Siniša Šteković, BSc

Reinforcement Learning With Deep Networks And A Robot

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dr. Vincent Lepetit

Institute of Computer Graphics and Vision
Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof

Graz, February 2018

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Acknowledgements

I am deeply grateful to my family, girlfriend and friends for supporting, encouraging and believing in me during the complete course of my studies. I am indepted to all of you and hope I can somehow return the favor.

I thank the group of researchers in the Prof. Lepetit's team at the Institute of Computer Graphics and Vision who accepted me and offered all kind of advices during the course of my master's thesis.

Special thanks go to Slaven Šteković, Darjan Salaj and Anil Armagan for preliminary reviewing the individual chapters of my thesis. I also want to thank Nataša Tadić for helping me with many design choices.

I would like to express my deepest appreciation to Prof. Lepetit for being an amazing supervisor, for finding enough time for me in his tight schedule and for guiding me responsibly throughout the whole thesis.

Abstract

The goal of this master's thesis is to investigate reinforcement learning methods for the task of autonomous robotic navigation in corridor environments through monocular images. In our setup, the agent explores the given corridor environment and learns to predict the rewards for the state-action pairs. During the learning process, we use the pose estimation and the bumper sensor of the robot to compute a reward signal. For predicting the reward, we only use monocular images. We utilize the experience replay approach to decorrelate the data during training. Herein, we introduce a novel method to show the importance of considering the sample similarity in the replay memory during the learning process. We propose an approach for managing the replay memory that discards only the similar samples from the replay memory. Therefore, we ensure that the samples in the replay memory are less correlated and offer good representations of the given environment. Due to the complexities of the evaluation in the real world, we perform our experiments in a simulated environment. We evaluate the ability of our agent to learn more complex corridor environments, step-by-step. First, we evaluate the learning process in a simple corridor and compare our replay memory management strategy to the standard discard phase for the replay memory. Second, we evaluate the capabilities of our agent for learning two corridor environments. Finally, we evaluate the learning process in a complex environment. The results indicate that the agent is able to apply the acquired knowledge well for the tasks at hand. Additionally, the replay memory management approach enhances the ability of the agent to learn multiple corridor environments.

Contents

List of Figures

1 Introduction

The autonomous navigation of robots in indoor environments is a challenging task. Usually, the methods propose the usage of complex hierarchies of sensors that enable a robot to complete long and smooth trajectories in the given environment. In this context, the related works often refer to Simultaneous Localization And Mapping (SLAM), as described in [20], the famous problem that correlates the task of navigation in a corridor environment with constructing a detailed map of the environment and providing a good localization estimation for the robot. In order to estimate the map of the environment, we need to be able to extract the information recorded by the available sensors precisely. The problem includes estimating the distances to the obstacles in the environment. The advanced solutions often utilize different sensors such as laser rangefinders or RGBD-sensors [6, 11] for such calculations. The solutions may therefore imply complex calibrations of the involved sensors and large costs for the deployment.

In contrast to the mentioned approaches, in this thesis we address the problem of mapless navigation and object avoidance in corridor environments using sensory data received from a monocular image. Through this, we reduce the required complexity of the underlying sensor hierarchy.

Even though calculating the distances in a monocular image is a hard task, humans are able to perceive the distances just by observing a given monocular image and can predict the most appropriate strategy for avoiding the obstacles by applying the previously learned context to the given image.

The goal of this thesis is to investigate the methods that we use today to extract the context from images and to develop a model which correlates the extracted context and learning through self-motion. The model is evaluated on a robot that learns to navigate in a given corridor environment. The detailed project specifications are presented in Chapter 2.

In this chapter we first cover one of the early neuroscience experiments that contributes to understanding the importance of self-motion for the learning progress. Further on, we discuss why reinforcement learning might be a good choice for modeling the concept of learning from self-motion. We briefly discuss the methods behind the Deep Q-Network (DQN) approach [13, 14] that exceeds the human-level of performance in the domain of playing ATARI 2600 games [1] and through that inspires our approach. We conclude the chapter by providing the problem statement for this thesis and we briefly introduce our approach and its advantages.

1.1 Motivation From Neuroscience

Learning from previous experiences influences our development. Even though it might be helpful to study the behavior of others to faster grasp the solution for the given task, sometimes we are able to find a solution only after we have experienced cycles of trial and error for the task at hand. What we can do today without a lot of thinking, might be the result of hard work. Although we do not even remember our own first steps, by observing the children today that go through the process of learning the same task, we can imagine how much work we had to invest for solving it.

To examine the influence of self-motion on the learning progress, Held and Hein conducted an experiment in 1963 that showed the importance of self-motion for learning the visual perception [5]. They performed the experiment on a pair of kittens, the active cat and the passive cat. The goal was to investigate the role of self-motion for the development of the action-perception. During the early stage of life, the cats were tied to a carousel-like structure. The active cat was given the freedom of walking around the structure. By moving freely, it was rotating the lever which in turn resulted in moving the passive cat, constrained to keep it from walking on its own. This way both the active and the passive cat received an equal amount of visual experience but the active cat was able to connect the visual experience to the range of movements it was performing during this time. The passive cat was not able to develop the functional sight during the experiment as it did not learn to perceive depth and was distinguishable

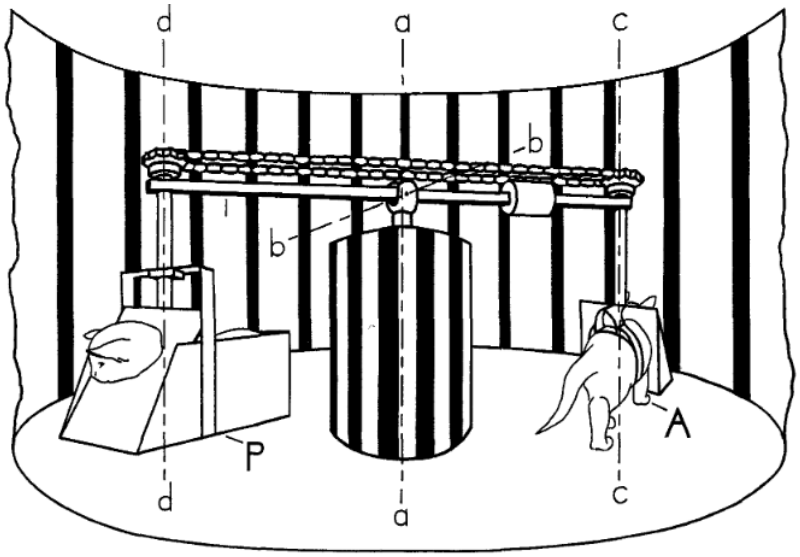


Figure 1.1: The active-passive cat experiment [5]. By walking around the corousel, the active cat A moves the lever which in turn moves the passive cat P.

among other cats by its performance in the open world. Active cat was able to perceive depth and perform in the world like any other cat. Only after the passive cat was given the freedom of the active movement for a prolonged period of time, it learned to perceive depth. The setting for the active-passive cat experiment is shown in Figure 1.1.

The experiment shows an example of the developmental process. The self-induced movement produces a variation of stimulus which influences the learning process in our brain. We ask ourselves the following question: As the self-motion is so important for our visual system, can we use it as an inspiration for developing machine visual systems that perform and perceive the world in a similar way as we do?

Herein, we investigate the autonomously learning systems that correlate to the notion of learning through self-motion. More specifically, we utilize the current progress in the field of reinforcement learning which fits the requirements for our model well.

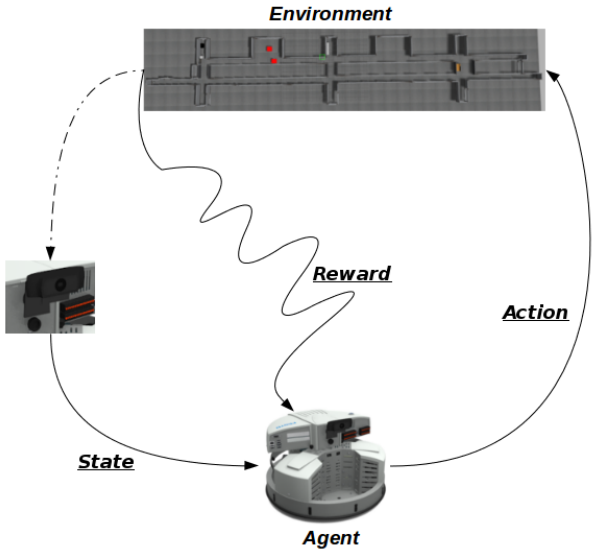


Figure 1.2: Reinforcement learning, typical scenario. The agent observes a state from the environment and chooses an action based on the observation. The agent associates a certain noisy reward to the state-action pair through the interaction with the environment.

1.2 Reinforcement Learning

When defining a machine learning problem, we have to think about the type of signal that is provided to the algorithm for the learning process. In the standard supervised learning task, the signals for the available data samples are precisely defined, usually with some sort of human interaction. The most typical examples of the supervised learning problems are the image classification tasks. In such tasks the learning process is guided by a dataset of images, carefully labeled to be categorized into the different classes that the algorithm should learn to recognize. The key component of the recent success in the field of image classification is the utilization of the Convolutional Neural Networks(CNN) [9]. The ImageNet Large Scale Visual Recognition Challenge [17] is just one such example task where outstanding results have been achieved in the last years.

In contrast to the supervised learning, in the field of *reinforcement learning* we deal with problems where it is hard or impossible to label the data

exactly. Therefore, the learning process is guided by an approximation of the correct signal, the *reward*. The typical setting for the reinforcement learning problems is as follows: An *agent* is placed into an *environment* where it gathers data samples consisting of an *observation state*, e.g. image sensory data, and an *action*, which leads to the transition to the next state. The agent receives a reward signal for each of the state-action pairs based on how the agent interacts with the environment for the chosen action in the given state. The reward in the most of the reinforcement learning scenarios is often delayed and noisy, making it difficult to design algorithms that deal with such problems. We illustrate a typical reinforcement learning scenario in Figure 1.2.

Different reinforcement learning methods have been successfully applied to the challenging task of playing ATARI 2600 games [1] recently. In such tasks, the observation state is usually the current game frame (or multiple game frames) and the reward is the score change after the agent performs a specific action. The DQN method [13, 14], for which the agent learned to play the ATARI 2600 games above the human-level, demonstrates a possible application of the reinforcement learning algorithms to solve this task.

