of collision avoidance in a fleet of quadcopters. We investigate and apply (RL) techniques to *learn* decentralized decision making strategies (also known as *policies*) that enable collision-avoidance in a swarm of self-interested autonomous quadcopters (in the following referred to as agents). We opted for a decentralized approach because of its *robustness* in comparison to a centralized system approach.

We test our approach in a simulation environment, show how different training methodologies affect the behavior of agents and provide a successful transfer of a control policy to a real-world application.

[1]UAV= unmanned aerial vehicle.

Contribution the main contributions of this work can be summarized as follow:

1. A simulation environment enabling the evaluation of single/multi-agent settings.

2. A new, biologically inspired, approach to solve collision avoidance in a non-cooperative setting via RL.

3. Transfer of learned policies to real quadcopters.

We begin with a background section chapter 2 where we provide the necessary information and formalism related to RL and Multi-Agent RL (MARL). We then proceed by describing the collision avoidance problem and present our approach to solve it chapter 3. Simulation and Hardware implementation are presented in chapter 4 and chapter 5 respectively. We conclude with a summary and an outlook chapter 6.

2 Foundations

> Nature has placed mankind under the governance of two sovereign masters, pain, and pleasure. It is for them alone to point out what we ought to do, as well as to determine what we shall do
>
> *Jeremy Bentham*
> *An Introduction to the Principles of Morals and Legislation*

2.1 Reinforcement Learning

The idea of RL can be intuitively seen as the attempt to build a hedonistic [1] [2] learning system [57]. A system that has the ability to adjust its behavior based on external signals equivalent to *pain* and *pleasure* so as to maximize exposure to *pleasing* experiences and minimize *painful* ones.
Formally, RL is a learning paradigm concerned with sequential decision making. In this setting, a controller/agent, acting in a given environment, learns an *optimal* behavior so as to maximize a numerical performance measure expressing a long-term objective [59]. In practice, the learning is *incremental*. Meaning that the agent learns via trial and error based on a feedback signal from the environment in a closed loop fashion.

The agent-environment interaction can be decomposed into three signals:

- A reward signal defining the goal/objective.
- A control signal reflecting the behavior of the agent.
- An observation signal: the basis on which actions are selected.

It is worth noting here that giving a scalar reward at each time-step t is sufficient to cover a wide range of applications to which RL can be applied. Using a reward signal to formalize a goal is one of the most distinctive features of RL. It builds on the *reward hypothesis*.

[1] Psychological or motivational hedonism claims that only pleasure or pain motivates us [41].
[2] RL can also be framed as a probabilistic inference, see [37].

"That all what we mean by goals and purposes can be well thought of as the maximization of expected value of the cumulative sum of received scalar signals (called reward)." [57]

Markov Decision Processes

Markov decision processes (MDPs) are a tool for modeling sequential decision making problems. A MDP is defined as a triplet $\mathcal{M} = (\mathcal{X}, \mathcal{U}, \mathcal{P}_0)$, where $\mathcal{X}$ is the non-empty set of states, $\mathcal{U}$ is the non-empty set of actions. The *transition probability kernel* $\mathcal{P}_0$ assigns to each state-action pair $(x, u) \in \mathcal{X} \times \mathcal{U}$, a probability measure over $\mathcal{X} \times \mathcal{U}$ denoted by $\mathcal{P}_0(\cdot \mid x, u)$. For $V \subset \mathcal{X} \times \mathcal{U}$, $\mathcal{P}_0(V \mid x, u)$ gives the probability that the next state and associated reward belongs to the set V provided that the current state is x and the action taken is u.
The transition probability kernel $\mathcal{P}_0$ gives rise to the *state transition probability kernel* $\mathcal{P}$, which for any $(x, u, y) \in \mathcal{X} \times \mathcal{U} \times \mathcal{X}$ triplet, gives the probability of moving from state x to some other state y when action u was chosen in state x

$$\mathcal{P}(x, u, y) = \mathcal{P}_0(\{y\} \times \mathbb{R} \mid x, u).$$

$\mathcal{P}_0$ also gives rise to the immediate reward function $r : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$, which gives the expected immediate reward received when action u is chosen in state x: if $(Y_{(x,u)}, R_{(x,u)}) \sim \mathcal{P}_0(\cdot \mid x, u)$ then

$$r(x, u) = \mathbb{E}[R_{(x,u)}]. \tag{2.1}$$

Generally, we assume that the rewards are bounded by some quantity $\mathcal{R} > 0$:
for any $(x, u) \in \mathcal{X} \times \mathcal{U}$, $R_{(x,u)} \leqslant \mathcal{R}$ holds almost surely[3]. An immediate result is that if the random rewards are bounded by $\mathcal{R}$ then $\|r\|_\infty = \sup_{(x,u) \in \mathcal{X} \times \mathcal{U}} r(x, u) \leqslant \mathcal{R}$ also holds. The interaction between the agent/controller and the system (the environment) is described as follow: let $t \in \mathbb{N}$ denote the current time/stage, let $X_t \in \mathcal{X}$ denote the random state of the system and $U_t \in \mathcal{U}$ the agent's action at time step t. Upon selecting an action, the system makes the following transition

$$(X_{t+1}, R_{t+1}) \sim \mathcal{P}_0(\cdot \mid X_t, U_t),$$

with $P(X_{t+1} = y \mid X_t = x, U_t = u) = \mathcal{P}(x, u, y)$ and $\mathbb{E}[R_{t+1} \mid X_t, U_t] = r(X_t, U_t)$. The agent observes the next state X_{t+1}, receives the reward R_{t+1}, chooses a new action $U_{t+1} \in \mathcal{U}$ and the process is repeated. The *behavior* of an agent is the rule it follows to select actions upon observing new states. The *return* at time step t underlying a behavior is defined as the total discounted sum of the rewards received

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.2}$$

where $0 \leqslant \gamma \leqslant 1$ is *the discount factor* indicating the relative importance of future rewards [57]. If $\gamma = 1$ then the MDP is called *undiscounted*.

[3]The statement holds with probability one everywhere on the probability space with the exception of a set of events with measure zero.

The Policy

Our goal is to find a rule for selecting actions that result in the highest expected total discounted reward [7]. This rule is commonly referred to as the *policy*. The policy defines how an agent selects its actions. Policies can be categorized under the criterion of being either *deterministic* or *stochastic*:

- In the *deterministic* case, the policy maps states to actions

$$U_t = \pi(X_t).$$

- In the *stochastic* case, the policy maps states to distributions over the action space $\mathcal{U}$

$$U_t \sim \pi(\cdot \mid X_t),\ t \in \mathbb{N},$$

where $\pi(x, u)$ denotes the probability of selecting action u in state x.

Value Functions

A straightforward approach to solve a MDP would be to list all policies and then identify the ones that result in the highest value for each initial state. This is clearly not a viable solution because of the big search space when dealing with high action/state spaces [4] and even intractable when dealing with continuous spaces.

A more fruitful venue is to define an *optimal* value function which then allows us to determine the optimal policy (the one resulting in the highest expected return) with relative easiness [57]. For a given MDP, the *state-value function* $V^\pi : \mathcal{X} \to \mathbb{R}$ underlying some fixed policy π

$$V^\pi(x) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+k+1} \mid X_0 = x\right],\ x \in \mathcal{X}, \tag{2.3}$$

is the expected return when an agent starts from a state x, and always follows policy π. [5] It is also useful to define an *action-value function*, $Q^\pi : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ underlying some fixed policy π

$$Q^\pi(x, u) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^k R_{t+k+1} \mid X_0 = x, U_0 = u\right],\ (x, u) \in \mathcal{X} \times \mathcal{U}, \tag{2.4}$$

which describes the average discounted cumulative reward when the agent starts in state x, takes an arbitrary action u, and then acts according to policy π.

[4] For a given MDP, the number of deterministic policies is $|\mathcal{U}|^{|\mathcal{X}|}$.

[5] Note that in order for the conditional expectation in Equation 2.3 to be well-defined for all states, $P(X_0 = x) > 0$ must hold for all states x.

The advantage function $A^\pi(x,u)$ defined as

$$A^\pi(x,u) = Q^\pi(x,u) - V^\pi(x),\ (x,u) \in \mathcal{X} \times \mathcal{U}, \tag{2.5}$$

measures the increase/decrease in expected return due to the agent having chosen action u in state x. A possible way for estimating V^π, Q^π, and A^π is via *Monte-Carlo methods* [59]. However, for computational efficiency and when dealing with applications involving high state/action spaces we commonly use parameterized function approximators (e.g. a neural network [27]) and learn a set of parameters θ that approximate the true value function (e.g. the weights and biases of a neural network). For example, we write $V^\pi_\theta(x)$ or $V^\pi(x;\theta)$ to indicate that the state-value function is approximated by a function with parameters θ.

Partial Observability

In practice, many situations involve settings where an agent indirectly or *partially* observes its environment, for example:

- A camera-only navigation robot
- A single swarm member

In these situations, we refer to the problem as a *partially observable Markov decision process* (POMDP).
In this case, the agent must *construct* its own state representation. For example by:

- Remembering the whole history $h_t = (X_t, U_t, \cdots, X_0, U_0)$.
- *Constructing* a sufficient statistic of the history e.g. a belief state over the next states [34].
- *Learning* a sufficient statistic of the history (for example using a recurrent network [27]).

2.2 Value Based Methods

Algorithms that rely on an approximate of the value function are called *value-based* (next we will discuss another family of algorithms that directly represent the policy).

Q-learning and DQN

To learn the optimal Q-function, Q-learning [64] makes use of the optimality equation for the *state-action-value* [59]. The optimal policy can then be recovered with

$$\pi^*(x) = \arg\max_{u \in \mathcal{U}} Q^*(x,u). \tag{2.6}$$

In Deep Q-Learning (DQN) [40], the Q-function is represented by a deep neural network $Q(x, u; \theta_k)$ where θ_k denotes the weights of the network at the $k^{\text{-th}}$ iteration. The weights are updated via stochastic gradient-descent [12] by minimizing the squared loss

$$L(\theta_k) = (Q(x, u; \theta_k) - Y_k^Q)^2 \tag{2.7}$$

with

$$Y_k = r + \gamma \max_{u'} Q(x', u; \theta_k^-), \tag{2.8}$$

where θ_k^- denotes the parameters of a target network which is kept constant for N iterations and x' is the next state. The weights update rule is

$$\theta_{k+1} = \theta_k - \alpha(Q(x, u; \theta_k) - Y_k^Q)\nabla_{\theta_k} Q(x, u; \theta_k), \tag{2.9}$$

where α is the *learning rate*. Actions are chosen by an action selector, typically implementing an ϵ*-greedy strategy* ($0 \leqslant \epsilon \leqslant 1$)

$$\pi(x) = \begin{cases} \arg\max_{u \in \mathcal{U}} Q^\pi(x, u; \theta_k) & \text{with probability } 1 - \epsilon \\ \mathcal{F}(\mathcal{U}) & \text{with probability } \epsilon. \end{cases}$$

$\mathcal{F}(\mathcal{U})$ denotes a sample from the uniform distribution over $\mathcal{U}$. In order to stabilize learning and improve sample efficiency, DQN also employs *experience replay*: a data set D is fed with tuples of experiences $(X_t, U_t, R_{t+1}, X_{t+1})$ and training occurs by sampling mini-batches of experiences from D. This also prevents the neural network from over-fitting to recent experiences. The possibility of employing an experience replay (sometimes called *experience buffer*) is one of the advantages of *off-policy* methods because it reduces their sample complexity.

