Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Reinforcement learning for multi-robot navigation

Kairat Bekbolinov

Supervisor: Ing. Vojtěch Vonásek, Ph.D.
Field of study: Cybernetics and Robotics
May 2024

Acknowledgements

I would like to thank Ing. Vojtěch Vonásek, Ph.D. for his help in creating this work during the whole semester.

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.

Prague, 24. May 2024

Abstract

In recent years, multi-robot systems have received significant attention due to their potential to solve complex problems in various fields, including intelligence, surveillance and logistics. Reinforcement learning (RL) has emerged as a promising approach for teaching robots to effectively navigate and cooperate in dynamic environments. In this study, we explore the application of RL techniques to enable autonomous navigation of multiple robots while avoiding collisions and efficiently exploring the environment.

Keywords: multi robot navigation, reinforcement learning, neural networks

Supervisor: Ing. Vojtěch Vonásek, Ph.D.

Abstrakt

V posledních letech se multirobotním systémům dostává značné pozornosti kvůli jejich potenciálu řešit složité problémy v různých oblastech, včetně zpravodajství, sledování a logistiky. Posílení učení (RL) se ukázalo jako slibný přístup pro výuku robotů, jak efektivně navigovat a spolupracovat v dynamických prostředích. V této studii zkoumáme aplikaci technik RL, abychom umožnili autonomní navigaci více robotů a zároveň zabránili kolizím a efektivně prozkoumali prostředí.

Klíčová slova: navigace s více roboty, posilovací učení, neuronové sítě

Překlad názvu: Navigace více robotů s využitím posilovaného učení

Contents

Figures

Tables

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Bekbolinov Kairat**	Personal ID number:	**507615**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Cybernetics and Robotics**		

II. Bachelor's thesis details

Bachelor's thesis title in English:

Reinforcement learning for multi-robot navigation

Bachelor's thesis title in Czech:

Navigace více robotů s využitím posilovaného učení

Guidelines:

. Study the path planning problem and mult-robot navigation task [1]. Get familiar with reinforcement learning and deep learning [2] and their application for multi-robot navigation [3,4,5].
2. Implement the method from [3] for multi-robot navigation. Consider 2D mobile circular robots in 2D environment. Create a suitable model for the environment. Use existing frameworks for deep learning and reinforcement learning (e.g., [6]).
3. Extend the method from task 2) to mobile robots with car-like kinematics.
4. Demonstrate the behavior of the implemented methods in various scenarios using all scenarios from [3].

Bibliography / sources:

[1] LaValle, Steven M. Planning Algorithms. 1st ed. Cambridge University Press, 2006. https://doi.org/10.1017/CBO9780511546877.
[2] Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, MIT Press, 2016.
[3] C. Jestel, H. Surmann, J. Stenzel, O. Urbann and M. Brehler, "Obtaining Robust Control and Navigation Policies for Multi-robot Navigation via Deep Reinforcement Learning," 2021 7th International Conference on Automation, Robotics and Applications (ICARA), Prague, Czech Republic, 2021, pp. 48-54, doi: 10.1109/ICARA51699.2021.9376457.
[4] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, “Emergent Tool Use From Multi-Agent Autocurricula,” arXiv:1909.07528, 2019.
[5] T. Fan, P. Long, W. Liu, and J. Pan, “Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios,” The International Journal of Robotics Research, vol. 39, no. 7, pp. 856–892, May 2020.
[6] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, “OpenAI Baselines,” https://github.com/openai/baselines, 2017

Name and workplace of bachelor's thesis supervisor:

Ing. Vojtěch Vonásek, Ph.D. Multi-robot Systems FEE

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

Date of bachelor's thesis assignment: **23.01.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Vojtěch Vonásek, Ph.D.	prof. Dr. Ing. Jan Kybic	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

Chapter 1

Introduction

Multi-robot navigation stands as a significant challenge within the field of mobile robotics. The task entails guiding robots to their respective goals while concurrently avoiding collisions with obstacles and other robots. For example, navigation of robots through a warehouse to sort goods. Amazon Robotics [12] uses a fleet of autonomous mobile robots (AMRs) to transport items in its fulfillment centers. Another field where multi-robot navigation is used is agriculture, where robots are used to automate tasks such as planting, watering, and harvesting crops. An example is Blue River Technology [11]. By using multi-robot navigation, Blue River Technology (John Deere) can automate the process of applying herbicides to weeds, reducing the need for manual labor and minimizing the use of chemicals. This improves the efficiency of farming operations, reduces costs, and helps to protect the environment. This work presents two types of robots, Circle which is a round robot like Turtlebot and Car-Like which represents a car [more in Chapter 3].

There are two primary approaches to address this challenge: centralized and decentralized methods. In the centralized approach, all robots are controlled by a single overarching system. However, this approach encounters scalability issues, restricting robot movement to predetermined areas. Additionally, reliable communication between the robots and the central system becomes imperative.

In contrast, the decentralized approach, which is the focus of this paper, circumvents these issues. It allows robots to operate independently of one another. The robots get information about the environment, in this case, mostly from sensors. Sensors are used because they allow you to determine

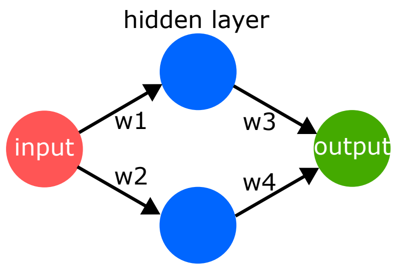

Figure 1.1: Simple Neural Network, where weights $\theta = [w_1, w_2, w_3, w_4]$ using which output will be calculated, input in our case is observation, output is an action.

distances to objects, which allows the robot to make decisions based on the data received from the sensor, one of the most popular sensors is LIDAR.

LIDAR (Light Detection and Ranging) sensors are remote sensing devices that use laser light to measure distances with high accuracy. LIDAR sensors are used in various applications, including autonomous vehicles, robotics, terrain mapping, forestry, agriculture, and environmental monitoring, due to their ability to provide highly accurate 3D measurements of objects and environments.

Our proposed solution involves employing Deep Reinforcement Learning algorithm to train our policy. In our case, policy means a neural network [Figure 1.1] that will take actions based on observations. Neural networks, roughly speaking, are functions with a number of configurable parameters called weights. Depending on these weights, our policy will return certain actions; accordingly, by training a policy we mean iterative updating of the weights in the neural network. The weights are updated by optimization algorithms called optimizers, which are also described in this work.

These policies get observations from the environment [Figure 1.2], in our case consisting of static obstacles and other agents, most of the literature uses the word agent, in this work robot and agent mean the same thing. This makes our environment dynamic, which means that for each iteration, the position of not only one robot changes, but also the rest. This adds complexity to training, since it is necessary to take into account the movement of other agents, to solve such problems people have memory, but modern architectures of neutron networks are fedforward, which means that they have no such thing as memory, this problem is solved in this work in the same way as in [2]. Each robot is assigned a distinct goal to achieve and possesses its own set of observations. These observations include data from LIDAR sensors in the form of laser readings, as well as relative directional vectors to the goal and the robot's orientation. This enables each robot to operate autonomously and make informed navigation decisions.