1.2.1 Deep Q-Network for playing ATARI 2600 games

The work by Mnih et al. in 2013 [13] introduces a novel model for reinforcement learning, the DQN, that masters a wide variety of ATARI 2600 games by using only the raw pixels of the image frame for the observation state. The idea is to extend the Q-Learning method, the acknowledged traditional approach, through deep learning.

The Q-Learning approach is the basis of the DQN method. As presented originally in 1989 by Watkins in his PhD thesis [22], the Q-Learning enables learning from the delayed rewards and shows convergence to the optimal solutions of the problems where the Markov property can be assumed for the defined observation states. The motivation for the Q-Learning approach is set from the assumption that the optimal action policy is the one that maximizes the future reward. Therefore, Watkins suggested that one way of learning such policy would be to propagate the delayed reward

through the previously visited state-action pairs. This idea is portrayed by the function $Q(s,a)$, also referred to as the action-value function in [13], that represents the estimation of the expected delayed reward for the given state-action pair (s,a). During the training phase, Q-Learning suggests the update of the action-value for the visited state-action pairs (s_t, a_t) at time step t, represented in Equation 1.1:

$$Q(s_t,a_t) \leftarrow \begin{cases} r_t & \text{if } s_t \text{ is terminal,} \\ (1-\alpha)Q(s_t,a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1},a)) & \text{otherwise,} \end{cases} \tag{1.1}$$

where α is the learning rate, r_t the immediate reward received at the time step t and $max_a Q(s_{t+1},a)$ the prediction of the future reward discounted by the factor γ. The state is terminal if there are no other states that follow. In other words, it is the state that ends the episode. For ATARI 2600 domain an episode is usually considered to be the interval between the start and the end of the game.

With such definition, the Q-Learning approach is limited to solving the tasks with very low state complexities. To solve this, for the DQN method the function Q is represented by a CNN. The observed state s is obtained by stacking the last 4 recorded image frames. The network takes the observed state as the input and outputs the predictions of the rewards for each of the defined actions in the environment.

The actual rewards for the ATARI 2600 games are represented in the DQN method by the change in the game score. The positive score changes are fixed to 1 and the negative score changes are fixed to -1. No score change implicates the 0 reward. The end of the game marks the end of the episode and afterwards the game restarts from the initial state.

The DQN method introduces the *experience replay* concept. The idea is to maintain a list of the previously visited state-action-reward triples, the *replay memory*, that contains the previous experiences of the agent gathered in the earlier iterations of the learning process. By randomly sampling a batch from the replay memory at the training time, the correlation between the data is smaller and therefore the experience replay ensures the low chances

of overfitting the network. As the replay memory is limited in size, the standard approach is to discard the oldest samples once the replay memory is full.

Additionally, the DQN also utilizes the ϵ-greedy action policy. With small probability ϵ, the agent takes random actions instead of the action for which the highest future reward is predicted. Through non-deterministic behavior, such policy encourages the agent to explore the environment and discover the potentially more rewarding state-action pairs.

Finally, the work of Mnih et al. from 2015 [14] makes one additional improvement on the work from 2013 [13] by introducing the *target network*. This approach separates the network for training, the training network, and the network that predicts the rewards, the target network. Through this, the samples are generated using the action policy that depends on the target network for multiple iterations making the variations between the rewards for the similar gathered samples smaller. After certain number of steps C, the target network is updated to fit the values of the training network. The work in [14] also presents the separate results of the training for a variety of ATARI 2600 games. One part of the visualization from [14], in Figure 1.3, shows the behavior of the learned agent for the game of Pong. The approach is still challenged by the games that require more demanding planning strategies where the learning does not converge to the optimal game strategies.

1.3 Problem Statement

In our reinforcement learning scenario, the agent learns to navigate in the given corridor environment. To solve this task, as implied throughout this chapter, we develop a model which introduces the aspect of learning from self-motions through a reinforcement learning approach inspired by the DQN method [13, 14].

Our agent explores the given corridor and it learns to navigate and complete long trajectories in the environment while avoiding obstacles by using the gathered experience. The reward signal is the distance that our agent covers

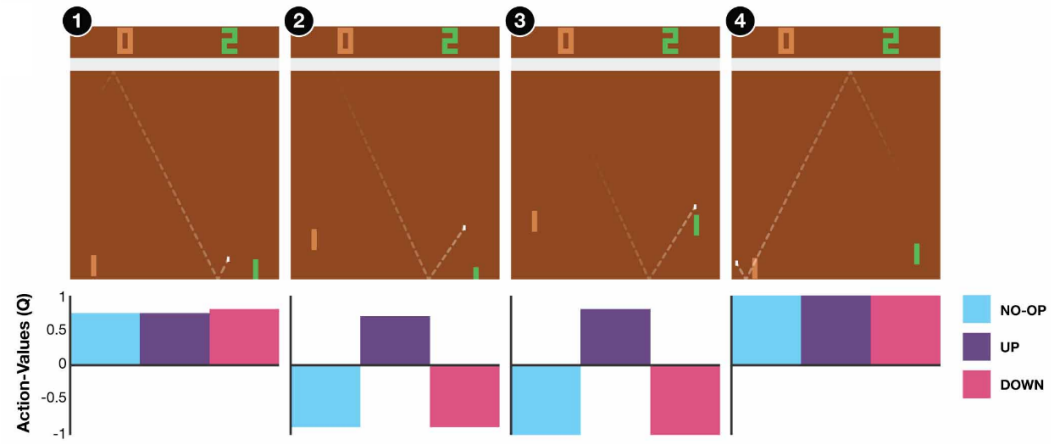


Figure 1.3: DQN, the learned predictions, adapted from [14]. The image shows the development of the learned Q-function for Pong in different situations of the game. In the first frame, the state is of a low risk and therefore the Q-function predicts high future rewards around 0.7 for all of the actions. As the risk increases in frames 2 and 3, the Q function predicts the negative scores for the non-optimal actions but keeps the score for the optimal action high. In the frame 4, the opponent is about to be beaten, therefore the predictions are ~ 1 for all actions.

for the corresponding state-action pair. In order to be able to estimate the reward, our agent has to learn to distinguish between the observation states defined by a single monocular image. During the learning process, we calculate the positive part of the reward by measuring the covered distance between different steps. We use the bump information as the negative part of the reward. The trained agent predicts the rewards for the state-action pairs based on a single monocular image. The agent chooses which action to take by relying on the defined action policy. Therefore, during the learning process we do not require the high-precision sensors. Our algorithm relies only on the camera mounted on the top of the robot, the bumper sensor and the noisy estimations of the covered distances. At the test time, the algorithm relies solely on the image data captured by the camera.

Although the usage of an continuous action space is arguably a better choice for the real world navigation scenarios, we choose to define a simple discrete action space for this problem. By allowing it to move forward with possibility of slight rotations to the left or to the right, the agent can choose between three different actions. Such simple definition of the discrete action

space allows the straightforward application of the techniques presented in the DQN approach for playing ATARI 2600 games in [13, 14]. The possibility of extending our approach to a more complex continuous action space is discussed in 5.3.1.

We build our model under the assumption that, in contrast to the game environments, the position of the robot in a real world cannot be just reset to the initial position. Therefore, the initial state for each trial differs from the previous one which might implicate additional complexity for our task.

Finally, learning the new corridor environments should not result in forgetting the knowledge about the previously learned corridors. Therefore, we adapt the standard experience replay approach and introduce the concept of replay memory management. By discarding only the similar samples from the replay memory, we ensure lower correlation between the samples. Hence, the agent can preserve the knowledge of the previously explored corridors. Similarly, the agent can preserve the knowledge about the less frequent situations in a single complex corridor environment.

We evaluate our agent on multiple challenges. First, we compare our approach that utilizes the replay memory management to the standard approach of discarding the oldest samples from the replay memory. We perform the comparison on the task of learning a simple corridor environment. Afterwards, we evaluate the ability of our agent to learn multiple corridor environments. Finally, we task our agent with learning a complex corridor environment. We demonstrate through the evaluations that our agent is capable of solving the tasks for which the standard experience replay approach is not fit by its definition. Hence, we conclude that considering the sample similarity in the replay memory enhances the possible applications for the experience replay.

2 Project Specifications

The specified problem for this thesis requires the definition of different aspects that enable the eventual implementation and the evaluation of our solutions. In this chapter we present various project definitions and choices considering the hardware and software components. We describe the omnidrive robot that we use to explore the corridor environments. Furthermore, we introduce the difficulties and the constraints that appear during the learning process in the real world. Therefore, we conduct our experiments in a simulated environment to overcome those difficulties. Herein, we also include the descriptions of different corridor environments for the evaluation of our approach. Each of the individual corridors is associated with different challenges that our agent must overcome in order to be able to maneuver smoothly. Finally, we present a software framework for establishing the communication channels between the robot and our software components.

2.1 Robotino®

The first step is to define the "body of the agent" through which the actions are transformed into the actual motions in the given corridor. In this section, we introduce Festo Robotino® [1] [19], a mobile robot platform that fits the requirements for our experiments. Figure 2.1 shows the model of Robotino.

Robotino® is equipped with three independent drive units to enable the omnidirectional drive, in other words the movement in all directions and the inplace rotation. HD Webcam, mounted on Robotino®, provides the

[1] http://www.festo-didactic.com/int-en/services/robotino

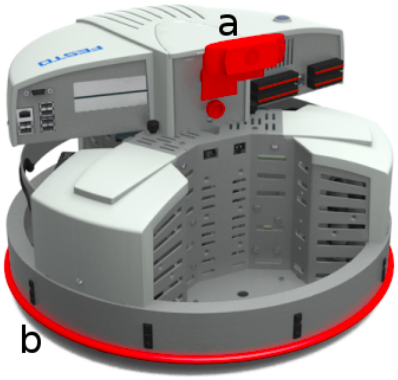


Figure 2.1: Robotino® model, the image adapted from the official web site [a]. Sensors marked red are required for conducting our experiments. a) Camera sensor is mounted onto the front panel of Robotino®. b) Bumper sensor surrounds the base of Robotino®.

[a] http://www.festo-didactic.com/int-en/services/robotino

observation state in our reinforcement learning scenario. The bumper sensor on the bottom edge of the chassis of Robotino® provides the bump information necessary for the definition of the negative rewards in Section 3.2. We use the built-in algorithms to estimate the change in position for the calculation of the positive part of rewards. The algorithms integrated in Robotino® estimate the position change by analyzing the data gathered through the odometry and the gyroscope sensor. Robotino® establishes the communication with the communication partner through its wireless access point.