2.3 Policy Gradient Methods

Let π_θ be some stochastic policy such that $\nabla_\theta \pi_\theta$ exists. *policy gradient* methods learn a set of parameters θ based on the gradient of some *performance measure* $J(\theta)$. They maximize performance by applying gradient *ascent* in J. The update rule is

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \widehat{J(\theta_t)}, \tag{2.10}$$

where again α is the *learning rate* and $\widehat{\Delta_\theta J(\theta_t)}$ is a stochastic estimate whose expectation approximates the gradient of the performance measure w.r.t. θ. The performance we usually care to enhance is the expected return [6]. Note that the return depends on the actions *and* on the state distribution, both induced by the policy. The state distribution. however, is usually unknown to the agent. Luckily, the *policy gradient theorem* provides the proof that the gradient of the performance we care to measure is independent of the gradient of the state distribution induced by π_θ [57].

[6] Some methods extend this performance measure with an entropy regularizer term to prevent the policy from becoming deterministic. See [28] for example

Policy Gradient Theorem

Let μ_0 be the distribution over start states, and let $\mu_{\pi_\theta}(x) = \sum_{t=0}^{\infty} \gamma^t P(X_t = x \mid X_0)$ with $X_0 \sim \mu_0$ denote the unnormalized states distribution induced by π_θ. For any MDP

$$\begin{aligned} \nabla_\theta J(\theta) &\propto \sum_{x \in \mathcal{X}} \mu_{\pi_\theta}(x) \sum_{u \in \mathcal{U}} Q^\pi(x,u) \nabla_\theta \pi_\theta(u \mid x) \\ &= \mathbb{E}\left[\sum_{u \in \mathcal{U}} Q^\pi(x,u) \nabla_\theta \pi_\theta(u \mid x) \right]. \end{aligned} \tag{2.11}$$

Proof in [57].

REINFORCE

The simplest policy gradient algorithm is called REINFORCE [66]. By reformulating Equation 2.11 we get

$$\begin{aligned} \nabla J(\theta) &= \mathbb{E}\left[\sum_{u \in \mathcal{U}} Q^\pi(x,u) \nabla_\theta \pi_\theta(u \mid x) \right] \\ &= \mathbb{E}\left[\sum_{u \in \mathcal{U}} \pi_\theta(u \mid x) Q^\pi(x,u) \frac{\nabla_\theta \pi_\theta(u \mid x)}{\pi_\theta(u \mid x)} \right] \\ &= \mathbb{E}\left[Q^\pi(x,u) \frac{\nabla_\theta \pi_\theta(u|x)}{\pi_\theta(u \mid x)} \right] \\ &= \mathbb{E}\left[G_t \frac{\nabla_\theta \pi_\theta(u \mid x)}{\pi_\theta(u \mid x)} \right]. \end{aligned} \tag{2.12}$$

The last equality follows from Equation 2.4 where G_t refers to the *finite-horizon* discounted *return* with horizon T

$$G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1}. \tag{2.13}$$

The update rule is then

$$\theta_{t+1} = \theta_t + \alpha \, \mathbb{E}\left[G_t \right] \frac{\nabla_\theta \pi_\theta(u \mid x)}{\pi_\theta(u \mid x)}. \tag{2.14}$$

Intuitively, the update rule increases/decreases the probability of actions in proportion to their expected return [57].

Because of this episodic return, REINFORCE suffers from high variance (caused by stochastic rewards) and thus slow convergence. One way to reduce variance is by subtracting a *baseline function* $b(x)$ that depends only on the state x. The new gradient then becomes

$$\nabla J(\theta) = \mathbb{E}\left[\sum_{u \in \mathcal{U}} (Q^\pi(x,u) - b(x)) \nabla_\theta \pi_\theta(u \mid x) \right].$$

Note that

$$\begin{aligned}\mathbb{E}\left[\sum_{u\in\mathcal{U}} b(x)\nabla_\theta \pi_\theta(u \mid x)\right] &= \mathbb{E}\left[b(x)\nabla_\theta \sum_{u\in\mathcal{U}} \pi_\theta(u \mid x)\right] \\ &= \mathbb{E}\left[b(x)\nabla_\theta 1\right] \\ &= 0.\end{aligned}$$

The second equality comes from the fact that the probability over all actions must sum to one. Consequently, adding a baseline $b(x)$ does not change the expectation in Equation 2.12 hence does not add *bias*.
A common choice for the baseline function is $V^\pi(x)$ Equation 2.3, which results in the following update rule

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha(G_t - V^\pi(x))\frac{\nabla_\theta \pi_\theta(u \mid x)}{\pi_\theta(u \mid x)} \\ &= \theta_t + \alpha(Q^\pi(x,u) - V^\pi(x))\frac{\nabla_\theta \pi_\theta(u \mid x)}{\pi_\theta(u \mid x)} \\ &= \theta_t + \alpha A^\pi(x,u)\frac{\nabla_\theta \pi_\theta(u \mid x)}{\pi_\theta(u \mid x)}.\end{aligned}$$

A^π has usually lower magnitudes than Q^π [23] which helps reduce the variance while not adding a bias.

Actor-Critic Methods

Actor-Critic methods (AC) are another flavor of policy gradients that, unlike REINFORCE, can be employed online. AC methods rely on *bootstrapping* [57]: the values of an estimate are updated based on previous estimations. This introduces bias but further reduces variance [57]. The main components of AC are :

- The *critic*: which represents an estimate of a value function (e.g. V^π)
- The *actor*: the policy

The *actor* uses gradients derived from the policy gradient theorem and adjusts the policy parameters θ_a, the critic, parameterized by θ_c, estimates the approximate value-function for the current policy ($Q_c^\pi(x,u) \approx Q^\pi(x,u)$).

We mentioned that value based methods use *experience replay* to stabilize training. In the case of AC methods, an approach to perform the policy gradient on-policy without experience replay has been investigated to mitigate the issue of over-fitting to recent *experiences*. With the use of *asynchronous* methods, multiple agents are executed in parallel and the actors are trained asynchronously.

The parallelization of agents ensures that each agent experiences different parts of the environment at any given time step t. In that case, the n-step return in Equation 2.13 can be used without introducing a bias. Parallelization can thus be applied to any learning algorithm that requires on-policy data removing the need for a *replay buffer* but does not increase the samples efficiency of on-policy methods [23].

2.4 Multi-Agent Reinforcement Learning

Until now, we presented RL as a single-agent-environment interaction loop. This model can be extended to take into account the simultaneous learning of $N > 1$ agents within the same environment.[7] Such systems are referred to as *multi-agent-system* (MAS) [21] and in the context of RL they are referred to as *multi-agent reinforcement learning* (MARL).
Borrowing game-theoretic definitions [46], we categorize MARL problems in 3 different categories:

- Cooperative games
- Zero-sum games
- Mixed or general-sum games

These differ in the way the reward (per agent) is computed (more on that in the next section).
Let's first define the MARL setting. Multi-agent systems can be formalized as a *stochastic game* also called *markov game* [39], $\mathcal{G}$. In general $\mathcal{G}$ is defined by a set of states $\mathcal{X}$, and a *collection* of action sets, $\mathcal{U}^1, \cdots, \mathcal{U}^N$, one for each of the N agents in the environment. The *state transition probability kernel* $\mathcal{P}$ is controlled by the current state and the joint action of the agents $\mathbf{U}$.
Each agent has an associated reward function, $r_i : \mathcal{X} \times \mathcal{U}^1 \times \cdots \times \mathcal{U}^N \to \mathbb{R}$, for agent i. Each agent is concerned with maximizing its expected sum of discounted rewards

$$\mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{i,t+k+1}\right].$$

We summarize some of the challenges arising in the multi-agent setting as follow:

- Non-stationarity
- Credit assignment
- Scalability
- Compact state representation

[7] Same, here, refers to the transition probability kernel $\mathcal{P}_0$ which conditions on the Joint action of all agents. The true state x however, is in general observed differently from any agent's perspective.

Non-stationarity arises from the fact that both the transition probability kernel and the reward function depend on the joint action of all agents. It follows that single-agent algorithms e.g. *q-learning*, when applied to multi-agent settings without considering $\mathbf{U}$, lose their convergence guarantees.

Credit assignment in the multi-agent setting refers to the problem of how to relate a single agent's action to the return it receives. When for example in a collision avoidance task, agent a makes a bad move and gets near an agent b but agent b reacts quickly and avoids a how should each one of the agents be rewarded?

Scalability in MARL refers to the issue of exponential growth in the joint-action representation [48]. Unambiguously representing the actions of N agents each with an action space $|U|$ results in a vector of length $|U|^N$.

Compact state representation differs from scalability in that now we are not concerned with the maximum number of agents a system can process but how to compactly represent a variable number of agents. Note that when using neural networks as a function approximator, the network expects a fixed length vector. The size of a swarm, however, can vary between consecutive time-steps .

2.4.1 The Multi-Agent Setting

In the following we describe the 3 categories into which we can categorize MARL problems, relate them to the aforementioned challenges and describe some recent work tackling these issues:

Cooperative Games

Cooperative games [46] denote tasks where, at any time step t, all agents receive the same reward

$$r^i(x, \mathbf{U}) = r^j(x, \mathbf{U}) = r(x, \mathbf{U}), \text{ for all } i, j \in N,$$

where $r(x, \mathbf{U})$ denotes a *team reward*. In such settings, N agents *learn* to cooperate, coordinate their actions and potentially communicate in order to maximize a shared payoff metric [22]. One challenge that arises in such settings is the *credit assignment* problem, where it is not always obvious which agent is responsible for the success/failure of the team. Consider for example the *lazy agent* problem in cooperative settings tackled in [56] with *value decomposition network* under the assumption that the joint action-value function can be additively decomposed into value functions across agents.

