AUTONOMOUS MAPPING AND NAVIGATION OF AN UNKNOWN ENVIRONMENT USING A REINFORCEMENT LEARNING APPROACH

R. (Rob) Schulte

MSC ASSIGNMENT

November, 2020

057RaM2020
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE   UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Summary

The capability of autonomous (mobile) robots to navigate their environment is a critical first step in the development towards deployment of mobile robots in our daily lives. Traditional path planners utilize a map of the environment in order to plan a collision-free path. However, a-priori map knowledge is not always precise or, depending on the task, not available at all. Other approaches which construct a map of the environment during the navigation process, such as (active) SLAM approaches, require pre-coded navigational directives or path planners in order to obtain a collision-free path. In this work we formulate the navigation and mapping task in a-priori unknown environments as a reinforcement learning problem such that the previously mentioned problems can be alleviated. In particular, a DDPG and SLAM approach is used in order to navigate and map known and unknown environments using sensory information obtained .

The main novelty of this work is aiding the robots navigational process by combining SLAM information and a concept of reachability in the state-space within the reward function definition. In this work, four different reward functions are compared on several domestic simulation environments with different sizes and complexities. The used reward structures are evaluated in the training environment in addition to a-priori unknown environments to test the generalization capabilities. The evaluation indicates that the proposed curiosity approach was able to cut down training time significantly within the used training environments. Other used performance metrics such as trajectories and success ratio also signify that the curiosity approach surpasses all other approaches in both the training and testing environments.

Contents

1 Introduction

1.1 General statement of the problem

Autonomous exploration of unknown environments is a challenging, but well-researched topic for mobile robots. In contrary to manual or operated robots, autonomous robots aim to reduce the amount of required human assistance, given a task, to a minimum. This could increase productivity (e.g. autonomous delivery drones) and in some cases boost safety levels (e.g. self-driving cars). However, these applications often require navigation in environments that are not inherently designed for robots - e.g. schools, offices, and cities. Hence, autonomous navigation is the first step towards a future with autonomous robots in our daily lives.

The problem of autonomous navigation is very broad and depends, amongst other things, on the type of environment (air, land, water), the involved sensors with which the robot can perceive its environment, and the type of actuation which is used to drive the robot through its environment. Furthermore, the robot needs to be able to localize itself within the environment and plan a proper, collision-free, path. In principle, this can be obtained if a map of the environment is accessible but in many real-world situations an accurate map of the environment is not readily available a-priori (e.g. the mars-rover used to explore Mars might have had an indication of its environment by training in the Nevada desert, but no proper map). Instead, a numerical representation of the environment is often constructed using the robot's onboard sensors to aid the robot with the navigation process.

The process of keeping track of the constructed map and localizing the robot within this constructed map is known as a Simultaneous Localization and Mapping (SLAM) problem. Many (approximate) solutions have been proposed within the context of mobile robots such as fast-SLAM (Montemerlo et al. (2002)), which uses a non-linear, Extended version of the Kalman Filter (EKF) in order to estimate the posterior distribution over the robots pose along with the positions of the landmarks. Where the landmarks are static locations within the environment described by two numerical values. As the environments grow larger so do the number of landmarks within that environment and consequently the computational complexity. Therefore, innovation in landmark selection was done (e.g. Guivant and Nebot (2001)). However, the methods still rely on pre-code navigational directives based on landmark positions to function properly (e.g. go straight when you recognize landmark x). One can see how coding these directives can be problematic if the environment is not known beforehand or not static.

Active SLAM methods, as the name implies, actively plans the path of the robot and at the same time localizes and builds the map of the environment. The most well-known active SLAM method is frontier-based exploration as introduced by Yamauchi (1997). It generates navigational targets (frontiers) located between the unknown and known areas of the built map and then navigates to those points using the aforementioned path planner. The generated trajectories are often sub-optimal and introduce a decision-making dilemma: to which frontier should the robot navigate next. Moreover, the path to the chosen frontier does not necessarily coincide with the optimal path (e.g. navigating to the closest frontier is not always optimal). Within the scope of this thesis, we aim to address some of these issues by using Reinforcement Learning (RL) in order to generate efficient trajectories and autonomously navigate and map an unknown environment.

Reinforcement learning is an approach, which learns to navigate the environment through a trial and error process by observing and interacting with its environment. The observations of the environment can come in many forms such as pixels from a camera image or positional data from an encoder. This information (state) is then used by the controller (agent) to determine the "optimal" control signal (action) and the agent receives a reward based on how good that action was. These reward signals are task dependant, for example bumping against a wall might give a negative reward. The maximization of the sequence of rewards $(r_0, r_1 ... r_t)$ is the goal of reinforcement learning. As the reward is the only feedback for the robot on how well it performs in any environment, so-called reward shaping (designing the reward function), is crucial for the success of the algorithm. This thesis contributes by proposing adaptations within the reward structure in order to successfully map, navigate, and explore an unknown environment using a mobile robot.

1.2 Problem formulation and challenges

Within the context of robotics, reinforcement learning can be successfully used to generate velocity commands for a mobile robot in order to navigate from a starting position A to desired position B, as was shown by Mustafa et al. (2019). Herein, the reward signal for the robot was based, amongst other things, on the distance to the desired point B. Furthermore, it was concluded that shaping the reward function based on the grid map obtained from SLAM can improve the training performance of the algorithm. However, this reward signal cannot be applied when autonomously mapping unknown environments since there are no fixed navigational targets available. Within the scope of the individual assignment, (Schulte, 2019) showed that obstacle avoidance behaviour can be achieved changing the reward signal to include a notion of how complete the map build by SLAM is. However, these results also show that the implemented reward signal does not sufficiently describe the quality of the trajectory and successful navigation of the environment was thus only partially achieved. This thesis aims to address these found issues by innovating reward signal further, as is discussed in Chapter 4 of the thesis. This can be graphically illustrated within the standard reinforcement learning context, as described in the previous section, as follows:

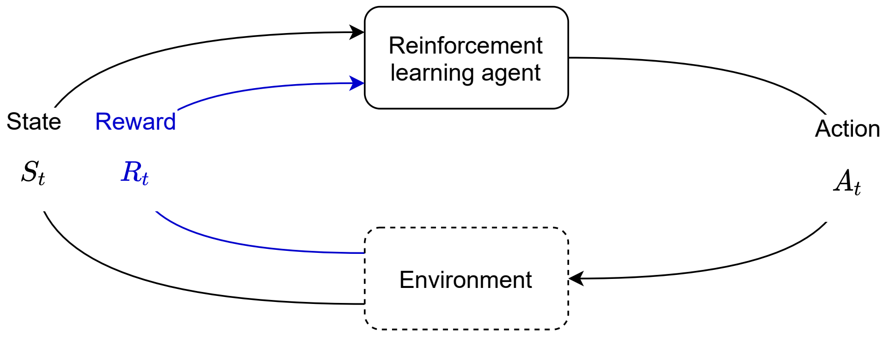


Figure 1.1: Reward signal within the context of the standard reinforcement learning framework

Within the reinforcement learning paradigm many choices for reinforcement learning agent do exist that enable a robot to learn to map an environment using sensory information present on the mobile robot. Most of the impressive results have been achieved by Deep-Q-Networks (DQNs) (e.g. Mnih et al. (2013)). Although DQNs function in continuous environments, the major downside is that they are not capable of handling continuous action spaces. In principle, we could discretize the action space to a forward, backward, left, and right motion, but this straightforward solution introduces problems as described by Lillicrap et al. (2015). Most notable among them is the exponential growth of the number of actions with the degrees of freedom. For example, a mobile robot with 2 degrees of freedom and the most unrefined form of discretization $a_i = \{-k, 0, k\}$ we obtain $3^2 = 9$ actions. If we require any finer speed settings e.g. 3 for each dof $a_i = \{-k, -\frac{2}{3}k, -\frac{1}{3}k, 0, \frac{1}{3}k, \frac{2}{3}k, k\}$ we would get $7^2 = 49$ possible actions. One can see how any refined action space would lead to an explosion in the action space in addition to throwing valuable information away about the nature of the action space. Instead, we opt for a more principled approach, which is capable of handling continuous action spaces by parameterizing the policy. The algorithm, which is used, is the Deep Deterministic Policy Gradient (DDPG), as introduced in Lillicrap et al. (2015). Despite these advantages, the authors of (Botteghi et al. (2020)) have shown that it possible to get state-of-the-art results with a DQN and DRQN approach when using an unrefined form of discretization. A comparison on performance between DDPG and DRQN/DQN is one of the supplementary contributions of the thesis.

The main goal of this project is to successfully explore, navigate and construct a 2-D map of an (unseen) 3-D environment using the raw sensory data from a mobile robot and information obtained by the SLAM algorithm. To achieve this, a reinforcement learning algorithm, DDPG (Lillicrap et al. (2015)) is used to map the observations of the robot to physical control actions. In addition, a Rao-Blackwellized Particle Filter (RBPF) SLAM algorithm (Murphy (2000) and Doucet et al. (2013)) is used to construct the map and localize the robot in the constructed map. Furthermore, we aim to find out the limitations of this approach when introducing the robot to previously unknown environments. Hence, the challenge of this project is twofold:

i **Reward Shaping**
To successfully navigate and built a map of the environment, it is paramount that we provide the agent with the proper exploration cues. Within the reinforcement framework, these can be supplied by shaping the reward function in such a way that the desired behavior is achieved. The challenge lies in incorporating the necessary knowledge (e.g. sensor data, spatial information, etc) within this reward function in order to achieve optimal performance. Altering the previously implemented reward function in this manner is considered the main contribution of this thesis.

ii **Generalization**
An important aspect to consider is that environments are dynamic and change over time. Even if we develop a perfect algorithm for a certain environment, we have no guarantee it will do well if this environment changes or the algorithm is used within a previously unknown environment. As the goal is to have the robot perform well in unseen environments beyond the performance of the algorithm in the trained environment we need to create an agent that can perform well in the trained environment as well as the unseen environment. Therefore, we will explore some commonly used strategies to improve generalization and observe how this affects the robot's performance both in training and testing environments within the scope of this thesis.

1.3 Research questions

To complete the previously described main goal of the project and overcome the described problems the following research questions have to be answered:

i What are state-of-the-art reward shaping novelties that can be successfully leveraged to aid the exploratory process within the used framework?

ii To what extent can we utilize SLAM in order to improve the reward function and increase the performance of the DDPG approach?

iii To which extent is the algorithm able to generalize in an unseen environment?

iv How does the reinforcement learning approach compare against a frontier based exploration approach?

v To which extent do the previously described theoretical concerns of DQN show in a practical setting and how does it perform against the used DDPG approach?

1.4 Limitations

Training a reinforcement learning in a practical setting would be a time-consuming process due to the handling of the robot. After every reset, the robot needs to be placed on its starting location and this would be very impractical considering the length of the training period. Therefore, an accurate simulation is used to approximate a real-world experience and speed up the learning process. This will give a significant boost to the training performance. Ideally, we would want to transfer the obtained policy during the training process on a real robot, however, due to the covid-19 regulations this can unfortunately not be done within the scope of this thesis.

The focus of this thesis is mainly on the reinforcement learning side of the project, meaning that the SLAM and frontier based exploration algorithms are not the main matter, but merely an instrument used to construct the map and in the latter case a comparison tool.

1.5 Assumptions

To make the proposed approach within this thesis possible, it is assumed herein that the state of the environment can be fully observed by the agent and hence are in an MDP. Although we are not blindfolded by the fact that many conditions might violate this assumption and some of these are also treated within the scope of this thesis.

1.6 Outline

The thesis is organized as follows:

Chapter 2. Background Starts with a brief explanation of the theory behind reinforcement learning (section 2.1) in general and later expands that framework towards poly-gradient methods (section 2.2) and finally the used DDPG approach (section 2.3). It continues with a brief explanation of the used SLAM approach (section 2.4) and a concise explanation of the frontier based exploration algorithm (section 2.5).

Chapter 3. Innovations in the reward function Within this chapter, possible innovations within the state vector from state of the art research is summarized. In section 3.1, the methodological approach for the used literature review is explained. The state-of-the-art research found with the literature review is summarized in section 3.2

Chapter 4. Method Details the methodology, used to answer main research questions. It begins by defining the used framework (section 4.1) and then proceeds with justifying the proposed reward function structures (section 4.2). Furthermore, the performed experiments and used environments are detailed (section 4.6), after an exhaustive treatment of the experimental setup (section 4.5) containing the simulation environment (section 4.5.1) and the used network architecture (section 4.5.2). The most important hyperparameters are also clarified within this section.

Chapter 5. Results The results of the most important experiments are evaluated, discussed, and analyzed. These include a comparison of the different reward structures (section 5.1), testing the generalization capabilities of these structures (5.2), a comparison with a DQN/DRQN algorithm (section 5.3) and an experiment parameter tuning (5.4).

Chapter 6. Conclusion This thesis ends with the conclusion to answer the separate research questions (section 6.1) and debates possible future research avenues and adaptation in section 6.2.

Appendices. Appendix A: Raw map-completeness data shows the raw map-completeness over time of the proposed approaches when training in the Apartment environment, an exhaustive list of used hyper-parameters is depicted in Appendix B: Hyper-parameters, Appendix C: Additional reinforcement learning architectures presents supplementary information of the state-of-the-art research section, and additional information to the neural network architecture is shown in Appendix D: Supplementary information of the neural network architecture. Lastly, Appendix E: Structured Literature review shows the literature review process for obtaining innovations in the reward structure.

2 Background

This chapter starts a brief introduction to reinforcement learning[1]. It then continues with the (mathematical) theory behind the used DDPG algorithm. Furthermore, the SLAM algorithm, used to build a map, will be discussed in section 2.4. Lastly, a brief explanation of frontier based exploration, used as comparison within this thesis, is given.

2.1 Reinforcement learning

In this thesis, we consider a standard reinforcement learning framework where we have an agent interacting with the environment. These interactions of the agent happen at discrete points in time t. At every timestep t the agent makes an observation $o_t \in \mathcal{O}$ of the environment, performs an action $a_t \in \mathcal{A}$ and receives a reward $r(o_t, a_t) \in \mathcal{R}$, as can be seen in Figure 4.1. This process is repeated until a terminal condition is reached and the episode ends. In the context of this thesis, the agent can be viewed as a controller with as inputs the sensory information of the robot and the information provided by the SLAM algorithm and as output the desired linear and angular velocity, which will drive the actuators of the robot. The goal of reinforcement learning is to learn an "optimal" behavior policy that maximizes the expected sum of rewards encountered during an episode. A key aspect is that environmental dynamics can be described as an MDP, which is a tuple consisting of all states, actions, a state transition model which describes how the environment changes given action a is performed from state s and, a reward model which yields the reward after action a is taken (Lillicrap et al. (2015)). Some formulations add the discount factor γ which stresses the importance of immediate rewards versus future reward (Silver (2015a)).

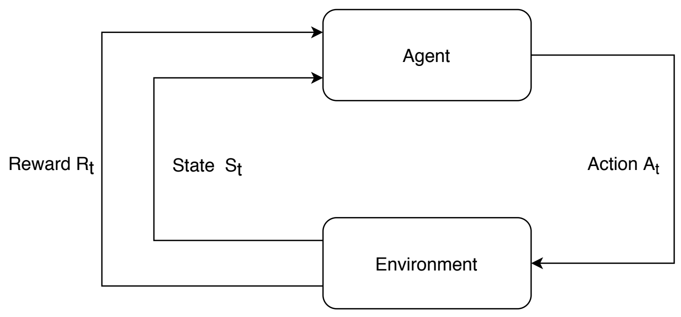


Figure 2.1: Reinforcement learning framework

There are many reinforcement learning algorithms that can be utilized to come to an (optimal) policy, however, we will only discuss those which are applicable within the context of the thesis. The well-known Q-learning algorithm (Watkins and Dayan (1992)) will therefore serve as a basis for both the DDPG and DRQN algorithm, explained later in this chapter. The Q-learning algorithm is a temporal-difference (TD) learning method which updates estimates based on previously learned estimates, without waiting for an outcome of the episode (bootstrapping). Furthermore, it is an off-policy algorithm meaning that it follows a behavior policy (sometimes referred to as exploration policy) with the intent of learning about a different target policy. The one-step Q-learning algorithm works as follows: at every step, the learned action-value function $Q(s, a)$ directly approximates the optimal value function Q^* and the difference between the estimate and the actual value is used to update the estimate:

$$\mathcal{Q}^\pi(s_t, a_t) \leftarrow \mathcal{Q}^\pi(s_t, a_t) + \alpha\{r_{t+1} + \gamma \max_a \mathcal{Q}^\pi(s_{t+1}, a_{t+1}) - \mathcal{Q}^\pi(s_t, a_t)\} \tag{2.1}$$

[1]For the basics of reinforcement learning the author refers the reader to Sutton and Barto (1998) and to Bishop (2009) for the basics of neural networks and machine learning

Where α determines how much the estimate should be updated towards the difference and the action-value function $Q^{\pi}(s,a)$ depicts how good it is to be in a particular state s, taking action a and thereafter following behavior policy π:

$$\mathcal{Q}^{\pi}(s,a) = E_{\pi}[\sum_{t=0}^{T} \gamma^{t-1} R_t | S_t = s, A_t = a] \tag{2.2}$$

A popular choice for exploration is a $\epsilon -$ greedy policy (or a derivative thereof) where the algorithm selects a random action with probability ϵ and acts greedily with respect to the action-value function with probability $1-\epsilon$:

$$a_t = \begin{cases} \text{random} & \text{with probability } \epsilon \\ \arg\max_a \mathcal{Q}^{\pi}(s_t, a_t) & \text{with probability } 1-\epsilon \end{cases} \tag{2.3}$$

The full Q-learning algorithm is then as follows:

```
Result: Q-learning algorithm as adapted from Sutton and Barto (1998)
Initialize $Q(s,a)$ at random
for all $\mathcal{N}^{episodes}$ do
    get initial observation $s_0$
    for $\mathcal{N}^{steps} \in \mathcal{N}^{episodes}$ do
        Select action $a_t$ sampled from behaviour policy (e.g. $\epsilon - greedy$)
        Execute action $a_t$
        Observe transition $\{a_t, s_t, r_t, s_t\}$
        Update the Q-values:
            $\mathcal{Q}^{\pi}(s_t,a_t) \leftarrow \mathcal{Q}^{\pi}(s_t,a_t) + \alpha\{r_{t+1} + \gamma \max_a \mathcal{Q}^{\pi}(s_{t+1},a_{t+1}) - \mathcal{Q}^{\pi}(s_t,a_t)\}$
    end
end
```

It is important to note that this method uses a tabular lookup table to represent the Q-values for every state-action pair. This is quite unfeasible for larger environments as the state-space will blow up and the amount of memory required equally so. Furthermore, there are many (practical) limitations to this approach which makes this method unfeasible. For example, as the table grows so does the amount of data that we have to loop trough, which is computationally expensive and undesirable. Additionally, the often employed $\epsilon - greedy$ exploration, which is a contextual multi-armed bandit problem, is only an approximate solution for this problem. It will suffer linear regret (the difference in action-value between taking an action - e.g. random - and the optimal one).

2.2 Policy gradient methods

In the reinforcement learning paradigm there a two types of ideas on how to select your actions: **value-based methods**, which explicitly approximate how much return you are going to get by taking a certain trajectory with action a_t from state s_t (e.g. 1000 reward for turning left and 850 for turning right from state s_t). Then, by parameterizing the value or action-value function (e.g. acting greedily) the agent knows how to behave in a certain environment. However, this is computationally costly for large action spaces and impossible for continuous action spaces. Secondly, there are **policy-based methods** which directly parameterize the policy $\pi_\theta(a|s) = \mathbb{P}(a_t|s_t,\theta)$. We consider policy-gradient methods that seek to maximize the performance of some scalar performance measure $\mathcal{J}(\pi_\theta)$. Most of the policy gradient methods, as their name implies, update their policy using the stochastic estimate whose expectation approximates the gradient of the performance measure, as explained in Sutton and Barto (1998). It then follows that the policy parameters θ can be updated such that:

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla \mathcal{J}(\theta_t)} \tag{2.4}$$

Given the following objective function $\mathcal{J}(\theta) = \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) \mathcal{Q}^{\pi_\theta}(s,a)$ we can find the gradient of the objective function:

$$\begin{aligned}\nabla_\theta \mathcal{J}(\pi_\theta) &\propto \nabla_\theta \sum_{s\in\mathcal{S}} \lim_{t\to\infty} P_r(s_t = s|s_0, \pi_\theta) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) \mathcal{Q}^{\pi_\theta}(s,a) \\ &= \nabla_\theta \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) \mathcal{Q}^{\pi_\theta}(s,a) \\ &= \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \mathcal{Q}^{\pi_\theta}(s,a) \\ &= \mathbb{E}_{s\sim d^{\pi_\theta}, a\sim\pi_\theta}[\underbrace{\nabla_\theta \log \pi_\theta(a|s)}_{\text{score function}} \mathcal{Q}^{\pi_\theta}(s,a)] \end{aligned} \tag{2.5}$$

Where $d^\pi(s) = \lim_{t\to\infty} P_r(s_t = s|s_0, \pi_\theta)$ is the stationary distribution of Markov chain for π_θ, $\mathcal{Q}^\pi(s_t, a_t)$ the action-value function. The key take-away is that the derivative of the expected reward is the expectation of the product of the reward and gradient of the log of the policy, also known as the **policy gradient theorem** Sutton et al. (1999b).Furthermore, the authors of Schulman et al. (2015) show that this result can be generalized for several expected reward functions including but not limited to the action-value function and value function $\mathcal{V}^\pi(s_t)$.

Most importantly, policy-based methods can operate in high-dimensional or continuous action spaces and have better convergence properties than value-based methods. The usage of policy gradient algorithms is widely used in the field of robotics since they allow for reinforcement learning in continuous action environments. They do, however, tend to converge to local optima for non-tabular environments, and evaluating them is typically inefficient and high in variance Silver (2015b). This is the case because the computed gradient depends on the randomly sampled trajectories. There are also methods that combine the ideas of policy approximation with value-function approximation called actor-critic methods, which is discussed in the next sections.