Reinforcement learning (RL) is a type of machine learning paradigm where an agent learns to make decisions by interacting with an environment. The agent learns to achieve a goal by maximizing cumulative rewards obtained through its actions. The agent's objective is to learn a policy $\pi(s)$ that maps states (s) to actions (a) in order to maximize the cumulative reward (R) it receives over time:

$$\pi(s) \to a$$

Where:

- $\pi(s)$ is the policy that maps states to actions.
- s represents the current state of the environment.
- a represents the action chosen by the policy π in state s.
- R is the cumulative reward the agent receives over time.

In RL, the agent explores the environment by taking actions and receives feedback in the form of rewards or penalties. Based on this feedback, the agent updates its strategy, aiming to maximize the total reward it receives over time. So in this work the following elements of Reinforcement learning were implemented:

- 2D Environment [Chapter 3] with two kind of agents, where we determine the parameters and kinematics of agents from the target and the collision between agents and obstacles. the entire environment was written in Python, all calculations were performed using the Numpy library created for working with vectors, the graphical part was made using the PyGame library,
- Reward system [Chapter 4] that will evaluate the taken action in each iteration, we receive information for assessment from the environment.
- The neural network [Chapter 5] that will represent our policy, it was implemented using the PyTorch library,

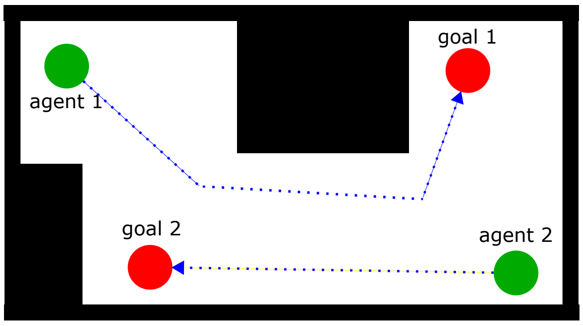

Figure 1.2: Example of an environment with two agents, where the black zones are obstacles, the green circles are the agents, the red circles are their goals, and the blue circles show the possible trajectories of the agents' movements to achieve the goals.

- A learning algorithm [Chapter 5] that will determine how we will train the neural network, it also was implemented using the PyTorch library.

Chapter 2

Related work

In recent years, multi-robot navigation has attracted significant attention due to its applications in various fields such as surveillance, search and rescue, and warehouse automation. Using reinforcement learning (RL) to coordinate multiple robots in complex environments has emerged as a promising approach to solve problems associated with decentralized decision making and coordination. In this section, we review previous studies that have studied multi-robot navigation using RL methods.

Mnih and his colleagues made a significant advancement in the realm of Deep Reinforcement Learning (DRL) through the development of Deep Q Learning (DQN), which demonstrated remarkable performance in playing Atari games [2]. Additionally, the actor-critic framework has been effectively utilized in addressing various DRL challenges [5]. Several enhancements to the actor-critic model, including Proximal Policy Optimization (PPO) [4], which will be describe later, have been proposed. In our DRL approach, we want to train the police to achieve the goal.

2.1 Single Robot Navigation using RL

Before discussing multi-robot navigation, it is essential to understand the advancements in single robot navigation using RL. Early studies such as [10] demonstrated the effectiveness of RL algorithms, such as Q-learning and Deep Q-Networks (DQN), in enabling robots to learn navigation policies in static environments.

2.2 Multi-Robot Navigation using Traditional Approaches

Traditional approaches to multi-robot coordination have focused on centralized planning algorithms such as auctions, consensus-based methods, and potential fields. While these approaches have shown promise in certain scenarios, they often suffer from scalability and coordination overhead issues, particularly in large-scale environments with a high number of robots. Here some traditional approaches:

Consensus-based algorithms: Consensus-based algorithms aim to achieve consensus among multiple robots by iteratively updating their positions or trajectories based on local interactions with neighboring robots. One example is the consensus-based pooling algorithm (CBBA), where each robot maintains a local set of tasks and negotiates with neighboring robots to resolve conflicts and distribute tasks efficiently [6].

Potential field methods. Potential field methods use artificial potential fields to guide robots toward a goal while avoiding obstacles and other robots. Each robot creates a repulsive potential around obstacles and other robots and an attractive potential towards the target. By summing these potentials, robots move towards the goal and avoid collisions. An example is the artificial potential field (APF) method, in which robots adjust their speed based on the potential field gradient generated by the environment [7].

2.3 Multi-Robot Navigation using RL

Recent research has explored the application of RL techniques to multi-robot navigation problems, aiming to overcome the limitations of traditional approaches. Here are a couple of examples of multi-robot navigation using reinforcement learning:

Decentralized navigation of multiple robots: In this approach, each robot independently learns its navigation policy using RL methods. Robots interact with their environment and learn to achieve their goals while avoiding collisions with obstacles and other robots. Through decentralized learning, robots can coordinate their movements without requiring explicit communication or central control. An example of this approach is the work of Gupta et al. [8], where multiple robots learn decentralized policies using deep reinforcement learning to navigate complex indoor environments.

Centralized Multi-Robot Navigation: In this approach, a centralized RL agent coordinates the movements of multiple robots to achieve common objectives. The centralized agent receives observations from all robots and learns a joint navigation policy that optimizes the overall performance of the robot team. An example is the work [9].

2.4 Challenges and Open Problems

Although significant progress has been made in the field of multi-robot navigation using RL, a number of challenges remain. These include challenges of scalability in large-scale environments, robustness to uncertainty and dynamic change, and the need for efficient exploration strategies in complex, high-dimensional state spaces. Addressing these issues requires further research into advanced RL algorithms, hierarchical coordination mechanisms, and real-world deployment considerations.

Chapter 3

Environment

The environment serves as the external system with which the RL agent engages, embodying the intricate dynamics of the surrounding world. It functions as the stage upon which the agent operates, furnishing it with observations, feedback, and rewards predicated on its actions and decisions. Characterized by a discrete action space comprising ten possible actions, as well as in [1], since they were not described in this article we chose actions [Table 3.1], and a continuous state space, the environment encompasses a diverse array of objects represented by polygons or circles. Within this dynamic landscape, two distinct types of agents are implemented, each tasked with navigating through the environment's obstacles and reaching designated goals.

In this environment setup, obstacles are depicted as polygons, serving as

	v	w
a_1	0	0
a_2	v_{max}	0
a_3	v_{max}	$0.25w_{max}$
a_4	v_{max}	$-0.25w_{max}$
a_5	v_{max}	$0.5w_{max}$
a_6	v_{max}	$-0.5w_{max}$
a_7	v_{max}	$0.75w_{max}$
a_8	v_{max}	$-0.75w_{max}$
a_9	v_{max}	w_{max}
a_{10}	v_{max}	w_{max}

Table 3.1: Possible actions, where v_{max}/w_{max} is the maximum linear/angular speed, each type has its own speed.

barriers that impede the agents' progress, while goals are represented as circles, signifying the targets that the agents aim to reach. The agents' overarching objective is to navigate through the environment, maneuvering past obstacles, and successfully reaching their designated goals.