Another issue is the above-mentioned *non-stationarity* of the environment from a single agent perspective. A straightforward approach to mitigate this problem is to make the assumption

that all agents are *homogeneous* and instead of having N different policies, a single *central-controller* maps joint observations to joint-actions. This solution scales linearly in the number of agents and also does not translate well to many real world scenarios where agents need to act based on local observations only. In the work of [54], N agents modeled as a single controller learn continuous communication protocols in a cooperative partially observable setting to solve a traffic junction task. In the work of [22] *centralized training decentralized execution* [45] and *difference rewards* [62] were investigated to tackle the issue of *credit assignment* where a centralized *critic* conditions on the joint action of N agents and efficiently (in a single neural network forward pass) computes the advantage function for each agent.

Centralized training decentralized execution is a commonly used technique in MARL where the setup allows for extra information to be shared between agents during training as long as agents do not require that extra information at execution time [45]. In the example of [22], the central-critic is only needed during training, at execution time, only the actors are used and they do not require the joint action as input.

In [48], a *Mixing Network* takes as input the action-state value of N agents and outputs a global state-action value $Q_{tot}(\boldsymbol{\tau} \mid \mathbf{U})$ enforcing the joint action-value function to be monotonic in the *per agent values* $(Q^1, ..Q^N)$ which allows tractable maximization of the joint-action value in off-policy learning and guarantees consistency between centralized and decentralized policies. One limitation of this work is that it only deals with discrete actions and also that it is suitable for tasks where each agent's best action does not depend on the other agent's action *at the same time*.

Zero-Sum Games

In zero-sum games, the sum over all agents' rewards is equal to 0 [46].

$$\sum_{i=1}^{N} r^i(x, \mathbf{U}) = 0.$$

The challenges of *non-stationarity* can be well illustrated in the adversarial setting. Consider the game of *rock-paper-scissors* [65]. Two agents *simultaneously* perform actions $u \in \{u_{paper}, u_{rock}, u_{scissor}\}$ and receive their respective reward $\{-1, 0, 1\}$ corresponding to loss, draw and win respectively. Each agent conditions on the game history to learn a policy π^i that maximizes their reward.
A reasonable strategy would be to exploit the opponent's behavior and come up with an adequate response. For example, if agent i shows preference to u_{paper}, then j would learn to play $u_{siccors}$ more often. j would then accumulate a history with that behavior, to which i reacts by changing its policy accordingly i.e. selects u_{rock} more often and the process continues leading to aforementioned non-stationarity as a consequence of both agents learning.

Mixed Games

A middle ground between cooperative and adversarial games are mixed (or general-sum) games, in which there is no clear or predefined relation between the reward of the N *self-interested* agents. However, during the task, agents may exhibit cooperative/competitive behavior. Recent work investigated general-sum games in the context of MARL [36; 11]. In [36] examples of *social dilemmas* [18] were studied. Specifically, the emergence of *cooperation* and *deflection* as a function of the environment parameters was investigated. Another interesting work was presented in [11] where the emeregence of *adversarial communication* was shown in settings where a self-interested-agent learns manipulative communication strategies allowing him to outperform a cooperative team of agents.

Centralized vs. Decentralized Control

In cooperative settings, we mentioned the possibility of substituting N agents with a single controller that maps joint observations to joint actions. This approach, while resolving the *non-stationarity* issue, suffers from the *curse of dimensionality* [6] in the action space, which decentralized systems escape because each policy conditions on individual local observations and outputs a single agent action vector.

Centralized systems are also not suitable for applications where agents have to selects individual actions based on their local observations. Further, in many situations, the agents are not necessarily fully cooperative but can also be self-interested which makes a central-controller approach impractical. Lastly, centralized systems suffer from the *single point of failure* risk whereas, in a decentralized system, a local failure affects the whole system's performance in a less harmful way.

2.4.2 Compact State Representation

Independent of the nature of the MARL setting (cooperative, adversarial or mixed), some applications require a *compact* way of representing the state space. Consider the task of collision avoidance: each agent would have to either condition its actions on observations about all other present agents which may vary in number or on a smaller subset of neighbors within a given *perception radius*.
The usually employed neural networks for value-functions and policy approximation expect a fixed size input vector. So a way to compactly represent a variable size state/observation space is needed. Related work on this issue suggested a variety of solutions for this issue by resorting to histograms, embeddings [32] and also long-short-term memory networks (LSTMs) [31] for representing a variable-length observation vector [19].

Histograms

Using histograms, it is possible to discretize the space of certain observations (e.g. distance to nearest agent...) into a fixed number of bins. This way, we can map a variable state space dimension into a fixed-size vector. However, this approach suffers from the *curse of dimensionality* which hurts its scalability. Another issue with histograms is the hard bin assignment which results in a non-smooth state representation. Using radial basis functions [14] (RBFs) with a fixed number of basis functions (e.g. a Gaussian kernel) evenly distributed over the observation space, we can achieve a more *fine-grained* space representation in comparison to histograms [32]. But we still face the same scalability problem when dealing with high dimensional spaces.

Mean Embeddings

The authors in [32] also suggested the use of *Mean Embeddings* as state representation for swarm systems. Assuming the observations an agent perceives about other agents is a sample drawn from a distribution that characterizes the current swarm state configuration, it is possible to use an empirical encoding of this distribution i.e. its arithmetic mean. This way, the input dimension to the function approximator (in [32] neural networks were shown to have the best performance) is given by the dimension of the mean vector. Figure 2.1 from [32] diagrams the mean-embedding for state representation.

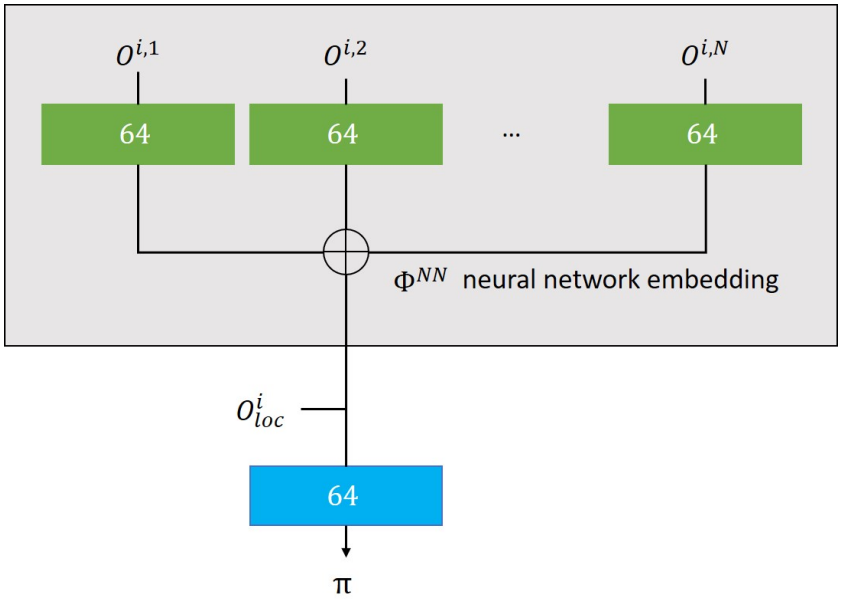


Figure 2.1: Illustration of the neural network mean embedding policy. The numbers inside the boxes denote the dimensionalities of the hidden layers. The color-coding highlights which layers share the same weights. The plus sign denotes the mean of the feature activations [32].

Long-Short-Term Memory (LSTM)

The work in [19] leverages the ability of LSTMS to process sequential data. Their approach first feeds observations of neighboring agents to the LSTM where a *heuristic* of feeding the nearest agent the last was employed. The last output of the hidden cell of the LSTM is then concatenated with the agent's own observation and fed to a 2-layer neural network which outputs discrete control actions (the *actor*) and a scalar state value (the *critic*) see Figure 2.2 from [19]. Note that in practice one must define a maximum sequence length for the LSTM which translates to a maximum number of agents the system can process.

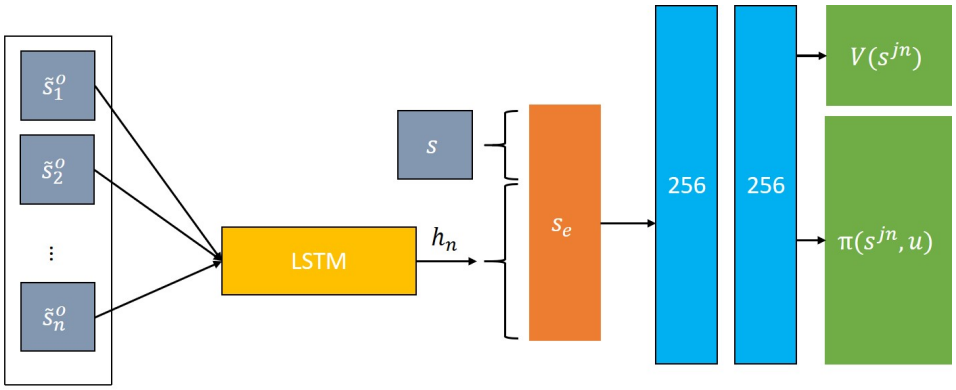


Figure 2.2: Illustration of the state representation with LSTMs. Observable states of nearby agents $\tilde{s}_i^o$, are fed sequentially into the LSTM. The final hidden state h_n is concatenated with the agent's own state s to form the vector s_e. The encoded state is fed into two fully connected layers (FC). The outputs are a scalar value function (top, right) and a policy represented a discrete probability distribution over actions (bottom, right) [19].

3 Multi-Agent Collision Avoidance

Safely negotiating interactions between dynamical agents is an important requirement if fleets of autonomous vehicles are to be deployed in real world applications. In this thesis, we investigate RL techniques chapter 2 to solve the collision-avoidance task in a fleet of self-interested quadcopters. In this chapter, we formulate our problem, describe related work and present our approach.

3.1 Problem Formulation

Our goal is to find a policy π_θ such that an agent, starting at a given position, reaches its target as quickly as possible while avoiding collision with other agents.
We can formulate the task as an optimization problem as follow

$$\begin{aligned} \arg\min_{\pi_\theta^i} \quad & \mathbb{E}\left[t_g^i \mid O^i\right] \\ \text{s.t.} \quad & \|\mathbf{p}_t^i - \mathbf{p}_t^j\|_2 \geq \epsilon_A \text{ for all } i \neq j, \text{ for all } t \\ & \mathbf{p}_{t_g}^i = \mathbf{p}_g^i \text{ for all } i \\ & \mathbf{p}_t^i = \mathbf{p}_{t-1}^i + \Delta t \pi_\theta^i(O_{t-1}^i) \text{ for all } i, \end{aligned} \tag{3.1}$$

where t_g^i denotes the time to goal for agent i, $\mathbf{p}_g^i$ is agent i's target position, and ϵ_A is the *minimal inter-agent distance* below which agents are considered to have collided. The first constraint ensures collision avoidance, but also introduces a coupling between the trajectories of individual agents rendering the problem *non-trivial* [29]. The second equation is the goal constraint which ensures the agent gets to its. The last constraint is the agent's kinematic-starget[1].