2.3 Actor-critic methods

Actor-critic methods combine the ideas of value-based and policy-based methods by following an approximate policy gradient. They estimate the action-value function using a critic $\mathcal{Q}^w(s,a) \approx \mathcal{Q}^\pi(s,a)$ with parameters w and use an actor to update the behaviour policy with parameters θ in the direction suggested by the critic. Compared to "vanilla" policy gradient method the objective function change to $\mathcal{J}(\pi_\theta) = \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) \mathcal{Q}^w(s,a)$ and the policy gradient from Equation 2.5 changes as follows:

$$\nabla_\theta \mathcal{J}(\pi_\theta) \approx \mathbb{E}_{s\sim d^{\pi_\theta}, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) \mathcal{Q}^w(s,a)] \tag{2.6}$$

This reduces variance by bootstrapping (Updating the value estimate of a given state depending on the estimated value of successive states). This does, however, introduce bias and therefore might not find the right solution Silver (2015b),Sutton and Barto (1998). For example, consider the aliased (grey squares) grid world example in Figure 2.2, where the objective is to get to the treasure and avoid the skulls. If we would use features $Q^w(s,a) = f(\phi(s,a), w)$ to describe where we are in the environment (e.g. $\phi(s_t = \text{wall to the north}, a_t = \text{moving east})$ and using an optimal deterministic policy, we can either move west or east in both grey states, because they cannot be distinguished. In such cases, where state aliasing occurs, the agent cannot observe the true environment state and we lose the Markov property.

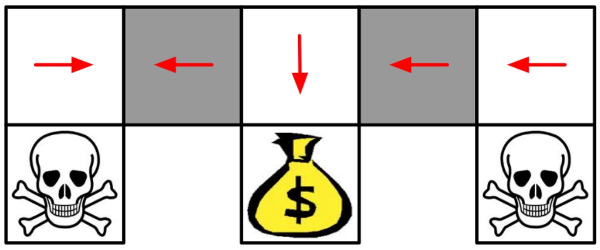


Figure 2.2: Aliased gridworld example with deterministic policy adapted from Silver (2015b)

2.3.1 DPG

Until now, we only considered stochastic policies $\pi_\theta(a|s)$ conditioned on parameter vector θ. However, these require integration of the whole state and action space, which require a large number of samples. Silver et al. (2014) introduce a **deterministic policy gradient** (DPG) method, as a special case of the stochastic one where $\sigma^2 = 0$, to compute the gradient more efficiently. The objective function, given a deterministic policy $a = \mu_0(s)$, can be described as $\mathcal{J}(\mu_{\theta)} = \int_{\mathcal{S}} d^\pi(s)\mathcal{Q}^\mu(s, \mu_0(s))\, ds$. The policy gradient is then calculated as follows:

$$\begin{aligned}\nabla_\theta \mathcal{J}(\mu_\theta) &= \int_{\mathcal{S}} d^\mu(s)\nabla_\theta\mu_\theta(s)\nabla_a\mathcal{Q}^\mu(s,a)\, ds \\ &= \mathbb{E}_{s\sim d^\mu}[\nabla_\theta\mu_\theta(s)\nabla_a\mathcal{Q}^\mu(s,a)] \end{aligned} \tag{2.7}$$

In contrary to the previous methods, the DPG algorithm can in principle be deployed for both on and off-policy learning. However, the authors mention that it is hard to introduce sufficient exploration for the on-policy case, but could potentially be used in environments with inherent noise to provide exploration. Instead one could use a stochastic behaviour policy $\pi(a, s)$ to learn about a deterministic target policy $\mu_\theta(s)$. This way, suitable exploration levels can still be ensured and the algorithm can still benefit from the efficiency of the policy gradient theorem. The off-policy policy gradient seen in Equation 2.7 changes since trajectories are sampled according to a behavior policy $\beta(a|s)$:

$$\begin{aligned}\nabla_\theta \mathcal{J}(\mu_\theta) &\approx \int_{\mathcal{S}} d^\mu(s)\nabla_\theta\mu_\theta(a|s)\nabla_a\mathcal{Q}^\mu(s,a)\, ds \\ &= \mathbb{E}_{s\sim d^\beta}[\nabla_\theta\mu_\theta(s)\nabla_a\mathcal{Q}^\mu(s,a)] \end{aligned} \tag{2.8}$$

Where $\mathcal{Q}^\mu(s, a)$ is substituted for the estimate of the critic $\mathcal{Q}^w(s, a)$ and is approximated using the Q-learning approach described in section 2.1. The parameters w and θ of the critic and actor network, respectively, can then be updated as follows:

$$w_{t+1} = w_t + \alpha_w\delta_t\nabla_w\mathcal{Q}^w(s_t, a_t) \tag{2.9}$$

$$\theta_{t+1} = \theta_t + \nabla_\theta\mu_\theta(s_t)\nabla_a\mathcal{Q}^w(s_t, a_t) \tag{2.10}$$

$$\delta_t = r_t + \gamma\mathcal{Q}^w(s_{t+1}, \mu_0(s_{t+1})) - \mathcal{Q}^w(s_t, a_t) \tag{2.11}$$

Where δ_t is known as the Temporal-Difference (TD) error. The results surpass that of the stochastic equivalent, however, is still limited to relatively simple environments due to linear function approximaters.

2.3.2 DDPG

Deep deterministic policy gradient (DDPG) extends DPG by using non-linear function approximators for the action-value function. It is well-known that this means convergence is no longer guaranteed, but the authors of Lillicrap et al. (2015) show that the use of a replay buffer and target networks, similar to the ones used in DQN (Mnih et al. (2013), Mnih et al. (2015)), stabilizes the learning process.

Replay buffer

The problem with most non-linear function approximators (e.g. neural networks) is that they assume samples that are i)independently and ii) identically distributed. Since the first clause is obviously violated whenever sequential data is used the assumption no longer holds. In order to alleviate this problem a replay buffer is used to break up the correlation between the samples and the agents experience $\{s_t, a_t, r_t, s_{t+1}\}$ at every time-step is stored in a memory $\mathcal{M}$ with finite size. For efficiency, several experiences (mini batch) are uniformly drawn from the buffer and used to update the critic and actor.

Target networks

It is a well-known fact that directly implementing Q-learning with neural networks proved to be unstable in many environments, because the same network $\mathcal{Q}(s, a|\theta^c)$ is used for both the network update and the target y_t (Mnih et al. (2015)).

$$L(\theta^c) = \mathbb{E}_{s\sim d^{\pi_\beta}, a\sim\pi_\beta}[(\overbrace{\mathcal{Q}(s_t, a_t|\theta^c) - \underbrace{r(s_t, a_t) + \gamma\mathcal{Q}(a_{t+1}, s_{t+1}|\theta^c)}_{\text{TD target} y_t})}^{\text{TD error} \delta_t})^2] \tag{2.12}$$

With critic network parameters θ^c. The authors introduce slower updating target networks $Q^{'}(s, a|\theta^{c^{'}})$ and $\mu^{'}(s|\theta^{p^{'}})$ in addition to the original critic and actor network $Q(s, a|\theta^c)$ and $\mu(s|\theta^p)$ to limit the variance in the target network. Using objective function $\mathcal{J}(\mu_{\theta^p}) = \int_{\mathcal{S}} d^{\pi}(s)\mathcal{Q}(s, \mu_0(s)|\theta^c)\, ds$ we can calculate the policy gradient:

$$\begin{aligned} \nabla_\theta \mathcal{J}(\mu_\theta) &\approx \int_{\mathcal{S}} d^{\mu}(s)\nabla_{\theta^p}\mu_\theta(s|\theta^p)\nabla_a \mathcal{Q}(s, a|\theta^c)\, ds \\ &= \mathbb{E}_{s\sim d^{\beta}}[\nabla_{\theta^p}\mu_\theta(s|\theta^p)\nabla_a\mathcal{Q}(s, a|\theta^c)] \end{aligned} \tag{2.13}$$

The four networks can then be updated using:

$$\theta^c_{t+1} = \theta^c_t + \alpha_{\theta^c}\delta_t\nabla_{\theta^c}\mathcal{Q}(s_t, a_t|\theta^c) \tag{2.14}$$

$$\theta^{c^{'}} = \theta^c + (1-\tau)\theta^{c^{'}} \tag{2.15}$$

$$\theta^p_{t+1} = \theta^p_t + \alpha_{\theta^p}\delta_t\nabla_{\theta^p}\mathcal{J}(\mu_\theta) \tag{2.16}$$

$$\theta^{p^{'}} = \theta^p + (1-\tau)\theta^{p^{'}} \tag{2.17}$$

With $\tau \in [0, 1]$. If $\tau = 1$ we recover the original network again and if $\tau = 0$ we will never update towards the original network. The authors recommend $\tau << 1$ to stabilize the learning process and slowly track the networks at the cost of a slower learning process. Since DDPG is an off-policy algorithm it allows for a stochastic behavior policy, whilst still learning about a deterministic target policy. Lillicrap et al. (2015) added serial correlated Gaussian noise, called Ornstein-Uhlenbeck noise Uhlenbeck and Ornstein (1930a), to the sampled action such that:

$$\mu^{'}(s_t) = \mu(s_t|\theta^p_t + \mathcal{N}_t) \tag{2.18}$$

This will ensure continual exploration where the next noise sample $\mathcal{N}_{t+1}$ is somewhat correlated to the previous noise sample $\mathcal{N}_t$. This is very useful for applications where the action directly corresponds to a physical process such as the velocity of a motor since it will not be so inconsistent. The full DDPG algorithm is then as follows:

2.4 SLAM

Building a map of an unknown environment is the objective of this thesis. This section will focus on the method used for tackling this problem. Furthermore, a state of the art active SLAM method (frontier based exploration) is introduced. The achieved results in this thesis are compared against this method.

```
Result: DDPG algorithm as adapted from Lillicrap et al. (2015)
Initialize actor $\mu(s|\theta^p)$ and actor target network $\mu^{'}(s|\theta^{p^{'}})$ at random
Initialize critic $Q(s,a|\theta^c)$ and critic target network $Q^{'}(s,a|\theta^{c^{'}})$ at random
initialize replay buffer $\mathcal{M}$ with size $\mathcal{N}$
for all $\mathcal{N}^{episodes}$ do
    get initial observation $s_0$
    for $\mathcal{N}^{steps} \in \mathcal{N}^{episodes}$ do
        Select action $a_t$ sampled from behaviour policy $\mu(s|\theta^p)$
        Execute action with exploration noise $a_t + \mathcal{N}$
        Observe transition $\{a_t, s_t, r_t, s_t\}$ and store in $\mathcal{M}$
        Sample $N$ transitions from memory $\mathcal{M}$
        Update the critic network by minimizing the loss:
            $L(\theta^c) = \frac{1}{N}\sum_i [(r(s_i,a_i) + \gamma \underbrace{\mathcal{Q}^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{p^{'}})|\theta^{c^{'}})}_{\text{slowly updating target network}} - \mathcal{Q}(s_i,a_i|\theta^c))^2]$
        Update the actor network using the policy gradient:
            $\nabla_\theta \mathcal{J}(\mu_\theta) \approx \frac{1}{N}\sum_i [\nabla_{\theta^p}\mu_{\theta^p}(s_i|\theta^p)\nabla_a \mathcal{Q}(s_i,a_i|\theta^c)]$
        Update the target critic network using:
            $\theta^{c^{'}} = \theta^c + (1-\tau)\theta^{c^{'}}$
        Update the target actor network using:
            $\theta^{p^{'}} = \theta^p + (1-\tau)\theta^{p^{'}}$
    end
end
```

SLAM methods attempt to construct a map of the environment, whilst estimating the pose of a robot at the same time. This poses a computational problem since a suitable map is required to localize properly and a good pose estimate in necessary for constructed a proper map. The map can be a topological (e.g. A map of the city) or metric representation (e.g. landmark-based map) of the environment, but in robotics, we generally consider 2-D/3-D grid-maps. A large variety of SLAM methods exist in literature to tackle this problem (Dissanayake et al. (2000); Montemerlo et al. (2003); Thrun (2000)), but in this thesis, we will utilize an RBPF-SLAM approach, as introduced by Murphy (2000) and Doucet et al. (2013). It uses a probabilistic approach and formally the objective can be described as estimating the pose x and map m of a mobile robot, given its observations z and movements u:

$$p(x_{1:t}, m|z_{1:t}, u_{0:t-1}) \tag{2.19}$$

Using the chain rule of probability we can factor it into a product of simpler distributions Murphy (2000):

$$p(x_{1:t}, m|z_{1:t}, u_{0:t-1}) = \underbrace{p(x_{1:t}|z_{1:t}, u_{0:t-1})}_{\text{localization}} \cdot p(m|x_{1:t}, z_{1:t}) \tag{2.20}$$

The key idea of RBPF-SLAM is that we can update the posterior of the map $p(m|x_{1:t}, z_{1:t})$ analytically and then represent potential trajectories of the robot $p(x_{1:t}|z_{1:t}, u_{0:t-1})$ using a particle filter. Where each particle j carries an individual map, build by observations $z_{1:t}$ and trajectory $x_{1:t}$. For each particle we then sample from proposal distribution $x_t^j \sim \pi(x_t|z_{1:t}, u_{0:t})$, assign a weight w to each particle, resample and estimate the map. The proposal distributions are typically approximated using the odometry motion model $p(x_{t+1}^j|x_t, u_t)$ since the closed form of this posterior is not accessible . However, the authors of Grisetti et al. (2005) claim this is not optimal if an accurate laser ray finder is used as observation. They propose to include the most recent sensor observation z_t in the proposal $x_t^j \sim \pi(x_t|x_{t-1}^j, z_t u_{t-1})$. Then weights can be assigned to each particle as:

$$w^j = \frac{p(x_{1:t}^j|z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^j|z_{1:t}, u_{1:t-1})} \tag{2.21}$$

They make up for the fact the proposal is presumably not identical to the target distribution. Since we are attempting to represent a continuous target distribution with a finite number of samples it is necessary to replace poor quality samples with higher quality samples. Generally, weights with low values are replaced by weights with higher values, however, this can lead to problems such as particle depletion Merwe et al. (2001). One intuitive solution to determine whether resampling is required is to check the variance of the weights. If there is low variance amongst the weights, it is likely that the samples are close to the true posterior and hence no resampling is required. Similarly, higher variance indicates that the sample set is a bad approximation of the true posterior. The authors of Liu (1996) propose the following metric to determine how well the sample set resembles the true posterior:

$$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N}(w^j)^2} \tag{2.22}$$

The resampling procedure can then prompted if the value of N_{eff} drops below a certain threshold. In Grisetti et al. (2005) they consider the value of $N/2$ to be an adequate threshold for initializing the resampling procedure. This procedure of resampling will allow the particle filer to converge on the "true" pose estimate, which can then be used to estimate the posterior of the map $p(m|x_{1:t}, z_{1:t})$. We can then represent the environment as an evenly spaced grid, with each cell of the grid indicating whether or not the cell is occupied, unoccupied or unknown. This concept is known as occupancy grid mapping and was first introduced by Moravec and Elfes (1985). Under the assumption that occupancy of each cell is independent we can represent each grid cell m_i individually:

$$p(m_i) = \begin{cases} 1, & \text{if occupied} \\ 0, & \text{if unoccupied} \\ 0.5, & \text{if unknown} \end{cases} \tag{2.23}$$

The probability distribution of the entire map is then the product of the individual cells:

$$p(m|x_{1:t}, z_{1:t}) = \prod_{i \in M} p(m_i|x_{1:t}, z_{1:t}) \tag{2.24}$$

Where M is the total number of grid cells present in the map. The resolution of the map should be selected such that the smallest obstacle can be accurately represented by a grid cell.

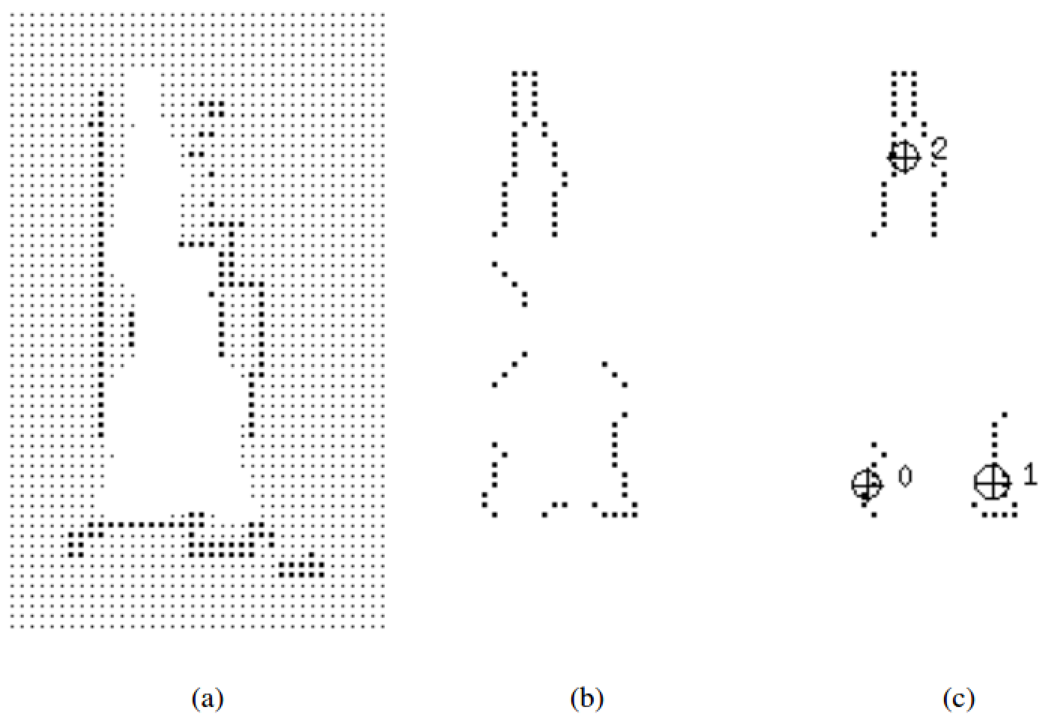


Figure 2.3: Frontier selection procedure with **a)** the occupancy grid map (size of dot indicates probability of occupation), **b)** frontier cells, **c)** frontier areas larger than the robots size. Adapted from Yamauchi (1997)

2.5 Frontier based exploration

Frontier based exploration, as already briefly mentioned in the introduction, is an active SLAM method, which attempts to guide the robot to the boundaries between the explored and unexplored areas of the map. In the case of an occupancy grid map, these are the boundaries between the known cells (unoccupied and occupied) and unknown cells of the grid map (not yet captured by the sensors of the robot or not yet processed). The problem frontier based exploration tries to solve is twofold:(i) deciding which of the (possible) multiple frontiers to navigate to in order to uncover the largest portion of the map and (ii) figuring out a collision-free path in order to reach that target within a reasonable amount of time. There are many frontier based exploration algorithms that have different strategies to solve the aforementioned problems such as random frontier selection, decision-theoretic selection of frontiers, and map segmentation based frontier selection. However, the rather naive approach of simply selecting the closest frontier, as first introduced by Yamauchi (1997), works surprisingly well and is able to compete with more sophisticated approaches whilst being computationally efficient Holz et al. (2010). The algorithm works as follows: any cell next to an unknown cell is considered a frontier cell. Bordering frontier cells are grouped together into frontier areas and if these areas are larger then the robot itself it is considered a frontier, as is depicted in Figure 2.3. Once a frontier has been detected it will determine the frontier which is closest to the robot:

$$\text{closest frontier} = \arg\min_{c \in F} P((x\ y)^T, \text{position robot}) \tag{2.25}$$

Where F denotes the set of found frontiers and $P((x\ y)$ the length of the shortest path from the robot to the frontier. This is typically done using a path planner, starting at the robot's current cell and utilizing a to traverse to the frontier cell. If the target cell is reached, the frontier is added to a list of visited frontiers and the next closest frontier is selected as a target. This procedure will repeat until there are no more frontiers remaining to visit. If for some reason a frontier cannot be reached within in some period of time it is marked as inaccessible and discarded from the frontier list.

2.6 Summary

In this chapter, reinforcement learning is introduced alongside the fundamental building blocks for the reinforcement learning architecture used within the scope of this thesis was discussed. Furthermore, the used SLAM and frontier based exploration algorithms are explained on a conceptual level.

3 Innovations in the reward function

In this Chapter, reinforcement learning techniques which improve the agent's ability to explore the environment and thereby make sure reinforcement learning is able to solve the underlying MDP are discussed. The intent is to use some of the innovations found in literature in order to improve the reward function. This Chapter contains a review of the current literature following the methodological approach described in Appendix E.

The balance of exploration versus exploitation is one of the fundamental problems of reinforcement learning. The conundrum is trying something new, which might lead to higher long-term benefits or doing something you know will give a reward, but you might lose out on potential higher rewards that are unknown to you. Hence, exploration increases the knowledge of the whole state space and long-term gain, at the expense of some short-term gain. Whilst exploitation leverages the current knowledge of the state space for short term gain. For example, visiting your favorite restaurant, which you know to have decent food, versus visiting a new restaurant, which might have even nicer food. It is important is to gather enough information about the state-space in order to make the best overall decision. The main purpose of exploration is to ensure that the agent's behavior does not converge prematurely to a local optimum. The remainder of this chapter summarizes state-of-the-art strategies to do so within the reinforcement learning paradigm.

3.1 Exploration at a policy level

Incentivizing exploration in reinforcement learning can be done in several ways. One of those ways is adjusting your action selection to make sure some form of exploration is present during the training phase. A well-known method for probabilistic policies is ϵ -greedy, where the agent takes the best action known to the agent from a given state with probability ϵ and a random action with probability $1-\epsilon$. This ensures that over time the whole state space is explored, but will also incur reward loss over time as a random action might not be the optimal one.

For algorithms with continuous action spaces, such as DDPG, adding random noise (e.g additive Gaussian noise) to the action space during the training process is very popular to make them explore better. However, one can imagine that for robotic applications and navigational tasks in particular this can lead to undesirable chattering of the robot. For this reason, the use of correlated noise or colored noise is very attractive in the field of robotics (**?**). Correlated noise is a stochastic process wherein values tend to be correlated with other values nearby in space or time. This leads to smoother transitions, which is necessary for better exploration when considering that the action space generally consists of velocities of the motors or derivatives thereof. Therefore, the norm in the field is to use correlated additive Gaussian action space noise (Ornstein–Uhlenbeck process Uhlenbeck and Ornstein (1930b)). When using some type of Gaussian action noise with mean μ and standard deviation σ^2, actions are sampled according to some policy π. This results in an action according to $a_t = \pi(s_t) + \mathcal{N}(\mu, \sigma^2 I)$. Therefore, it can be the case that one would yield a completely different action for every time that state is sampled. This can be undesirable and the authors of Plappert et al. (2017) suggest replacing action space noise with so-called parameter space noise. Parameter noise adds adaptive noise to the parameters of the neural network policy instead of the action space, as can be observed in Figure 3.1.