Furthermore, it's worth noting that the environment scale is defined as $5[pix] : 100[mm]$, providing a crucial metric for interpreting the spatial dimensions within the environment. This scale facilitates a coherent understanding of the relationship between the pixel-based representations of objects and their real-world counterparts, ensuring accurate navigation and interaction within the environment.

3.1 Agents

A collision between an obstacle agent or another agent is represented as an intersection of figures; if there is an intersection, then a collision has occurred. A successful episode is considered if the agent managed to approach the target at a certain distance in a certain number of steps. Movement of both agents represented by vector that consist of two numbers first one is linear and second is angular velocity.

3.1.1 Circle agent

This kind of agents represented by circle. The movement of this robot occurs as follows: first, we rotate the coordinate system by an angle equal to the angular velocity and then we shift the center of the circle along the X axis by a vector whose length is equal to the linear velocity.

$r[pix]$	$w_{max}[rad/it]$	$v_{max}[pix/it]$
30	$\pi/6$	5

Table 3.2: Circle agent parameters.

3.1.2 Car-like agent

A car-like robot [Figure 3.1], often referred to as a car-like mobile robot, is a type of wheeled robot designed to move in a similar manner to a car. These

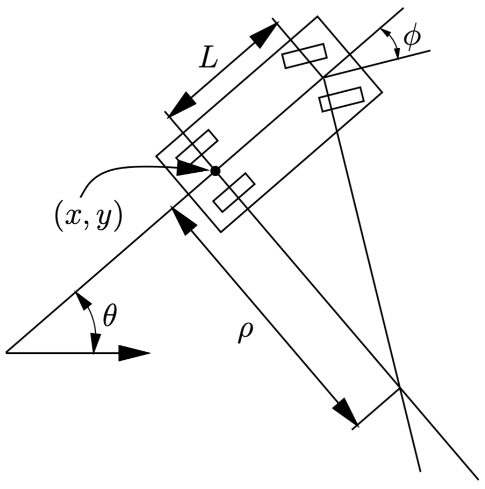

Figure 3.1: Simple car kinematic explanation. Image is taken from [15]

robots usually have two or more wheels. This work presents a model with 4 wheels and dimensions similar to those of an average car. Car-like robots are popular in robotics research and development due to their simplicity, versatility, and ease of control. They provide an excellent platform for experimenting with different navigation algorithms and control strategies. This type of agent is represented by a rectangle, movement is given by the equations:

$$\dot{x} = v\sin(\theta_t) \tag{3.1}$$

$$\dot{y} = v\cos(\theta_t) \tag{3.2}$$

$$\dot{\theta} = \frac{v}{L}\tan(w) \tag{3.3}$$

$W[pix]$	$H[pix]$	$L[pix]$	$w_{max}[rad/it]$	$v_{max}[pix/it]$
93	223	159	$\pi/6$	10

Table 3.3: Car-like agent parameters.

3.2 Observation

We are dealing with a dynamic environment, where for each iteration several robots move at once, each robot initially does not know in which direction the robot will move. Therefore, it is impossible to get a complete picture of the environment from the current observation, because in this case it is impossible to determine in which direction the other robots are moving. To solve this problem, I used the approach from [2], in which the neural network receives the last 4 observations from sensors as input, thanks to which the robot will be able to determine its movement and the directions of movement of nearby robots. Each agent's observation consists of laser lengths o_l, direction vector to goal o_d, and orientation of agent o_{or}. Limitation for laser lengths is $l_{max} = 500$ px, each agent has 16 laser beams. Observation has following form:

$$o_t = [o_{l_t}, o_{l_{t-1}}, o_{l_{t-2}}, o_{l_{t-3}}, o_{d_t}, o_{or_t}] \tag{3.4}$$

where o_l is vector of length 16, o_d is vector of length 2, o_{or} single number, and at $t = 0$ observation will be

$$o_0 = [o_{l_0}, o_{l_0}, o_{l_0}, o_{l_0}, o_{d_0}, o_{or_0}]$$

at $t = 2$:

$$o_2 = [o_{l_2}, o_{l_1}, o_{l_0}, o_{l_0}, o_{d_2}, o_{or_2}]$$

and so on.

3.2.1 Laser observation part

Lasers are represented by many segments having one common vertex and also the angles between adjacent segments are equal. Afterwards, the intersection between the segments and obstacles and robots is searched, and if there are any, then the new length of the segment is equal to the distance from the common vertex to the point of the closest intersection point. Each agent has its own position where we will send lasers from. In case of circle agent it is its center, in case of car-like agent it is point between forward wheels.

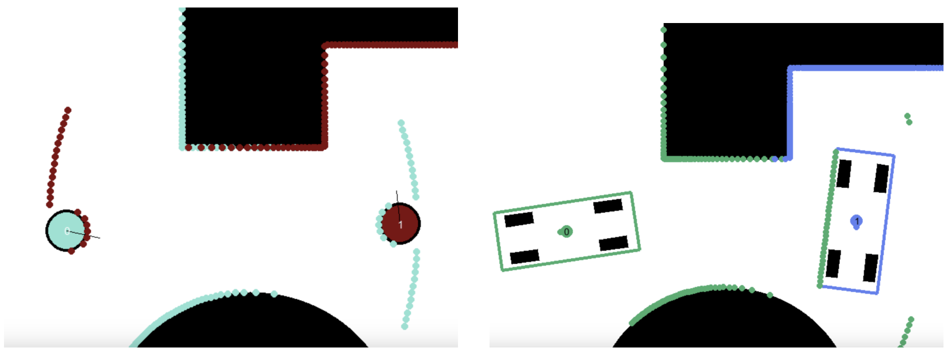

Table 3.4: Laser observation, where the agents are indicated and the ends of the lasers of each agent are indicated by the corresponding color.

3.2.2 Direction vector and agent orientation

The direction vector is defined as the difference between the robot's position and its target. Thanks to it, the robot will be able to determine in which direction it needs to move. Orientation means the angle by which the robot is rotated relative to the world coordinate system. Thanks to it, the robot will be able to understand what angle it needs to turn to move in the desired direction.

3.2.3 Data normalization

Data normalization is a preprocessing technique used to rescale the values of numerical features to a standard range without distorting the differences in the ranges of the original data. It involves adjusting the values of the features so that they fall within a similar scale.. This process ensures that all functions contribute equally to the analysis and modeling process, preventing certain functions from becoming dominant due to their larger scale. The normalization in our case will be in following form.

$$o_{l_{norm}} = \frac{o_l}{l_{max}} \tag{3.5}$$

$$o_{d_{norm}} = \frac{o_d}{\|o_d\|} \tag{3.6}$$

$$o_{or_{norm}} = \frac{o_p}{2\pi} \tag{3.7}$$

Further, when mentioning observation, we mean its normalized version.