Furthermore, even though we do not rely on some of the functionalities for our approach, Robotino® provides interesting features that might be useful for the extensions of our approach. The basis of Robotino® is equipped with nine distance sensors to examine the immediate surrounding. Additional modules, such as the robotic arm, are available to extend the number of possible use cases.

However, the constraints of the real world environments imply difficulties for the learning process. First of all, due to the battery consumption, the duration of one experiment is limited to several hours only. Second, the real environments might imply variations of the observation state, e.g. the

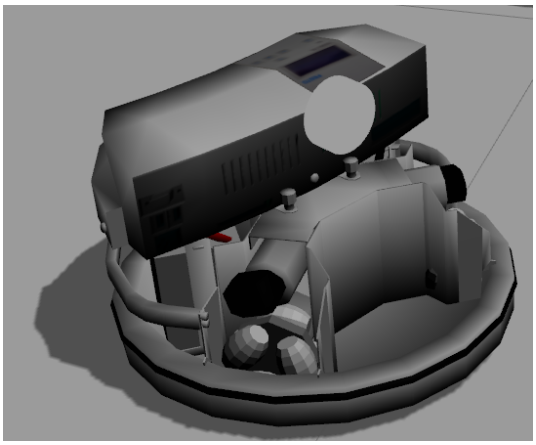


Figure 2.2: Simulated Robotino® model[a] from [24] was adapted to fit our experiments .
[a] http://www.robocup-logistics.org/

illumination changes or the moving obstacles in the scene. While these variations certainly should not be neglected in the real case scenarios, such setting makes the learning and the evaluation processes much harder. Hence, we look for a simulation environment that is fit to fulfill the requirements for the task.

2.2 Gazebo Simulation

Gazebo [2] is a tool for running robotic simulations with integrated physics engine and quality graphics. Because of its simple programmatic interface and its open source nature, it is a good simulator choice for conducting our experiments.

Moreover, for the purpose of creating a simulation environment for RoboCup Logistics League (RCLL) [3], Zwilling et al. presented in 2014 a simulated Robotino® model[24] that can be integrated into Gazebo simulations. Figure 2.2 shows the simulated Robotino® model. As the model lacks the simulation of a bumper sensor, we adapt the model to include this feature.

[2] http://gazebosim.org/
[3] http://www.robocup-logistics.org/

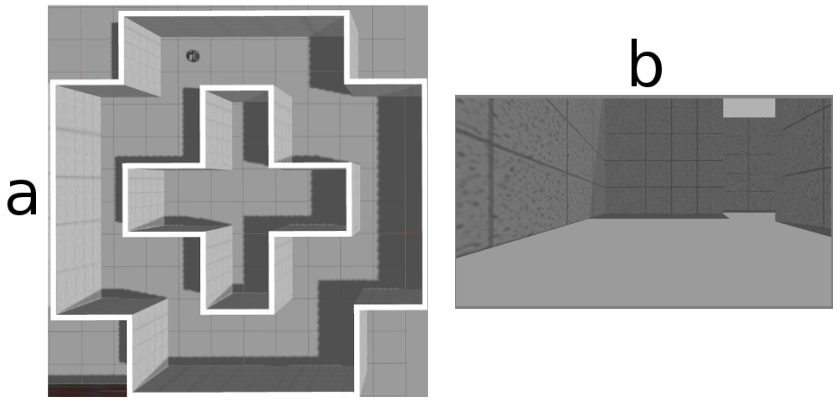


Figure 2.3: Plus corridor. a) The model of the corridor. b) The corridor as seen by the agent. Due to the shape of the corridor, the agent will often encounter situations where the wall corners are not in its field of view. Therefore, in order to perform long and smooth trajectories, it is useful to learn to avoid such states. Additionally, the agent might switch the direction of traversing the corridor, implying additional complexities to the learning process.

The adapted model is therefore suitable to provide the required sensory data for the simulated environment. Additionally, in the Gazebo simulation environment, we are able to reduce the execution time for each action for the agent and therefore reduce the time for performing the experiments significantly.

2.2.1 Corridor Models

For the purpose of evaluation, we present three corridor models, shown in Figures 2.3, 2.4, 2.5. Each of the models induces certain challenges that test different skills that are required for the task of navigation in corridor environments. Herein, we test the agent on difficult challenges, such as:

- nearby wall corners that are not in the field of view,
- dead ends,
- different wall textures,
- windows,
- obstacles,
- and more.

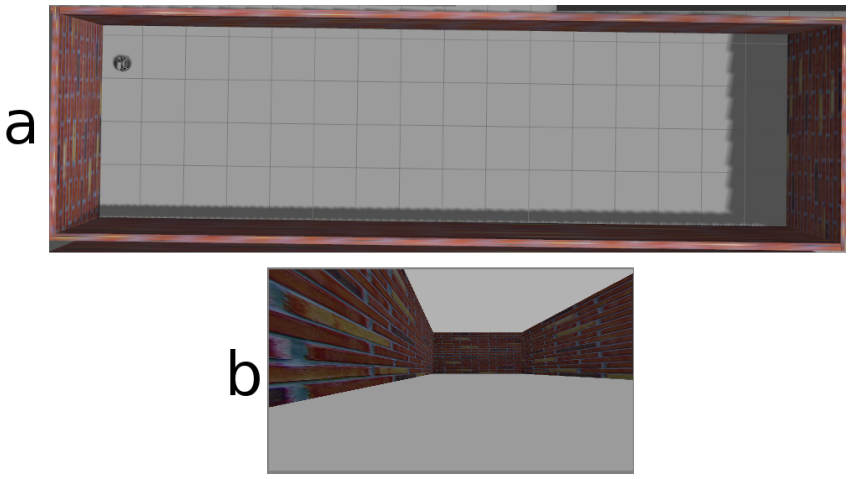


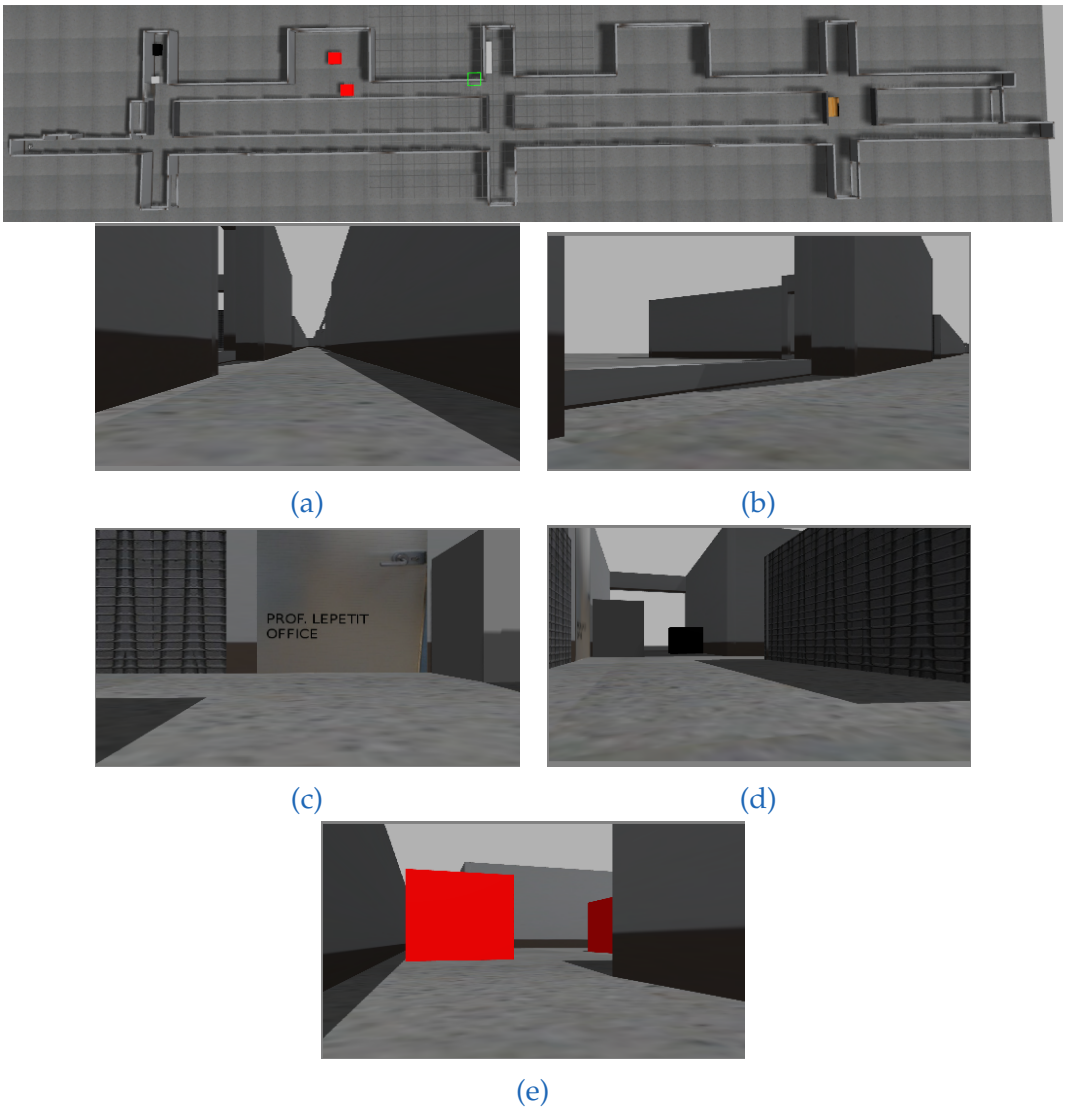
Figure 2.4: Minus corridor. a) The model of the corridor. b) The corridor as seen by the agent. The agent has to learn to perform the U-turn. In other words, it has to reverse the movement direction once it has reached the dead end. At the same time, the definition of the reward should not encourage the agent to drive in small circles around one point.

2.3 Robot Operating System

Robot Operating System (ROS) [4] [15] is a software framework widely used as an communication interface between the robotic application and the robot itself. The communication between the nodes is based on subscribing and publishing to topics, the named buses over which the nodes exchange the messages. Additionally, ROS offers a large set of tools for diagnosing and visualizing the data about the robot but which is of less interest for the purpose of this thesis.

As Robotino® is compatible with ROS, we use it for setting up the communication channels. In our case, we subscribe to the topics for reading image frames, bumper and pose data. We publish to the topic that controls the individual wheel engines of Robotino® and, in turn, induces motions in the desired direction.