The parameter vector of the current policy is pertubed by additive Gaussian noise, such that:

$$\hat{\theta} = \theta + \mathcal{N}(0, \sigma^2 I) \tag{3.1}$$

With policy parameters θ and pertubed policy parameters $\hat{\theta}$. It is demonstrated that both off-policy and on-policy methods can benefit from this approach and is particularly easy to adapt

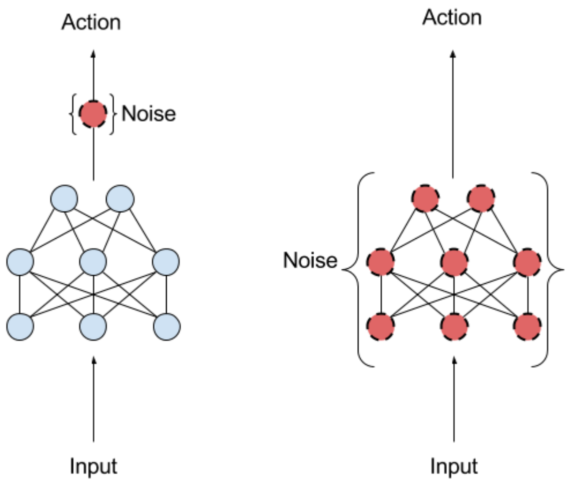


Figure 3.1: Action space noise (left network) versus parameter space noise (right network) adapted from Plappert et al. (2017)

for off-policy methods such as DDPG. Furthermore, experimental results show increased performance compared to traditional Ornstein–Uhlenbeck -approaches, particularly in environments with sparse or very sparse rewards. Despite the theoretical advantages of parameter space noise, Mustafa (2019) argue that it does not always work in practice due to sensitivity of the neural network to parametric perturbations.

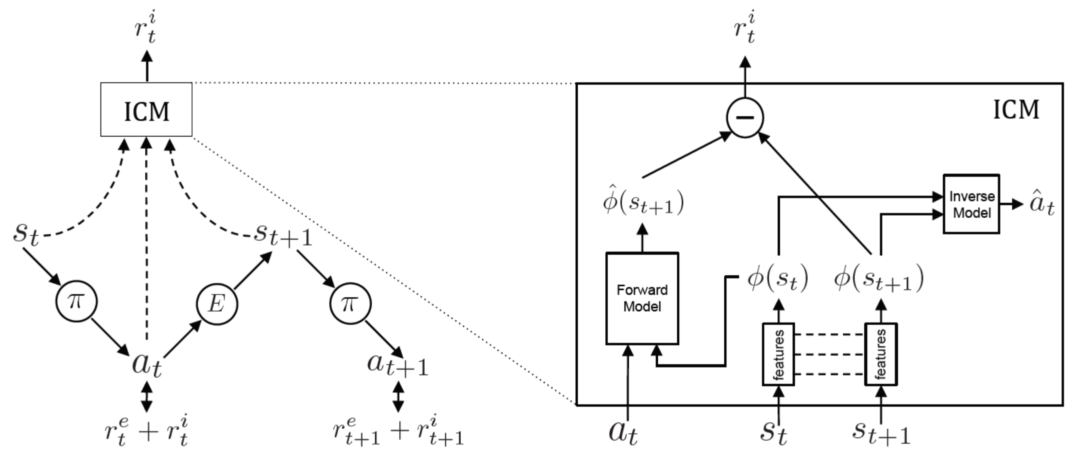


Figure 3.2: ICM module, adapted from Pathak et al. (2017)

3.2 Exploring by reward shaping

Curiosity

Expecting to navigate to a goal or map of an environment successfully with a semi-random exploratory process is likely to be fruitless in all but the simplest environments. In any environment with a sparse reward setting or with a complex task, it is important to encourage the exploration of the state-space. Adding some form of intrinsic reward in addition to the extrinsic (environmental) reward can help the agent to explore "novel" states. Furthermore, it can help the agent reduce the uncertainty in the transition dynamics of the environment by encouraging actions that are uncertain to the agent. In order to achieve this, the authors of Pathak et al. (2017) propose an intrinsic reward signal based on the prediction error of the agent's uncertainty about its environment. They determine the prediction error by calculating the difference between a predicted action $\hat{a}_t = g(s_t, s_{t+1}; \theta_I)$ and the actual action sampled from the policy $a_t \sim \pi(s_t; \theta_P)$, such that:

$$\min_{\theta_I} L_I(\hat{a}_t, a_t) \tag{3.2}$$

With learned function g (also known as inverse dynamics model), loss function L_I and neural network parameters θ_I and θ_P, respectively. They also stress the importance of not encoding the raw states (e.g. laser data, pixels of a camera, etc.) directly because this would likely lead to undesired behavior if the environment dynamics are hard to model. They propose to train another neural network that takes as input a feature vector $\phi(s_t)$ in addition to the action a_t to predict the feature encoding at the next timestep $\hat{\phi}(s_{t+1}) = f(\phi(s_t), a_t; \theta_F)$, by minimizing the loss function L_F:

$$L_F(\phi(s_t), \hat{\phi}(s_{t+1})) = \frac{1}{2}||\hat{\phi}(s_{t+1}) - \phi(s_{t+1})||_2^2 \tag{3.3}$$

Finally, the intrinsic reward r_t^i can be calculated as $r_t^i = \eta \; L_F$ for some scaling factor $\eta > 0$, as shown in Figure 3.2. The major limitation of this method as hinted to before are stochastic dynamics in the environment. If the dynamics are truly random no model can predict the environment transistions properly. Even if they are not truly random, the agent will search for transitions with the highest prediction error, which can be problematic for sub-optimal learning algorithms and POMDP problems. For example, Burda et al. (2018) showed that an agent, using an ICM approach, preferred watching a noisy TV placed in an environment over exploring the rest of the state-space. The authors of Savinov et al. (2018) propose the concept of novelty trough reachability to overcome the aforementioned issues. The approach uses an episodic memory to store past observations and compare them against the current observation of the agent in order to determine the novelty of the observation. To be exact, they use a neural network approximator to determine if the current observation can be reached within the k-steps of the stored observation, as shown in Figure 3.3. They propose to calculate the intrinsic reward as follows:

$$r_t^i = \mathcal{B}(M,e) = \alpha(\beta - \mathcal{C}(M,e)) \tag{3.4}$$

Where $\mathcal{C}(M,e)$ is then the similarity score $\in [0,1]$ between the memory in the buffer and what is reachable with the current observation. The hyperparameters $\alpha \in \mathbb{R}^+$ and $\beta \in \mathbb{R}$ should be picked according to the scale of the task. The ICM and episodic curiosity methods show similar performance across the used environments. Moreover, the latter shows significant improvements in terms of convergence rate (2x as fast) and does not suffer from the aforementioned environmental issues.

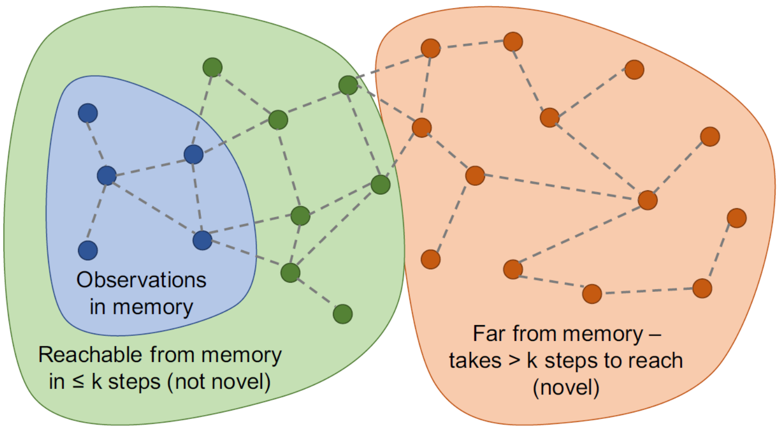

Observations in memory
Reachable from memory in ≤ k steps (not novel)
Far from memory – takes > k steps to reach (novel)

Figure 3.3: k-step reachability, adapted from Savinov et al. (2018)

3.3 Summary

This chapter introduced basic concepts for state space exploration by adjusting both the action space and by tweaking the reward function. The concepts of episodic curiosity and OU-noise are promising concepts and are leveraged in the remainder of this thesis to innovate the reward function.

4 Method

The DDPG algorithm is capable of handling continuous action space, but the exploratory process is a major challenge for DDPG and reinforcement learning in general. We have seen in Chapter 2 that DDPG is capable of exploring an environment through the use of stochastic behaviour policy to learn about a deterministic target policy. As the exploratory process is vital in achieving a good performance and can affect the learning rate, Chapter 3 introduced two ways of aiding the exploratory process: exploring on a policy level and exploring by reward shaping. This chapter presents the used architecture in section 4.1 and then proceeds by explaining the choices for the innovation within the reward function definition in section 4.2. To that end, four different reward structures are introduced within that section. The details of the simulation, neural network architecture and the performed experiments are explained in the later sections of this Chapter.

4.1 Framework

The main purpose of this research is to design a reinforcement learning algorithm capable of mapping known and unknown indoor environments with a flat surface using its on-board sensors. To achieve this goal, the DDPG algorithm, discussed in Chapter 2, is utilized. The state of the agent, S_t, is comprised of the sensory information provided by the robot's onboard sensors (laser data), the pose estimate provided by the SLAM algorithm and the previous action [1]. This is then passed through the actor neural network to obtain the next primitive control action A_t, which consists of the desired angular and linear velocity of the mobile robot.These are then fed through a PID controller to ensure the desired movements are properly executed by the robot's actuators. In a sense the reinforcement learning can be viewed as a setpoint generator for the PID controller loop. When the interaction with the environment is completed, new sensory information is obtained and the cycle continues until the whole area is mapped. To improve the behaviour of the actor over time, the critic network is used to assess the quality of the possible actions given the current state. The network parameters of the actor is then slowly be adjusted in the direction of the gradient of the action-value function, as proposed by the critic. This process is repeated to a satisfactory performance level or optimal policy is achieved. The SLAM algorithm is utilized to obtain a pose estimate of the mobile robot within the map, given the odometry and laser data from the robot (Figure 4.1).

4.2 Reward Shaping

4.2.1 Old reward function

It has been concluded that the previously investigated reward function from Schulte (2019) is not indicative enough of the optimal trajectory through the environment. In the previous work, the reward of the agent was formulated as:

$$R(s_t) = \begin{cases} r_{\text{map completed}}, & \text{if } c_t \geq C \\ r_{\text{crashed}}, & \text{if robot crashed} \\ r_{\text{dense}}, & \text{otherwise} \end{cases} \tag{4.1}$$

Where the reward for completing the map $r_{\text{map completed}}$ was assigned whenever the environment was mapped to a certain threshold $c_t \geq C$. A penalty r_{crashed} was given whenever the robot crashed or came too close to a wall and a reward was assigned, r_{dense}, proportional to the % of the map completed at that given time-step. The latter reward, r_{dense}, turned out to be troublesome as even standing still would grant a reward. Exploring more of the environment would in principle grant more reward compared to standing still, however, if the robot crashed

[1]The sensory information is explained in-depth in Section 4.4

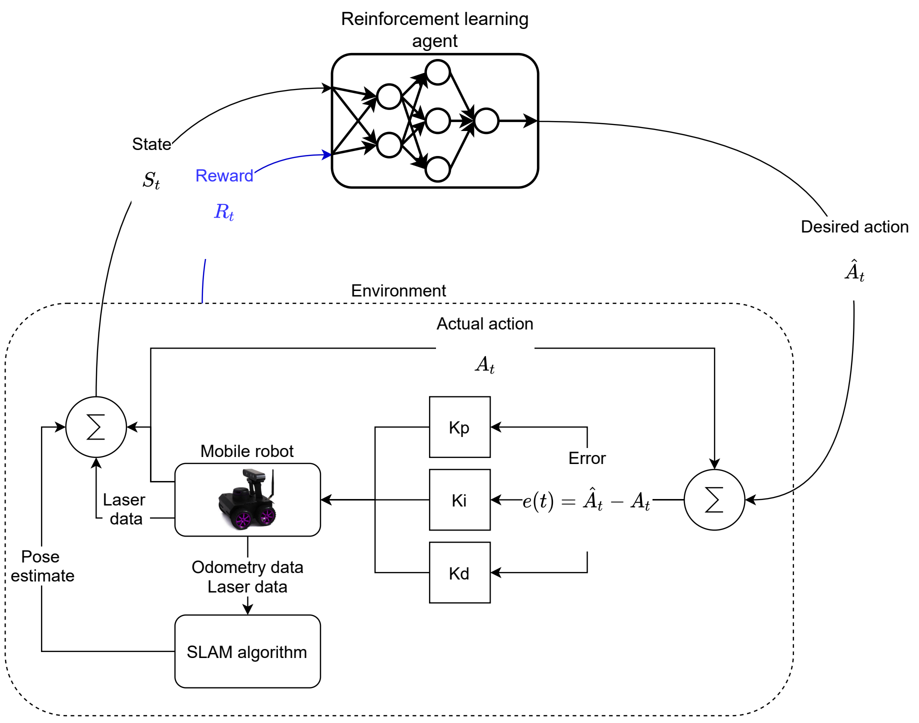


Figure 4.1: The proposed architecture for autonomous mapping and navigation of an unknown environment. The reinforcement learning agent is the high level controller which uses sensory information from the robot and SLAM algorithm to determine its next action. The PID controller then makes sure the desired action is executed by the actuators of the robot.

the penalty would always be larger then the dense bonus. Since the initial room of the environment was rather difficult to escape, the algorithm would learn that driving circles within the room would optimize the amount of reward. This way it would never find the large reward for completing the map and it was stuck in a local optimum. Such behaviour was also shown by (Matheron et al. (2019)) in a simpler 1-D environment. Suggested problems which are known to cause issues are sparse rewards and inefficient exploration. The authors of Matheron et al. (2019) have established the less trivial fact that, even if exploration does find the reward consistently but not early enough, an actor-critic algorithm can get stuck into a configuration from which rewarded samples are just ignored. In this Section, we propose several reward structures and aim to find out which works best for office and apartment-like environments.

4.2.2 Baseline scenario

Since the reward function has a direct impact on the exploratory behaviour of the agent and convergence time of the algorithm, as shown in Chapter 3, we employ four reward structures to assess the impact on the results. The first reward structure was created to evaluate relative performance compared to the starting point of this project. The *Baseline* scenario has a sparse reward setting in the form of a penalty or bonus whenever a terminal state has been reached, such that:

$$R(s_t) = \begin{cases} r_{\text{map completed}}, & \text{if } c_t \geq C \\ r_{\text{crashed}}, & \text{if } l_t \leq l_{\text{min}} \forall l_t \in L \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

For some value C and l_{min}. This reward setting will provide no additional incentive to explore the environment beyond the bonus for completing the map. The hope is that by learning to avoid obstacles it is able to drive through the environment and complete the map. Any improvement of exploratory behaviour or convergence time over this reward setting can in this way directly be contributed to the difference in reward structure.

4.2.3 Oracle scenario

In the second scenario, called *Oracle*, it is assumed that the total area of the environment is known. This way, we can reward the agent for exploring new areas of the map by giving a reward proportional to the newly discovered area of the map in addition to the "sparse" rewards of the *Baseline* scenario. The major difference between this reward and the one found in Equation 4.1 is that the latter incentives standing still and this one does not. This should drive the robot forward to "unknown" areas of the map and greatly increase exploratory behavior compared to the first scenario. The reward function can then be described as follows:

$$R(s_t) = \begin{cases} r_{\text{map completed}}, & \text{if } c_t \geq C \\ r_{\text{crashed}}, & \text{if } l_t \leq l_{\text{min}} \forall l_t \in L \\ c_t - c_{t-1}, & \text{otherwise} \end{cases} \tag{4.3}$$

This reward function has the caveat that it is necessary to define the area of the environment beforehand in order to obtain the map completeness. In a practical setting, an estimation of the size of the area can be obtained from the SLAM algorithm if this is unknown.

4.2.4 Information-gain scenario

The third reward function is a decision-theoretic approach where we take advantage of the map that slam provides. Every cell within the map is assigned a probability value of being occupied, as explained in Chapter 2. High values indicate that the cell is likely occupied by an obstacle and low values indicate that the cell is likely unoccupied. Furthermore, a value of 0.5 indicates that it is not known whether the cell in question is occupied or unoccupied, as show in the following Figure: Using the uncertainty of this posterior Using the uncertainty of this posterior

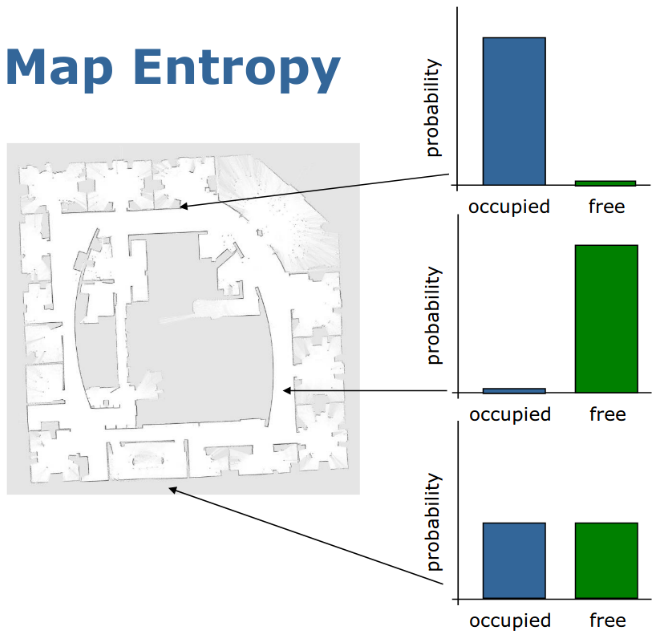


Figure 4.2: Occupied, unoccupied and unknown cells within a constructed map adapted from (Abbeel, 2006)

(Entropy), we can calculate the uncertainty in the map:

$$H(m) = -\sum_{c \in M} p(c) \log(p(c)) + (1 - p(c)) \log(1 - p(c)) \tag{4.4}$$

If a large portion of the map is unknown, the majority of the cells will have a value of $p(c) \approx 0.5$ and $H(m)$ is approximately -1. Controversly, if the map is largely known the term $H(m)$ will approach 0. Hence, the rationale is that by making the reward proportional to the reduced uncertainty of the map from time t to $t+1$ a good exploratory behaviour should develop. This is also known as (expected) *information-gain*:

$$R(s_t) = \begin{cases} r_{\text{map completed}}, & \text{if } c_t \geq C \\ r_{\text{crashed}}, & \text{if } l_t \leq l_{\text{min}} \forall l_t \in L \\ \zeta(H_t - H_{t-1}), & \text{otherwise} \end{cases} \tag{4.5}$$

With some scaling factor ζ.

4.2.5 Curiosity scenario

Lastly, we investigate a "naive", but practical adaptation[2] of the *curiosity* reward function, proposed by Savinov et al. (2018), to improve the exploratory behaviour of the algorithm and simultaneously remove the a-priori information about the map from the reward function. In contrast to the method in Savinov et al. (2018), we can prevent using the complete state and use only that part of the state which is comprised of the location of the robot.

The method works as follows: at the beginning of each episode, the initial position is appended to a novelty memory. At every step during that episode consecutive positions of the robot are compared against all novel positions in memory. We will consider a position "novel", if the distance between the current position s_t and the positions in memory is larger then some distance k (Figure 4.3). If a position is determined to be "novel", the agent receives a bonus and the novel position is added to memory.

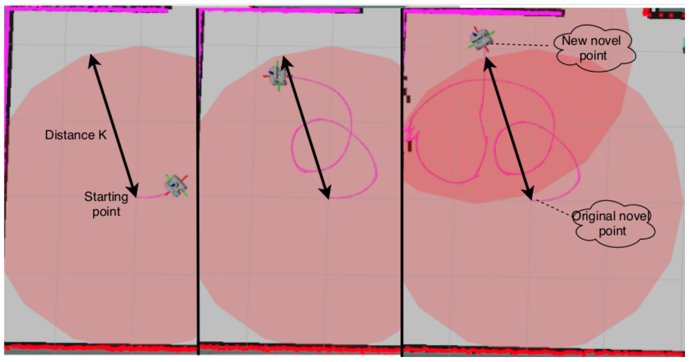


Figure 4.3: Three sequential screenshots of the mobile robot depicting distance K, which imposes the red area as non-novel. In the third screenshot a new novel point is reached and added to the novelty memory, which imposes an additional area as non-novel. The path of the robot is depicted in purple.

This way anything that is quickly reachable by the robot ($< k$) will become uninteresting since no further reward can be obtained from there. Whilst reaching positions further away from the robot ($> k$) are encouraged. This should drive the robot to unexplored areas by using the pose estimate from the SLAM algorithm. Positions within some distance of the walls are not considered as novel positions since this would encourage undesired behaviour. A benefit of

[2] To overcome practical issues mentioned in the paper and on the Github page such as requiring 10M training steps and northwards of 50gb RAM

this is that the robot should have a natural tendency to stay in the middle of the room.