Chapter 4

Rewards

In reinforcement learning, rewards are used to indicate the success or failure of an agent's actions in the environment. Reward is a measure of the quality of action in a given position. Reward does not have units of measurement, but only a numerical value, so in this chapter all calculations are made without taking into account units of measurement. Positive rewards usually mean that the agent has performed a desired action or achieved a certain goal. Negative rewards indicate that the action taken was undesirable or took the agent away from the goal.

The agent's goal is to learn a policy, mapping between states and actions, that maximizes the total reward it receives over time. By receiving feedback in the form of a reward, the agent can learn which actions are more favorable in a given environment. In order to stimulate the agent to achieve the goal as quickly as possible, discounted rewards are used.

$$R_T = \sum_{k=0}^{T} \gamma^k r_k \tag{4.1}$$

The reward function is defined as

$$r = \begin{cases} r_{\text{goal}} & \text{goal reached} \\ r_{\text{collision}} & \text{collision} \\ 0 & \text{timeout} \\ r_{\text{laser}} + r_{\text{direction}} + r_{\text{distance}} & \text{else} \end{cases} \tag{4.2}$$

Where r_{goal} represents reward for reaching the goal. And $r_{collision}$ represents negative reward for collision with world or other agent. No reward is given if a timeout occurs. But this will not be enough to train the policy, therefore there is also a 4th point that evaluates the quality of the action.

Laser reward

In order to understand in which direction robot can move and in which direction robot cannot move, because a collision will occur, we use the laser reward.

$$r_{\text{laser}} = \begin{cases} (r_{\text{collision}} + l_{\text{laser}} - l_{\text{min}})(-l_{\text{neg}}) & l_{\text{min}} < r_{\text{collision}} + l_{\text{laser}} \\ 0 & \text{else} \end{cases} \tag{4.3}$$

this part is equal to [1] (equation 9), where l_{neg} is a manually chosen scaling factor and l_{min} is the measured distance of the laser beam currently measuring the lowest distance of all beams.

Direction reward

In order to stimulate the agent to move along the shortest possible path to the goal, its movement for each action is estimated relative to its maximum linear speed. because each agent for each step can get as close as possible to the target at a distance numerically equal to the maximum linear speed.

$$r^{t+1}_{\text{direction}} = (r_{\text{dir\_max}}) \frac{d^t_{\text{goal}} - d^{t+1}_{\text{goal}}}{v_{\text{max}}} \tag{4.4}$$

where $r_{\text{dir\_max}}$ is manually chosen, d^t_{goal} is distance from agent position to goal. This formula allows you to receive rewards in a certain interval, which prevents you from receiving too large numbers.

Distance reward

It seemed that the two upper parts would be enough to train the policy, but as it turned out later at the training stage, sometimes this led to the agents simply starting to spin around in place, which is why the whole reward became approximately equal to zero. Therefore, I also calculate the reward for the distance at which the agent approached the goal relative to its initial position.

$$r^t_{\text{distance}} = (r_{\text{dist\_max}}) \frac{d^0_{\text{goal}} - d^t_{\text{goal}}}{d^0_{\text{goal}}} \tag{4.5}$$

Final states

Our main goal is to achieve the goal without collisions; therefore, it is necessary to give a negative reward for a collision, and a collision with an agent will be considered worse than a collision with an obstacle.

$$r_{\text{collision}} = \begin{cases} -c_{\text{world}} & \text{collision with world} \\ -c_{\text{agent}} & \text{collision with agent} \end{cases} \tag{4.6}$$

Chapter 5

Training

The Actor-Critic algorithm is a popular reinforcement learning technique that combines aspects of both value-based methods (like Q-learning) and policy-based methods (like Policy Gradient). In Actor-Critic, the agent has two main components:

1. Actor: The policy network, which learns to select actions based on the current state of the environment. It directly maps states to actions.
2. Critic: The value function, which evaluates the actions taken by the actor by estimating the expected return (cumulative future rewards) from a given state. It provides feedback to the actor by critiquing its actions.

5.1 Network architecture

Architecture [Figure 5.1] of our neural network can be divided into two parts, Feature extractor and actor-critic parts, where actor and critic will share the same feature extractor, thus they will receive the same features as inputs. Extractor has observation as an input, where o_l will go through two one dimensional layers with kernel size of 4 and stride 1, but fist one has 16 output channels and second one 32 output channels. o_d and o_r will together go through the fully-connected layer with 32 output units, then we

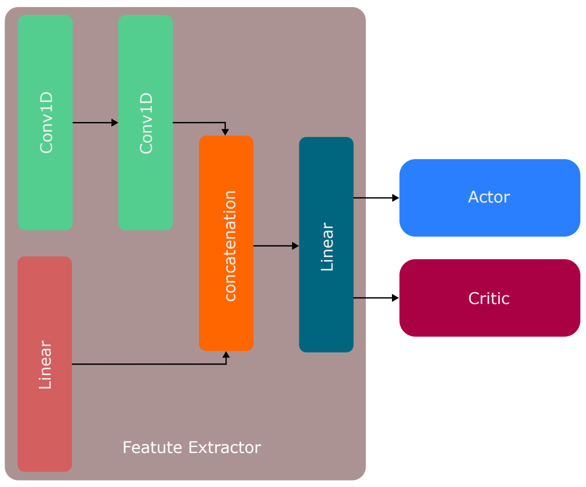

Figure 5.1: actor-critic policy architecture. The architecture of network is similar to the architecture of [1], differing only in the number of cores in convolutional layers and the sizes of linear layers.

concatenate the features and this result will go through fully-connected layer with 352 output units.

Actor is represented by fully-connected layer with 10 output units and critic has 1 output unit.

5.2 PPO algorithm

Proximal Policy Optimization (PPO) [Algorithm 1] is an advanced reinforcement learning algorithm that aims to improve the stability and sample efficiency of Policy Gradient methods like Actor-Critic. PPO addresses the problem of unstable policy updates by introducing a clipped objective function that constrains policy updates to a trust region. This prevents large policy changes and leads to more stable training.

Usually when we talk about PPO algorithm we mean CLIP version, which mean Proximal Policy Optimization Contrastive Learning for Policy Improvement. It is an advanced reinforcement learning algorithm that combines Proximal Policy Optimization with Contrastive Learning for improved policy learning. It leverages contrastive learning to enhance the policy improvement step, resulting in more efficient and stable training. PPO-CLIP has shown promising results in various challenging environments, making it a popular choice for training complex reinforcement learning agents.

To deal with this algorithm we need to define some definitions.

5.2.1 Distribution space

Directly using the outputs of a neural network (deterministic actions) can limit exploration, potentially causing the agent to get stuck in local optima. If the policy were deterministic, it would be challenging to compute meaningful policy gradients, as the gradient estimation relies on the probability of actions. PPO leverages probability distributions to model the policy. This allows for sampling different actions according to their probabilities, fostering better exploration during training.