3.2 Related Work

Previous work in decentralized collision avoidance can be categorized into fully-cooperative and general-sum methods.

[1] Satisfying one constraint (for example the collision avoidance constraint) often leads to the violation of another (e.g. the goal constraint). To ensure the feasibility of the trajectories, the deviation of the final agent's state from its target is often penalized.

General-Sum Methods

General-sum methods solve the collision avoidance problem in two steps: first, other agents' motions (or paths) is predicted and, based on that prediction, a collision-free trajectory is generated. This approach, while suitable for situations where agents are self-interested, may result in the *freezing-agent* problem [61]. Briefly, the *freezing-agent* problem means that once the environment surpasses a certain complexity threshold, the planer (the path generator) decides that all paths are unsafe and the robot freezes. This problem was mitigated in [61] by making the robot engage in *joint collision avoidance*: different agents *cooperate* to make room for each other creating feasible paths.

While the first assumption was non-cooperativeness between the agents, the solution has a cooperative side. This *mix* between cooperation and self-interest was recently exploited in the work of [43] where, framing the collision-avoidance problem as a *sequential-social-dilemma*, led to the design of a reward function that encourages the robot to (i) passively avoid other pedestrians but also to (ii) actively address other agents (e.g. pedestrians) to *make room*.

Cooperative-Methods

Assuming full-cooperation is feasible (see discussion in section 2.4.1), cooperative methods solve the *freezing-agent* problem by modeling interactions between agents and taking into consideration the *mutual* influence of agents' actions. These methods can be further divided into *reaction-based* and *trajectory-based* methods.

Reaction-based approaches rely on one-step interaction rules based on geometry or physics [63; 50]. For example, in reciprocal velocity obstacles (RVO) [63] each agent adjusts its own velocity vector to ensure collision avoidance while staying as close as possible to its preferred velocity vector (e.g. the vector pointing from the agent to its target).
In force-based collision avoidance, agents are modeled as point charges thus creating repulsive forces when near each other. The resulting repulsive force vector is added to the vector pointing from the agent to its target ensuring collision avoidance between agents. Reaction-based methods, while computationally efficient, are *myopic in time* thus do not *anticipate* other agents movements which may result in non-efficient paths (i.e. longer times to target).

Trajectory-based approaches [29] compute paths on longer time scales which requires either knowledge of hidden states (e.g. other agents' goals) or computationally expensive models [19] but they deliver better trajectories (i.e. shorter time to target) [19]

An orthogonal third family of methods to cooperative and non-cooperative methods are *learning-based* approaches.

Learning-based approaches [19; 43] aim to learn *interaction-rules* encoded in a policy. Learning-based approaches are computationally efficient because learning can be done offline (e.g. in a simulator) and at execution time, actions are efficiently queried from the extracted policy.

3.3 Proximal Policy Optimization (PPO)

Recent work on collision avoidance assumed discrete action space [43; 19]. Although good as a starting point and proof of concepts, continuous action spaces offer more freedom to the agent and remove the need to decide on the discretization of the actions space. Policy gradient algorithms (PG) section 2.3 are suitable for continuous control tasks because they can deal with continuous action spaces. We planned a *benchmarking* of some of the state of the art algorithms (PPO, SAC, TD3..) but due to time constraints, we only show results for the proximal policy optimization (PPO) and leave the benchmarking for future work.

One issue with PG is the risk of performance collapse if the step size in Equation 2.14 is set high. On the other hand, low α values result in slow convergence. Another issue is that computing multiple *stochastic gradient descent* epochs on the same batch can result in the divergence of the policy parameters, hurting PG's sample efficiency. To mitigate these challenges *Trust Regions Policy Optimization* TRPO was introduced [52]. TRPO penalizes the difference in the distribution of consecutive policies measured with kullback-leibler divergence. The objective to be maximized in TRPO is subject to a fixed constraint on the Kullback-Leibler (KL) divergence of the policy before and after the parameter update, ensuring the policy parameters θ are bounded.

$$\begin{aligned} \max_{\theta} \quad & \mathbb{E}_t \left[\frac{\pi_\theta(u_t,|x_t)}{\pi_{\theta_{old}}(u_t|x_t)} \hat{A}_t \right] \\ \text{s.t.} \quad & \mathbb{E}_t \left[KL(\pi_{\theta_{old}}(\cdot \mid x_t) \parallel \pi_\theta(\cdot \mid x_t)) \right] \leqslant \delta, \end{aligned} \tag{3.2}$$

where δ is a hyperparameter. The theory justifying TRPO, suggests using a penalty in the objective function resulting in the following unconstrained optimization problem.

$$\max_{\theta} \quad \mathbb{E}_t \left[\frac{\pi_\theta(u_t,|x_t)}{\pi_{\theta_{old}}(u_t|x_t)} \hat{A}_t - \beta KL(\pi_{\theta_{old}}(\cdot \mid x_t) \parallel \pi_\theta(\cdot \mid x_t)) \right] , \tag{3.3}$$

for some coefficient β. One issue with this optimization problem is that it is hard to choose a single β value that transfers well across different problems.
PPO aims to mitigate this issue with a *first-order* algorithm that *emulates* the monotonic improvements of TRPO by modifying the objective in Equation 3.3. Let $r_t^{RATIO}(\theta)$ denote the probability ratio

$$r_t^{RATIO}(\theta) = \frac{\pi_\theta(u_t \mid x_t)}{\pi_{\theta_{old}}(u_t \mid x_t)},$$

PPO's *clipped* loss is given by

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min(r^{RATIO}(\theta)\hat{A}_t, \text{clip}(r_t^{RATIO}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right], \tag{3.4}$$

where ϵ is a hyperparameter controlling the degree of change between the policies' parameters and $\hat{A}_t$ is a truncated version [52] of *generalized advantage estimation* [**?**] for a T-steps *return*

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-1} V(x_T). \tag{3.5}$$

The PPO algorithm is shown below

Algorithm 1 Algorithm 1 PPO, Actor-Critic

```
for iteration=1,2,··· do
  for actor=1,2,···, N do
    Run policy π_{θ_old} in environment for T timesteps
    Compute advantage estimates Â_1, ··· , Â_T
  end for
  Optimize L^{CLIP}(θ), with K epochs and minibatch size M ⩽ NT
  θ_old ← θ
end for
```

Next, we describe our approach to solve the decentralized collision avoidance task.

3.4 Our Approach

Our main goal is to learn decentralized decision-making strategies (policies) that can achieve collision-avoidance in a fleet of self-interested quadcopters. The solution should ideally be scalable and computationally efficient to run on edge-dvices (e.g. nano-quadcopters[25]). We use deep reinforcement learning in a multi-agent setting where each agent considers other agents as part of the environment. To make our method scalable, we pre-process *peers observations* and, at each time-step t, for each member of the swarm, we only consider its m-nearest neighbors. This way, the observation space of the policy has a fixed dimension independent of the swarm size.

3.4.1 Biological Inspiration

Flocks of birds, fishes, and many other animals provide examples of elegant [2] and robust collective behaviors capable of adaptation in challenging environments [5; 9; 47; 17; 16; 55]. Previous work investigated the mechanisms behind swarm behavior and it is believed that it emerges from simple local interactions between elements of the swarm (see for example [50]). One important characteristic of flocks is the ability of its members to maintain *cohesion* despite being subjected to uncertain information about the behavior of their neighbors. Thanks to the analysis of position [5] and velocity [9] correlations in data collected from large flocks of starlings (*Sturnus vulgaris*) [4] it has been shown that each bird responds to a *fixed number*, seven or six, of its nearest neighbors.

As to why exactly this number(although less relevant for the thesis at hand), the authors in [68] show that when each member of the swarm pays attention to its 6 or 7 nearest neighbors,

[2] This is a subjective opinion, nevertheless, we make the assumption it is a common one.

it minimizes the balance between group cohesion and individual effort (expressed in costs of sensing and attention). For our approach, the main insight is that some biological flocks rely on a *topological* interaction rule (interacting with a fixed set of m nearest neighbors) rather than on a *metric* interaction rule (interacting with neighbors within a fixed radius). The observation model is illustrated in Figure 3.1

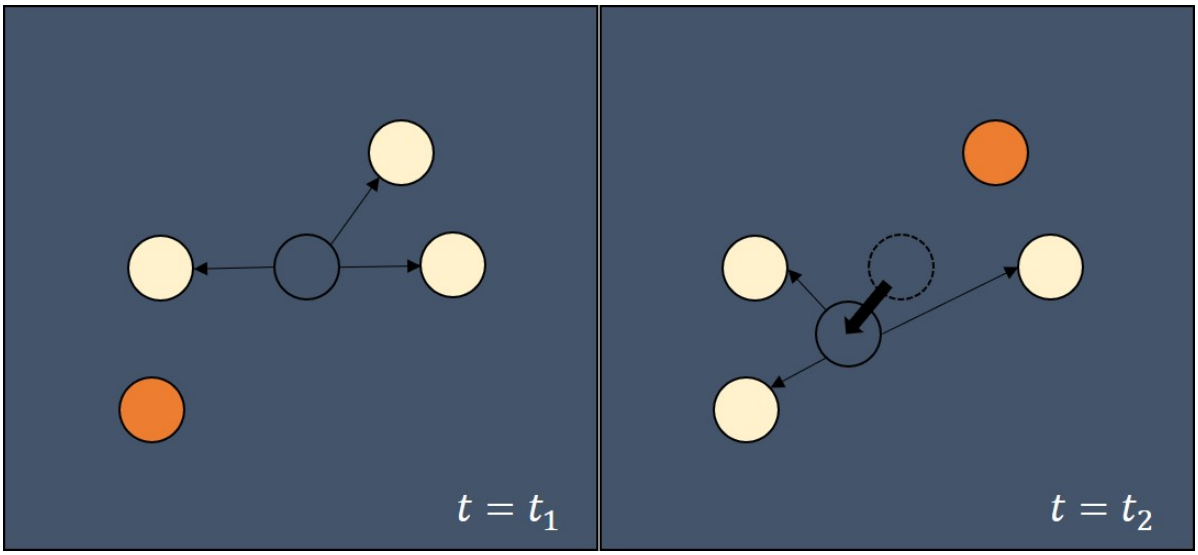


Figure 3.1: Illustration of the m-nearest neighbors ($m = 3$) observation model. The agent (the circle in the middle) conditions its policy on its 3 closest neighbors (white circles) ignoring the orange agent at $t = t_1$. Upon choosing an action and receiving a new observation the agent's 3-nearest neighbors are updated.