We can increase the exploratory behaviour of the robot further by making the bonus dependent on the average distance to all novel states in memory. This will motivate the robot to drive away from the set of novel states. The "intrinsic" reward function can then be calculated as $r_t^i = \alpha\ d(\mathcal{P}_0, \mathcal{P}_1)$ for some scaling factor α and distance between two points $d(\mathcal{P}_0, \mathcal{P}_1)$. Given an episodic environment with maximum amount of steps $\mathcal{N}^{\text{steps}}$ and $\mathcal{N}^{\text{episodes}}$ amount of episodes, the complete adaptation is as follows:

Algorithm 1: Episodic Curiosity Reward

```
Result: Write here the result
initialize episodic memory buffer $\mathcal{B}$ with size $\mathcal{N}$ ;
for all $\mathcal{N}^{episodes}$ do
    for $\mathcal{N}^{steps} \in \mathcal{N}^{episodes}$ do
        if $d(\mathcal{P}_t, \mathcal{P}) > k\ \forall\ \mathcal{P} \in \mathcal{M}$ then
            $\mathcal{M} \leftarrow \mathcal{P}_t$;
            $r_t^i = \frac{\alpha}{\mathcal{M}} \sum_{\mathcal{P} \in \mathcal{M}} d(\mathcal{P}_t, \mathcal{P})$;
            return $r_t^i$;
        else
            return $r_t^i = 0$ ;
        end
    end
end
```

Where the simplest form to calculate the distance between two points is picked: $d(\mathcal{P}_0, \mathcal{P}_1) = \sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2}$. Intuitively, k should be picked such that the reward is dense (a reward that gives value to most of the transitions), however, picking it too small will have a significant impact on performance since all the novel states are appended too a memory. Picking it too large (e.g. $k = \infty$) will make the reward sparse again providing no additional benefits to exploration. Moreover, α can control the scaling of the bonus and in a sense the urgency with which to drive away from the set of known states. Additionally, the sparse structure of the *Baseline* scenario is maintained and the complete reward is then given as follows:

$$R(s_t) = \begin{cases} r_{\text{map completed}}, & \text{if } c_t \geq C \\ r_{\text{crashed}}, & \text{if } l_t \leq l_{\text{min}} \forall l_t \in L \\ r_t^i, & \text{otherwise} \end{cases} \tag{4.6}$$

4.3 Generalization

As the goal is to have the robot perform well in unseen environments beyond the algorithms performance in the trained environment we need to come up with a good way to test generalization capabilities. In traditional machine learning, one would split training and testing data sets to test performance. However, reinforcement learning generally does not have split training and testing sets. Instead, it is accepted practice to examine performances directly on the training environments (Zhang et al. (2018)). In some circumstances, such as the Atari (Bellemare et al. (2012)) environment, it can make sense since the application environment is the same as the training environment. However, this is not a good way to test the abstraction capabilities of the agent and certainly would not extent well into real-world environments. Therefore, to test the generalization properties of the algorithm we clearly distinguish training environments Θ_{train} from the testing environment Θ_{test} and formalize the robustness to unseen environments as:

$$\eta_{p_{test(\Theta)}}(\mu^*)|[\mu^* \arg\max_{\mu} \eta_{p_{train(\Theta)}}(\mu), p_{train}(\Theta) \neq p_{test}(\Theta)] \tag{4.7}$$

With $\eta_{p_{(\Theta)}}(\mu)$ denoting the expected return under environmental distribution $p\Theta$:

$$\eta_{p_{(\Theta)}}(\mu) = \mathbb{E}_{\Theta \sim p(\Theta)}[\mathbb{E}_{\tau \sim (\mu)}[\sum_{t=0}^{T} \gamma^t r(s_t, \mu(s_t))]] \tag{4.8}$$

This metric is used to compare the ability of the algorithm to generalize in different environments and with different reward settings. Similarly, they are compared on map-completeness as this is the objective of the algorithm.

4.4 State vector design

In this framework, the sensory information will consist of the pose estimate p_t and the laser data l_t, respectively. The former is comprised of the x-y coordinates and the yaw (rotation around the z-axis within the Cartesian coordinate system). The laser data will consist of 720 laser points, equally distributed in a 360-degree scan field obtained from the robot's Lidar system. However, adding such a large number of laser data to the state vector would inadvertently prolong the training time unnecessarily. The main goal of adding the laser data is to detect obstacles, and during the early experiments, it has been found that 80 laser points are sufficient information for the robot to detect them consistently. Consequently, the output is the desired angular and linear velocity of the robot in order to successfully navigate the environment. The state vector will then be the following:

$$s_t = [l_t,\ p_t,\ a_{t-1}] \tag{4.9}$$

The state must be comprised of all information necessary to make an informed decision, which means only the current state s_t in the case of an MDP. However, there are many minor discrepancies in this simulated environment (e.g. between the command velocity and actual velocity of the robot, noisy sensor information etc.) which can make an MDP assumption unstable. Moreover, in a finite horizon MDP, where the problem can be solved within a set number of time-steps T it can be important to make the robot aware of the terminal conditions such as the number of time-steps remaining. The authors of Pardo et al. (2017) showed that in finite horizon problems, time limits should be included in the state vector to avoid state aliasing and violation of the MDP assumption. This can be particularly important if the notion of time is a part of the optimal policy (e.g. find the fastest route from point A to B). Since the running assumption is that we are in an MDP environment, we should include this information in our state vector. In this thesis, the reinforcement learning problem is considered a finite horizon MDP, where the problem can be solved within a set number of time-steps T. In this case, it becomes important to make the algorithm aware of the terminal conditions by appending relevant information to the state vector as explained by Pardo et al. (2017). In our case this means the number of time-steps remaining $T - t$ in any given episode alongside the percentage of the map still to be completed $C - c_t$ should be included in the state vector, such that it becomes:

$$s_t = [l_t,\ p_t,\ a_{t-1},\ C - c_t,\ T - t] \tag{4.10}$$

4.5 Experimental setup

4.5.1 Simulation environment

The open-source Gazebo simulation platform offers dynamics simulation capabilities with the physics engine and realistic rendering of environments. This 3-D engine is used to simulate the robot's dynamics. The algorithm is constructed in Python 2.7 using the "standard" RL package of Tensorflow, Cuda, Cudnn and OpenAI-gym. Tensorflow are used to create the neural network allowing for relatively simple construction of complex networks and integrates easily with Python. The Robot Operating System (ROS), which allows for low-level device control, is used to obtain sensory information (Lidar and odometry) and provide data exchange from python to Gazebo using its parameter server. Additionally, ROS allows for easy integration with SLAM packages and the gmapping package is used to create the map.

4.5.2 Neural network architecture

To obtain the next action, the state vector, described in chapter 4, is passed through a neural network as depicted in Figure 4.4 for the actor-network. A similar network architecture is used in (Mustafa et al., 2019) to successfully navigate to a pre-set point with a mobile robot. The actor-network is comprised of three fully connected dense (fully connected) layers with 512 neurons each. All of the layers are activated with a Rectified Linear Unit (Relu) activation function. Relu activation function is a (piece-wise) linear function that will output the input directly if is positive, otherwise, it will output zero. It can be viewed as a bounded linear function, which simply returns the input if positive:

$$g(x) = \max(0, x) \tag{4.11}$$

For any input x. It has become the staple activation function within the reinforcement learning paradigm resulting in more efficient and better performances. They preserve many of the properties that make linear models easy to optimize with gradient-based methods (e.g. they are differentiable and generalize well). The actor network outputs a 2-dimensional vector comprised of the desired linear and angular velocity of the robot. The outputs are constrained using the (non-linear) hyperbolic tangent or sigmoid activation function. The hyperbolic tangent can be used to transform the input into a value between -1 and 1. Much larger values are transformed to 1 and values much smaller are converted to a -1. Similarly, the sigmoid activation function can be used to constrain the output of the network between 0 and 1. This effectively allows us to control the degrees of freedom of the robot and constraint the linear velocity movement to only forward with the sigmoid activation function. The angular velocity will use the hyperbolic tangent (tanh) to allow turning (around the z-axis) in both directions. The actor's output will directly be imposed on the robot's actuators as can also be observed in Figure 4.4. The laser scanner (LiDAR) of the robot has a field of view of 360 degrees, which allows the robot to effectively map the environment. The LiDAR will have a minimum range of 0.2 and a maximum range of 10m. This will allow efficient exploration within the used training and testing environment .

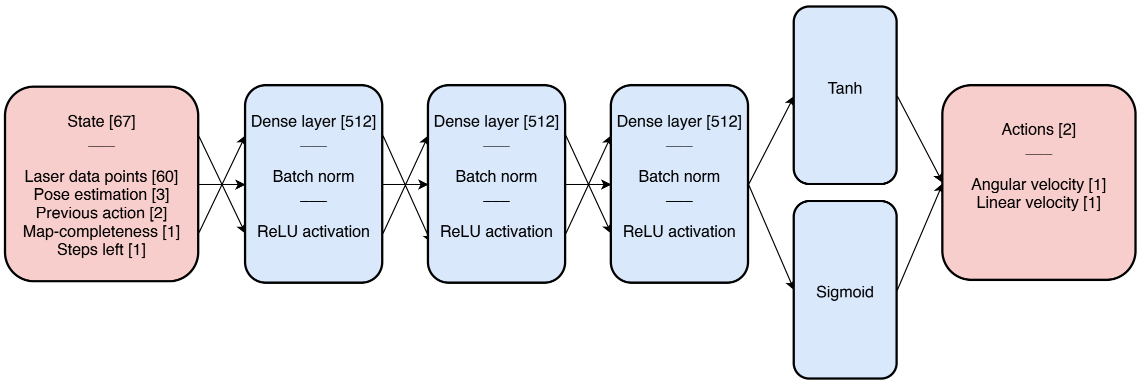


Figure 4.4: Actor network architecture

The critic network will have a similar composition to the actor-network in that it will have three fully connected dense layers of 512 neurons. However, the actions are also appended in the second layer as output. With the intent to let the first layer learn a good representation from the states alone. Instead of the hyperbolic and sigmoidal activation functions in the actor, a linear activation function is used to obtain the state-action (Q-value) as output. The critic network is shown in Figure 4.5.

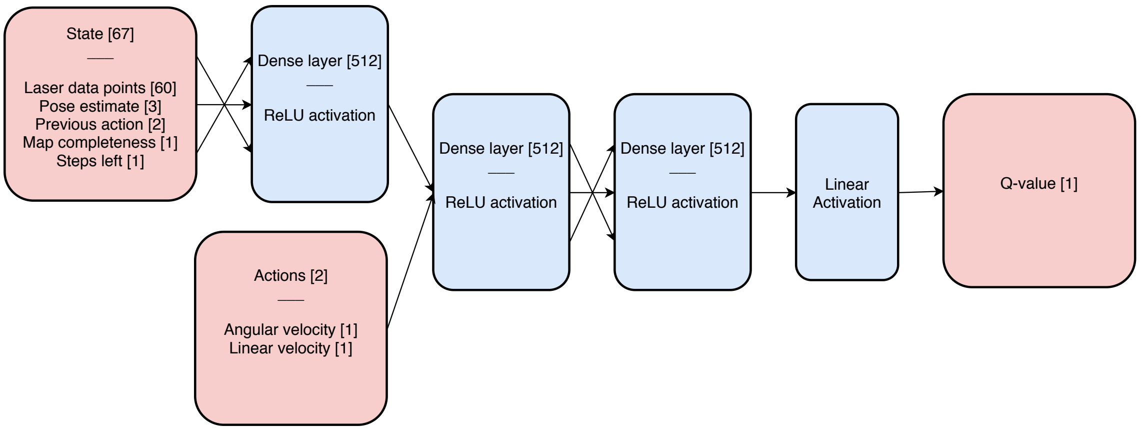


Figure 4.5: Critic network architecture

The weights of the neural network are initialized at random and normalized using batch layer normalization after each dense layer. The normalization is important since all the layers are fully connected and even small fluctuations of the independent variables (input) can be amplified throughout the network to affect the later hidden layers. This can be problematic when earlier layers receive updates and therefore the inputs to the later layers change of value, known as covariate shift [3]. For all experiments, the learning rate γ is set to $1e-3$ and $1e-4$ for the actor and critic network, respectively. The learning rates of the target networks τ are set to $1e-3$ for both networks. To perform the gradient descent for both the actor and critic, the Adaptive Moment Estimation (Adam) is used (Kingma and Ba, 2014). It is a method that computes flexible learning rates for each parameter. Also, it keeps an exponentially decaying average of past gradients and thereby attempts to approximate second-order gradient-based optimization at the computational cost of a first-order optimizer [4]. A full list of hyperparameter used within this thesis can be found in Appendix B.

Target network updates

In the original DDPG implementation both target networks are updated in accordance with Equation 2.17 and 2.15 with a decay of $1-\tau$:

$$\theta^{\text{target}} = (1-\tau)\theta^{\text{target}} + (\tau)\theta \tag{4.12}$$

Using a decay of 0.999 allows us to update the original network parameters θ slowly towards θ^{target}. However, we can do better than just making a copy of the variables by applying a post-processing step which maintains an exponential moving average of the original network parameters:

$$\theta_{\text{ema}} = (1-\lambda)\sum_{i=0}^{N} \lambda^i \theta_{N-i} \tag{4.13}$$

[3] for a detailed explanation of batch layer normalization see Appendix D Section 1

[4] More details on the gradient descent optimizer can be found in Appendix D Section 2

Where $\lambda \in [0,1]$ is the decay rate and θ_N the neural network weight after N steps. Using a decay close to 1, the calculation is similar to Equation 4.12. Over time the network updates become smaller and less consequential, which improves performance.

4.6 Experiments

First, this section will describe the environments used to test the algorithm during training and testing. In the second part of this section, the performed experiments are explained which are used to assert the effectiveness of the reward structures.

4.6.1 Simulation environments

The experiments are trained and tested on a variety of environments with different complexities to test the capabilities and limitations of the algorithm and each scenario individually. All the environments are indoor environments, comprised of a set of walls with a flat floor. The four environments are named Maze, Apartment and Apartment2, as shown in Figure 5.7. The *Maze* environment has a total area of 22 m^2 with few walls and serves as a relatively "simple" starting point. The challenge within this environment is provided by the small and narrow starting room (2x2.5) making it hard for the robot to escape. Using this environment has two advantages: 1. improvement over the previous work of Schulte (2019) can be shown since this environment is identical to the one used therein. 2. due to it's relatively small area training time is limited to a minimum and hence some parameter tuning is feasible in order to determine what is functioning well and what is not feasible.

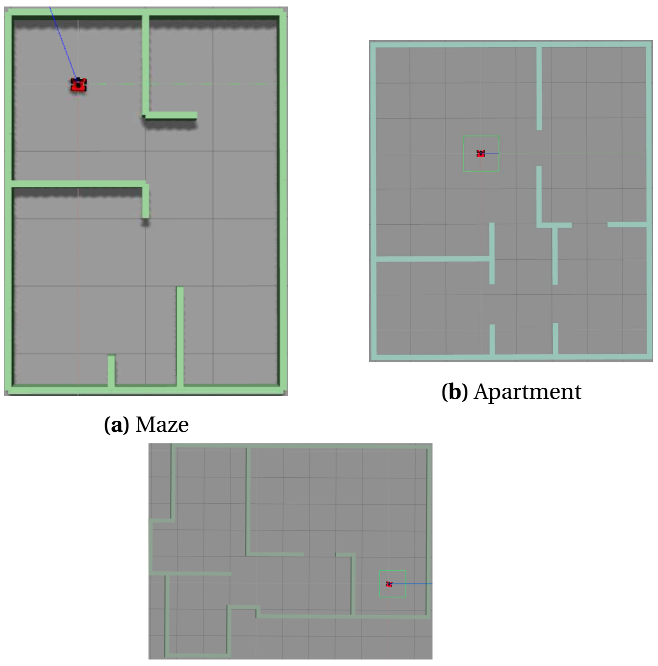
The layout of the second environment, *Apartment*, is based on a real-life, self-contained housing unit with a single floor from the Dutch housing website funda (2020). Although all clutter except the walls is omitted, the intent is to showcase the capabilities of the algorithm in a semi-realistic environment. It is almost triple the size compared to the maze environment (65 m^2) and is comprised of a hallway, living room and three additional rooms. Although there is more space to navigate compared to the Maze environment, mapping the whole environment will provide a significant challenge since the navigational (OU) noise is less likely to be impactful. This puts an additional strain on the reward structure to provide the exploration incentives, which is the intent.

The third environment, *Apartment2*, is similar to the second environment and with a total area of 68.5 m^2 it features slightly more open-space with less enclosed rooms. This environment will mainly be used to test the generalization capabilities of the agent when trained on the comparable Apartment environment.

4.6.2 Experiment: Comparison of different reward functions

The four different reward structures mentioned in the Method section of the thesis are evaluated in this experiment with the intent to (partially) answer research question ii. Hence, it is paramount to determine the quality of the obtained trajectory and exploratory behaviour. Intuitively, proper exploratory behaviour will lead to a successful mapping of the environment. Additionally, a strong exploration policy should be able to map the whole environment in a minimum of actions and with the shortest possible trajectory. Hence, the quality of the trajectory is assessed by the length of the trajectory, the number of collision samples and whether or not the environment is successfully mapped.

First, the four different reward structures are trained on the maze environment until a satisfactory performance level is achieved. After the training period, the reward structures are tested on the same environment for 10 episodes whilst omitting the exploration noise. Thereafter, the reward structures are trained and tested on the Apartment environment in a similar fashion. In both instances, they are compared to the non-reinforcement learning approach of frontier based exploration, as introduced in Chapter 3.


(a) Maze

(b) Apartment

(c) Apartment2

Figure 4.6: Used environments

4.6.3 Experiment: Generalization

In this experiment, the ability of the algorithm to generalize is tested with the intent to answer research question iii. Therefore, the learned policy is directly transferred to a second apartment-like environment showcased in Figure 4.6c. Herein, the area of the environment is similar, however, the room structure is drastically different. It is expected that the agent would suffer in terms of performance since (some) encountered states will unfamiliar to the agent.

4.6.4 Experiment: DDPG vs DRQN

As mentioned in the Introduction (Chapter 1), the authors of Lillicrap et al. (2015) argue that, in theory, discretization of the action space will lead to several issues. However, it is not obvious how this will hold up in practice if the action-space is *very coarsely* discretized. The aim of this experiment is not to disprove any statements made within the paper, but an attempt to showcase possible limitations, advantages or drawbacks of discretization with regards to training phase and testing phase. Specifically, exploratory behaviour and quality of the trajectories are considered when evaluating the experiments. The results of the experiments for the DRQN architecture were performed during an earlier stage of the thesis and published in Botteghi et al. (2020). The Curiosity reward structure was not specifically considered herein and therefore not compared. Furthermore, only the results of the Maze environment are compared, because the Lidar ranged was capped at 4m in the paper, making a comparison on the larger environment improper. Moreover, the paper compared a DQN and DRQN architecture. Although the latter also features a Long Short-Term Memory (LSTM), the reported results of the DRQN are only marginally better in terms of learning curve and variance in terms of map-completeness (Botteghi et al., 2020). This, in addition to the inherent weakness of the DDPG to non-Independent and Identically Distributed (iid) data and thus inability to profit from the LSTM layer justify this comparison.

4.6.5 Experiment: parameter tuning

As already introduced, parameter tuning is a vital part of the reinforcement learning paradigm. If not done properly the performance of the algorithm might suffer significantly or in some cases not converge at all. In principle, it would be best to test these parameters in different settings and compare the results. However, the sheer number of parameters to tune is always a major limitation to this approach. Not to mention that different environments require different parameters for optimal performance. Additionally, there is variance present with each training session and hence the time investment necessary and the computing force required is massive. Only big tech companies like Google can pull off large parameter sweeps and thereby come to the best results. Nevertheless, an adequate amount of parameter tuning has been done during this thesis and this experiment will show some of the recorded results. In particular, the tuning experiments for the Curiosity approach with different values for k and novelty bonus are shown.

5 Results and Discussion

In this chapter, the result of the performed experiments are depicted, discussed and analyzed. Section 5.1 compares the different reward functions on both the Maze and Apartment training environment. The results herein are split between the training of the policy (training phase) and the evaluation in that same environment (performance evaluation in the training environment). The generalization experiment in Section 5.2 compares the performance evaluation in training environment to the performance in the unknown environment Apartment2. Furthermore, the obtained results are compared to a DRQN approach in section 5.3. The influence of the parameter K for the curiosity approach is discussed in Section 5.4 and in the last Section all the overall results are evaluated.

5.1 Experiment: Comparison of different reward functions

5.1.1 Maze

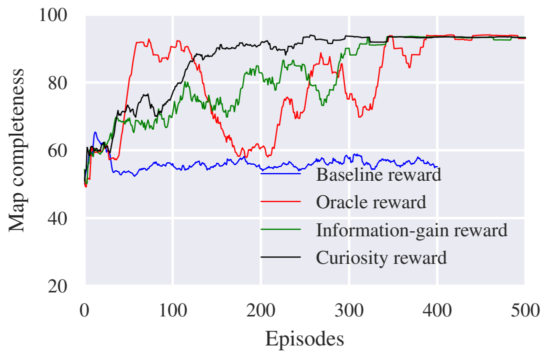
Training phase

The comparison of the Baseline, Oracle, Information-gain and Curiosity reward structures in the maze environment, during the training phase, are depicted in Figures 5.1a - 5.1b. It is apparent, from Figure 5.1a and Table 5.1c, that the Baseline reward function does not converge to the optimal solution within the environment and therefore was stopped early at 400 episodes of training. It was trained for 100 episodes less then the other reward functions, however, almost required double the amount of steps (*398.923* in total) to achieve this amount. This can also clearly be observed in Figure 5.1b, where the baseline reward structure requires the maximum number of steps (1000) nearly every episode, indicating it almost always didn't experience the bonus for completing the map. This can be contributed to the "sparse" reward setting which doesn't promote exploration of the environment and hence the agent gets stuck in a local optimum. The Information-gain and Oracle reward structures have similar convergence times with a total of *277.028* and *266.048* steps, respectively. Although these reward structures initially also do not experience the reward bonus for completing the map, they do get reward uncovering unknown parts of the map and hence progressively work towards that goal. With *176.151* steps, the Curiosity reward structure trains significantly faster than all the other reward structures.

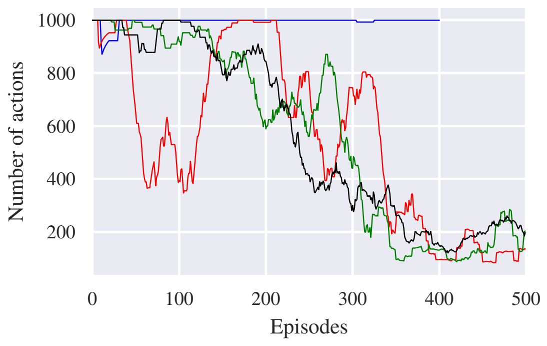
Performance evaluation in the training environment

The trained reward structures are tested in the same environment as they are trained in: the maze (Figure 4.6a). The testing phase was comprised of 11 episodes of at most 1000 actions where the correlated Orhnstein-Uhlenbecker noise was omitted to exclude any randomness and make the comparisons as proper as possible. Similar to the training phase, the robot was spawned in the same position every time the episode was started.

In Table 5.2a the map-completeness, reward, amount of crashes, and success rate during testing is illustrated. All reward functions (except Baseline) are able to complete the map *100%* of the time with no significant difference in the average map-completeness. The baseline reward structure was unable to map the whole environment during training time and consequently unable to do so during testing. Furthermore, the trajectories during testing (a sample trajectory is shown in Figure 5.3), and the amount of crashes (Table 5.2a) indicate that the robot did learn obstacle avoidance behaviour by circling around in the initial room. This, in addition to the small variance in map-completeness suggest that the algorithm was stuck in a local optimum. The intuitive explanation for the learned behaviour is that the bonus reward for completing the map was never experienced during training, and by only imposing an angular velocity the walls will not be hit and therefore no penalty is incurred.