Distribution spaces are mathematical spaces used to represent sets of probability distributions. These spaces are equipped with mathematical structures that allow probability distributions to be compared, manipulated and analyzed. They are needed in fields such as statistics, machine learning and signal processing to understand and work with uncertain data.

Due to we have 10 discrete possible actions we have deal with discrete distribution space [Figure 5.2]. A discrete distribution space refers to a set of outcomes or events, each of which has a specific probability assigned to it. In the context of reinforcement learning, particularly in algorithms like Proximal Policy Optimization (PPO), a discrete distribution space is used to model the probabilities of selecting different actions from a finite set of possible actions.

5.2.2 Generalized advantage estimation

Generalized Advantage Estimation (GAE) is a method used in reinforcement learning to estimate the advantages of taking a particular action in a given state. It addresses the issue of high variance in estimating advantages, which can lead to unstable training when using Policy Gradient methods.

The advantage A_t of taking action a_t in state s_t at time step t is estimated

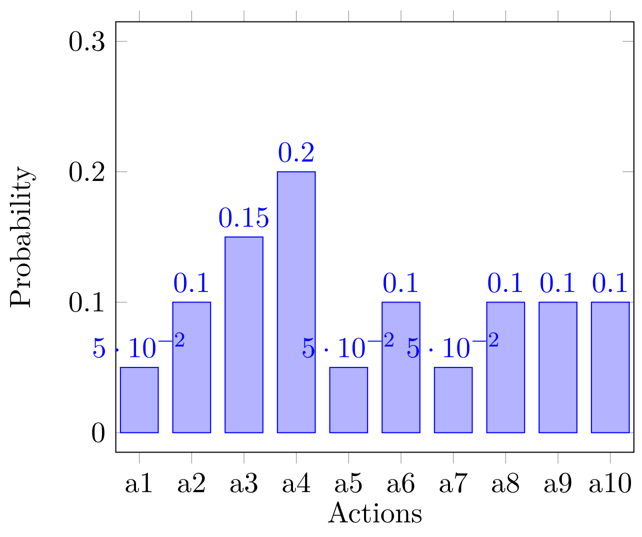

Figure 5.2: Example of discrete distribution with ten possible actions.

using the following formula:

$$A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{5.1}$$

Where:

$$\delta_{t+l} = r_{t+l} + \gamma V(s_{t+l+1}) - V(s_{t+l}) \tag{5.2}$$

$V(s_t)$ represents the estimated value of being in state s_t. γ: discount factor, λ : GAE parameter. The generalized advantage estimator for $0 < \lambda < 1$ makes a compromise between bias and variance, controlled by parameter λ.

5.2.3 Value loss

The value loss measures the error between the predicted value of a state $V(s_t)$ and the estimated return $\hat{V}(s_t)$. It is minimized during training to make the value function V accurately predict the expected cumulative reward from a given state.

The value loss function is often the mean squared error (MSE) between the predicted value and the estimated return:

$$L_{val} = \frac{1}{N} \sum_{t=1}^{N} \left(V(s_t) - \hat{V}(s_t) \right)^2 \tag{5.3}$$

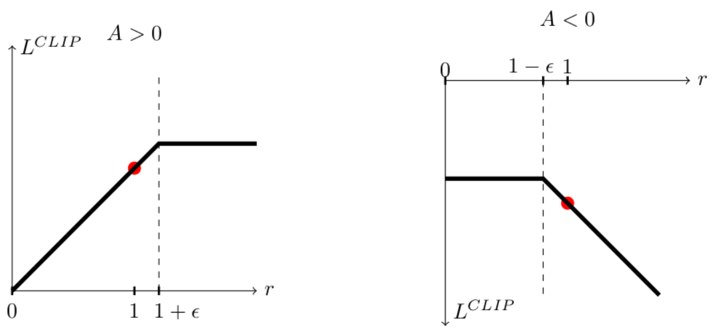

Figure 5.3: Clip visualization with $A > 0$ and $A < 0$, image is taken from [14].

in our case N is amount steps per epoch. Minimizing the value loss helps the value function to better approximate the expected future rewards, which in turn improves the performance of the actor in the Actor-Critic algorithm.

5.2.4 PPO Loss "CLIP"

In the Proximal Policy Optimization (PPO) algorithm, the PPO loss is the objective function used to update the policy network. The goal of the PPO loss is to improve the policy while ensuring that the policy updates are not too large, preventing instability during training.

$$L^{CLIP}(\theta) = -\mathbb{E}_t \left[\min\left(r_t(\theta) \cdot A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot A_t\right)\right] \tag{5.4}$$

where $\pi_\theta(a_t|s_t)$ is probability of action a_t in state s_t returned by policy π with weights θ, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new policy and the old policy. A_t is the advantage of taking action a_t in state s_t. ϵ is a hyperparameter that controls the size of the policy update.
The clip function ensures that the probability ratio $r_t(\theta)$ is bounded between $1-\epsilon$ and $1+\epsilon$. This prevents large changes in the policy during training, stabilizing the learning process.

5.2.5 Optimizers

Optimizers in the field of machine learning, or more specifically in our case, in the case of reinforcement learning, are algorithms or methods for updating weights in neutron networks based on the outputs of loss functions. Below is an explanation of how they work, moving smoothly to the optimizer used in this work.

Gradient Descent (GD) and Stochastic Gradient Descent (SGD)

Gradient Descent (GD) differs from stochastic gradient descent (SGD) only in that in the second case we divide the dataset into batches and weight optimization occurs after each batch. These algorithms are based on finding the gradient of the loss function based on the weights of our model, and the weights are updated as follows:

$$g_t = \nabla L(\theta_t) \tag{5.5}$$

$$\theta_{t+1} = \theta_t - \alpha g_t \tag{5.6}$$

where α is learning rate, and due to the fact that the gradient is always directed towards the growth of function, we will subtract it to minimize the loss function. But these algorithms suffer from the local minimum problem, which means that if a local minimum is found, our optimization will stop before reaching the global minimum [13].This problem is solved by the method of moments:

$$m_t = \beta m_{t-1} + (1-\tau)g_t \tag{5.7}$$

where $\beta \in [0,1]$, τ is damping, this is how much we will lower the current gradient, and if $t = 0$ then $m = g$, and the update of the weights looks like this:

$$\theta_t = \theta_{t-1} - \alpha m_t \tag{5.8}$$

Root Mean Square Propagation (RMSProp)

The tasks in which optimizers are used are quite different and the gradient values can differ thousands of times, and each time you have to select the learning rate. To solve this problem, an algorithm with an adaptable learning rate called RMSProb was invented:

$$v_t = \gamma v_{t-1} + (1-\gamma)g_t^2 \tag{5.9}$$

this formula is called Exponential Moving Average, and v is called square average where $v_0 = 0$, and usually $\gamma = 0.999$, then update of the weights has folloving equatation:

$$\theta_t = \theta_{t-1} - \alpha \frac{g_t}{\sqrt{v_t}} \tag{5.10}$$

subsequently this optimizer formed the basis of the most popular optimizer called Adam.