3.4.2 Implementation Details

We make the following assumptions: the time is discrete, the agents are modeled as single-point-masses, and able to *instantly* change their velocity. We assume fast update rates so the agents can react quickly and also that the change in the data distribution (due to the learning of other agents) is slow so that agents can adapt to it. Each agent regards other agents as part of the environment and thus agents are *independent*. The independence assumption can be seen as a special case of *bounded rationality*: the agents do not recursively reason about one another's learning [36; 26].

Agent's Model

We assume an agent can control its linear and angular velocity. The agent's goal is to get from point A (start) to point B (goal) as quickly as possible while avoiding other dynamical agents. We assume a general-sum setting section 2.4.1. We also assume an agent i can perceive the following quantities: its position $\mathbf{p}^i = [\mathrm{p}_x^i, p_y^i]$ its velocity $\mathbf{v}^i = [\mathrm{v}_x^i, v_y^i]$ its bearing to its

target b_T^i and its distance from the target d_T^i. In addition to these *ego-centric* observations, we assume each agent can access a set of *peers-observations*:
for any other agent j in the environment, agent i has access to following quantities with respect to it: its distance $\mathrm{d}^{i,j}$, its bearing to it $\mathrm{b}_A^{i,j}$ and its relative velocity $\mathbf{v}^{ij}$

$$\mathbf{v}^{i,j} = [v_x^i - v_x^j, v_y^i - v_y^j],$$

and

$$b_A^{i,j} = \arctan(\frac{p_y^j - p_y^i}{p_x^j - p_x^i}) - \phi^i,$$

where ϕ^i is agent i's orientation. Agent i's action vector $\mathbf{u}^i$ is given by

$$\mathbf{u}^i \sim \pi_\theta^i(\cdot \mid O^i),$$

where $\mathbf{u}^i = [v_x^i, v_y^i, \omega^i]$, ω^i is the agent's angular velocity measured in radians per second and O^i denotes the concatenation of *egocentric* and *peers* observations on which i conditions a stochastic policy modeled as multivariate Normal distribution with mean μ_θ and a diagonal covariance $\mathbf{\Sigma}$. We approximate the policy (actor) and the value-function (critic) with a 2-layer feedforward neural network (64 hidden units for each layer with a tanh activation) each.
The policy's outputs are clipped between $[-1, 1]$ and then multiplied by the maximum linear/angular velocity, resulting in the following updates

$$\begin{aligned} v_{x,t+1}^i &= v_{x,t}^i + \mathrm{clip}(v_x^i, -1, 1) \cdot v_{max} \\ v_{y,t+1}^i &= v_{y,t}^i + \mathrm{clip}(v_y^i, -1, 1) \cdot v_{max} \\ \omega_{t+1}^i &= \omega_t^i + \mathrm{clip}(\omega^i, -1, 1) \cdot \omega_{max}, \end{aligned}$$

where v_{max} and ω_{max} denote the maximum linear and angular velocity of the agent respectively.

We also assume *homogeneity* between the agents. This assumption allows for *parameter sharing* and thus accelerates the learning process (instead of training N different neural networks, we train the same network weights with different observations collected from the learning agents). This assumption is sometimes called *reciprocity* in the literature [15]. The multi-agent problem is solved with proximal policy optimization (PPO).

Reward Mechanism

The reward function we employed decoupled the task into two sub-tasks: (i) get to the target as fast as possible and (ii) keep a minimum distance from other agents. The first sub-task translates in the following reward function per agent

$$f_t^i = \alpha \langle \mathbf{v}^i, \mathbf{u}^i \rangle \text{ for all } i \in N, \tag{3.6}$$

where $\alpha > 0$ controls how much we reward/penalize the agent for moving toward/away the/from target. $\mathbf{v}^i$ is the agent's velocity vector, $\mathbf{u}^i$ is the vector pointing from the agent

to its target. This reward function was borrowed from the unity ml-agents examples [2] and resulted in a relatively fast learning. for the second sub-task we employ the following reward function

$$g_t^a = \beta \min(0, d^{ij} - \epsilon_A), \tag{3.7}$$

where $\beta > 0$ controls how much to penalize getting too close to other agents and d^{ij} is the euclidean distance between i and j. We get the following reward function

$$R_t^a = f_t^a + g_t^a. \tag{3.8}$$

α and β were found via trial and error. For a fixed α, a low β will result in agents *pushing* their way to the target. A high β will lead to the agents *freezing*. but the interval of suitable values for β is relatively broad (between 1 and 10 for $\alpha = 0.003$)

Observations

The observations an agent perceives also alter its behavior. In some setups, agents had access to the orientation of other agents which resulted in agents spinning more often (especially when they get near each other) than when they did not had access to that extra information. We suspected it is an example of communication through actions but proving this will require a clear definition of what we mean by communication and how to quantify it. Nevertheless, that behavior resulted in agents escaping the *freezing agent problem*. In the next section, we show the results for the observations we used for the hardware implementation defined in section 3.1.

Curriculum learning (CL) [8] was employed to gradually increment the number of agents from 2 to 25. And also to increase the initial distance to target. We also used CL in the observation space. Specifically, the *peers* observations start with a perception radius above which they (the observations) get a value of 0. We gradually increment the perception radius to cover the whole training area. This technique substantially helped the agents learn faster. We suspect that at the beginning of learning, the *abrupt* change from 0 to another number helps the agents associate peers-observations with the penalty they get for getting too close to them faster.

4 Learning in a Simulation

In this section we present the simulation environment and evaluate our results.

4.1 The Simulation Setup

We built the simulation environment in the Unity Game Engine [33] which, in addition to a NVIDIA's physics engine *Nvidia PhysX* [3], offers a new open-sourced *ml-agents* package [2].

Unity's *ml-agents'* open-source project allows for the creation of sophisticated MARL environments (see open-sourced examples [2], [33]) , and the integration of the created environment with *Gym* [13], a popular RL python library.

With the Unity editor, we can create agents and game objects (e.g obstacles, targets...) and also define the game logic (e.g reward mechanism, teams definition, hierarchical teams...).

We decided to build the simulation environment in the game engine Unity because, besides a high fidelity physics and rendering engine, Unity's open-sourced package *ml-agents* allows for the creation of sophisticated multi-agent environments with flexible social structures. The game logic can be defined in C# whereas the created game object allows for interaction via a python-API in a gym-style fashion [13]. We created two tasks: *AtoB* and *Move And Collect*. In *AtoB* agents start in a given formation (e.g. a circle) and navigate to their respective targets. Whereas in *Move and Collect*, upon reaching the target, agents receive a new randomly resapwned target. Videos from the simulation can be found in the link provided in the abstract.

For the PPO implementation section 3.3, we used RLlib [38], an open-source RL and MARL library. RLlib, besides offering efficient implementations of state of the art RL algorithms (SAC, PPO, TD3...) and a ready-to-use multi-agent python class, also supports scalability with a parameter-server policy-workers architecture. The policy workers periodically receive updated policy weights and send back new trajectories to the parameter-server (the central-learner) on which multiple stochastic gradient descent (SGD) iterations are computed. Figure 4.1 illustrates the server-workers architecture. RLlib also supports cluster training where similar to the server-workers architecture, a *head-node* plays the role of the central-learner interacting with the rest of the cluster nodes. Especially clusters managed with SLURM [67] are natively supported with RLlib.

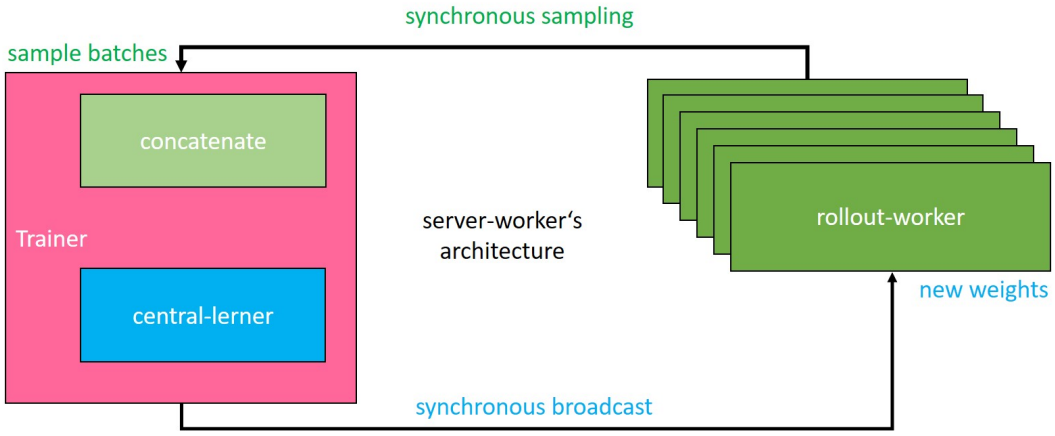


Figure 4.1: Learner-workers architecture: the Trainer class coordinates the distributed workflow of running rollouts and optimizing policies. RLlib uses rollout-workers to scale training from a single core to many thousands of cores in a cluster.

4.2 Results

We investigate the scalability of our approach with the following question: how does the number of agents affect the performance, measured in time to target and minimum distance to other agents, of the policy?

We tried different reward functions to solve this task. One thing that we learned is that the agents can exploit the functions in ways we don't always anticipate. Two remarkable behaviors are worth noting here:
In one of the setups, we ended the episode without a penalty each time the agent hits a boundary (e.g. wall or floor), we gave a reward for getting to the target and a step penalty to reinforce quick execution. During training, agents would fly down and hit the floor each time the reward for getting to the target is less than the sum of the step-penalties.

In another setup, we didn't end the episode when hitting the boundaries but re-spawned the agents and gave them a penalty. Some of the agents learned to push other agents (fly toward them) till they hit the wall and thus clear their way toward the target. They exploited the fact that other agents are trying to avoid them. We assume that because agents condition on different observations and at each time step have different returns, we can expect the emergence of different behaviors even when agents share the same parameters. We can not unambiguously quantify these behaviors because our actions are spatial movements but one can (in line with [35]) asses the *behavior signature* of the agents.