(a)



(b)

Reward structure	Convergence time [in steps]
Baseline	398,923†
Oracle	266,048
Information-gain	277,027
Curiosity	176,151

(c)

Figure 5.1: a) Smoothed (moving-average) map-completeness of the Baseline, Oracle, Information-gain and Curiosity reward structures over time whilst training on the maze environment, **b)** Smoothed number of actions of the Baseline, Oracle, Information-gain and Curiosity reward structures over time whilst training on the maze environment, **c)** Convergence time of the reward structures in steps during training. † did not convergence

The other reward structures are consistently able to map the whole environment with success rates *100%*, *100%* and *100%* for Oracle, Information-gain and Curiosity, respectively. On closer inspection of the trajectories, the Oracle reward function shows some tendency to venture close to the wall near the entrance of the first room (Figure 5.3). Looking at the crash data confirms that, with an average of 15 timesteps per episode spent too close to the wall ($\leq 0.2m$). This is obviously not optimal behaviour, however, makes sense given that the agent perceived more reward by uncovering a greater part of the map taking this trajectory. The bonus for uncovering part of the map outweighs the small penalty accrued due to crashing. Hence, the balance of the reward structure can ultimately affect the developed behaviour. In principle we can train the algorithm for a longer period of time such that better behaviour is achieved, but this makes the algorithm prone to overfitting, hurting the generalization capabilities as we will discuss in Experiment 2. Overall, Curiosity performs the best of the four reward structures within the Maze environment. With the least amount of crashes (*7*) and highest reward (*257.9*) on average[1].Furthermore, the variance σ^2 of the obtained reward and amount of crashes for both the Curiosity and Information-gain reward structure suggest that a longer training time might be beneficial to improve performance even more.

The results of the "best" trajectory from each reward structure are compared to a trajectory of the frontier-based exploration algorithm, as explained in chapter **??**. The frontier-based exploration algorithm had the same settings as previously explained for the other reward structures to make the comparison unbiased. The "best" trajectory in this context is defined as the trajectory of the episode which obtained the highest reward. In Figure 5.3a, these tra-

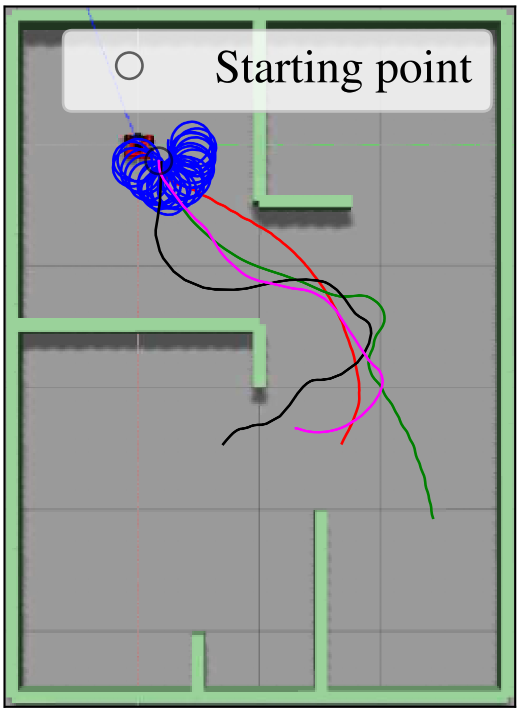
[1]It must be noted that one cannot directly compare reward, since different reward structures yield different rewards for the same task and thus would be an improper comparison. However, the reward in combination with the amount of crashes and the trajectory length gives an improved picture.

jectories are depicted together with the length of the trajectory (Figure 5.3b). Herein it can be observed that the length of the taken trajectories does not differ much between Oracle (3.31m), Information-gain (4.49m) and Curiosity (4.72m). One could argue Oracle performs best here, however, as hinted at before, shows a significantly higher number of collisions, in contrary to the Information-gain and Curiosity reward structures. Curiosity shows a greater tendency to stay further away from the walls, incentivized by it's reward structure to not obtain reward whenever close to a wall. Furthermore, Information-gain shows similar behaviour staying in the middle of the room, which is a desirable property whenever exploring. All of these reward structures have similar trajectory lengths to the frontier-exploration algorithm with a trajectory length of 3.90m. Hence, it can be concluded they perform on par with a staple active SLAM algorithm. Baseline, like the performance during training, was not able to map the entire environment at all and thus had a large trajectory length (28.43m).

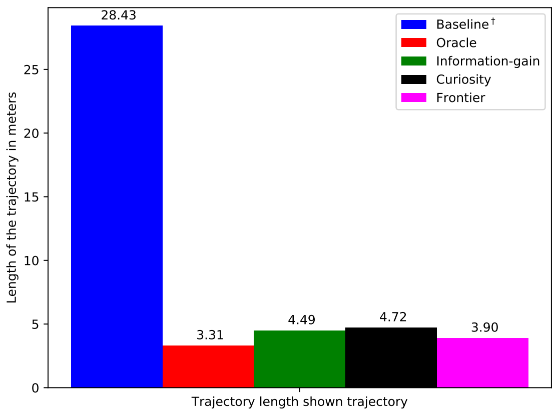
Reward structure	Map-completeness [in %]		Reward [per episode]		Amount of crashes [per episode]		Succes rate [in %]
	μ	σ^2	μ	σ^2	μ	σ^2	
Baseline	48.1	5.7	-12.2	758.0	12	762.2	0
Oracle	**94.2**	0.4	236.0	3.1	15	1.5	**100**
Information-gain	93.1	0.02	195.3	157.2	9	174.0	**100**
Curiosity	93.4	0.2	**257.9**	379.8	**7**	32.4	**100**

(a)

Figure 5.2: a) Map-completeness, reward, amount of crashes, and success rate of the Baseline, Oracle, Information-gain and Curiosity reward structures over time whilst testing on the maze environment. Where μ denotes the average, and σ^2 the variance. The map is considered complete if the explored area >= 93 %.



(a)



(b)

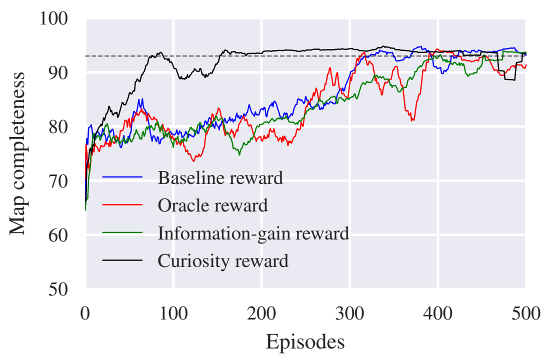
Figure 5.3: a) Trajectory comparison between the Baseline (blue), Oracle (red), Information-gain (green), Curiosity (black) reward structures and frontier based exploration (magenta) over time whilst testing on the Apartment environment, **b)**Trajectory length comparison of the best trajectory in meters. † indicates the environment was not successfully mapped

5.1.2 Apartment

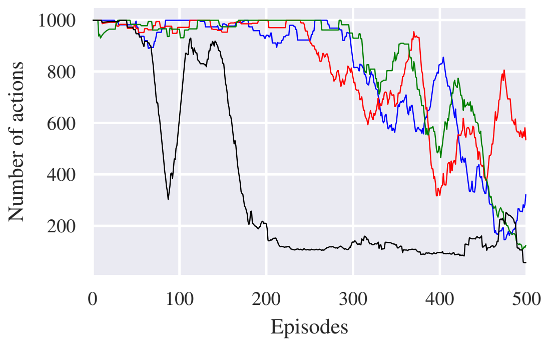
Training phase

The comparison of the Baseline, Oracle, Information-gain and Curiosity reward structures in the Apartment environment, during the training phase, are depicted in Figures 5.4a - 5.4b. The map-completeness during training (Figure 5.4a) is surprisingly similar between the Baseline, Oracle and Information-gain reward structures. Whilst it would be expected that the Oracle and Information-gain converge faster to a solution since they incentivize exploration. A possible explanation can be found when looking at the raw data for the Baseline reward[2], instead of the moving-average, as shown in Figure 5.4c. Herein, it can be observed that the Baseline reward completes the map multiple times within the first 100 episodes, probably due to fortunate pathing trough the environment and thereby experiencing the bonus for completing the map. Multiple training runs should be performed to support this hypothesis, however, this is not done in this thesis due to time-constraints and required training time.

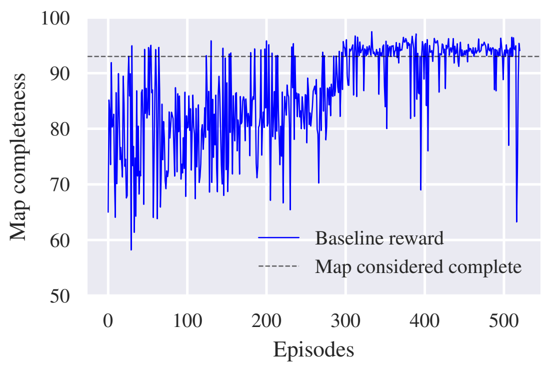
[2]Raw data for the Oracle, Information-gain and Curiosity reward function can be found in Appendix A



(a)



(b)



(c)

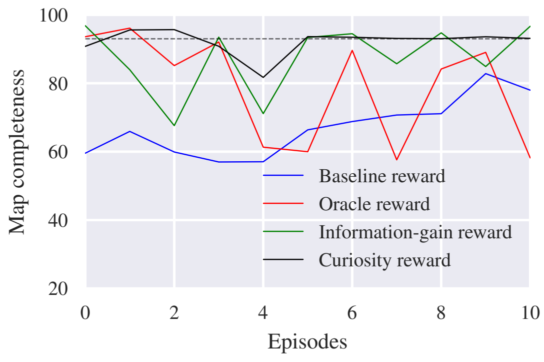
Reward structure	Convergence time [in steps]
Baseline	391,928
Oracle	404,957
Information-gain	407,913
Curiosity	173,459

(d)

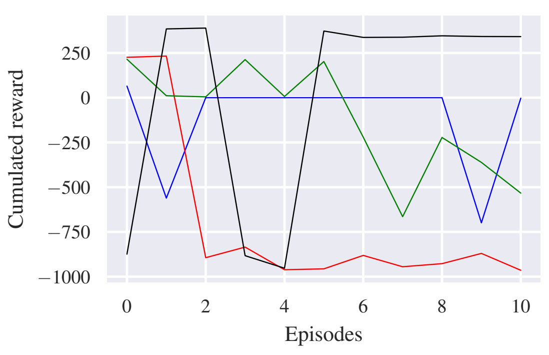
Figure 5.4: a) Smoothed map-completeness of the Baseline, Oracle, Information-gain and Curiosity reward structures over time whilst training on the apartment environment, **b)** Smoothed number of actions of the Baseline, Oracle, Information-gain and Curiosity reward structures over time whilst training on the apartment environment, **c)**Raw map-completeness of the Baseline reward structure over time whilst training on the apartment environment with a visual indicator in black showcasing the percentage of the environment required to be mapped in order to be considered complete, **d)** Convergence time of the reward structures in steps during training.

The Curiosity reward structure clearly outperforms the other reward structures in terms of convergence time, completing the map consistently from the 160th episode onwards. This also becomes apparent when looking at the number of actions taken per episode, as shown in Figure 5.4b. The number of actions per episode sharply drop for the Curiosity reward structure and only slightly decrease after roughly 200 episodes. This implies it quickly reaches a solution for the environment due to the need to drive away from the set of known "novel" states. The Information-gain reward structure requires about the same actions after 500 episodes of training whilst the Baseline and Oracle reward structures perform significantly worse. This indicates they have found sub-optimal solutions for the task and require additional training or they are stuck in a local-optimum, which is not uncommon for policy-gradient methods. It is noteworthy that the Curiosity reward structure requires less than half the total amount actions throughout the training phase *173.459* compared to the others *391.928*, *404.957* and *407.913* for Baseline, Oracle and Information-gain respectively (Table 5.4d).

As the training time of the algorithms is one of the bottlenecks for reinforcement learning, the Curiosity reward structure offers a significant improvement over the others. To put this in perspective, using my personal computer and setup as an example, every 100.000 actions require around 9 hours to complete saving approximately more then *18 hours* when using Curiosity.



(a)



(b)

Reward structure	Map-completeness [in %]		Reward [per episode]		Succes rate [in %]
	μ	σ^2	μ	σ^2	
Baseline	67.0	64.7	-108.9	615,533.0	0
Oracle	78.8	229.0	-706.8	196,268.4	18
Information-gain	87.5	93.5	-122.7	83,785.5	54
Curiosity	**92.2**	13.2	**12.8**	315,243.0	**73**

(c)

Figure 5.5: a) Map-completeness of the Baseline, Oracle, Information-gain and Curiosity reward structures over time whilst testing on the apartment environment. The black dotted line shows the percentage of the environment required to be mapped in order for it to be considered complete,**b)** Cumulated reward of the Baseline, Oracle, Information-gain and Curiosity reward structures over time whilst testing on the apartment environment, **c)**Map-completeness, reward, and success rate of the Baseline, Oracle, Information-gain and Curiosity reward structures over time whilst testing on the maze environment. Where μ denotes the average, and σ^2 the variance. The map is considered complete if the explored area >= 93 %.

Performance evaluation in the training environment: overall performance

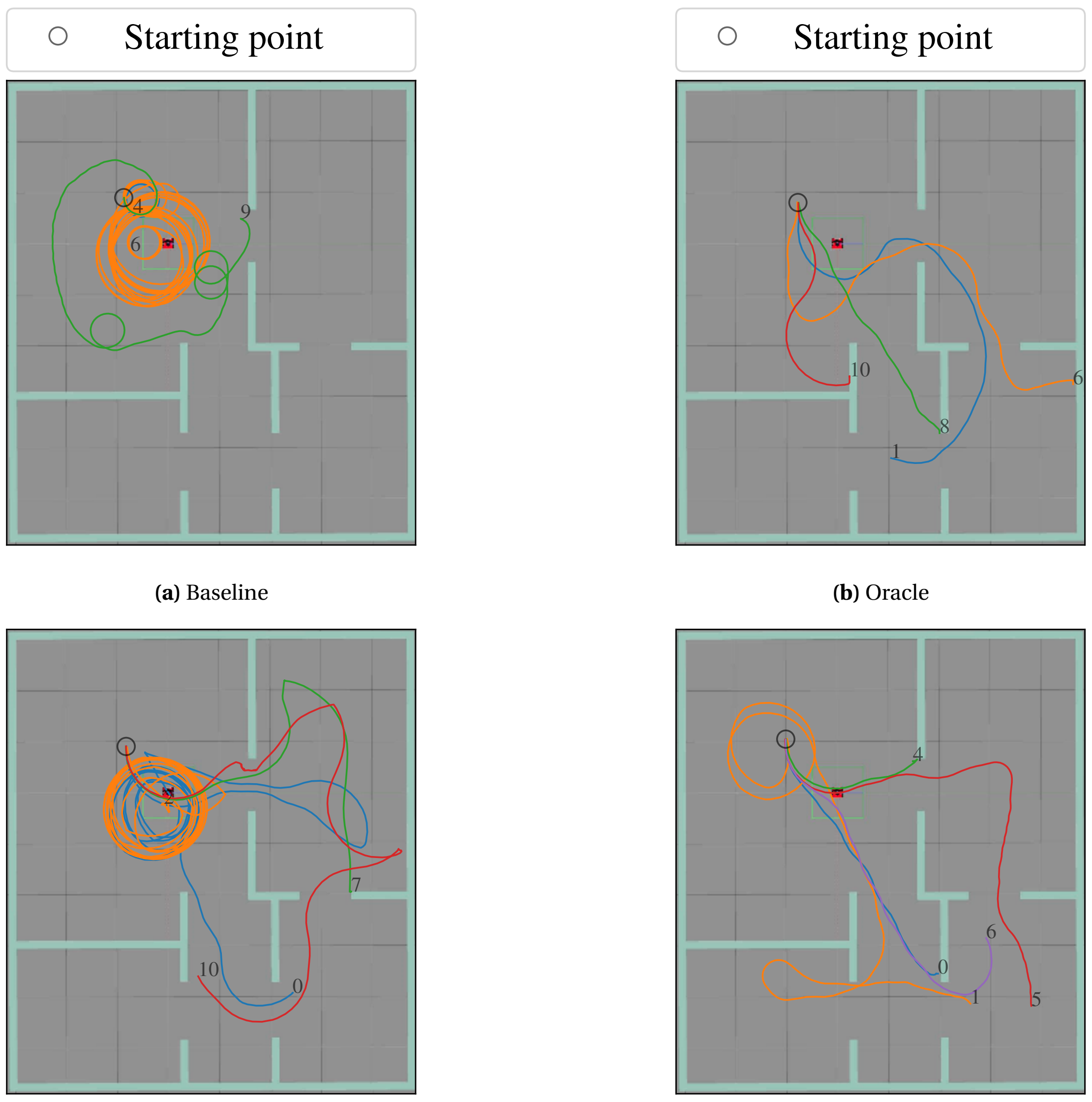
The trained reward structures are tested in the same environment as they are trained in: the apartment (Figure 4.6b). The testing phase was comprised of 11 episodes of at most 1000 actions where the correlated Orhnstein-Uhlenbecker noise was omitted to exclude any randomness and make the comparisons as proper as possible. Similar to the training phase, the robot was spawned in the same position every time the episode was started.

In Figure 5.5a the map-completeness during testing is illustrated with a black dotted line to indicate the threshold for the environment to be considered mapped. In Figure 5.5b, the cumulated reward per episode can be observed. Moreover, the succes ratio is reported in Table 5.5c alongside the mean, and variance of the map-completeness and cumulated reward. The Curiosity reward outperforms the others with a success ratio of *73%*, average map-completeness of *92%*, and an average cumulated reward of *12.8*. It is meaningful to mention that the average map-completeness is a slightly biased metric, because map-completeness can go over the required completeness of 93% depending on the path taken, speed of the robot and trigger of the SLAM algorithm to update the map. Information-gain is a close second in terms of average map-completeness *88%*, however, was only able to complete the map 6 out of 11 times with a success ratio of *54%*. The Baseline and Oracle are the worst performers with average map-completeness of *67%* and *79%*, respectively. The former shows particularly poor performance never being able to successfully complete the map within the 11 episodes whilst the latter achieves a success rate of *18%*. From the low average and high variance in reward per episode it becomes readily apparent that the determinstic policy can be trained further to improve performance if desired. Furthermore it indicates a drastic performance difference between episodes, which are investigated in the following Section.

Performance evaluation in the training environment: trajectories

Next we want to take a closer look at the trajectories of the robot during the 11 test episodes so we can start to analyze the quality of the trajectories and exploratory behaviour. Some sample trajectories are depicted in Figures 5.6a - 5.6d to show the variance in trajectories for the Baseline, Oracle, Information-gain and Curiosity reward structures, respectively[3]. The Baseline scenario shows the expected obstacle avoidance behaviour due to the penalty for crashing, however, shows no interest in exploring the other rooms or completing the environment. The Oracle and Information-gain rewards show improved exploratory behaviour traversing multiple rooms in most instances. This can be attributed to utilization of map-dependent knowledge driving the robot to unexplored areas of the map. The latter still illustrates significant circling behaviour, which indicates it did not yet converge to optimal behaviour. Furthermore, both are showing a significant increase in the number of crashes - trajectories 2, 4, 5, 6, 7, 9, 10 for Oracle and 5, 6, 7, 8 and 9 for Information-gain - compared to the Baseline scenario. An explanation can be found in the fact that there is an supplementary exploration reward, which somewhat seems to reduce the impact the penalty of crashing, if there is sufficient bonus gathered from uncovering unknown parts of the map. Furthermore, trajectories are sequential and therefore states located far away from the initial position are visited less often. This could be alleviated by additional training time, however, this will not guarantee a solution to this problem. This is because once convergence is achieved it is unlikely to return to sub-optimal parts of the state-space and thereby only learning from a certain set of observations over and over. Increasing the noise can help frequent the state-space more often, but this will introduce obvious problems such as increasing the incurred regret (opportunity loss) and training time with only small deviations to the optimal trajectory. One possible solution is randomizing the starting position of the robot to some degree. This will force the robot to search solutions from different parts of the state-space and this option is considered in Experiment 5.2.

[3]Due to a little variance in the odometer and by plotting the trajectories on an image, some trajectories might be a little skewed



(a) Baseline

(b) Oracle

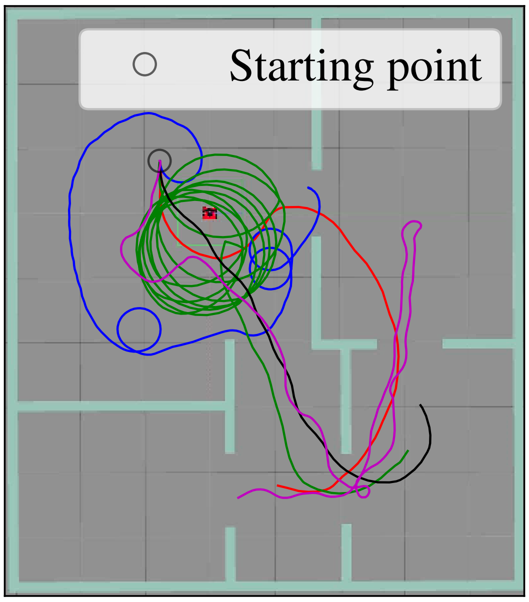
(c) Information-gain

(d) Curiosity

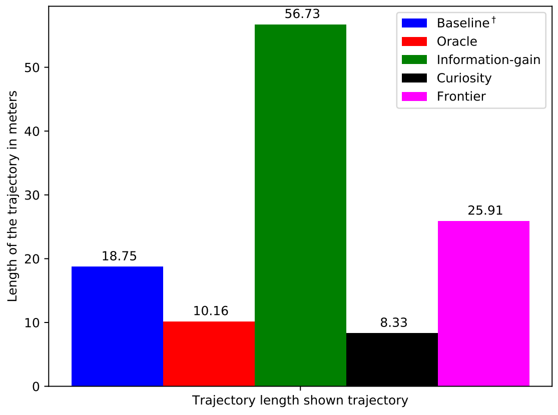
Figure 5.6: Trajectories of the **a)** Baseline, **b)**Oracle, **c)**Information-gain and **d)**Curiosity reward structures over time whilst testing on the apartment environment. The labels on the trajectories correspond to the results found in Figures 5.5a and 5.5b.