Adaptive Moment Estimation (Adam)

The optimizer used in this work, thanks to its efficiency, Adam converges faster and performs well with large datasets or parameters, and robustness, it works well in practice and is less sensitive to the choice of hyperparameters compared to other optimization algorithms. Roughly speaking, Adam is SGD with momentum and RMSProb and one feature:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{5.11}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{5.12}$$

Bias-corrected estimates are used in the Adam optimizer to counteract the biases that occur in the estimates of the first and second moments during the initial stages of training. Then the weights update is:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{5.13}$$

5.2.6 Pseudo code

Now using the points above we can describe the learning algorithm.

Algorithm 1 Proximal Policy Optimization (CLIP)

```
1: Initialize policy network $\pi_\theta(a|s)$ with parameters $\theta$, value function $V_\phi(s)$
   with parameters $\phi$
2:
3: Initialize hyperparameters, buffer, and optimizer settings
4: for each iteration do
5:     Collect trajectories using current policy: $(s_t, a_t, r_t, s_{t+1}, d_t)$
6:     Compute advantages $A(s_t, a_t)$ and target values $V_{\text{target}}(s_t)$ using re-
       wards and value function
7:     for each epoch do
8:         Compute policy loss $L^{CLIP}(\theta)$ and value loss $L_{val}(\phi)$ using col-
           lected data
9:         Compute total loss $L = L^{CLIP}(\theta) + c_1 L^{VF}(\phi)$
10:        Update policy and value function parameters using optimizer
11:    end for
12: end for
```

Usually values of $L^{VF}(\phi)$ much higher then $L^{CLIP}(\theta)$ so there is coefficient c_1 which will minimize $L^{VF}(\phi)$. In this work PPO has following hyperparameters:

Learning rate	0.0003
Discount γ	0.99
GAE parameter α	0.95
PPO clipping	0.2
c_1	0.5

Table 5.1: PPO hyperparameters.

Chapter 6

Experiments

The learning process was constrained by the computational complexity inherent in the laser observation calculations. These calculations were executed using Python libraries, which perform operations on the processor. This reliance on processor-based computations, as opposed to utilizing more efficient parallel processing units such as GPUs, resulted in limitations on the number of agents that could be effectively trained. Calculating 126 step of one Circle agent takes 2.8 seconds, 3 seconds for Car-Like, these calculations were performed on the Apple M2 chip, macOS version 14.2. This complicated the process of selecting the necessary hyperparameters, so most of the training did not produce results.

Initially, in one epoch we initialized one environment and as a result we had N*M, where N is number of steps per one epoch, M is number of robots per initialization, pairs (a_t, s_t),this led to the fact that the training was very unstable and did not give any results. Therefore we initialized 20 environments at each epoch, which gave 20 times more pairs (a_t, s_t). This made training more stable and produced greater results. Furthermore, the dependency of the learning process on the specific environmental conditions necessitated a tailored approach in this study. Unlike the setup described in [1], this work introduced several variations, including a different number of lasers, distinct maps, and varying robot parameters. Consequently, it was imperative to independently determine the appropriate values for the reward function to suit these unique conditions. The reward function parameters I employed are detailed below.

r_{goal}	2
c_{world}	1.2
c_{agent}	1.5
r_{dir_max}	1
r_{dist_max}	0.75
l_{neg}	0.2

Table 6.1: Reward parameters.

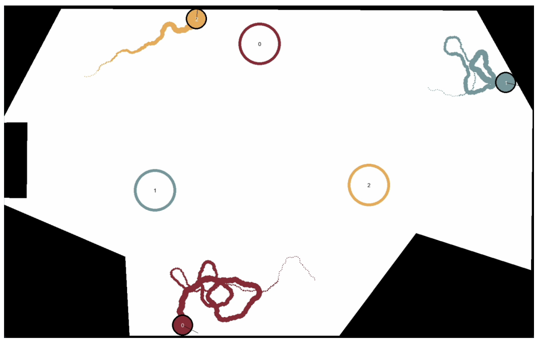

Figure 6.1: Robot movement on an initialized policy without training.

Circle agent training

Initially, the training involved three agents operating on the map illustrated on Figure 6.1. The movement in which the agent moved from the moment of its initialization can be seen from the following: the agent started moving from the thin part and stopped in the thicker part. During this stage, the maximum number of steps was restricted to 126. For each iteration, targets were assigned to the agents in a random sequence. Additionally, the position of each agent was adjusted within a specific radius, and their orientation was modified within a 30-degree range. This randomized approach aimed to enhance the robustness and adaptability of the agents' learning process by exposing them to a variety of positional and directional scenarios.

The training process extended over 1300 epochs. During the initial stages, the average reward exhibited significant fluctuations, oscillating dramatically from small positive values to negative values [Figure 6.2]. This variability in the average reward likely reflects the agents' attempts to explore and learn from the environment, adjusting their strategies and behaviors in response to the different scenarios they encountered. Such fluctuations are common in the early phases of reinforcement learning as the agents are actively experimenting to find optimal policies.

The training process yielded the following results [Table 6.2], which were quantified by calculating success percentages based on 100 random attempts.

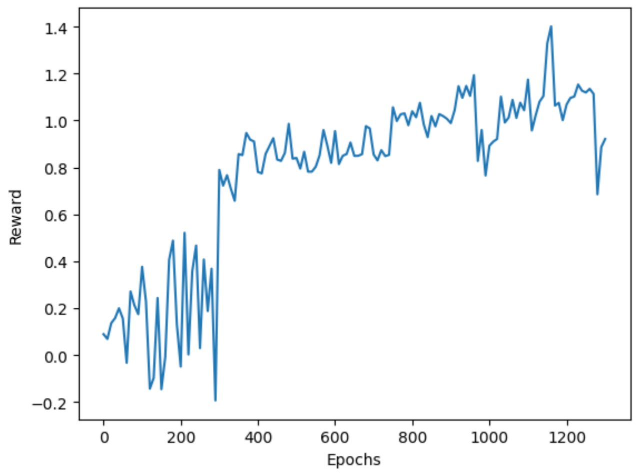

Figure 6.2: Reward during training with 3 robots on map given by Figure 6.1.

This approach provided a robust measure of performance, ensuring that the results were not influenced by any specific sequence of events or conditions within the environment. We also want to evaluate how well our policies work, for this we initialize 100 environments and the success rate is equal to the number of agents who achieved their goals divided by the total number of agents multiplied by 100%, the remaining percentages are calculated in a similar way [Table 6.2].