[4] http://www.ros.org/



(a) (b) (c) (d) (e)

Figure 2.5: Inffeldgasse 16c (First Floor) corridor (I16c). The model is based on a real corridor, located at Inffeldgasse 16c, 8010 Graz, Austria. a) The narrow corridor encourages the agent to move straight forward. b,c,d) For the largest part, the wall textures are similar but, occasionally, the agent runs into different wall textures, e.g. the bridges that connect the different parts of the corridor, windows, or the doors to Prof. Lepetit's office. The areas that are not encountered often are harder to learn. Therefore, the agent might overfit to the areas that it visits often and not react well in the less frequently visited areas. c) The texture of the doors to Prof. Lepetit's office only occurs once in the complete corridor. e) Additionally, the corridor contains simple obstacles that the agent should learn to bypass. Finally, the dead ends and the narrow pathway for the agent in this corridor imply that the agent requires the possibility of rotating in place at the test time.

2.4 Online Repository

The repository containing the adapted simulated model of Robotino®, the corridor models and videos that demonstrate the behavior of the agent for some of the experiments can be found at https://github.com/sinisastekovic/Reinforcement-Learning-With-Deep-Networks-And-A-Robot.git (State: February 2018).

3 Reinforcement Learning With Deep Networks And A Robot

In this chapter, we present our approach for learning corridor navigation with monocular images. The agent explores the given corridor and learns over time to estimate the optimal actions to take for the given observation states. Throughout this chapter, we present our reinforcement learning approach and the individual components that we utilize during the learning process. The main goal of our approach is to show the importance of decreasing the correlation of the data for the training phase.

Section 3.1 presents the overview of our approach. We present the intuition of using experience replay approach for the task of corridor navigation. The later sections introduce the individual components of the algorithm and explain the algorithmic choices for the components.

3.1 Learning Corridor Navigation With Monocular Images

In this section, we give an overview of our approach for learning corridor navigation with experience replay, with the algorithm flowchart and the pseudo code shown in Figure 3.1, 3.2 respectively.

In the initialization phase, the algorithm sets the parameters of the neural networks, see Section 3.4. The concept of the train and the target network is presented in Section 3.4.3. Then, the algorithm initializes the replay memory to contain N samples. The main advantage of the replay memory is that in the train phase we do not consider only the last gathered samples. By

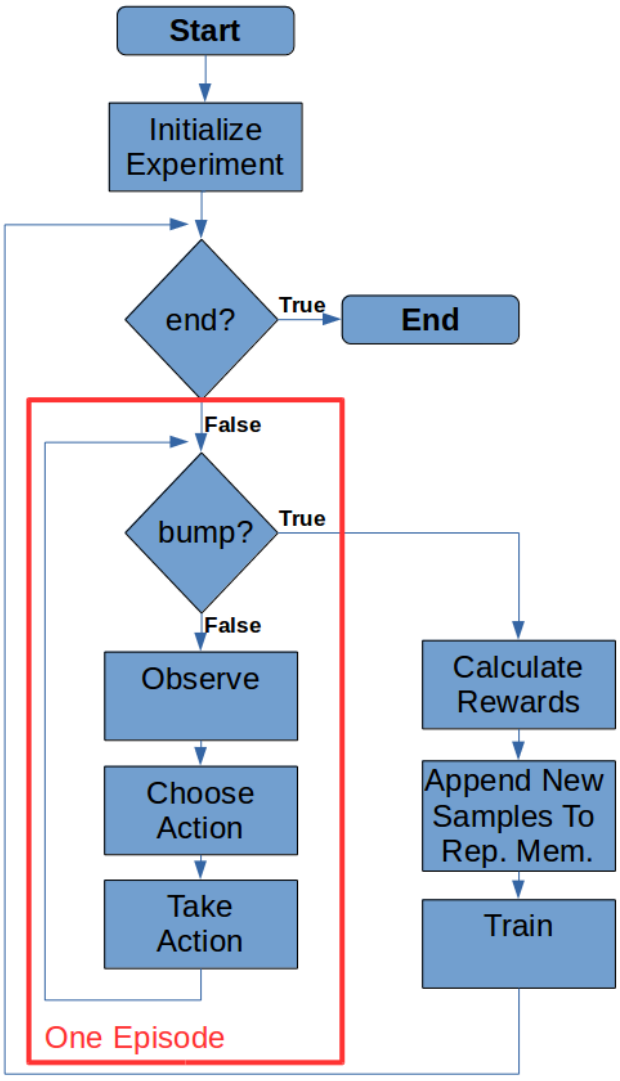


Figure 3.1: Simplified algorithm flowchart for learning the corridor navigation with experience replay

```
1  Function learnCorridor():
2      Initialize the target network parameters θ
3      Initialize the train network parameters θ⁻
4      Initialize the replay memory 𝒟 to the full capacity N
5      Initialize the empty ebatch list for the data gathered in one episode
6      Loop :
7          get the current state s, position pos, bump data bump
8          Loop while bump == False and len(ebatch) < 500:
9              Choose a based on the given action policy
10             Perform action a and observe s', pos', bump'
11             Append tuple (s, a, s', pos) to ebatch
12             s ← s'
13             pos ← pos'
14             bump ← bump'
15         EndLoop
16         if bump == True then
17             Calculate the rewards for all samples in ebatch
18             Append the corresponding state-action-reward triples to 𝒟
19             Reset ebatch to the empty list
20         else
21             Calculate the rewards for the first 450 samples in ebatch
22             Append the corresponding state-action-reward triples to 𝒟
23             Remove the first 450 samples from ebatch
24         end
25         if len(𝒟) > N then
26             Discard samples from 𝒟
27         end
28         Update the parameters θ⁻ based on 𝒟
29         if bump == True then
30             θ = θ⁻
31         end
32     EndLoop
```

Figure 3.2: The overview of the algorithm for learning the corridor navigation with experience replay. We present the individual components in the corresponding sections.

maintaining a list of the previous experiences, we reduce the correlation of the samples and allow the agent to learn more general strategy for moving through the given corridor environment.

The algorithm alternates between the two phases: the gathering of new samples and the training phase. By training the network using the gathered samples, our agent learns better reward predictions which lead to further increasing the performance level of the agent and higher rewards in the next iteration.

The agent observes its current state, position and bump information for each action in the corridor. The agent decides the next action based on the given action policy, see Section 3.3. We refer to the phase of moving smoothly in the corridor as an *episode*. The bump event signals the end of an episode. Additionally, we end the episode every time the agent has gathered 500 unprocessed samples to increase the frequency of training steps. At the end of each episode, the agent processes the newly gathered samples by calculating the corresponding rewards and adding the individual state-action-reward triplets into the replay memory. If the episode has ended due to a large number of gathered samples, we calculate the rewards only for the first 450 gathered samples and keep the remaining unprocessed samples for the next episode. The reason is that, as we describe in Section 3.2, in case there is no bump, the agent has to look 50 steps in the future to calculate the corresponding reward. Therefore, the reward for the last 50 samples cannot be computed. Finally, the replay memory is limited to hold a fixed number of samples. Hence, if the memory is full, the agent discards samples based on the discard strategy described in Section 3.5. The training step, as described in Section 3.4.2, uses the samples from the replay memory to update the parameters of the network. The updated parameters lead to different reward predictions which, in turn, induce different behavior of the agent during the next episode.

3.2 Reward Calculation

The rewards help the agent to recognize good and bad situations in the corridor. In other words, the reward definition must ensure that the agent

learns to distinguish between the state-action pairs that lead to good and bad performance. Therefore, the agent should associate the bad rewards with the state-action pairs that lead to a bump. It should associate the good rewards with the state-action pairs that maximize the traveled distance in the given amount of time steps. Finally, the reward should regularize the observation by a factor of previous knowledge.

We define the reward to be a scalar in range $[-1, 1]$. For each time step t and state-action pair (s_t, a_t), the agent receives the reward r_t. The final reward is given in Equation 3.1 as the weighted sum that fits the requirements for the definition of the reward:

$$r_t = \begin{cases} -1 & \text{if } s_t \text{ terminal} \\ 0.8(\text{eucl\_dist}_t + \text{bump\_dist}_t) + 0.2 \max_a Q(s_{t+1}, a) & \text{otherwise}, \end{cases} \tag{3.1}$$

where Q-function predicts the reward for the given state-action pair. We define the Q-function through a neural network in Section 3.4, similarly to the DQN approach [13, 14]. State s_t is the terminal state if a bump event has occurred at that time step. $\max_a Q(s_{t+1}, a)$ is the prediction of the maximum reward for the next state and introduces the aspect of previous knowledge into the reward. eucl_dist is the positive and bump_dist is the negative part of the reward.

The positive part of the reward is based on the traveled distance. It is the normalized euclidean distance given by Equation 3.2:

$$\text{eucl\_dist}_t = \max_i \frac{\sqrt{(x_t - x_i)^2 + (y_t - y_i)^2}}{max\_dist}, \; t < i < k, \tag{3.2}$$

where k is the time step of the next bump occurrence. If the bump does not occur in the next 50 steps, we set $k = t + 50$. $p_t = (x_t, y_t)$ and $p_i = (x_i, y_i)$ are the 2D positions of the agent at time steps t and k. *max_dist* is a constant defined as the maximum possible distance that the agent can travel in 50 steps. Therefore, *eucl_dist* calculates the largest normalized distance the agent has reached from the initial position p_t after a certain amount of time steps.

Equation 3.3 represents the negative part of the reward and is based on the amount of time steps to the nearest bump occurrence in the future:

$$\text{bump\_dist}_t = \begin{cases} (-0.9)^{(t_b - t)} & \text{if } t_b - t < 50 \\ 0 & \text{otherwise,} \end{cases} \tag{3.3}$$

where t_b, $t_b > t$, is the time step of the first bump occurrence after the time step t. Therefore, *bump_dist* decreases the final reward based on the number of time steps before the first bump occurs. Hence, the final reward r_t is the weighted sum of the observed reward and the prediction of the reward for the next state.

3.3 Action Policies

An action policy is the algorithm upon which the agent decides which action to take. Even though, it is intuitive to take actions for which the agent predicts the maximum reward, it does not introduce an exploration factor into the system. The agent is not encouraged to search for the solutions that might be better than the current one. In this section, we present different action policies that encourage the exploration.

3.3.1 ϵ-Greedy Action Policy

The ϵ-greedy action policy, in Figure 3.3, suggests that the agent should take the action that maximizes the reward prediction for the current state but with some probability, given by ϵ, the agent should take a random action instead. Therefore, ϵ represents the exploration factor for such approach.