Since the robot is only allowed to move forward with a minimum speed of *0.1 m/s* it is unable to drive any further whenever a collision occurs. Hence, getting stuck means that it will only receive negative reward from that point moving forward and thus circling behaviour is encouraged. The inability of the robot to escape obstacles also clarifies why there are a number of episodes where the endpoint is a wall in case of Oracle or Information-gain. Allowing the robot to have negative linear speed would allow it to recover from a crash at the cost of some algorithmic complexity. However, this is not done within the scope of this thesis due to time-constraints. The Curiosity reward structure does not seems to suffer from that same issue as points close to wall cannot be considered novel points. Therefore, it will try not to venture too close to a wall as can be observed in the trajectories. Moreover, in trajectories 1 (orange) it shows some backtracking behaviour which can be desirable in more complex and unseen environments.

Lastly, the results of the "best" trajectory from each reward structure are compared to a trajectory of the frontier based exploration algorithm, as explained in chapter 2. Where "best" is defined as trajectory of the episode with the highest reward, where the environment was successfully mapped, as shown in Figure 5.5b. The frontier-based exploration algorithm had the same parameters[4] as the other reward structures to make the comparison fair. In Figure 5.7a, these trajectories are depicted together with the length of the illustrated trajectory. Similar to previous comparisons, Curiosity has the best performance with a trajectory length of *8.3 meters*. Oracle is a close second with a length of 10.2 meters, taking a slightly sub-optimal route trough the right-top room and thereafter travelling down. Both Curiosity and Oracle outperform Frontier-based exploration (*20.2 meters*) and follow significantly shorter paths. The other reward structures, Baseline and Information-gain, either where unable to map the environment in the case of Baseline or took a very long path (Information-gain *56.7* meters).



(a)



(b)

Figure 5.7: a) Trajectory comparison between the Baseline (blue), Oracle (red), Information-gain (green), Curiosity (black) reward structures and frontier based exploration (magenta) over time whilst testing on the apartment environment, **b)**Trajectory length comparison (in meters). † indicates the environment was not successfully mapped

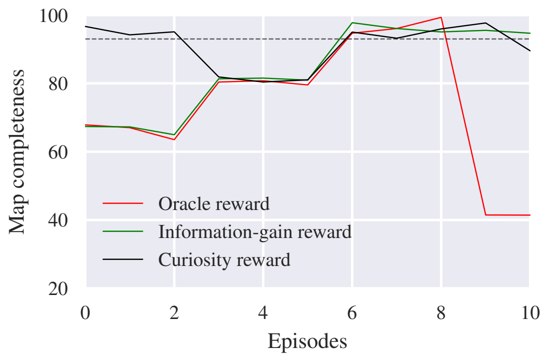
5.1.3 A note on the different environment configurations

At first glance, the objective for the robot seems relatively simple in the maze environment: getting out of the room. However, it has been shown that, due to its narrow configuration, escaping the initial room is a challenge, especially for the Baseline reward structure. It took the other reward structures also relatively long to converge considering the size of the maze environment, with Information-gain and Oracle taking around ≈ 130,000 steps less to converge, compared to the larger Apartment environment. This can again be contributed to the narrow environment, leaving little room for maneuvering and consequently a relatively long training time. It can be concluded that the configuration of the environment significantly impacts the training performance and it might be beneficial to take this into account when shaping the reward function.

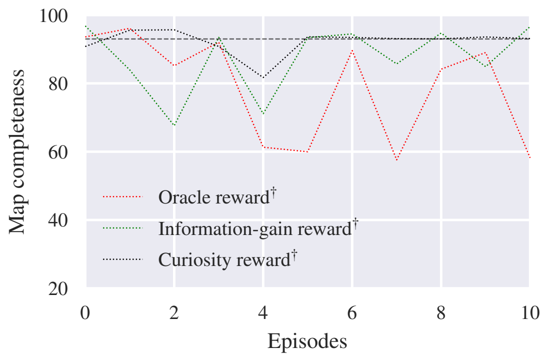
[4]Shown in appendix B

5.2 Experiment: Generalization

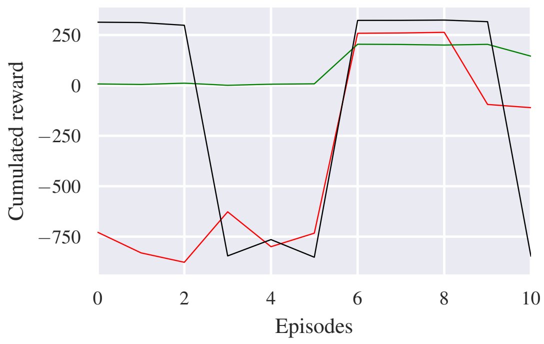
In this section, the generalization of the learned policies to unseen environments are discussed in more details. In the first part, the (successful) policies trained on the Apartment environment, discussed in Experiment 5.1, is directly transferred to unseen environment of Figure 4.6c. The robot was allowed to start 3 times from 4 different pre-picked position for a total of 12 times. The results are depicted in Figure 5.8a for the map-completeness over time. It can be observed that the Curiosity reward structure performs best on average across all the starting positions 91.0 %, whilst Information-gain and Oracle have a lower average of 83.9 and 73.8 %, respectively. Curiosity consequently has the highest average success ratio (64 %) against 46 % of Information-gain and 28 % of the Oracle reward structure.



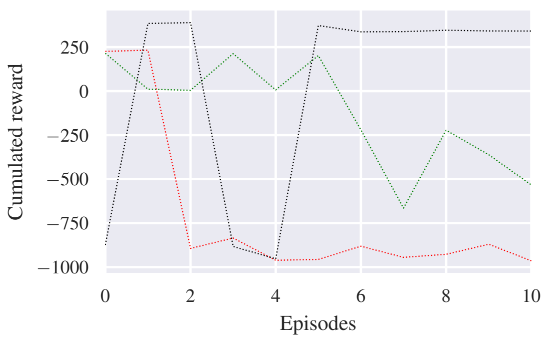
(a)



(b)



(c)



(d)

Figure 5.8: a) Map-completeness of the Oracle, Information-gain and Curiosity reward structures over time, whilst being tested on the Apartment environment,**b)**Map-completeness of the Oracle, Information-gain and Curiosity reward structures over time whilst being tested on the **unknown** Apartment2 environment, **c)** Cumulated reward of the Oracle, Information-gain and Curiosity reward structures over time whilst testing on the Apartment environment, **d)** Cumulated reward of the Oracle, Information-gain and Curiosity reward structures over time whilst testing on the **unknown** Apartment2 environment. The black dotted line shows the percentage of the environment required to be mapped in order for it to be considered complete and the daggers † indicates testing performance on the unknown environment

Reward structure	Map-completeness					Reward					Succes rate	
	Env1		Env2			Env1		Env2			Env1	Env2
	μ_1	σ_1^2	μ_2	σ_2^2	$\mu_2-\mu_1$	μ_1	σ_1^2	μ_2	σ_2^2	$\mu_2-\mu_1$	%	%
Oracle	78.8	229.0	73.8	362.6	-5.0	-706.8	196,268.4	-365.3	210,622.2	**341.5**	18	28
Information-gain	87.5	93.5	83.9	149.6	-3.7	-122.7	83,785.5	**90.4**	8,733.6	213.1	54	46
Curiosity	**92.2**	13.2	**91.0**	40.6	**-1.3**	**12.8**	315,243.0	-100.1	302,892.1	-112.8	**73**	**64**

Table 5.1: Map-completeness, reward, and success rate of the Baseline, Oracle, Information-gain and Curiosity reward structures over time whilst testing on the Apartment and Apartment2 environment. Where μ denotes the average, σ^2 the variance, and $\mu_2 - \mu_1$ the performance difference during testing between the unseen Apartment2 (Env2) and Apartment (Env1).

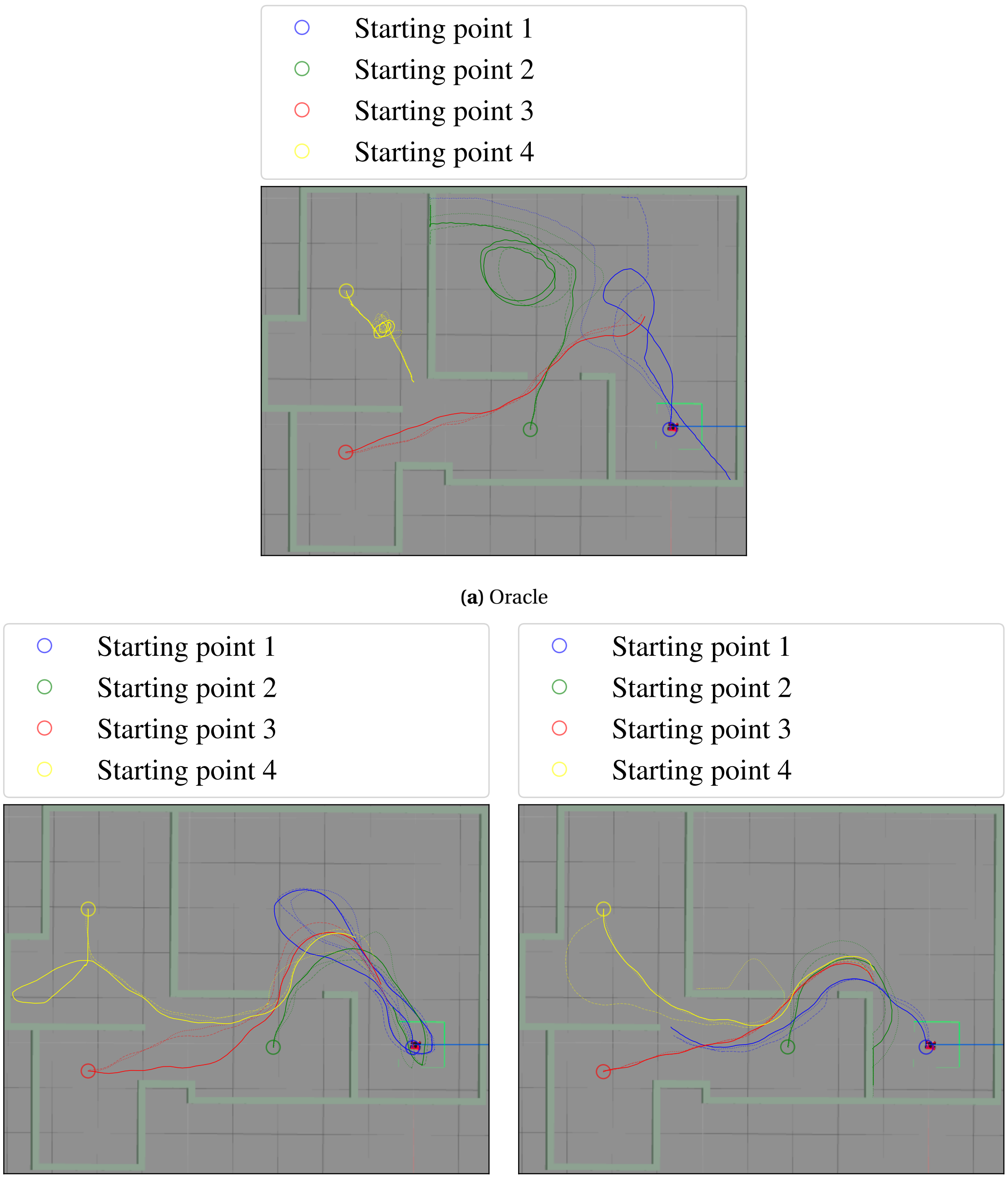
In this section we are more interested in the generalization capabilities of the respective agents. The performances are set side to side in Figures 5.8a - 5.8b for map-completeness and Figures 5.8c - 5.8d for the results of testing on Apartment (solid lines) and Apartment2 (dotted lines), respectively. In terms of map-completeness, directly transferring the policy on a similar environment seems to have only a slight negative impact on average performance: -1.3 % for Curiosity, -3.7 % for Information-gain and -5 % for Oracle. As one might expect the reward gained (on average) per episode is lower (-87.3) in the Apartment2 environment for Curiosity. Surprisingly, the Oracle reward function gets, on average, 341.5 more reward per episode in the unseen (Apartment2) environment compared to the seen (Apartment) environment. Furthermore, the Information-gain reward function also performs better in the unseen environment with a positive reward difference of 213.1 (Table 5.1).

We are going to explore the possible reasons why by taking a look at the trajectories in Figures 5.9. Herein, the Oracle reward function has relatively long trajectories with only a few time-steps spent crashing.The results of section 5.1 show some short trajectories with a lot of crashing, which is penalized heavily. Hence, this can possibly explain the positive difference in reward between the two environments. Moreover, starting position 1 features a lot of open space, which gives the Information-gain and Oracle structures a boost in map-completeness and thus reward. Overall, it can be concluded that the ability of the proposed reward structures is satisfactorily with relatively small drops in performance (map-completeness) and in some instances a better performance (reward). This opens up potential for the agent to train on a single environment and to do well in multiple environments by direct policy transfer, if a diverse enough training environment is constructed. This can save immense amounts of training time and consequently computer resources, which is the major downside of reinforcement learning as of today.

It is noteworthy to mention that none of the algorithms were able to perform well from starting position 2. A possible reason for that is that the agent is not trained on backtracking and looping back trough it's starting area in this fashion. The Curiosity reward structure has particular trouble with starting position 2. Whilst starting from this position, the only way for the robot to map the environment is by looping back trough the already explored part. This might imply a weakness of the reward structure in that previous novel position do not provide additional benefits to exploring already explored areas of the map without training for this explicitly. Starting position 4 was also difficult for the agents, because of the narrow opening of the room. This made it particularly challenging to get out of, similar to the Maze environment. Overall, Curiosity clearly has the best performance being the only reward structure to completely map the environment from 3 out of 4 starting positions.

Sample videos of the performance can be found on Youtube:

1. *Curiosity*
2. *Information-gain*
3. *Oracle*



(a) Oracle

(b) Information-gain

(c) Curiosity

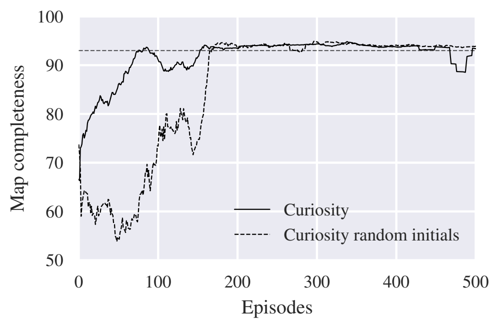
Figure 5.9: Trajectories of the **a)** Oracle, **b)**Information-gain and **c)**Curiosity reward structures over time whilst testing on the Apartment2 environment.

5.2.1 Improving generalization

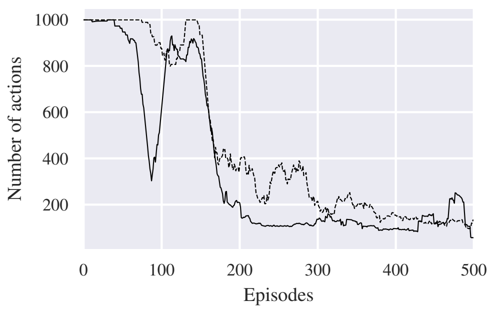
In this section, the Curiosity reward structure is specifically (re)trained for generalization by starting from a random initial position with random orientation within the Apartment environment. The initial position was uniformly sampled from 4 possible starting locations, as can be observed in Figure 5.10c. The intent here is to observe if (i) the shortcomings (e.g. backtracking) of the Curiosity reward structure within the unknown environment can be resolved by visiting all parts of the state-space more frequently during training, and (ii) to see what effect this strategy has on the trained environment (Apartment) and on the generalization performances in the unknown environment (Apartment2).

Training phase

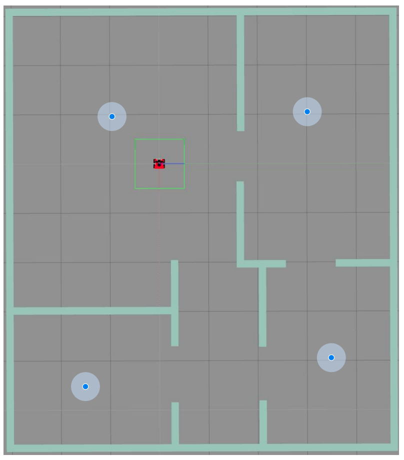
During the training, the Curiosity approach with random initial positions and orientation took initially longer to learn, which makes sense given that it had to learn from a more diverse sample distribution (Figure 5.10a). It also depicts a steeper learning curve, compared to the Curiosity without random initials. This is because once the learned trajectories from the different starting positions connect, the robot is instantaneously capable of navigating through multiple rooms. Consequently, the map-completeness then rises quickly, as can be observed around 50 and particularly around 150 episodes. The result is that the Curiosity with random initials does not require many additional steps to converge, compared to the Curiosity approach (Table 5.10d). Furthermore, the trend in the number of actions during training is similar between the two approaches (Figure 5.10b).



(a)



(b)



(c)

Reward structure	Convergence time [in steps]
Curiosity	176,151
Curiosity random initials	222,835

(d)

Figure 5.10: a) Smoothed (moving-average) map-completeness of the Curiosity and Curiosity with random initials over time whilst training on the maze environment, **b)** Smoothed number of actions off the Curiosity and Curiosity with random initials over time whilst training on the maze environment, **c)** The four possible starting positions of the mobile robot within the Apartment environment, **d)** Convergence time of the Curiosity and Curiosity with random initials in steps during training.

Performance evaluation in the training and testing environment
The testing performance in terms of reward and map-completeness for both approaches is illustrated in Figures 5.11a - 5.11d for the tests performed on both the Apartment and Apartment2 environment. Furthermore, the success rate, average map-completeness and cumulated reward is reported in Table 5.2.

It becomes immediately clear when observing the results that the Curiosity with random initials approach performs better in both the unknown and training environment. Furthermore, the small amount of variance in map-completeness and reward (5.2) indicate that the taken trajectories are quite consistent and that any noise in the sensors, particularly the pose estimation, is very well dealt with. This can be contributed to the the samples accrued in positions it would otherwise (without random starting position and orientation) less likely encounter. This does come at the cost of a longer training time (50.000 extra steps).

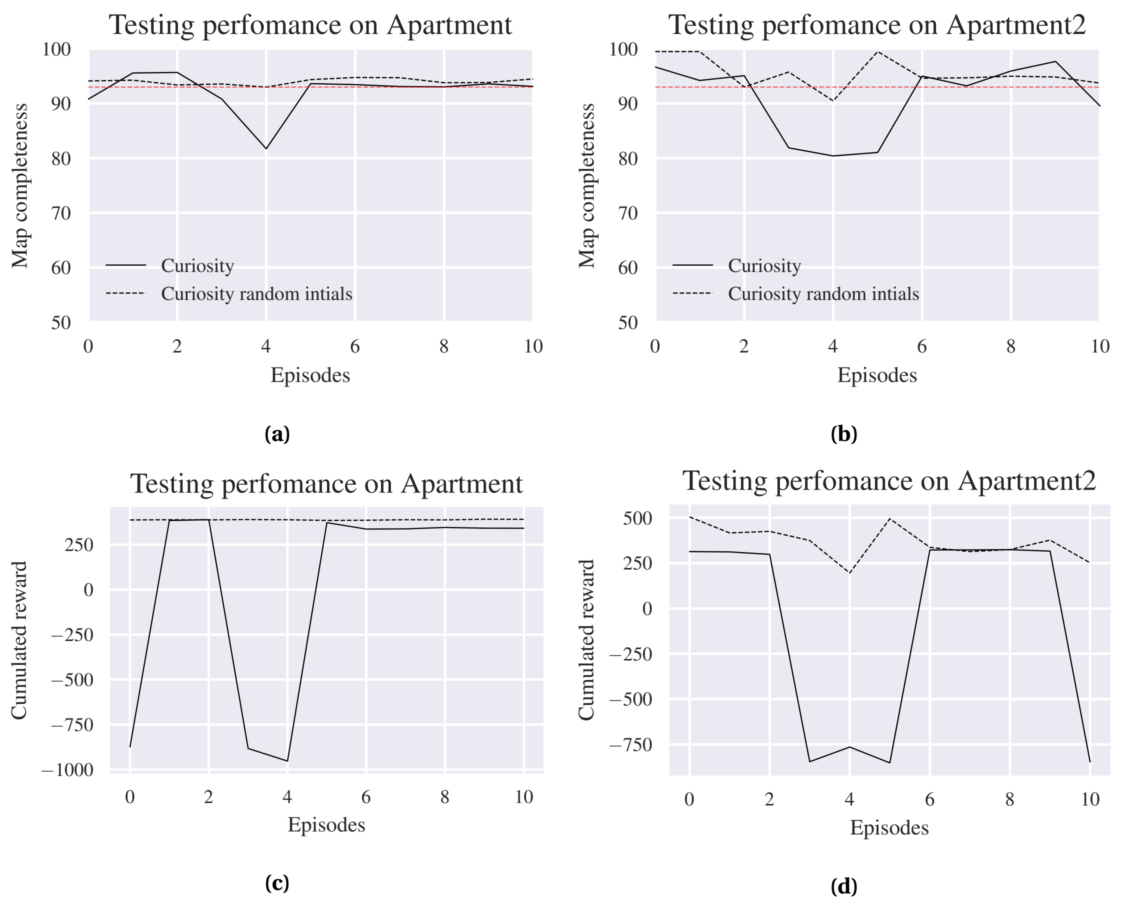
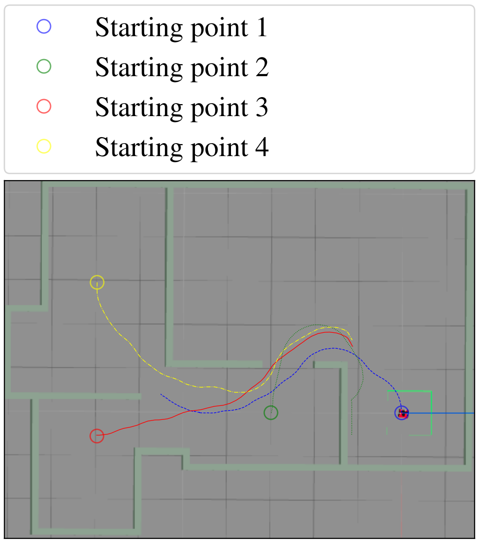


Figure 5.11: a) Map-completeness of the Curiosity and Curiosity with random initials over time, whilst being tested on the Apartment environment,**b)**Map-completeness of the Curiosity and Curiosity with random initials over time whilst being tested on the **unknown** Apartment2 environment, **c)** Cumulated reward of the Curiosity and Curiosity with random initials over time whilst testing on the Apartment environment, **d)** Cumulated reward of the Curiosity and Curiosity with random initials over time whilst testing on the **unknown** Apartment2 environment. The red dotted line shows the percentage of the environment required to be mapped in order for it to be considered complete.