For the case of 1 nearest agent, we recovered the following trajectories

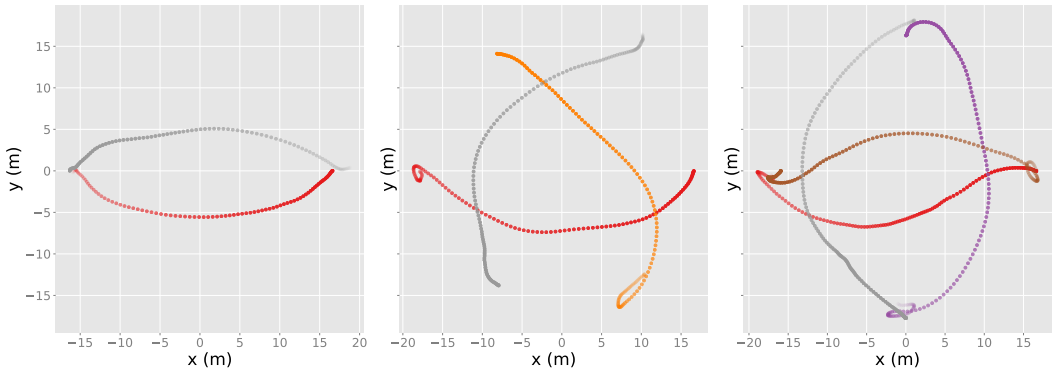


Figure 4.2: 1-nearest neighbor trajectories with $n \in [2, 3, 4]$ agents

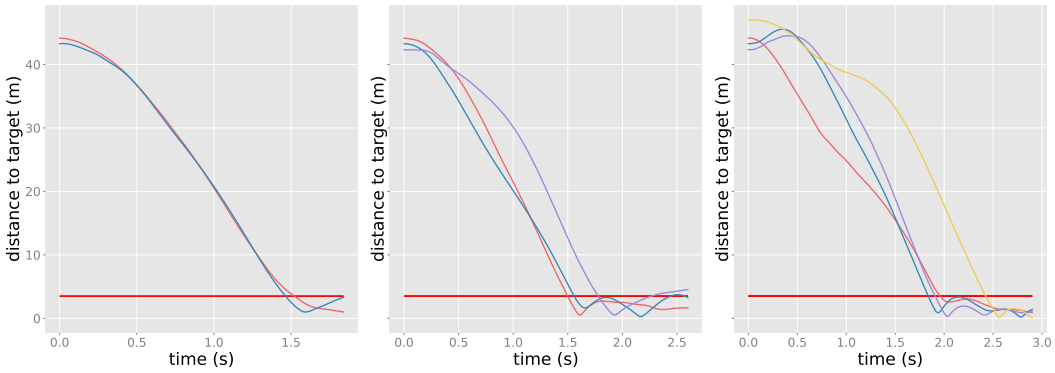


Figure 4.3: 1-nearest neighbor distance to target with $n \in [2, 3, 4]$ agents. The red line denotes the minimal distance to target: 3.5 m.

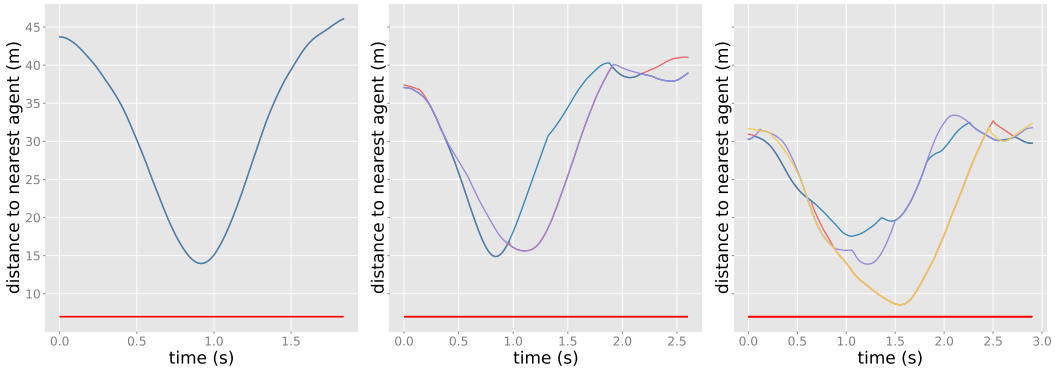


Figure 4.4: 1-nearest neighbor distance to nearest agent with $n \in [2, 3, 4]$ agents. The red line denotes the *minimum inter-agent distance*: 7 m.

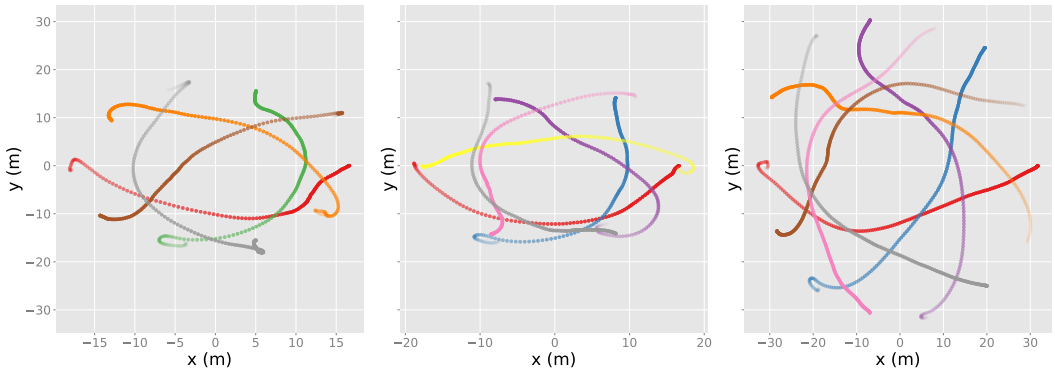


Figure 4.5: 1-nearest neighbor trajectories with $n \in [5, 6, 7]$ agents

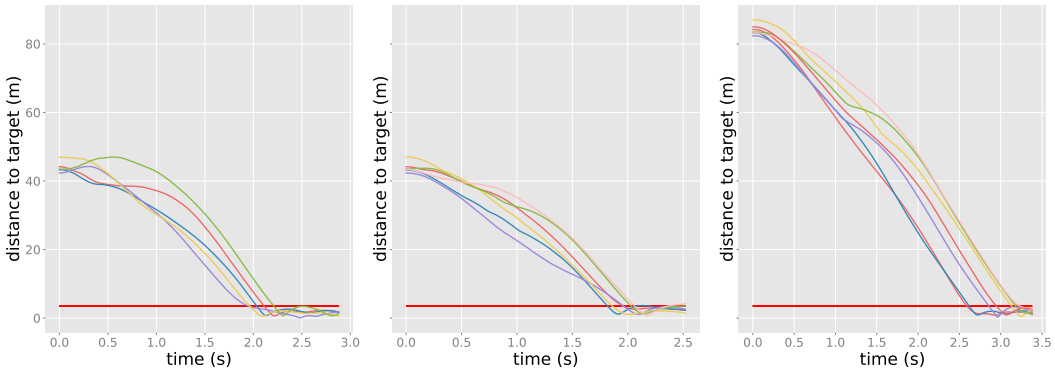


Figure 4.6: 1-nearest neighbor distance to target with $n \in [5, 6, 7]$ agents. The red line denotes the minimal distance to target: 3.5 m.

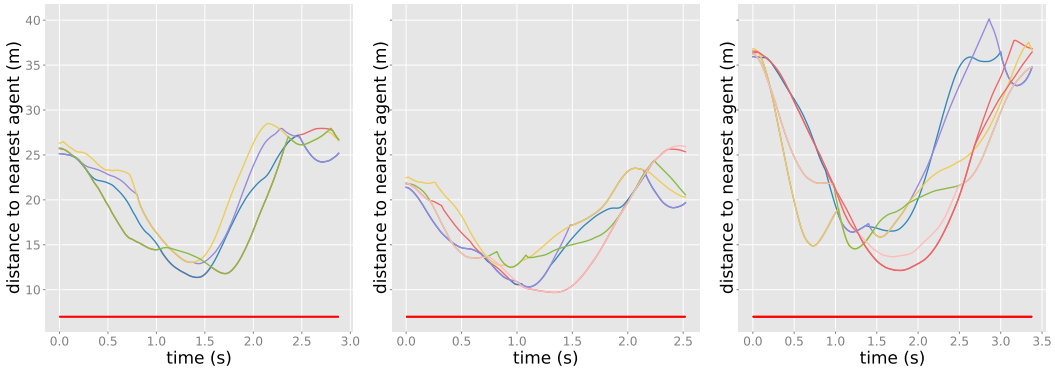


Figure 4.7: 1-nearest neighbor distance to nearest agent with $n \in [5, 6, 7]$ agents. The red line denotes the *minimum inter-agent distance* : 7 m.

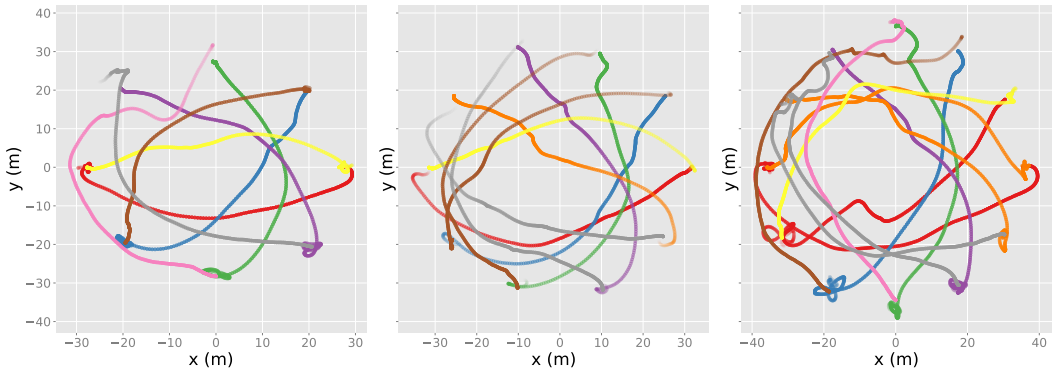


Figure 4.8: 1-nearest neighbor trajectories with $n \in [8, 10, 12]$ agents

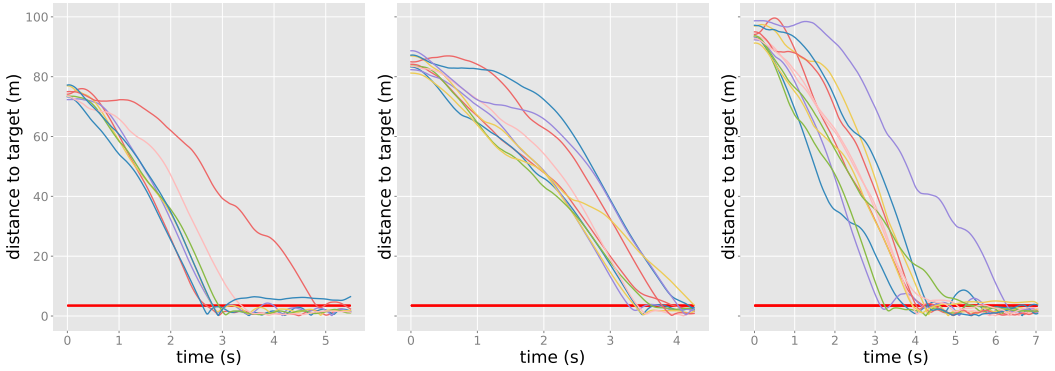


Figure 4.9: 1-nearest neighbor distance to target with $n \in [8, 10, 12]$ agents. The red line denotes the minimal distance to target.