Function *chooseEpsGreedy(state s, set of actions $\mathcal{A}$):*
  Select *randval* $\in [0, 1]$ uniformly at random
  if *randval* $\geq \epsilon$ **then**
    Choose action a^* such that $a^* = \max_a Q(s, a)$ for $a \in \mathcal{A}$
  end
  else
    Uniformly at random choose action a^* from $\mathcal{A}$
  end

Figure 3.3: ϵ-greedy action policy. We set ϵ to 0.1. Therefore, the agent takes a random action with probability of 10%. In contrast, the DQN method [13, 14] suggests decaying ϵ through time from 1 to 0.1.

Function *chooseRoulette(state s, set of actions $\mathcal{A}$):*
  For $\mathcal{A}$ calculate the probability distribution table $\mathcal{P}$:
  $p_i = \frac{exp(Q(s,a_i)/0.01)}{\sum_{a \in \mathcal{A}} exp(Q(s,a)/0.01)}$ for $a_i \in \mathcal{A}$, $p_i \in \mathcal{P}$
  Choose action a^* from $\mathcal{A}$ with probabilities from $\mathcal{P}$

Figure 3.4: Roulette action policy. We choose the factor 0.01 as such to additionally regularize the effect of the reward predictions on the final probability distribution. Often, it is referred to as the temperature and is altered during the learning process [12]. We keep it constant to reduce the number of hyper parameters for the algorithm.

3.3.2 Roulette Action Policy

The roulette action policy, in Figure 3.4, suggests to calculate the probability for choosing the individual actions based on the predicted reward. Therefore, the agent does not always take the action that maximizes the reward prediction. The exploration factor varies based on the difference between the reward predictions for the individual actions.

Function *chooseEpsGreedyRoulette(state s, set of actions $\mathcal{A}$):*
  Select *randval* $\in [0, 1]$ uniformly at random
  if *randval* $\geq \epsilon$ **then**
    Choose action $a*$ such that $a* = max_a Q(s, a)$ for $a \in \mathcal{A}$
  end
  else
    For $\mathcal{A}$ calculate the probability distribution table $\mathcal{P}$:
    $p_i = \frac{exp(Q(s,a_i)/0.01)}{\sum_{a \in \mathcal{A}} exp(Q(s,a)/0.01)}$ for $a_i \in \mathcal{A}$, $p_i \in \mathcal{P}$
    Choose action a^* from $\mathcal{A}$ with probabilities from $\mathcal{P}$
  end

Figure 3.5: ϵ-greedy roulette action policy. By combining the ϵ-greedy and the roulette action policies we overcome the disadvantages of both approaches.

3.3.3 ϵ-Greedy Roulette Action Policy

The third possibility is the mixture of the two policies from Section 3.3.1, 3.3.2. Therefore, we call it the ϵ-greedy roulette action policy. The algorithm is shown in Figure 3.5. Due to the disadvantages of the other policies, as we explain later in Section 4.2, we use it as a compromise solution to overcome the difficulties.

3.4 Network Architecture

The Q-function is a CNN. Hence, we use the notation $Q(s, a|\theta)$ as the reward prediction for state-action pair (s, a) changes with the network parameters θ. The network takes a grayscale image as one input and an action as a separate input and outputs the prediction of the reward. With such architecture, the network model is easily adjustable to a higher dimensional or continuous action space for future work. It is possible to use a network architecture which only takes an image and outputs three values, one per reward prediction for each of the defined actions. However, it does not generalize to the action spaces of higher dimensionality.

We present a detailed description of the individual layers for the network model in Figure 3.6. As the model suggests, the network processes the grayscale image input through a number of convolutional layers and one fully connected layer. The action input is a binary vector of size three:

- Vector (1,0,0) represents the forward motion,
- Vector (1,1,0) represents the forward motion and, at the same time, applies a slight motion to the left (forward/left motion)
- Vector (1,0,1) represents the forward motion and, at the same time, applies a slight motion to the right (forward/right motion).

The network processes the action vector through a fully connected layer. Finally, it concatenates the preprocessed inputs, performs additional processing through a fully connected layer and outputs the reward prediction. Additionally, we clip the output to the interval $[-1, 1]$ to fit the range of the possible reward values. The action policies in Section 3.3 make use of the reward prediction in order to decide the next action.

3.4.1 Preprocessing The Image Data

Before feeding it to the neural network, we first preprocess the image frame of shape $ch \times h \times w$ (number of channels $\times$ image height $\times$ image width). In our case, the frames recorded by the camera are of shape $3 \times 720 \times 1280$. First, we rescale the image to the size $3 \times 50 \times 75$. Second, we convert it to the grayscale image. Additionally, we ensure that the individual pixel values are in the range $[0, 1]$.

3.4.2 Training The Network

At the end of each episode we perform a training step. We randomly sample a mini-batch of size 128 from the replay memory and calculate the loss based on the mean square error function in Equation 3.4.

$$\text{mse}(mb) = \frac{1}{128} \sum_{(s,a,r) \in mb} (r - Q(s, a; \theta))^2 \tag{3.4}$$

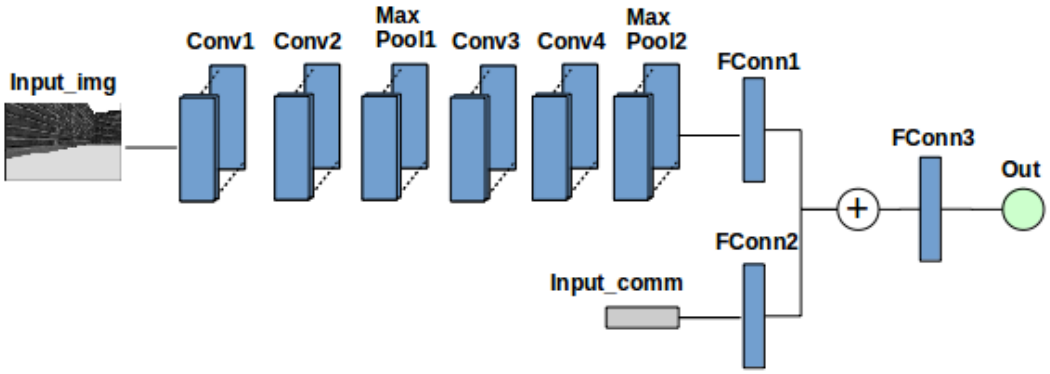


Figure 3.6: Network architecture. All trainable layers, with the exception of *Out* layer, use ReLU [7] as the activation function. The learning parameters are initialized using the Glorot uniform initializer (also known as Xavier uniform initializer) [3]. *Input_img* is the input layer into the network for the image data of shape $(1 \times 50 \times 75)$, where 1 is the number of image channels, 50 and 75 are the image height and the image width respectively. *Conv1-4* are the convolutional layers with 64 filters of size 3×3 each. *Max_pool1-2* layers perform max pooling of size 2×2. *FConn1* is a fully connected layer with 128 output units that is the last step of the individual processing for the image data. *Input_comm* is the input layer into the network for the action data of shape $(1, 3)$ to match the dimensions of the action vector. *FConn2* is the fully connected layer with 32 output units and the only individual processing step for the action data. By concatenating the outputs of *FConn1* and *FConn2* layers we obtain a vector of size 160 that is the input into *FConn3*, the fully connected layer with 128 output units. *Out* is the output fully connected layer with 1 output unit and no activation function.

where mb is the mini-batch sampled from the replay memory, θ are the parameters of the network. The optimization is done by minimizing the loss function by RMSprop [21], the gradient descent based method with adaptive learning rates. The learning rate is 0.00025 as in the DQN approach [13, 14].

3.4.3 Target Network

The target network approach relies on two networks of the same architecture, the training network and the target network. As proposed for the DQN in [14], the update steps are performed on the training network. The target network is used for the reward predictions. Occasionally, the parameters of the target network are assigned to the parameters of the training network. In our approach, we update the parameters of the target network after every bump.

Occasionally adjusting the parameters of the target network implies that the agent uses the same state of the network for multiple episodes. As the work in [14] suggests, such approach adds a delay before the update influences the behavior of the agent and reduces the policy oscillations.

3.5 Replay Memory Management

As the replay memory is limited to hold a certain number of samples, there are different strategies to handle the memory in case it gets full. In the both of the strategies that we consider, once the memory is full, we keep 99% of the maximum replay memory size and discard the rest of the samples.

The simplest strategy is to discard the oldest samples in the memory. In other words the strategy follows the First-In-First-Out (FIFO) principle. Hence, we refer to this approach as No Replay Memory Management (NoRMM). The main idea is that the newer samples in the replay memory contain more accurate rewards. Therefore, it is intuitive to keep those samples in memory until the newer, more accurate samples are recorded. Even though this strategy is successfully applied in the DQN approach [13, 14], it

has certain drawbacks. The samples that appear only occasionally may be neglected during the learning process. The definition of the strategy implies that the agent is not able to learn two different corridor environments. After being exposed for a longer period of time to only one corridor, the agent will discard the samples from the previous corridor and eventually overfit to the current corridor. Furthermore, when the agent learns the task to a certain extent, the upcoming episodes may contain a small amount of the examples that lead to a bump. Therefore, the agent may overfit to the samples containing positive rewards and eventually decrease the performance level.

In this thesis, we introduce a different approach. We propose discarding samples based on their similarity to the other samples in the replay memory. By doing so, we further reduce the correlation between the samples in the replay memory. Therefore, we reduce the possibility of overfitting to the most recent samples. We refer to this approach as the Replay Memory Management (RMM).

3.5.1 Calculating The Discard Candidates

The RMM method is partially motivated by the BRIEF approach [2] for calculating and matching the image features with binary vectors. Our method presents a basic framework for the calculation of the discard candidates for the replay memory through binary feature vectors. We discuss the possible improvements in Section 5.3.5.

We recalculate the new discard candidates every time the number of candidates for a certain decision falls below 1000. In order to calculate the discard candidates, we first extract the image features by taking the output of the *MaxPool2* layer of our network that gives us a feature map of size $64 \times 9 \times 15$. Then, we divide the feature map into the sub-feature maps of size $4 \times 9 \times 15$ and we sum over the individual values of the sub-feature maps. Finally, the comparison between the individual values results in a 120 dimensional binary vector. The abstraction of the calculation process is shown in Figure 3.7.