It was also observed that when the agent failed to hit the target, it tended

success rate	78%
collision with world	0.66%
collision with agents	4%
time out	17.33%

Table 6.2: Results on map given by Figure 6.3 after 1300 epochs.

to execute large turns. This behavior stems from the agent's inability to move backward and the limited availability of sharply turning maneuvers. In situations where the target was missed, the agent encountered difficulty in adjusting its trajectory smoothly, resulting in exaggerated turning motions.

In the case of testing one robot from different positions and orientations and without obstacles, the robot always reaches the goal, but it made too much of a turn [Figure 6.4]. In my opinion, the fact that the agent moves in a circle even taking into account that there are no other agents and obstacles is due to the fact that when training three agents [Figure 6.3], the optimal strategy was to move in a circle, which prevented collisions with other agents.

However, on more complex maps, with narrow passages and a large number of robots, this model cannot cope. Therefore, taking the weights of the trained

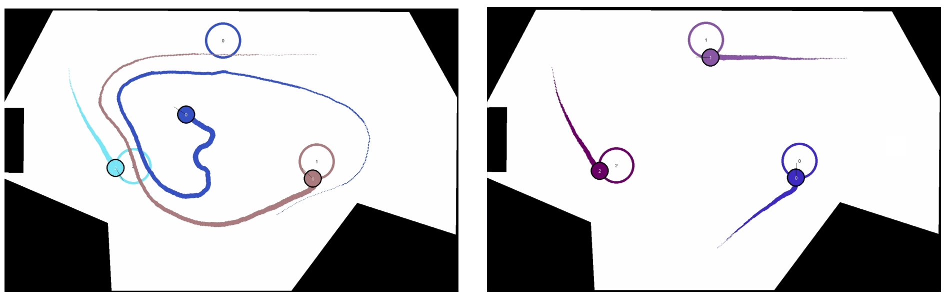

Figure 6.3: Random attempts of pretrained policy.

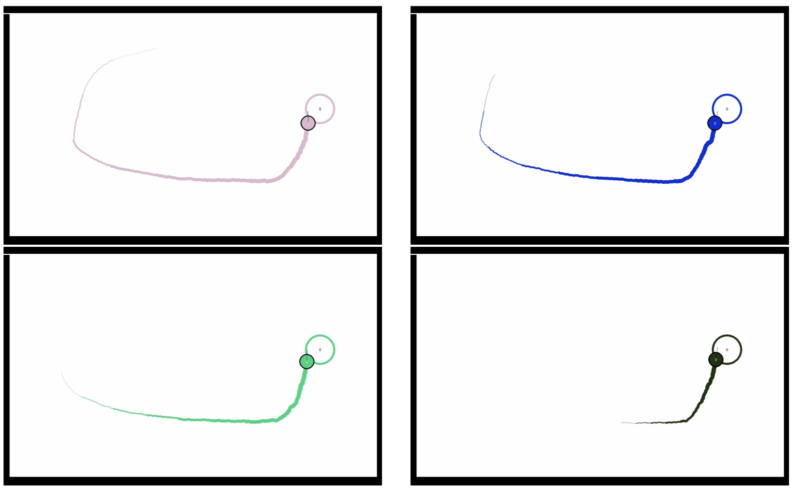

Figure 6.4: One robot from different positions without obstacles.

model, I launched a new training with a large number of steps per epoch equal to 256 [Figure 6.5].

The rate at which the reward increased was notably sluggish, and regrettably, the robots were unable to reach the goal within the allocated time frame. However, a positive outcome emerged from the situation: instances of collisions were exceedingly rare, and, overall, the robots exhibited consistent movement toward the designated goal location. Despite falling short of the ultimate objective, this outcome underscores a notable achievement in the navigation process, indicating a level of proficiency in navigating the environment and avoiding collisions [Figure 6.5].Furthermore, I hold the belief that by augmenting the number of epochs in the training process, we can enhance the capability of our models to handle increasingly intricate scenarios. This extended training duration will afford the models more opportunities to learn and adapt to a wider range of situations, ultimately leading to improved performance in navigating complex environments.

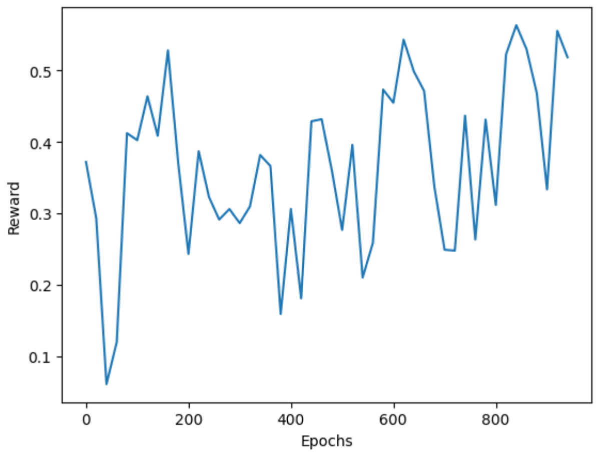

Figure 6.5: Reward during training with 4 robots on map given by Figure 6.6.

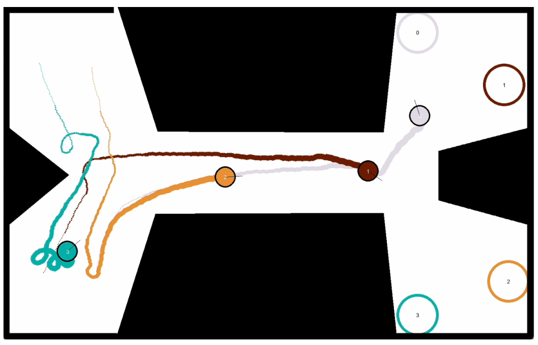

Figure 6.6: 4 agents in difficult map.

Car-Like

Due to the fact that agents have different methods of movement, they should have different policies. To begin with, the model was trained for straight driving [Figure 6.7], with obstacles on both sides with a maximum number of steps of 126, and the following results were achieved [Table 6.3].

success rate	97%
collision with world	2%
time out	1%

Table 6.3: One Car-Like agent in map given by Figure 6.7.

Car-like agents were trained [Figure 6.8] with a maximum of 126 steps per episode. The training process was conducted across multiple maps [Figure 6.9], which were selected randomly for each training session.

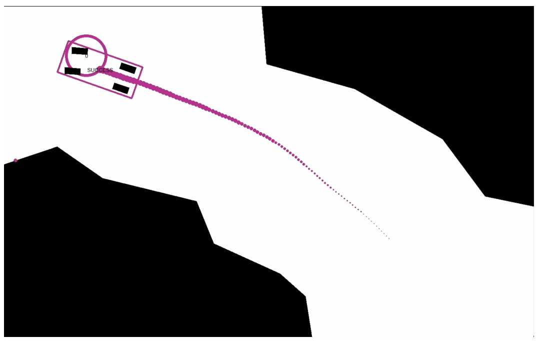

Figure 6.7: One Car-Like agent in simple map.