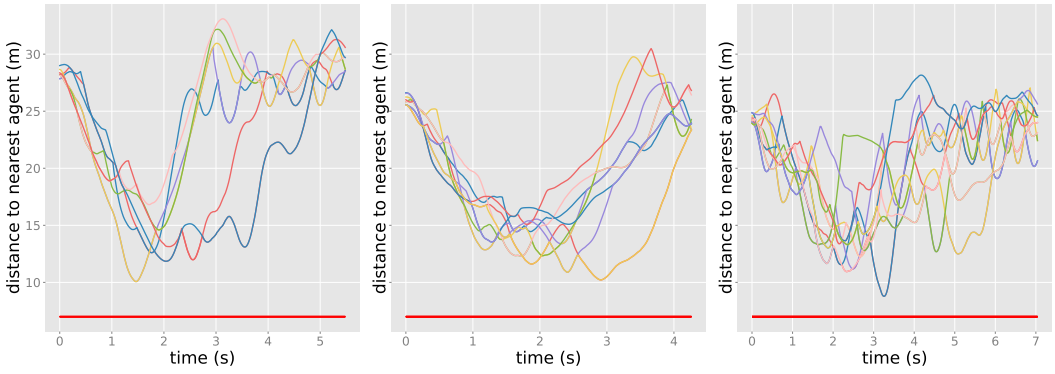


Figure 4.10: 1-nearest neighbor distance to nearest agent with $n \in [8, 10, 12]$ agents. The red line denotes the *minimum inter-agent distance* $7m$.

4.3 Discussion

This section presented results for the multi-agent collision avoidance with continuous action space. Our approach scales reasonably well with the number of agents (a higher number of agents results in slightly longer times to target and slightly affects the minimum distance between the agents). To the best of our knowledge, this is the first application of multi-agent reinforcement learning to solve collision-avoidance with continuous action spaces. The promising results motivated us to further investigate a hardware implementation as a proof of concept. Nevertheless, we need to also list some of the issues and limitations of our approach.

The most relevant one is that with a learning-based approach we can not *guarantee* zero-collision. We can *empirically* assess the behavior of the agents but the risk of collision can not be excluded. Simply because we theoretically can not cover all possible real-world situations and thus must rely on the generalization capabilities of the policies. Nevertheless, we can opt for a hybrid solution combining RL and for example force-based approaches [63]. The second issue is the lack of benchmarks with other baselines and approaches that would help us further assess the quality of our approach. This can be done in future works, because of time constraints we gave priority to the hardware implementation.

5 Hardware Experiments

High-Level Control

The Drone Lab at the Bioinspired communication systems (BCS) is equipped with a fleet of *crazyflies* [25], a nano-quadcopter that can safely operate in indoor environments.
For our experiments we opted for a high-level control which means at each time step t, our extracted policy outputs a velocity setpoint vector S_t corresponding to the linear velocities in the 2-d space and the angular velocity[1] which are then executed by a low-level proportional integral derivative controller (PID) controller. Figure 5.1 illustrates the control mechanism.

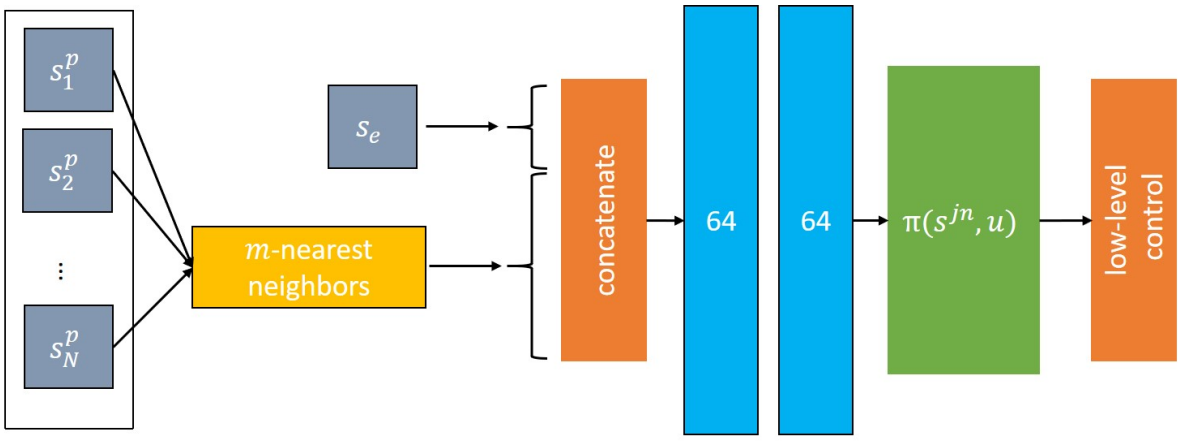


Figure 5.1: Illustration of the state representation with the m-nearest neighbors approach in a high-level control setting.

The weights of the recovered policy are saved in the static memory of the drone. For the last layer, we only consider the mean of the multivariate normal distribution because, at execution time, we generally opt for exploitation and thus employ *deterministic* policies. The policy can be queried efficiently ~ 1 ms which allows for real-time applications.

[1] For a good explanation of drones dynamics, see [24; 1].

Communication Protocol

In order to construct the observation space as in the simulation, we need to access other agents' positions and velocities (to compute euclidean distance and relative velocities). The crazyflie drones are equipped with broadcasting capabilities so we opted for a *round-robin* approach: each drone gets a time slot to broadcast its position and velocity to all other present drones. Meanwhile, the receiving agents compute distances and relative velocities for the neural network evaluation. This technique is also known as time division multiple access (TDMA) [20]. Figure 5.2 illustrates the communication protocol with 3 agents.

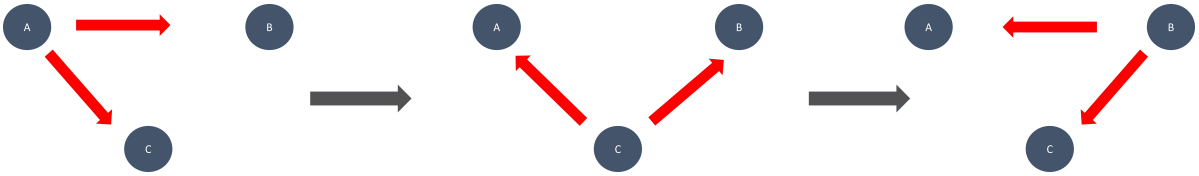


Figure 5.2: The TDMA communication protocol with three agents sharing the same radio frequency. At each step, one agent (the transmitter) broadcasts its position and velocity to the remaining agents (receivers).

State Construction

The crazyflie 2.x is an open-firmware product which allowed us to further modify the source code to access extra state information. We made two modifications to the source code to read the velocities of the quadcopters directly from the *kalman-estimator* [42]. And also to be able to access the actual yaw of the drone. Position and Velocity measurements where done with the lighthouse-positioning system. A good description of the workings and calibration of the lighthouse at the Drone Lab can be found here [58].
To increase realism, sensor noise and other random disturbances had to be taken into account. First, we fly the drones in a circle at different velocities and log the position, and velocity measurements, we then compute the measurements standard variation, and next, when running the simulation we add white noise to the observations with three times the computed standard deviations. The standard deviations from the sensor readings are listed in Table 5.1 and Table 5.2

Table 5.1: Crazyflie 2.x sensor measurements' standard deviation.

readings	bearing	position in m	distance in m
standard deviations	0.000265	0.000184	3.91e−5

Table 5.2: Measurements' standard deviation with different velocities.

velocity in m/s	0.1	0.4	0.8	1
standard deviations	0.00425	0.005187	0.005594	0.008696

Results

We conducted two experiments (*AtoB* and *Move and Collect*). For the *AtoB* task, we simultaneously start all drones (placed in a circle) and assign to each one of them a target in the opposite direction. For the *AtoB* task, we saw that the policies successfully transferred to the real-world: agents learned to fly around in a circle in order to avoid each other. For the second experiment, we saw that the agents also successfully learned to negotiate conflict situations, for example when both agents get assigned to the same position one of them waits for the other to get to the target and clear the way.