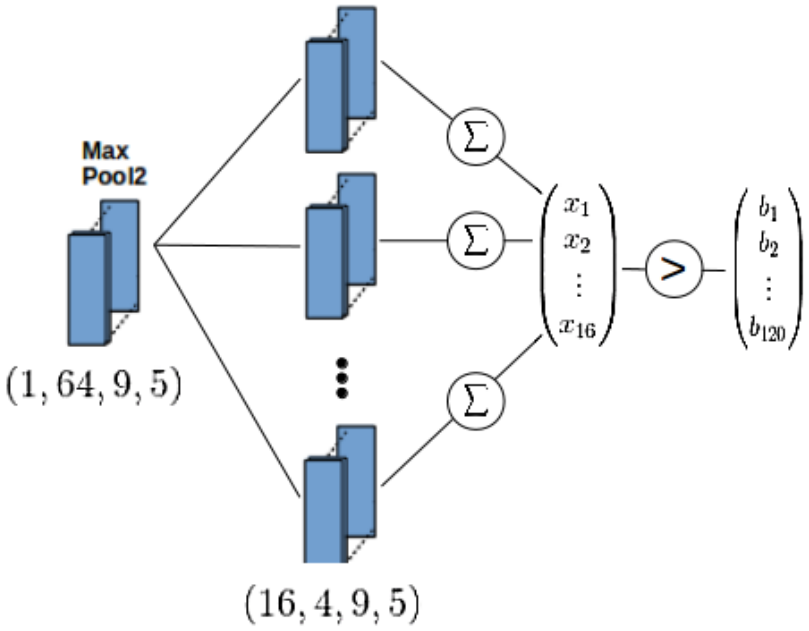


Figure 3.7: Calculation of the binary descriptors for RMM.

We compare the binary vectors by calculating the Hamming distance between the descriptors. Hence, if the distance exceeds the given threshold, we mark the sample with a lower index in the replay memory as the discard candidate. We compare only the samples which contain the same action. The distance threshold, measured in percentages of the full size of the binary descriptor that is 120 bits, is set to 1%(or equivalently 1 different bit) at the beginning of the learning and is increased by an additional percent when the agent is not able to find more than a certain number of discard candidates for any of the actions. In all of the experiments, we set this number to 2000. For the conducted experiments, the Hamming distance threshold never exceeds 7%.

In the discard phase, we only consider the candidates of the individual decisions for which the number of samples exceeds one third of the replay memory. Therefore, each decision keeps 99% of its replay memory space by discarding the oldest discard candidates. By doing so, our approach ensures that all decisions are represented with equal number of samples in

the replay memory.

As the initial network parameters produce similar feature maps for arbitrary images, it is useful to pre-train the network before utilizing this approach. Otherwise, the agent marks the majority of the samples as the discard candidates during the first calculation as the standard network initializations output similar features for all inputs. Therefore, the advantages of the RMM approach are first notable after the agent has performed the calculation of the discard candidates for the second time and has trained for a larger amount of iterations.

3.5.2 Reward Equalization

In comparison to the NoRMM, for the RMM the samples may remain in the replay memory for uncertain amount of time. The older samples often contain rewards which are inaccurate and can therefore interfere with the learning process. To circumvent this problem, we propose to perform one reward equalization step for every 50 episodes in Equation 3.5:

$$r_t = 0.7r_t + 0.3Q(s_t, a_t|\theta). \tag{3.5}$$

We perform the equalization for every state-action-reward triple (s_t, a_t, r_t) in the replay memory. In case that the target network approach is used, we propose using the target network for the equalization process.

3.5.3 Fitting The Network To The Replay Memory

High correlation of the samples in the replay memory might still occur in some situations for the NoRMM approach. In such cases, training more frequently can lead to degrading the level of performance to the state from the beginning of the training. If we consider a perfect theoretical model of the agent that does not bump, the newly gathered samples with maximized reward would eventually overfill the replay memory. By training a large number of epochs for such setting, the network learns that the best solution is always to predict the maximum rewards.

As the RMM considers the similarity of the samples for its discard strategy, the correlation between the samples of the replay memory is significantly reduced when compared to the NoRMM. Therefore, we expect that such samples are good representations of the given corridor environment. This allows us to perform multiple training steps without considering the possibility of overfitting the network to the most recent experiences. In our experiments, we introduce a fitting phase after every 100 episodes. In the fitting phase, we perform 3 epochs. Each of these epochs randomly divides the replay memory into the mini-batches of size 128 that are individually used for training the network. Through the fitting phase, we apply much larger training updates which, in turn, results in learning the environment in significantly smaller number of episodes.

4 Evaluation

In this part of the thesis, we evaluate our approach. In the first evaluation we compare the action policies to define the one which is the most suitable for the rest of the experiments. The goal of the rest of the experiments is to evaluate, step by step, if our agent is able to learn to navigate in a complex corridor environment using the RMM approach. For this part, we first evaluate the ability of the agent to learn a simple corridor by utilizing the NoRMM and RMM approach for discarding the samples from the replay memory. The main focus is showing the importance of considering the similarity of the samples. Second, we investigate if the agent is able to learn two simple corridors at the same time with the RMM. This is tightly related to the agent exploring a large corridor where the individual parts are not encountered as often as the others. Finally, we evaluate our agent in a complex corridor environment.

4.1 Fitness Definition

In all of the experiments we define the fitness as the number of actions the agent performs without bumping. The maximum fitness is 500 as that is the maximum number of actions that the agent performs during a single episode. It is not the actual measure for the performance level as the trained agent should also prefer traversing the long distances which we cannot observe from the fitness measures. Still, such fitness definition is simple to visualize and to interpret for the evaluation of the learning process. One additional observation for interpreting the results is that, after the agents bumps, it finds itself in a difficult position for which any action leads into additional bumps. Therefore, the fitness plots for the individual experiments in the following sections appear degraded by this factor.

Number of episodes	Replay memory size
1000	2000

Figure 4.1: Action policies evaluation settings.

Action policy	ϵ-greedy ($\epsilon = 0.1\%$)	Roulette	Roulette (4000 ep.)	ϵ-greedy roulette ($\epsilon = 0.1\%$)
maximum fitness (average on 100 episodes)	**38**	16	16	36

Figure 4.2: Comparison of action policies. Agent learns the Plus corridor through the given action policy. The performance is measured based on the maximum fitness level of the agent. In the table, we show the fitness values averaged on 100 episodes. The bold value indicates the highest fitness among the experiments. All the agents have completed 1000 episodes, except for the experiment where it is explicitly stated that the agent has completed 4000 episodes.

4.2 Evaluation: Comparison Of Action Policies

In order to choose the appropriate action policy for the rest of the evaluations, in the first step, we evaluate the action policies introduced in the previous sections. For each of the policies, the agent learns the Plus corridor. The settings for the experiment are shown in Figure 4.1. The initial weights for the neural networks and the initial state of the replay memory are equal for all of the experiments. We present the results in Figure 4.2.

The ϵ-greedy strategy shows the highest fitness level. However, even though the reward predictions might be correct, due to its definition, the random actions can lead to the bumps for this policy. On the other hand, the roulette strategy does not seem to show any learning progress. Even though we let it to run for 4000 episodes, the roulette strategy still fails to reach higher fitness level. Therefore, we choose the ϵ-greedy roulette strategy for the rest of the experiments as it reaches the similar fitness level as the ϵ-greedy strategy but explores the corridor in a way that always considers the reward predictions.

Number of completed episodes	Replay memory size	Completed steps	Highest number of actions without bumps
15000	45000	1584893	8719 (episode 8685)

Figure 4.3: Plus corridor, NoRMM, evaluation details.

4.3 Evaluation: Comparison Of The NoRMM And The RMM Approach

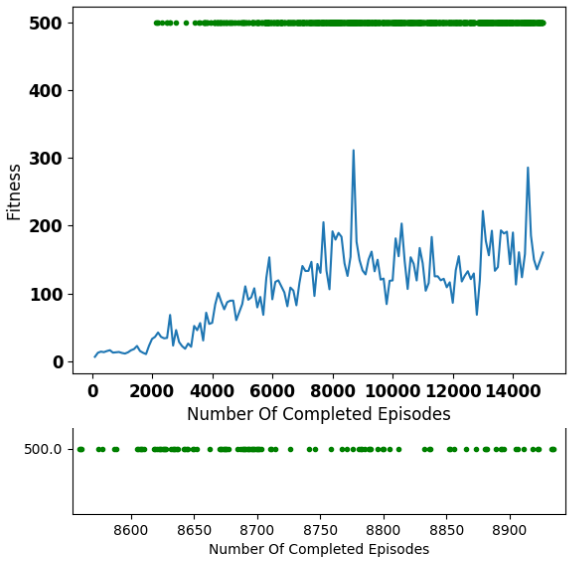
After deciding to use the ϵ-greedy roulette policy, we evaluate the ability of the agent to learn the Plus corridor environment and compare the presented methods for managing the replay memory. The Section 2.2.1 defines the challenges for the navigation in the Plus corridor. These experiments provide the necessary visualizations of the data for understanding intuitively the importance of sample selectivity for the replay memory. In order to provide a better insight into the fitness level of the agent, we observe that the agent completes a lap in the Plus corridor by completing approximately 150 actions.

4.3.1 Evaluation: NoRMM For The Plus Corridor

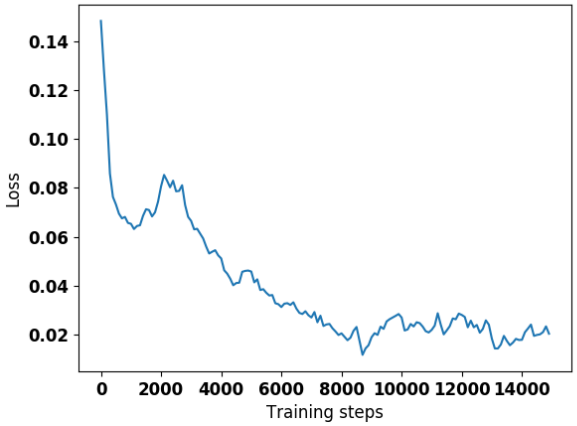
In order to evaluate the NoRMM, we train the agent for 15000 episodes in the Plus corridor. The details of the experiment are shown in Figure 4.3.

The development of the agent's performance is shown in Figure 4.4a. The progress of the loss function is presented in Figure 4.4b. We observe the progress of the agent during the training to explain the behavior of the fitness progress.