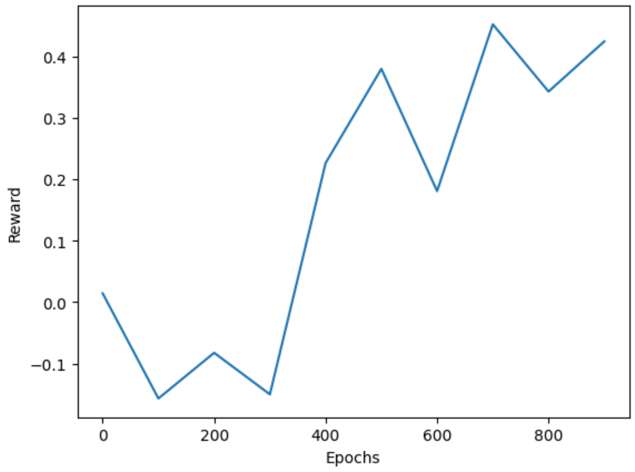

Figure 6.8: Reward during training policy for Car-Like agent on maps given by Figure 6.9.

However, an unexpected problem emerged during this phase. Certain situations occurred with disproportionate frequency during training, leading to an overfitting issue. As a result, the model became overly specialized in responding to these frequently encountered scenarios, at the expense of its generalizability. This overfitting caused the agents to perform poorly in less common situations, often resulting in the agents moving in incorrect directions when faced with unfamiliar or less frequently encountered circumstances. This highlighted the need for a more balanced and varied training dataset to ensure robust and adaptable agent performance. Also, the inability to drive back created additional problems, since there were places where the agent would no longer be able to get out. Evaluation of this policy [Table 6.4].

success rate	34.5%
collision with world	7%
collision with agents	5%
time out	53.5%

Table 6.4: Two Car-Like agents in maps given by Figure 6.9.

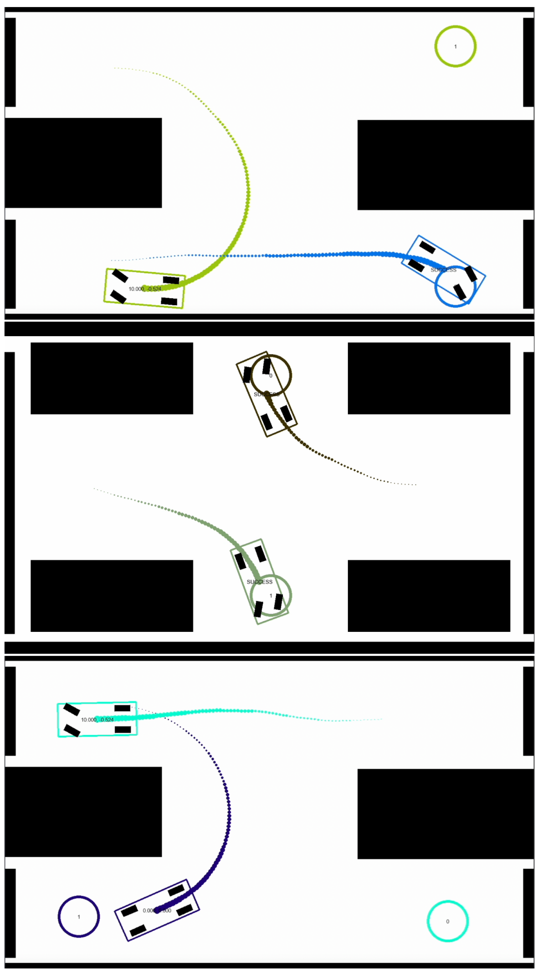

Figure 6.9: Performance of pretrained policy on map with two Car-Like agents

Chapter 7

Conclusion

Reinforcement learning holds immense promise for robot navigation, as evidenced by the remarkable achievements showcased in this study. In particular, we were able to attain a success rate of 100% for the Circle agent and 97% for the Car-like agent in navigating simple scenarios with a single robot. However, challenges become more pronounced in complex environments. Nevertheless, as demonstrated above, even under these challenging conditions, the potential of reinforcement learning remains evident. I believe that increasing the number of epochs as well as increasing the variety of maps will improve the results, because neural networks will learn to identify more general features, which will allow them to navigate effectively on unknown maps.

The main obstacle encountered in improving these results was optimizing the computation required for the environment while meeting time constraints. Also specifically in our work, there was a problem with debugging errors due to the fact that many things were implemented independently. Which led to uncertainties during training. For example, it often happened that the reward did not change and it was initially unclear where the error was; it could be in the implementation of the environment, the architecture of the neural network, the reward system, the learning algorithm, or the selected parameters. Despite these challenges, the results highlight the promising potential of reinforcement learning in developing the navigation capabilities of robots.

In the future, it is necessary to consider the possibility of moving backwards for a Car-Like agent, which would help solve the problem associated with places from which the agent will no longer be able to get out. It is also worth conducting training with continuous action space, which could potentially lead to smoother movement of agents and a greater number of possible actions.

Chapter 8

References

[**1**] C. Jestel, H. Surmann, J. Stenzel, O. Urbann and M. Brehler, "Obtaining Robust Control and Navigation Policies for Multi-robot Navigation via Deep Reinforcement Learning," 2021 7th International Conference on Automation, Robotics and Applications (ICARA), Prague, Czech Republic, 2021, pp. 48-54, doi: 10.1109/ICARA51699.2021.9376457.

[**2**] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236

[**3**] Schulman, John et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation." CoRR abs/1506.02438 (2015): n. pag.

[**4**]J.Schulman,F.Wolski,P.Dhariwal,A.Radford,andO.Klimov.ProximalPolicyOptimization Algorithms. arXiv 1707.06347 [cs.LG], 2017

[**5**] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, Proceedings of The 33rd International Conference on Machine Learning, volume 48 of Proceedings of Machine Learning Research, pages 1928–1937, New York, New York, USA, 2016.

[**6**] Gerkey, B., Mataric, M. J. (2004). "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems". International Journal of Robotics Research.

[**7**] Khatib, O. (1986). "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots". International Journal of Robotics Research.

[**8**] Gupta, J. K., Egorov, M., Kochenderfer, M. J., Levine, S. (2017). "Learning decentralized controllers for multi-agent navigation". Conference on Robot Learning.

[**9**] Orr, J.; Dutta, A. Multi-Agent Deep Reinforcement Learning for Multi-Robot Applications: A Survey. Sensors 2023, 23, 3625. https://doi.org/10.3390/s23073625

[**10**] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). "Human-level control through deep reinforcement learning". Nature.

[**11**] Blue River Technology: `https://bluerivertechnology.com/ourmethods/`

[**12**] Amazon Robotics: `https://www.amazon.science/latest-news/amazon-robotics-auto`

[**13**] Visual explanation of optimizers `https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-a`

[**14**] Clip explanation: `https://ai.stackexchange.com/questions/37608/why-clip-the-ppo-objective-on-only-one-side`

[**15**] LaValle M. Steven. Planning Algorithms. *Cambridge University Press.* 2006 `https://lavalle.pl/planning/`