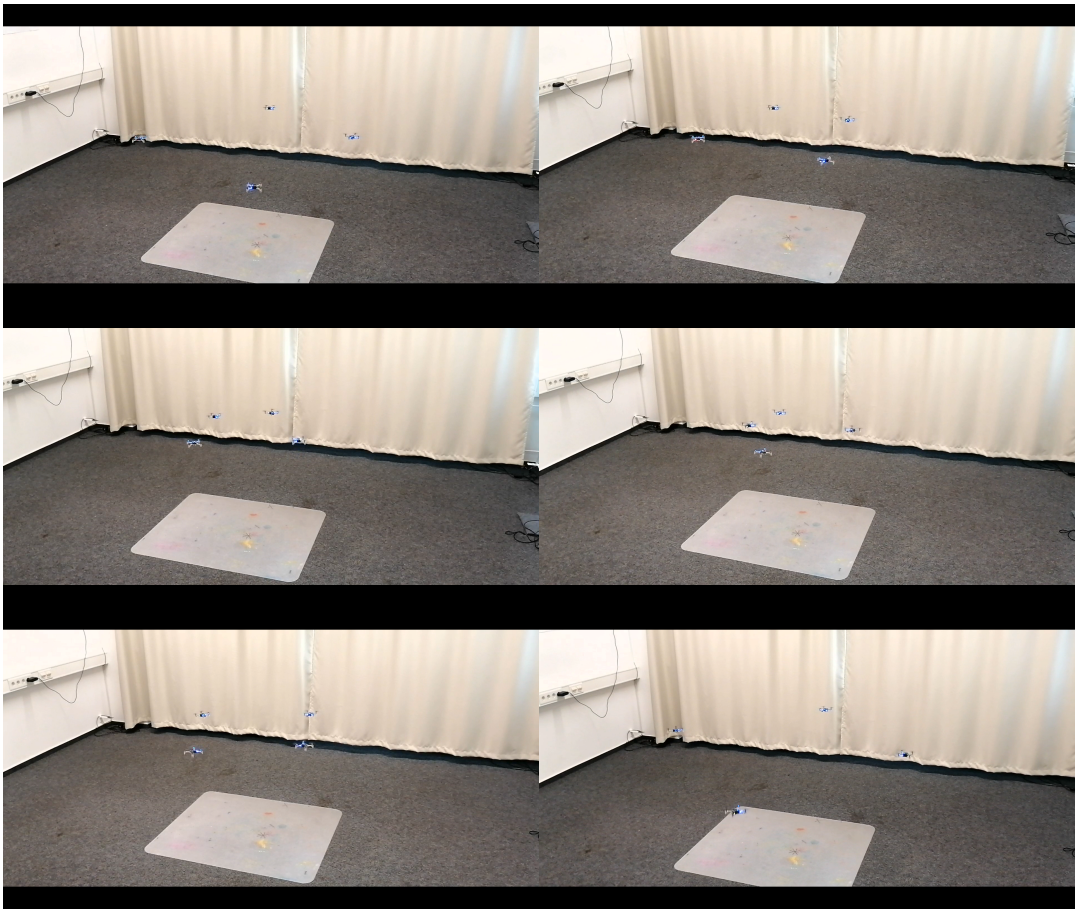


Figure 5.3: 4 quadcopters running the extracted policy on-board in a *AtoB* setting. at each time step t, each agent conditions its actions on its nearest neighbor. The agents form a symmetric pattern in the center of the room.

Discussion

In this chapter, we provided a proof of concept for the sim-to-real transfer of policies learned in a simulation. The implementation on real hardware gave us insights about the stability of the system and the scalability of our approach. The main drawback is that with TDMA, the communication between the agents will become the bottleneck because the time between two neural network evaluations (for any agent) grows linearly with the size of the fleet. The advantage of our approach lies in the degree of freedom with continuous action space and in the fact that we don't assume full-cooperation between the agents which is a more general case.

6 Summary and Outlook

In this thesis, we investigated the application of reinforcement learning to solve the task of collision-avoidance in a fleet of self-interested agents. To deal with the potentially variable size of the fleet we opted for a *topological* interaction rule. We tested our approach in a simulation environment that enables the creation of complex multi-agent settings. We opted for an *Independent-Actors* approach, meaning, that each agent considers other agents as part of the environment. The independence assumption can be seen as a special case of *bounded rationality* [26]: agents do not recursively reason about one another's learning. In future work we can benchmark this approach with other baselines (e.g. a central-critic approach).

We tested different reward mechanisms which resulted in the agents behaving in different, sometimes unexpected ways. Suggesting the emergence of *cooperation* and *deflection* as the agents encounter different situations. Because our actions are spatial movements, we need a way to extract the behavior signature of the agents to further investigate this claim. We also tried different *peers*-observations (other agents' relative position, agents' orientations...) the observations we ended up using resulted in the best learning scenario. Resorting to *curriculum learning*, we were able to scale-up to > 10 agents.

The encouraging simulation results motivated a hardware implementation where agents had to communicate in a round-robin way in order to construct their observation vector. The embedded policies managed to solve the collision avoidance task with 4 agents. Our method is considered a high-level control where our policy outputs continuous actions (velocities) to a low-level controller (PID). It would be interesting to see if we can substitute the low-level controller with a RL policy and test a hierarchical control strategy. Another interesting future work would be to extend this method to the 3-d space.

Our approach, while delivering promising results, can not guarantee collision avoidance, so we need to either combine it with other rule-based methods or resort to *heuristics*. For example, stopping if the inter-agent distance is less than some threshold.

Lastly, we want to point out the fact that such systems can be employed to wage wars and surveillance in totalitarian regimes, but the same can be said about practically any other technology: we choose how to use them. We also believe that the open-source movement and the democratized access to information is our best hedge against the misuses of such technologies.

Bibliography

[1] Demystifying drone dynamics! — bitcraze. https://www.bitcraze.io/2018/11/demystifying-drone-dynamics/. (Accessed on 10/28/2020).

[2] ml-agents learning environment examples. https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Examples.md. (Accessed on 10/28/2020).

[3] Nvidia physx sdk 4.1 documentation. https://gameworksdocs.nvidia.com/PhysX/4.1/documentation/physxguide/Index.html. (Accessed on 10/28/2020).

[4] Starling - wikipedia. https://en.wikipedia.org/wiki/Starling. (Accessed on 10/28/2020).

[5] M. Ballerini, N. Cabibbo, R. Candelier, A. Cavagna, E. Cisbani, I. Giardina, V. Lecomte, A. Orlandi, G. Parisi, A. Procaccini, M. Viale, and V. Zdravkovic. Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study. *Proceedings of the National Academy of Sciences*, 105(4):1232–1237, 2008.

[6] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[7] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[8] Y. Bengio, Jérome Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. volume 60, page 6, 01 2009.

[9] William Bialek, Andrea Cavagna, Irene Giardina, Thierry Mora, Edmondo Silvestri, Massimiliano Viale, and Aleksandra M. Walczak. Statistical mechanics for natural flocks of birds. *Proceedings of the National Academy of Sciences*, 109(13):4786–4791, 2012.

[10] Information Technology Biology Bio-inspired Communication Systems, Electrical Engineering. *Modelling and Simulation of Drone Dynamics*.

[11] Jan Blumenkamp and Amanda Prorok. The emergence of adversarial communication in multi-agent reinforcement learning, 2020.

[12] Léon Bottou. Stochastic gradient descent tricks. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2012.

[13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[14] Martin D Buhmann. *Radial basis functions: theory and implementations*, volume 12. Cambridge university press, 2003.

[15] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P. How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning, 2016.

[16] Iain D Couzin, Jens Krause, Nigel R Franks, and Simon A Levin. Effective leadership and decision-making in animal groups on the move. *Nature*, 433(7025):513–516, 2005.

[17] Felipe Cucker and Ernesto Mordecki. Flocking in noisy environments. *Journal de mathématiques pures et appliquées*, 89(3):278–296, 2008.

[18] Robyn M Dawes. Social dilemmas. *Annual review of psychology*, 31(1):169–193, 1980.

[19] Michael Everett, Yu Fan Chen, and Jonathan P. How. Collision avoidance in pedestrian-rich environments with deep reinforcement learning, 2020.

[20] D. D. Falconer, F. Adachi, and B. Gudmundson. Time division multiple access methods for wireless personal communications. *IEEE Communications Magazine*, 33(1):50–57, 1995.

[21] Jacques Ferber and Gerhard Weiss. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading, 1999.

[22] Jakob N Foerster. *Deep multi-agent reinforcement learning*. PhD thesis, University of Oxford, 2018.

[23] Vincent FranÃğois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and TrendsÂő in Machine Learning*, 11(3-4):219âĂŞ354, 2018.

[24] Andrew Gibiansky. Quadcopter dynamics, simulation, and control. *Andrew. gibiansky. com*, 2012.

[25] W. Giernacki, M. SkwierczyÅĎski, W. Witwicki, P. WroÅĎski, and P. Kozierski. Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. In *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 37–42, 2017.

[26] Gerd Gigerenzer and Reinhard Selten. *Bounded rationality: The adaptive toolbox*. MIT press, 2002.

[27] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[28] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019.

[29] Michael Hamer, Lino Widmer, and Raffaello Drandrea. Fast generation of collision-free trajectories for robot swarms using gpu acceleration. *IEEE Access*, PP:1–1, 12 2018.

[30] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity, 2019.

[31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[32] Maximilian HÃijttenrauch, Adrian ÅăoÅąiÄĞ, and Gerhard Neumann. Deep reinforcement learning for swarm systems, 2019.

[33] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents, 2020.

[34] Leslie P. Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[35] Max Kleiman-Weiner, Mark Ho, Joseph Austerweil, Michael Littman, and Joshua Tenenbaum. Coordinate to cooperate or compete: Abstract goals and joint intentions in social interaction. 01 2016.

[36] Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas, 2017.

[37] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review, 2018.

[38] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning, 2018.

[39] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, page 157âĂŞ163, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

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

[41] Andrew Moore. Hedonism. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2019 edition, 2019.

[42] M. W. Mueller, M. Hamer, and R. D'Andrea. Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1730–1736, 2015.

[43] Mai Nishimura and Ryo Yonetani. L2b: Learning to balance the safety-efficiency trade-off in interactive crowd-aware robot navigation, 2020.

[44] Beth Simone Noveck. *The single point of failure*. Springer, 2011.

[45] F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289âĂŞ353, May 2008.

[46] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*, volume 1 of *MIT Press Books*. The MIT Press, December 1994.

[47] Brian L Partridge. Internal dynamics and the interrelations of fish in schools. *Journal of comparative physiology*, 144(3):313–325, 1981.

[48] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning, 2018.

[49] Rasmus V Rasmussen and Michael A Trick. Round robin scheduling–a survey. *European Journal of Operational Research*, 188(3):617–636, 2008.

[50] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25âĂŞ34, August 1987.

[51] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[52] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[53] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[54] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation, 2016.

[55] David JT Sumpter, Jens Krause, Richard James, Iain D Couzin, and Ashley JW Ward. Consensus decision making by fish. *Current Biology*, 18(22):1773–1777, 2008.

[56] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017.

[57] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[58] Florian Swienty. Decentralized Collision-Free Flight of a Quadcopter Swarm. Master's thesis, Fachbereich Elektrotechnik und Informationstechnik Bioinspired Communication Systems, Technische Universität Darmstadt, Germany, 04.

[59] Csaba Szepesvari. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.

[60] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

[61] P. Trautman and A. Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 797–803, 2010.

[62] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 330–337, Honolulu, HI, May 2007.

[63] J. van den Berg, Ming Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935, 2008.

[64] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.

[65] Wikipedia. Rock paper scissors — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Rock%20paper%20scissors&oldid=984838785`, 2020. [Online; accessed 28-October-2020].

[66] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[67] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 44–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[68] George F. Young, Luca Scardovi, Andrea Cavagna, Irene Giardina, and Naomi E. Leonard. Starling flock networks manage uncertainty in consensus at low cost. *PLOS Computational Biology*, 9(1):1–7, 01 2013.