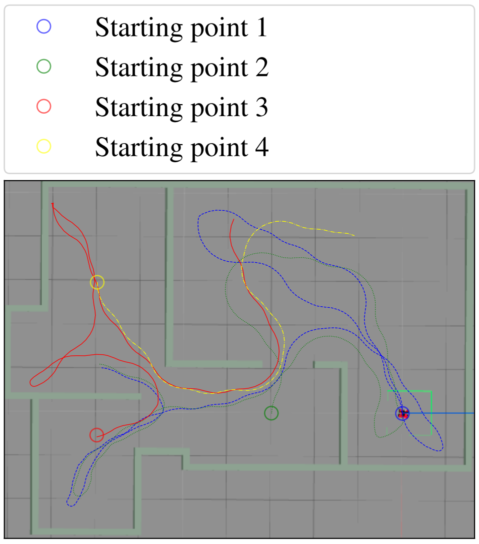
Reward structure	Map-completeness					Reward					Success rate	
	Env1		Env2			Env1		Env2			Env1	Env2
	μ_1	σ_1^2	μ_2	σ_2^2	$\mu_2 - \mu_1$	μ_1	σ_1^2	μ_2	σ_2^2	$\mu_2 - \mu_1$	%	%
Curiosity	92.2	13.2	91.0	40.6	-1.3	12.8	315,243.0	-100.1	302,892.1	-112.8	73	64
Curiosity random	**94.0**	0.3	**95.5**	7.7	**1.5**	**388.4**	3.6	**365.0**	8,155.6	**-23.4**	**100**	**90.1**

Table 5.2: Map-completeness, reward, and success rate of the Curiosity and Curiosity with random initials over time whilst testing on the Apartment and Apartment2 environment. Where μ denotes the average, σ^2 the variance, and $\mu_2 - \mu_1$ the performance difference during testing between the unseen Apartment2 (Env2) and Apartment (Env1).

The Curiosity trained with random initials shows significantly longer trajectories from all of the four different starting locations (Figure 5.12b). Furthermore, it exhibits a tendency to go to the corner of any particular room and thereafter circle back in the direction it came from. This behaviour seems to appear even if the room in question is already mapped (e.g. top-left room for the red trajectory or circling back to the starting point in case of the blue trajectory). One could argue this is sub-optimal behaviour compared to the Curiosity which is not trained with random initials (Figure 5.12a). However, from the agent's point of view this is considered "optimal" behaviour since it achieves a better result (Table 5.2), both in terms of reward and map-completeness, behaving this way. This is because there is no penalty for taking additional steps within a particular episode given that there are no crashes involved and extra reward can be accrued by collecting excess novel points. One could consider this a flaw in the reward structure design if short trajectories are desired, which could be resolved by adding a term which increases the urgency of exploration (e.g -1/timestep) and thereby negating this effect. However, it shows the best exploratory behaviour, by carefully inspecting every room within the state-space, which allows it to traverse complicated rooms. Thus, the generalization capabilities of the Curiosity with random initials is better in every way compared to the Curiosity with normal training procedure. A sample video of the Curiosity with random initials on the Apartment2 can be found on Youtube: *Curiosity trained with random intials*



(a) Curiosity

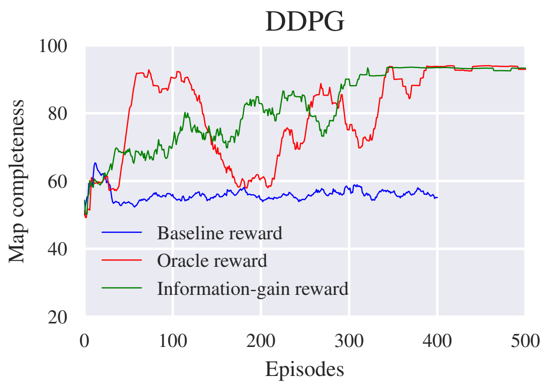


(b) Curiosity trained with random initials

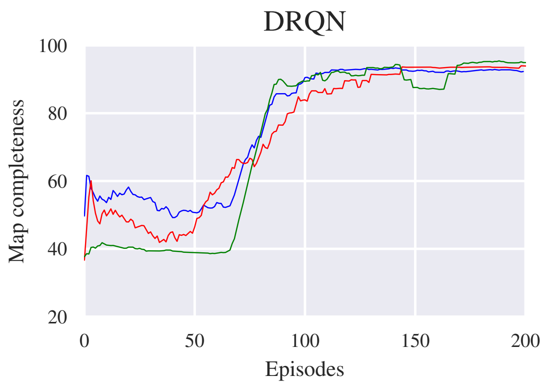
Figure 5.12: Side to side comparison of the best trajectories from each of the four starting positions of the **a)** Curiosity, **b)**Curiosity trained with random initial position and orientation whilst testing on the Apartment2 environment.

5.3 Experiment: DDPG vs DRQN

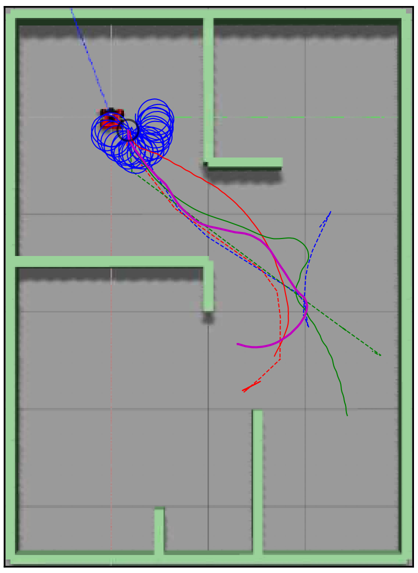
In this section, the results of the DDPG algorithm are compared to a DRQN algorithm, which are both trained and tested on the Maze environment 4.6a. The training results of the DRQN algorithm are shown in Figure 5.13a, whilst Figure 5.13b depicts the familiar training results of DDPG. It is abundantly clear that the DRQN algorithm shows better results during training time, achieving convergence at around 100 episodes for all reward structures. This is more than 250 episodes faster then the best performing DDPG algorithm, converging at around 350 episodes. Similar differences can be found in terms of actions performed during the training period averaging 108,369 actions for the DRQN algorithm across all reward structures and averaging 313,999 actions for DDPG (Figure 5.13d). The differences can be explained due to the discretization of the action-space $(0, k)$ for the linear and $-k, 0, k$ for the angular velocity) and thus training is a lot faster then training over a continuous action range.



(a)



(b)



(c)

Reward structure	Average convergence time [steps]
Baseline DDPG†	398,923
Oracle DDPG	266,048
Information-gain DDPG	277,027
Baseline DRQN	99,543
Oracle DRQN	114,540
Information-gain DRQN	111,025

(d)

Figure 5.13: a) Smoothed map-completeness of the Baseline, Oracle, Information-gain and Curiosity reward structures over time whilst training on the maze environment utilizing the DRQN architecture (Botteghi et al. (2020)), **b)** Smoothed map-completeness of the Baseline, Oracle, Information-gain and Curiosity reward structures over time whilst training on the maze environment utilizing the DDPG architecture, , **c)** Best trajectories of the Baseline DDPG (blue), Baseline DRQN (blue dotted), Oracle DDPG (red), Oracle DRQN (red dotted), Information-gain DDPG (green), Information-gain DRQN (green dotted) and Frontier (magenta) reward structures over time whilst testing on the maze environment

The results in terms of best trajectories are shown in Figure 5.13c. The Baseline reward structure performs better for DRQN, compared to DDPG. However, the Oracle reward structure performs better for DDPG (3.3m against 4.7m). In the case of information-gain the DRQN has the upper hand (4.1m against 4.7m). One could conclude that in fact there are only minimal difference between the trajectories.

The used action space is not massive with when using a 2 dof robot. Hence, the described theoretical concerns do no hold if the action-space is sufficiently small. It must be noted that within this environment, which is mostly comprised of straight sections of walls, the coarse discretization is not really noticeable among the trajectories. However, one could imagine that maneuvering in a more complex environment would prove more difficult and a finer discretization might be required.

Performance evaluation Apartment and Apartment2 environment
As stated in Chapter 5, we cannot really compare the training results of the Apartment directly, because of the reduced Lidar range used by the authors of Botteghi et al. (2020). However, we can compare the test results in the Apartment and unknown Apartment2 environment in terms of average map-completeness, success ratio and general navigational behaviour (ignoring trajectory length) for the two architectures. Since the Baseline (called Sparse in Botteghi et al. (2020)) performs very poorly for both architectures in the Apartment and Apartment2 environment, these results are omitted.

The DRQN architecture shows better testing results in the environment it was trained in (Table 5.3). With much better success rate and map-completeness on average, however, it's performance drops considerably when testing on the Apartment2 environment (-43.6 and -31.7 map-completeness for Oracle and Information-gain, respectively). Hence, they generalize significantly worse than their DDPG counterparts. In both instances, the Information-gain reward structure seems to be the superior choice. It can be concluded that the DRQN architecture is a feasible choice if the training environment is the same as the application environment.

Reward structure	Map-completeness [difference]					Succes rate	
	Env1		Env2			Env1	Env2
DDPG	μ_1	σ_1^2	μ_2	σ_2^2	$\mu_2 - \mu_1$	%	%
Oracle	78.8	229.0	73.8	362.6	-5.0	18	28
Information-gain	87.5	93.5	**83.9**	149.6	**-3.7**	54	**64**
DRQN							
Oracle	93.7	0.5	50.1	562.7	-43.6	**100**	11
Information-gain	**94.3**	0.4	62.6	221.5	-31.7	**100**	11

Table 5.3: Map-completeness, reward, and success rate of the Oracle and Information-gain reward structures (for both the DDPG and DRQN architecture) over time whilst testing on the Apartment and Apartment2 environment. Where μ denotes the average, σ^2 the variance, and $\mu_2 - \mu_1$ the performance difference during testing between the unseen Apartment2 (Env2) and Apartment (Env1).

5.4 Experiment: parameter tuning

In this section some interesting parameters regarding the Curiosity approach are discussed and analyzed. In particular, some of the parameters affecting the reward process directly such as novelty parameter K.

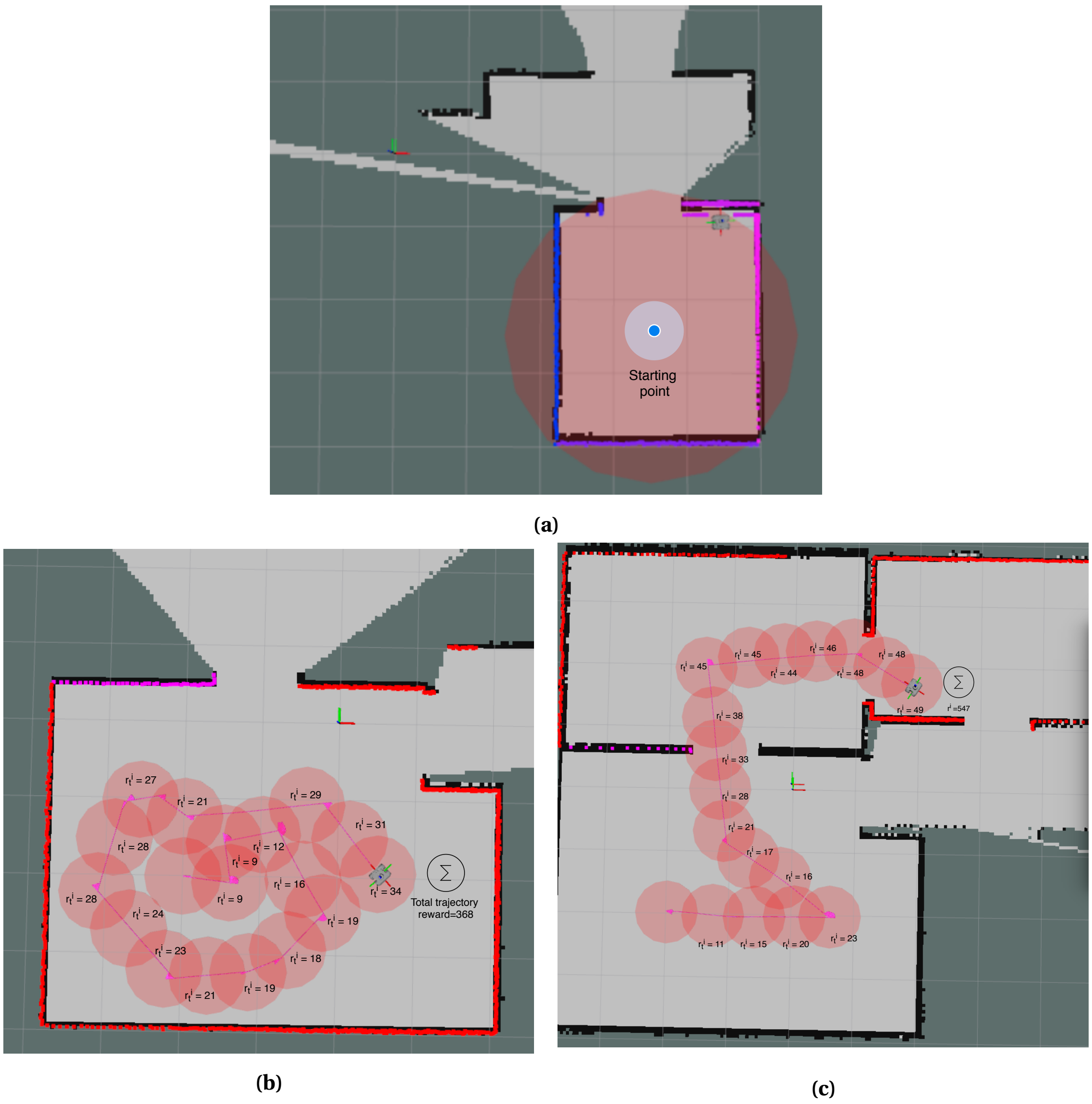


Figure 5.14: a) Depiction of a too large parameter K considering the starting room for the Curiosity approach within the Apartment environment, **b)** Intrinsic curiosity reward per time-step and total reward depicted for manually driven trajectory whilst staying in a singular room, **c)** Intrinsic curiosity reward per time-step and total reward for a trajectory depicted for manually driven trajectory trough multiple rooms

As hinted at before, a weakness of the Curiosity approach is it dependency on selecting the appropriate novelty distance K. The appropriate value for this hinges on many factors, such as the topology of the rooms within the environment, the starting position of the robot within the environment, the training methodology (e.g random starting positions), and the range of your LiDAR. The parameter K effectively determines the radius of the novelty circle (area on the map where no new novelty point can be obtained) with as origin the position of the novel point. As the first novel point is the starting point of the robot within an environment, it can be beneficial for the training process to select K such that several novel points can be obtained within the starting room. Selecting K too large compared to the starting room of the robot can be detrimental to the learning process. For instance, consider the starting location depicted in Figure 5.14a, where the robot first has to escape the room to obtain additional novel points. Given the Curiosity reward structure (4.6), the robot has to exclusively rely on the exploratory noise in this instance in order to escape the room. This reverts the reward structure back to a sparse setting (similar to the Baseline reward structure), which can be problematic in some instances (e.g. training performance of Baseline in the Maze environment 5.1a).

Similarly, selecting K too small can lead to situations where an abundance of novel points can be collected within the confines of the same room (Figure 5.14b). Herein it can be observed that a relatively large intrinsic (curiosity) reward can be collected (*368*) by following the depicted trajectory, with near to no progress made with respect to the objective (mapping the environment). This can lead to a longer converge time during training, however, it is unlikely that no progress is made since the height of the reward is dependant on distance travelled from the novel points in memory. To illustrate this, the intrinsic reward obtained from two distinct trajectories collecting the same amount of novel point are depicted in Figures 5.14b - 5.14c. Herein it can be observed that novel points further away from the already collected set of novel points yields a greater reward (Figure 5.14c) compared to novel points collected in the same approximate area as the already collected novel points (5.14b). Hence, due to the distance term in the curiosity reward structure there is some built-in robustness to the selection of K. To conclude, the parameter K should be selected small enough to make several novelty circles possible within the confines of the starting room to aid the training process, but large enough such that it does not affect behaviour in a detrimental way (e.g. too many circles possible).

5.5 Evaluation

In this section, the strengths and weaknesses and improvements of the proposed approach are analyzed and discussed. One could conclude the following from the made observations so far:

5.5.1 Different reward structures

i It is evident that the Curiosity reward structure has the best results during both training and testing phase. In terms of quality of the trajectory and exploratory behaviour the approach significantly outperformed all other methods including the state-of-art Frontier-exploration.

ii Although Information-gain has higher overall performance during training and testing in comparison to the Oracle reward structure, the quality of the taken trajectories which are successful are significantly worse compared to the Oracle method.

iii Sparse reward functions do not function well in complex and big environments

Furthermore, it seems important that the algorithm is able to map the whole environment within the first 100 episodes as otherwise it seems develop wall avoiding behaviours mainly, which can hurt the performance of the algorithm. It was observed that learning this circling behaviour will make it very difficult for the robot to escape that local optimum in consecutive episodes. Hence, one could conclude that, due to the robot's inability to escape the walls, it develops behaviour which is detrimental to the goal. This is also confirmed by the performance of the Baseline reward structure as showed before. Although obstacle avoidance in and of itself is not undesirable, the current implementation is sub-optimal. Possible solution to improve are:

1. Adding a backwards action to the linear velocity and thereby allowing it to escape walls making the penalty less impactful at the cost of computational complexity.

2. Omitting the penalty for collision entirely or replacing it with an alternative which suits the objective better (e.g. a time-step penalty instead of (or in addition to) the collision penalty).
3. Better balancing of the existing reward functions to decrease the impact of the collision.

5.5.2 Generalization

i The proposed Curiosity approach adapts well to previously unknown (simulation) environments using direct policy transfer. Given that the used environment is similar to the trained environment.

ii The generalization of the Curiosity approach can be improved if started from different positions and with random orientation during training. This comes at the cost of additional time to convergence time.

5.5.3 DDPG vs DRQN

The low-level controller can output continuous actions in the case of the DDPG architecture leading to improved trajectories over a DRQN approach. This helps the generalization capabilities of the algorithm at the cost of additional training time an hence decreased training performance compared to DRQN.

6 Conclusion and future work

6.1 Conclusion

- What are state-of-the-art reward shaping novelties that can be successfully leveraged to aid the exploratory process within the used framework?

  As a first step towards innovation in the reward function, a form of episodic Curiosity in the context of this framework was identified as a prime candidate to significantly improve the exploration process of the robot.

- To what extent can we utilize SLAM in order to improve the reward function and increase the performance of the DDPG approach?

  The proposed reward structures (Baseline, Oracle, Information-gain and Curiosity) have been trained and tested on the Maze and Apartment environment to evaluate the performance within the training environment. It can be concluded that all of the proposed approaches that capitalized on the map information (Oracle, Information-gain, Curiosity), provided by SLAM, are able to successfully navigate and map the Maze and Apartment environments, contrary to the Baseline counterpart. Furthermore, by combining SLAM information with an episodic memory and a concept of reachability in the state-space within the reward definition, the convergence time during training was cut down significantly (more than 50% in the Apartment environment) in both environments.

- To which extent is the algorithm able to generalize in an unseen environment?

  To check the generalization capabilities of the reward structures to different environments, the policy (trained in the Apartment environment) was directly transferred to the a-priori unknown Apartment2 environment. The results indicate that the tested algorithms (Oracle, Information-gain and Curiosity) are able to complete the map in the unseen Apartment2 environment trough direct policy transfer. Particularly the Curiosity approach was generally able to complete the map with shortest trajectories and a tendency to navigate toward doorways. It was also shown that changing the training strategy by letting the robot start from a random position and with random orientation - thereby increasing the diversity of the seen samples - greatly increases the generalization capabilities of the agent at the cost of longer but safer trajectories.

- How does the reinforcement learning approach compare against a frontier based exploration approach?

  The Curiosity reward structure is able to map the environment using a shorter trajectory compared to the state-of-the-art frontier based exploration in the Apartment environment. All reward structures which leverage SLAM information achieve similar results in terms of trajectory length in the Maze environment compared to frontier based exploration. The major benefit of the proposed approach is that the it does not require pre-coded navigational directives in order to function compared to the frontier based exploration algorithm.

- To which extent do the previously described theoretical concerns of DQN show in a practical setting and how does it perform against the used DDPG approach?

  Lastly, the Oracle and Information-gain reward structures were compared to results obtained using a DRQN architecture within the Maze, Apartment and Apartment2 environment. In the compared Maze environment, the concerns of blowing up the actions-space and consequently unsuccessful training of DRQN networks do not show. Paradoxically, the DRQN network trains faster, in terms of training steps, and achieves better results than the proposed DDPG approach within the trained environment. However, the DRQN fails to generalize well, making the proposed DDPG architecture a better candidate for exploring unknown environments.

All in all, the proposed SLAM + Curiosity reward structure can be used to generate safe but efficient trajectories for exploring, navigating and constructing a map in both known and unknown environment. The major drawback of this approach is the additional novelty parameter K. Making this parameter adaptive and deploying this approach in real-life circumstances is the next step for this framework.

6.2 Future work

In this sections directions for future work are discussed.

6.2.1 Real-world experiments

The proposed approach works well for the used simulation environments but extending this work towards deploying the obtained policy on a real robot in real-life scenarios is the next step. The learned policy can hypothetically directly be transferred to the real robot, however, a difference model to capture the difference between the real and simulated system such as the one used in Mustafa (2019) is a good candidate. Additionally, testing the proposed approach in more interesting (dynamic) environments could be a step up towards this.

6.2.2 Adaptive naive curiosity

One of the drawbacks of the naive curiosity approach is the dependency on the novelty parameter k. For every environment, this has to be manually tuned for every specific environment and thus if the unseen environment has a divergent structure, compared to the training environment, the performance can suffer. Making this parameter adaptive (e.g. moving average of the laser readings) could help overcome this difficulty. However, this adaptive distance might uncorrelate the received reward and state-action taken which might require a different network structure or more training time.

6.2.3 Hierarchical reinforcement learning

A weak point of the proposed naive Curiosity approach is mainly looping back trough areas which are already explored as these will provide no additional reward. A hierarchical approach such as option-critic, briefly mentioned in Appendix C, could help overcome this issue. For example, by utilizing the map of SLAM you could have a specialized option for returning to areas which are already explored if the naive exploration (e.g Curiosity) option is not making any further progress.