Initially, the agent does not differ between facing a wall and looking at a long straight corridor. After a short learning period we observe that the agent is first able to learn to predict the rewards for the situations when a bump is immediate. As there is no much variance in the rewards for



(a) Plus experiment, NoRMM, fitness progress. The upper plot shows the fitness progress for the 15000 completed episodes. The blue curve shows the fitness values averaged every 100 episodes. The green points indicate the episodes where the agent has reached maximum fitness value of 500. The lower plot concentrates on the maximum fitnesses reached around the most successful episodes.



(b) Plus experiment, NoRMM, loss progress. The plot shows the loss progress for the 15000 completed episodes. The results are averaged for every 100 episodes.

Figure 4.4: Plus experiment, NoRMM, fitness and loss progress.

such samples, the agent is able to quickly grasp the appropriate reward predictions. Harder to grasp are the situations where the agent needs to maneuver around a corner with a precise set of actions and may require much larger amount of training. Finally, as the agent uses a single monocular image as the observation state, some situations are impossible to handle without overfitting the network to the corridor environment. Due to the specific properties of the Plus corridor, there are lots of situations where there is a wall corner in the immediate surrounding which the agent is not able to see. the agent predicts a high reward but the movement results in a bump. Therefore, there is a lot of variance between the rewards for resembling observation states. Learning such features of the individual corridor is hard and the performance of the agent varies during the training due to such cases. Additionally, we notice that the performance of the agent when traversing the corridor in the clockwise direction does not match the performance in the counter-clockwise traversal. The performance for the both directions varies throughout the episodes.

The slow development of the fitness throughout the first 15000 episodes implies the need for increasing the number of training sessions per episode. As discussed in Section 3.5.3, for the NoRMM approach, increasing the number of training iterations in a single episode can theoretically degrade the fitness to the initial level. We perform additional 5000 episodes and include a fitting phase every 100 episodes. Additionally, we perform 5 standard training steps after every episode instead of only 1 with 5 different mini-batches sampled from the replay memory. As demonstrated in Figure 4.5, the fitness level never reaches the highest possible level. Therefore, the replay memory never contains only the samples with positive rewards. Hence, the network does not overfit to the samples with positive rewards and we never experience a large drop in performance during the experiment either. However, we notice a sudden increase of fitness as we include the fitting phase in the approach. This motivates the fitting phase for the RMM approach where the correlation between the samples is significantly reduced and we expect that the replay memory is able to provide a better representation of the corridor environment.

Additionally, in Figure 4.6 we observe that the number of samples in the replay memory for the individual actions is not equal. Depending on the properties of the corridor, the agent might develop a certain preference for

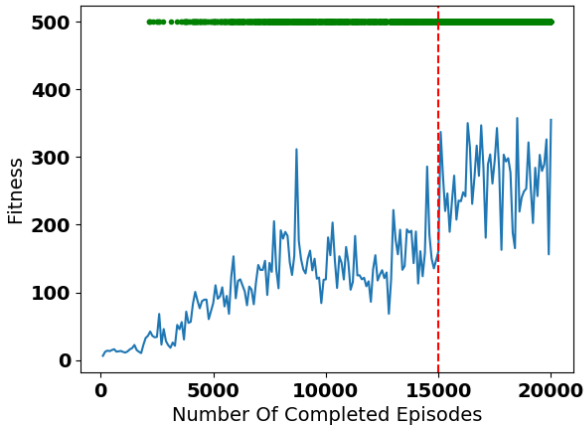


Figure 4.5: Plus experiment, NoRMM with fitting phase, fitness progress. The first 15000 episodes are the results earlier presented for the NoRMM without fitting phase. Vertical red dashed line represents the first episode where we use NoRMM with fitting phase. We observe that the agent is able to reach higher values more frequently. The fitness level never drops to the very low levels as the agent never reaches such high levels of performance where the fitting phase would lead to overfitting.

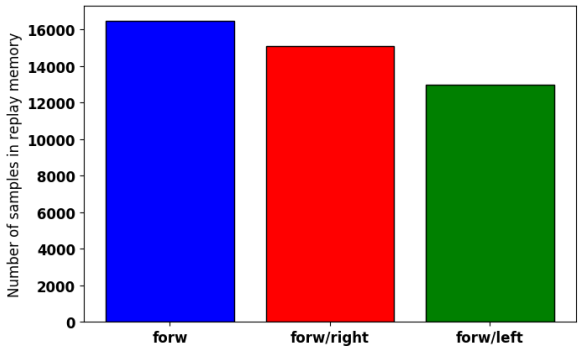


Figure 4.6: Plus experiment, NoRMM, replay memory samples per action. The presented statistics refer to the state of the replay memory after 15000 episodes. Depending on the observed episode, it is possible that one action dominates the replay memory by larger margins.

a single action which, in turn, becomes the dominant action of the replay memory. Hence, we address this issue in the RMM approach by defining the limit for the number of samples for each of the actions.

Finally, the learned reward predictions of the agent for the different observation states in the corridor are presented in Figure 4.7. Due to the oscillations in the performance, the figure shows the predictions for the network parameters, for which the network had reached the highest performance. Even though, the agent performs a very long trajectory, the reward predictions might not be very satisfactory for the more challenging situations for some of the actions. One of the causes might be the unequal distribution of the samples in the replay memory. This further motivates the RMM approach.

The experiment in the Plus corridor with NoRMM shows the possible disadvantages of the discard strategies that follow the FIFO principles. Through this experiment, we point to the advantages of tackling the problem from a different perspective. Therefore, in Section 4.3.2 we evaluate the agent that utilizes the RMM strategy in the Plus corridor.

4.3.2 Evaluation: RMM For The Plus Corridor

We evaluate the RMM approach by training the agent for 7000 episodes (significantly less than for the NoRMM) in the Plus corridor. The details of the experiments are shown in Figure 4.8. We use the same set of the initial network parameters as for the NoRMM evaluation. As already explained in Section 3.5.1, it is useful to perform a pre-training phase for the RMM approach. Therefore, we vary the different approaches for training the agent int the first 3000 episodes. For the first 1000 episodes the agent uses the NoRMM approach with the replay memory of the size 2000 to give the agent a small initial boost. Next, the agent increases the replay memory size limit to 45000, additionally explores the envirionment to fill the memory and then continues to use the NoRMM approach for the episodes 1000-2000. Finally, for the episodes 2000 to 3000 the agent uses the RMM approach without ever performing a train step in order to decrease the correlation of the data in the replay memory. We perform the first fitting phase in episode 3000 that marks the end of the pre-training phase. The agent then trains

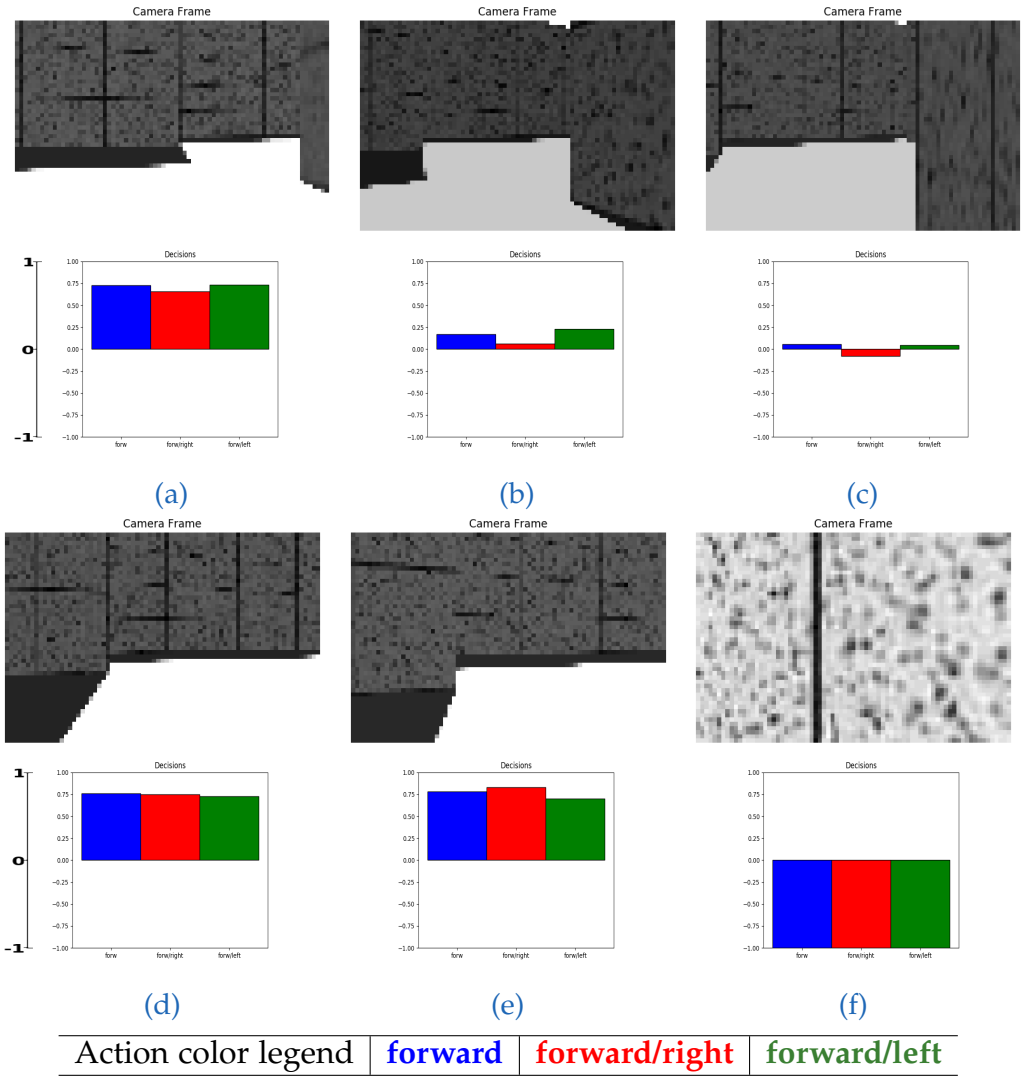


Figure 4.7: Plus experiment, NoRMM, reward prediction examples. Each of the image pairs a-f shows in the upper image the observation state and in the lower image the reward predictions for each of the actions. a) The agent is in a safe situation. Therefore, all of the predictions are relatively high. b,c) The agent is getting closer to the wall to the right. Even though the motion to the left would move the agent into a safe state, the reward for that motion is still relatively low. d,e) The agent is in the safe position, all rewards are high. f) The agent is about to bump and predicts the minimum rewards.