6.2.4 Imitation learning

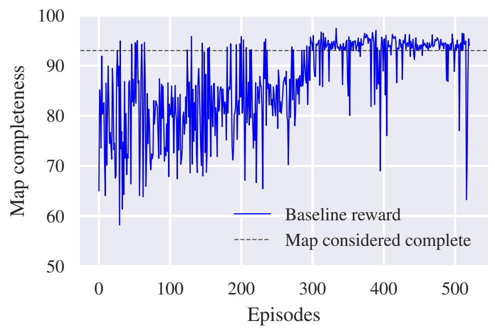
Learning from an expert is not possible in every case, but could be interesting if the environment allows for it. So-called Imitation Learning, as introduced in Appendix C, could significantly reduce convergence time for the proposed approach by letting reinforcement learning agent learn from expert trajectories in a supervised learning setting. This could also be applied as a post-processing step to further improve the generated trajectories

6.2.5 Frontier based exploration

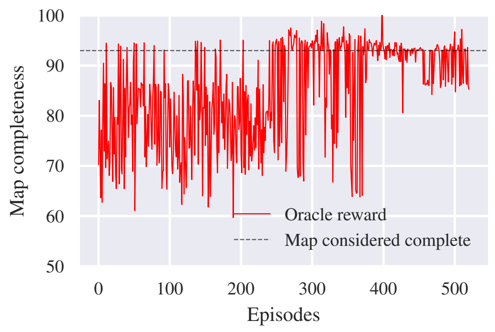
Frontier based exploration is capable of generating a decent navigational target, which could be utilized by the reinforcement learning agent as a target to navigate instead of the traditional, computationally expensive, path planner.

A Raw map-completeness data

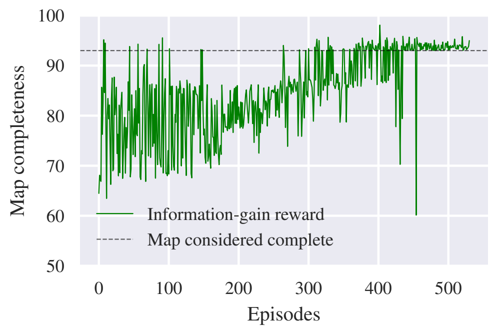
In Figures A.1a - A.1d, the non-smoothed map-completeness over time can be observed. Herein it can be observed that the Baseline, Oracle, Information-gain and Curiosity approaches indeed do complete the map within the first 100 episodes. This is something which cannot be seen in the smoothed map-completeness, however, is vital for convergence of the algorithm.



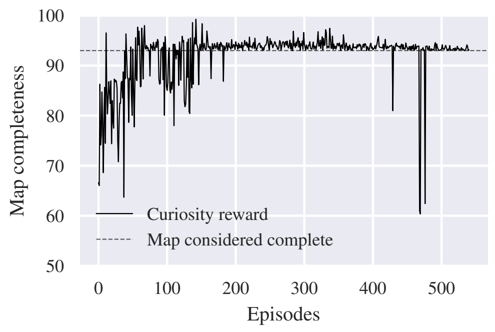
(a) Baseline



(b) Oracle



(c) Information-gain



(d) Curiosity

Figure A.1: Raw map-completeness of the **a)** Baseline, **b)**Oracle, **c)**Information-gain and **d)**Curiosity reward structures over time whilst training on the apartment environment. Dotted line indicates the threshold for the map to be considered complete

B Hyper-parameters

In Table B.1 the hyper-parameters for the SLAM and reinforcement learning approach are showed. It must be noted that dense layer bias is only applied whenever no batch layer normalization is used. This is done because batch layer normalization applies bias on its own and this would forgo the benefits of using the batch layer normalization.

RL and SLAM parameters	Value
optimizer	ADAM
DDPG actor learning rate	10^{-3}
DDPG critic learning rate	10^{-4}
Dense layer bias initialization min value	$\frac{-1}{512}$
Dense layer bias initialization max value	$\frac{1}{512}$
Dense layer weight initialization min value	$\frac{-1}{512}$
Dense layer weight initialization max value	$\frac{1}{512}$
discount factor γ	0.99
Batch size	64
Replay buffer size	$1e6$
particles	80
process scan threshold translation	0.05
process scan threshold rotation	0.05
grid cell size	0.05m×0.05m
occupancy threshold	0.60
LiDAR max. range	10m
LiDAR min. range	0.2m
collision threshold	0.2m
map completed threshold	93 %
novelty distance k maze	0.8
novelty distance k apartment	1.8

Table B.1: Parameters of the experiments.

C Additional reinforcement learning architectures

C.1 Hierarchical reinforcement learning

There is converging evidence in developmental psychology that newborns, primates, children, and adults rely on the same cognitive systems for their basic knowledge. These cognitive systems include entities, agents, actions, space, social structures and intuitive theories. During open-ended games such as stacking up physically stable block structures, toddlers will use this knowledge to set sub-goals. To achieve these goals, toddlers seem to generate sub-goals within the space of their basic knowledge, engaging in temporal abstraction . This is precisely the approach Hierarchical Reinforcement Learning (HRL) takes in order to solve larger and complex problems. The authors of Sutton et al. (1999a) coin the term Options[1] for generalizing primitive actions to include temporally extended courses of action. An Option is a triple $<\mathfrak{I}, \pi, \beta>$ consisting of an Initation set $\mathfrak{I}$, a policy π and a terminal condition β. Options can be taken if they are available from state $s_t \in \mathfrak{I}$ and henceforth follow option policy π until an option terminal condition β is reached. From there a new option can be selected. In this way, specialized and specific tasks can be constructed such as "search the hallway" option depicted in Figure C.1. This enables planning on a per room basis instead of a per cell basis, which significantly speeds up the learning process.

C.2 Imitation learning

Although most achievements in the field focus on including deep learning in reinforcement learning methods, the authors of Hussein et al. (2017) show that learning from an example can result in significant reduction in the policy space. This approach, generally referred to as Imitation Learning, is a form of supervised machine learning and can lead to an efficient learning experience and effective policy from demonstration, if sufficient training examples are available. The authors of Vinyals et al. (2019) show that by combining learning from demonstrations and experience, together with a multi-agent approach, they are able to surpass human level performance in the game of Starcraft II, which is considered the benchmark environment (for games) with 10^{26} possible actions each timestep and a massive solution space. In the area of robotics, the authors of Hussein et al. (2017) show that learning from demonstrations can outperform methods like A3C and DQN in a navigational task. However, one of the big problems with imitation learning in general is the Independent and Identically Distributed (i.i.d.) assumption: while supervised learning assumes that the state-action pairs are distributed i.i.d., in MDP an

[1]For the scope of this thesis only Options are treated, however, other HRL methods include Feudal learning, Hierarchical Abstract Machines, MAXQ and many others

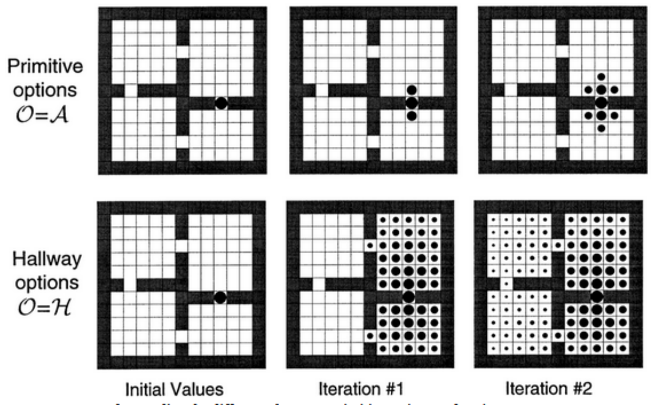


Figure C.1: Different options in the context of a gridworld, adapted from Sutton et al. (1999a)

action in a given state induces the next state, which breaks the previous assumption. This also means, that errors made in different states add up, therefore a mistake made by the agent can easily put it into a state that the expert has never visited and the agent has never trained on. In such states, the behaviour is undefined and this can lead to catastrophic failures as can be seen in Figure C.2. In the case of unknown environments the applicability and usability is limited since it requires human interference, which makes the environments known by definition.

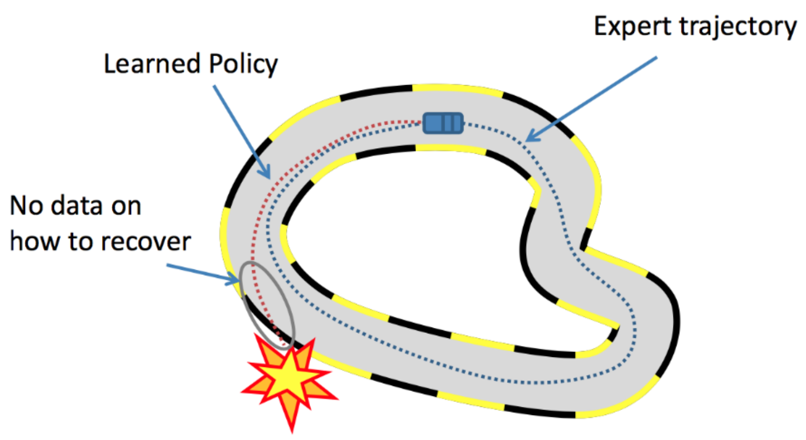


Figure C.2: Failure in imitation learning, adapted from Brunskill (2020)

D Supplementary information of the neural network architecture

D.1 Batch layer normalization algorithm

Co-variate shift can be illustrated by the following example: imagine you are building a neural network and train it on images of cats. You would like to classify images by either cat or non-cat, however, only train it on images of black cats. If you would then show the classifier images of a coloured cats, the classifier would likely not perform very well. The reason is that the distribution of the pixel intensity vector has shifted considerably. By using batch normalization, we can prevent covariate shift by normalizing the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation

```
**Algorithm 2:** Batch normalization
**Result:** $y_i = \text{Batchnorm}_{\gamma,\beta}(x_i)$
$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i \in m} x_i$
$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i \in m}(x_i - \mu_{\mathcal{B}})^2$
$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$
$\mathrm{y}_i \leftarrow \gamma \hat{x}_i + \mathcal{B}$
```

Where ϵ is used to avoid a division by zero (typically in the range of $1e-5$ and parameters γ and β are learned during training along with the model by letting the gradient descent optimizer change only these two weight for each activation instead of changing all weights. This will allow a greater range in the selection of parameter selection and speeds up training, as showed in Ioffe and Szegedy (2015).

D.2 Gradient descent optimizer

For the Adam optimizer, the parameters θ of the stochastic objective function are updated such that:

$$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{D.1}$$

Where α is the step size,$\hat{m}_t$ and $\hat{v}_t$ are the first and second bias-corrected raw moment estimates:

$$\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t} \tag{D.2}$$

$$m_t \leftarrow \beta_1 m_{t1} + (1\beta_1) g_t \tag{D.3}$$

$$\hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2^t} \tag{D.4}$$

$$v_t \leftarrow \beta_2 v_{t1} + (1\beta_2) g_t^2 \tag{D.5}$$

$$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1}) \tag{D.6}$$

Where g_t is the gradient with respect to the objective function $f_t(\theta)$, β_1 and β_2 the exponential decay rates of the moment estimates. The authors propose default values of 0.9 for β_1, 0.999 for β and $10e^{-8}$ for ϵ. They show empirically that Adam works well in practice and compares favorably to other adaptive learning-method algorithms. Furthermore, the robustness of the default values is the reason for Adam to be so popular as it is. With other, comparable (on performance level), methods requiring intensive tuning of the parameters.

E Structured Literature review

In order to establish a proper research question and find an answer to that research question, a structured literature review has been conducted. This method has been chosen because the benefits of conducting a structured literature review are manifold. First of all, by making use of a structured literature review to dive into a novel research area, the current state of research on this topic can be determined. In order to find out to what extent a certain topic has already been studied and gather the existing research results, this method is most adequate. Moreover, the experts on this specific topic can be recognized as well as key questions that still need an answer. Consequently, the way is paved for future research.

E.1 Search terms

The way of researching will involve (accessible) online databases trough which we can gather literature and scientific studies. An obvious choice is the online research database FindUT, which is comprised of multiple scientific libraries. Primarily, the scope of the present research needs to be specified in terms of search terms and entered in the research database.

The pool of search terms includes terms that might provide valuable insight in the problem statements and research questions and are directly related to these. The used terms are “Mobile robots”, “Reinforcement learning”, “SLAM”, “Mapping”, “Navigation”, “Exploration”, "Reward shaping", "Reward function" and "DDPG". In order to receive the most relevant results, two search term combinations have been compiled and entered into the online research database. The quotation marks have been added to the search terms while browsing the different online databases in order to make sure that they appear as the whole established term in the articles and lead to the most relevant search results.

The used combinations of search terms can be found in Table E.1. The first search term combination of "reinforcement learning" , "DDPG" and "exploration" is specifically aimed at finding exploratory strategies suitable for the the DDPG framework. Furthermore, the second search term combination is aimed at finding exploratory strategies which are not directly applicable in the DDPG framework, but with some adaptation are usable in the framework. These terms include "reinforcement learning", "exploration", "mobile robots" and "navigation". In addition papers and derivatives in the previous work and recommendations by the supervisors are analyzed.

E.2 Inclusion criteria

Several general inclusion criteria have been determined as means to decide on which articles to use in this structured literature review. The inclusion criteria were: (i) The language in which the articles are composed had to be English in order to make the analysis more convenient and coherent and provide a set of references in a language the readers of this research are able to understand; (ii) the articles had to be peer-reviewed; and (iii) the publications have to be recent for which we selected a time-frame of 5 years.

Table E.1: Used search term combinations for the literature review process

Search terms combinations	
1st	"Reinforcement Learning" AND "DDPG" AND "exploration"
2nd	"Reinforcement Learning" AND "exploration" AND "mobile robots" AND "navigation"

Table E.2: Reduction process of the found articles for the first search term

Stepwise reduction of articles (first term)	Number of residual articles
Searching for articles in Databases	277 hits
Filtering on type:article	14 hits
Filtering on published last 5 years (2015 - 2020)	12 hits
Checking titles for adequacy	5 hits
Analyzing abstracts concerning the inclusion criteria	2 hits

Table E.3: Reduction process of the found articles for the second search term

Stepwise reduction of articles (second term)	Number of residual articles
Searching for articles in Databases	421 hits
Filtering on type:article	410 hits
Filtering on published last 5 years (2015 - 2020)	142 hits
Checking titles for adequacy	7 hits
Analyzing abstracts concerning the inclusion criteria	2 hits

Bibliography

Abbeel, P. (2006), Lecture notes on gmapping.
https://people.eecs.berkeley.edu/~pabbeel/cs287-fa11/slides/gmapping.pdf,

Bellemare, M. G., Y. Naddaf, J. Veness and M. Bowling (2012), The Arcade Learning Environment: An Evaluation Platform for General Agents, *CoRR*, **vol. abs/1207.4708**.
http://arxiv.org/abs/1207.4708

Bishop, C. M. (2009), *Pattern Recognition and Machine Learning*, Springer Science+Business Media, LLC, ISBN 978038731073-2.

Botteghi, N., B. Sirmacek, R. Schulte, M. Poel and C. Brune (2020), REINFORCEMENT LEARNING HELPS SLAM: LEARNING TO BUILD MAPS, *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, **vol. XLIII-B4-2020**, pp. 329–335, doi:10.5194/isprs-archives-XLIII-B4-2020-329-2020.
https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLIII-B4-2020/329/2020/

Brunskill, E. (2020), Lecture 7: Imitation Learning in Large State Spaces.
http://web.stanford.edu/class/cs234/slides/lecture7.pdf

Burda, Y., H. Edwards, D. Pathak, A. Storkey, T. Darrell and A. A. Efros (2018), Large-Scale Study of Curiosity-Driven Learning.

Dissanayake, G., H. Durrant-Whyte and T. Bailey (2000), A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem, in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pp. 1009–1014 vol.2.

Doucet, A., N. de Freitas, K. Murphy and S. Russell (2013), Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks.

funda (2020), https://www.funda.nl/, https://www.funda.nl/ (accessed: April 2020), house finding website in the Netherlands.

Grisetti, G., C. Stachniss and W. Burgard (2005), Improving Grid-based SLAM with Rao-Blackwellized Particle Filters By Adaptive Proposals and Selective Resampling, pp. 2432–2437, doi:10.1109/ROBOT.2005.1570477.

Guivant, J. E. and E. M. Nebot (2001), Optimization of the simultaneous localization and map-building algorithm for real-time implementation, **vol. 17**, no.3, pp. 242–257.

Holz, D., N. Basilico, F. Amigoni and S. Behnke (2010), Evaluating the Efficiency of Frontier-based Exploration Strategies, pp. 1 – 8.

Hussein, A., E. Elyan, M. Gaber and C. Jayne (2017), Deep imitation learning for 3D navigation tasks, *Neural Computing and Applications*, **vol. 29**, doi:10.1007/s00521-017-3241-z.

Ioffe, S. and C. Szegedy (2015), Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *CoRR*, **vol. abs/1502.03167**.
http://arxiv.org/abs/1502.03167

Kingma, D. and J. Ba (2014), Adam: A Method for Stochastic Optimization, *International Conference on Learning Representations*.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra (2015), Continuous control with deep reinforcement learning.

Liu, J. S. (1996), Metropolized Independent Sampling with Comparisons to Rejection Sampling and Importance Sampling.

Matheron, G., N. Perrin and O. Sigaud (2019), The problem with DDPG: understanding failures in deterministic environments with sparse rewards.

Merwe, R., A. Doucet, N. Freitas and E. Wan (2001), The Unscented Particle Filter, *NIPS*, **vol. 13**.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. A. Riedmiller (2013), Playing Atari with Deep Reinforcement Learning, *CoRR*, **vol. abs/1312.5602**. http://arxiv.org/abs/1312.5602

Mnih, V., K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis (2015), Human-level control through deep reinforcement learning, *Nature*, **vol. 518**, pp. 529–33, doi:10.1038/nature14236.

Montemerlo, M., S. Thrun, D. Koller and B. Wegbreit (2002), FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem, in *In Proceedings of the AAAI National Conference on Artificial Intelligence*, AAAI, pp. 593–598.

Montemerlo, M., S. Thrun, D. Koller and B. Wegbreit (2003), FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges, *Proc. IJCAI Int. Joint Conf. Artif. Intell.*

Moravec, H. and A. Elfes (1985), High resolution maps from wide angle sonar, pp. 116 – 121, doi:10.1109/ROBOT.1985.1087316.

Murphy, K. (2000), Bayesian Map Learning in Dynamic Environments.

Mustafa, K. A. A. (2019), Towards Continuous Control for Mobile Robot Navigation: A Reinforcement Learning and SLAM Based Approach. http://essay.utwente.nl/79803/

Mustafa, K. A. A., N. Botteghi, B. Sirmacek, M. Poel and S. Stramigioli (2019), TOWARDS CONTINUOUS CONTROL FOR MOBILE ROBOT NAVIGATION: A REINFORCEMENT LEARNING AND SLAM BASED APPROACH, *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, **vol. XLII-2/W13**, pp. 857–863, doi:10.5194/isprs-archives-XLII-2-W13-857-2019. https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-2-W13/857/2019/

Pardo, F., A. Tavakoli, V. Levdik and P. Kormushev (2017), Time Limits in Reinforcement Learning, *CoRR*, **vol. abs/1712.00378**. http://arxiv.org/abs/1712.00378

Pathak, D., P. Agrawal, A. A. Efros and T. Darrell (2017), Curiosity-driven Exploration by Self-supervised Prediction.

Plappert, M., R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel and M. Andrychowicz (2017), Parameter Space Noise for Exploration.

Savinov, N., A. Raichuk, R. Marinier, D. Vincent, M. Pollefeys, T. P. Lillicrap and S. Gelly (2018), Episodic Curiosity through Reachability, *CoRR*, **vol. abs/1810.02274**. http://arxiv.org/abs/1810.02274

Schulman, J., P. Moritz, S. Levine, M. Jordan and P. Abbeel (2015), High-Dimensional Continuous Control Using Generalized Advantage Estimation.

Schulte, R. (2019), Autonomous mapping and navigation of a mobile robot using Reinforcement Learning - Individual Assignment.

Silver, D. (2015a), Lecture notes on Markov Decision Processes. https://www.davidsilver.uk/teaching/

Silver, D. (2015b), UCL Course on RL: Lecture 7 - Policy Gradient Methods. https://www.davidsilver.uk/wp-content/uploads/2020/03/pg.pdf

Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller (2014), Deterministic Policy Gradient Algorithms, *31st International Conference on Machine Learning, ICML 2014*, **vol. 1**.

Sutton, R., D. Precup and S. Singh (1999a), Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning, *Artificial Intelligence*, **vol. 112**, pp. 181–211.

Sutton, R. S. and A. G. Barto (1998), *Introduction to Reinforcement Learning*, MIT Press, Cambridge, MA, USA, 1st edition, ISBN 0262193981.

Sutton, R. S., D. McAllester, S. Singh and Y. Mansour (1999b), Policy Gradient Methods for Reinforcement Learning with Function Approximation, in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, MIT Press, Cambridge, MA, USA, NIPS'99, p. 1057–1063.

Thrun, S. (2000), An Online Mapping Algorithm for Teams of Mobile Robots, *Int. J. of Robot. Research*, **vol. 20**.

Uhlenbeck, G. E. and L. S. Ornstein (1930a), On the Theory of the Brownian Motion, *Phys. Rev.*, **vol. 36**, pp. 823–841, doi:10.1103/PhysRev.36.823.
`https://link.aps.org/doi/10.1103/PhysRev.36.823`

Uhlenbeck, G. E. and L. S. Ornstein (1930b), On the Theory of the Brownian Motion, *Phys. Rev.*, **vol. 36**, pp. 823–841, doi:10.1103/PhysRev.36.823.
`https://link.aps.org/doi/10.1103/PhysRev.36.823`

Vinyals, O., I. Babuschkin, W. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. Agapiou, M. Jaderberg and D. Silver (2019), Grandmaster level in StarCraft II using multi-agent reinforcement learning, *Nature*, **vol. 575**, doi:10.1038/s41586-019-1724-z.

Watkins, C. and P. Dayan (1992), Technical Note: Q-Learning, *Machine Learning*, **vol. 8**, pp. 279–292, doi:10.1007/BF00992698.

Yamauchi, B. (1997), A frontier-based approach for autonomous exploration, in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pp. 146–151.

Zhang, C., O. Vinyals, R. Munos and S. Bengio (2018), A Study on Overfitting in Deep Reinforcement Learning, *CoRR*, **vol. abs/1804.06893**.
`http://arxiv.org/abs/1804.06893`