Number of completed episodes	Replay memory size	Completed actions	Highest number of actions without bumps
7000	45000	1299722	18211 (episode 3260)

Number of pre-train episodes	Number of train episodes	Initial Hamming threshold
3000	4000	1%

Figure 4.8: Plus corridor, RMM, evaluation details.

using the RMM strategy for additional 4000 episodes. During this time, the Hamming distance threshold for determining the discard candidates reaches the maximum of 3% of the descriptor size. Due to a large number of training steps that occurs in the fitting phase every 100 episodes, we skip the standard phase of performing one training step per episode during the experiment. Therefore, in the episodes between the two fitting phases, we only perform additional filtering of the samples in the replay memory. Hence, the target network method is obsolete for such approach.

The fitness progress is shown in Figure 4.9a. The progress of the loss function is presented in Figure 4.9b. Again, we observe the agent during the learning process to analyze the behavior of the fitness.

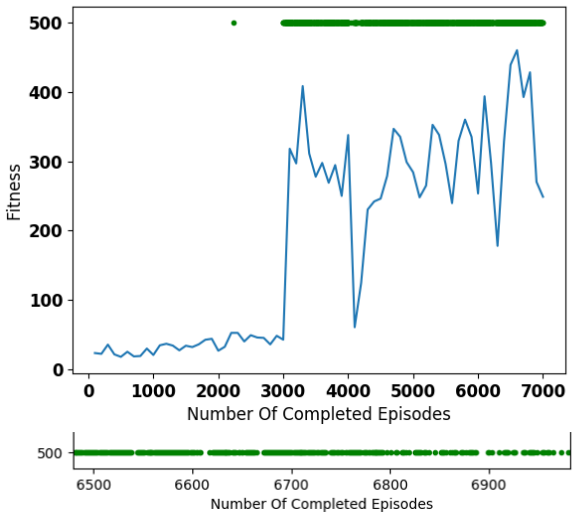
Immediately after the first fitting phase, we observe a large increase in the performance of the agent that is close to the best average fitness for the standard NoRMM evaluation. The agent continues to perform well in the corridor afterwards. Around the episode 4000, what appears as a large drop in fitness, is when the agent starts to grasp the movements for the opposite direction of the corridor. The fitness improves fast and the agent is able to learn the both traversal directions in a short amount of time. We observe that the performance may vary for both of the traversal directions throughout the episodes. Still, after learning it for the first time, the agent is able to recover the good performance for the affected direction after the first fitting phase. In the end, the wall corners that are in the blind spot of the agent still represent the unsolved obstacle for the agent. The performance in such

cases varies throughout the episodes.

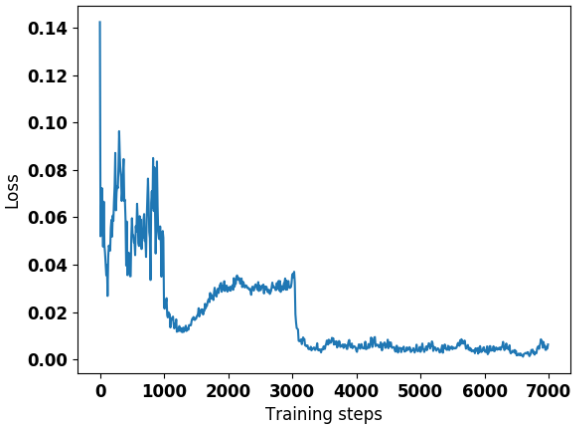
The learned reward predictions for the NoRMM and the RMM strategies are shown in Figure 4.10. For each of the strategies, we use the set of the network parameters for which the agent has reached the best performance. The agent that utilizes the RMM strategy is able to predict more accurate rewards when compared to the NoRMM strategy for critical observation states. We show in Section 4.4 that for the RMM approach we must not necessarily use the set of the best network parameters in order to predict the accurate rewards. Furthermore, Figure 4.11 shows that the RMM strategy achieves the equal distribution of the samples in the replay memory for each of the actions.

To sum up the role of the fitting phase for the RMM approach, we run an experiment for RMM without the pre-training phase for 5000 episodes and observe the required number of completed episodes before the agent starts reaching higher fitnesses. Additionally, we sample 5 mini-batches from the replay memory and perform 5 training steps after every episode to be able to provide the similar setting to the NoRMM approach that uses the fitting phase for a better comparison. The results of the fitness comparisons of the two RMM approaches and the NoRMM approach are presented in Figure 4.12. We observe that for both RMM approaches the fitness level still includes a certain level of variance. However, by comparing the maximum fitness levels, we conclude that the fitting phase has significant benefits for the RMM approach when compared to the NoRMM approach.

Through the experiments in the Plus corridor, we show that the RMM strategy is able to replace the NoRMM strategy for the standard task of learning a simple corridor environment. In Section 4.4, we continue to show that, by utilizing the RMM strategy, the agent is able to learn the tasks for which the NoRMM strategy is not fit to learn by its definition. The agent learns the navigation in two different corridors.



(a) Plus experiment, RMM, fitness progress. The upper plot shows the fitness progress throughout the 7000 completed episodes. The blue curve shows the fitness values averaged every 100 episodes. The green points indicate the episodes where the agent has reached maximum fitness value of 500. The lower plot concentrates on the maximum fitnesses reached around the most successful episodes.



(b) Plus experiment, RMM, loss progress. The plot shows the loss progress for the 7000 completed episodes. The results are averaged for every 100 episodes. We observe a sudden drop of loss value after the first fitting phase in episode 3000.

Figure 4.9: Plus experiment, RMM, fitness and loss progress.

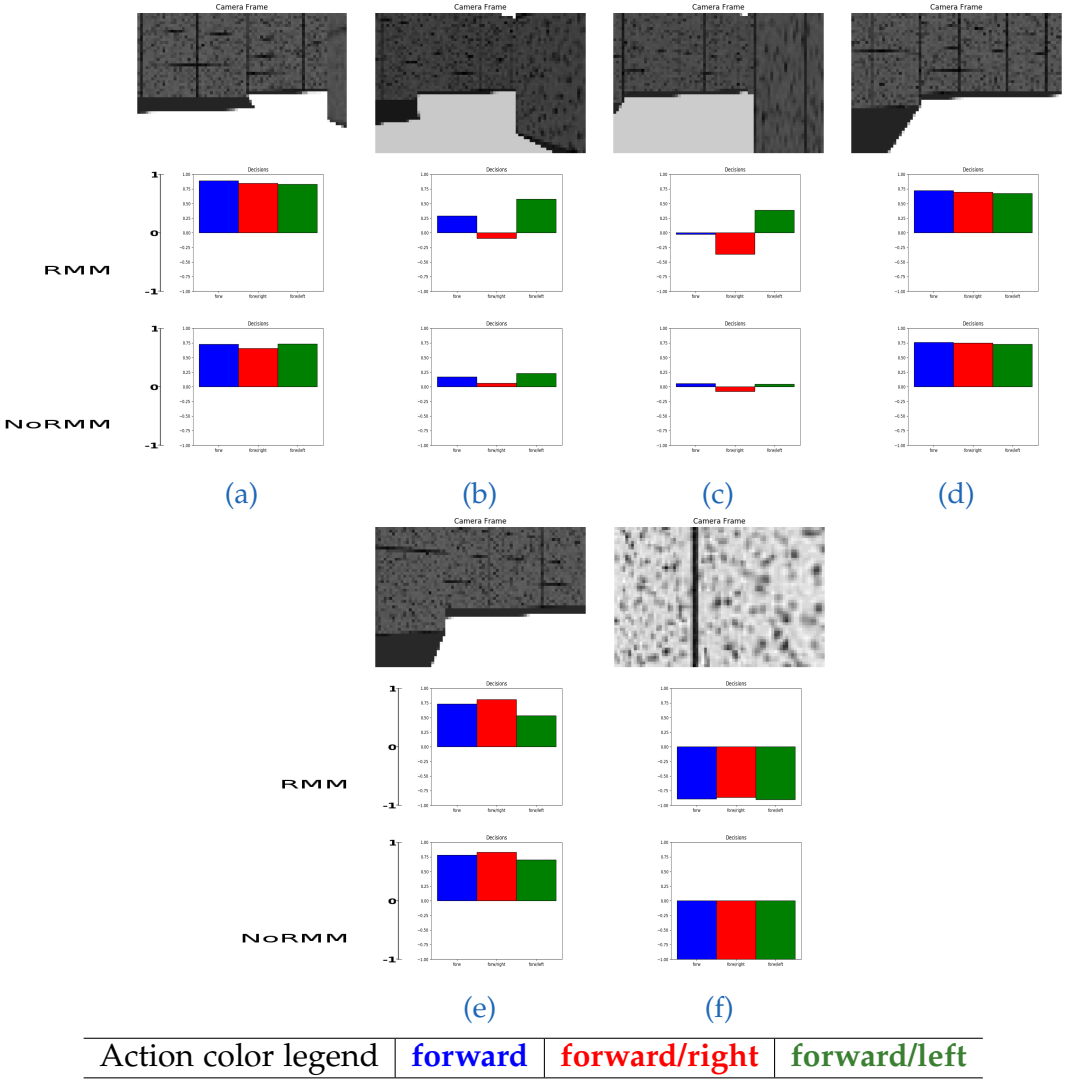


Figure 4.10: Plus experiment, RMM and NoRMM, reward prediction examples. Each of the image triples a-f shows in the upper image the observation state, in the middle image the reward predictions for each of the actions for RMM approach and in the lower image the reward predictions for each of the actions for NoRMM approach. For the non-critical observation states in examples a) and d), both networks predict similar rewards. For the more critical observation states in examples b), c) and e), the network trained with RMM outputs better predictions.

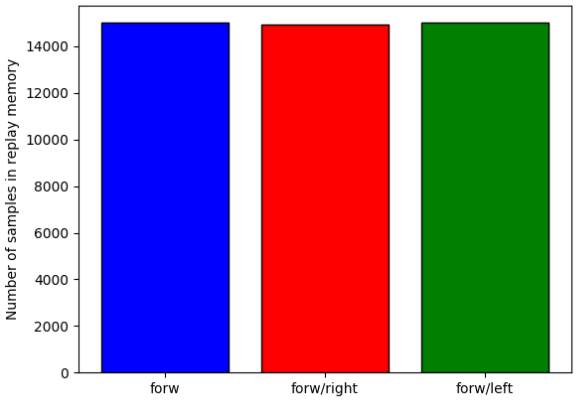


Figure 4.11: Plus experiment, RMM, replay memory samples per action. The presented statistics refer to the state of the replay memory after 7000 episodes. Throughout the learning process, the number of samples in the replay memory for each action remains balanced.

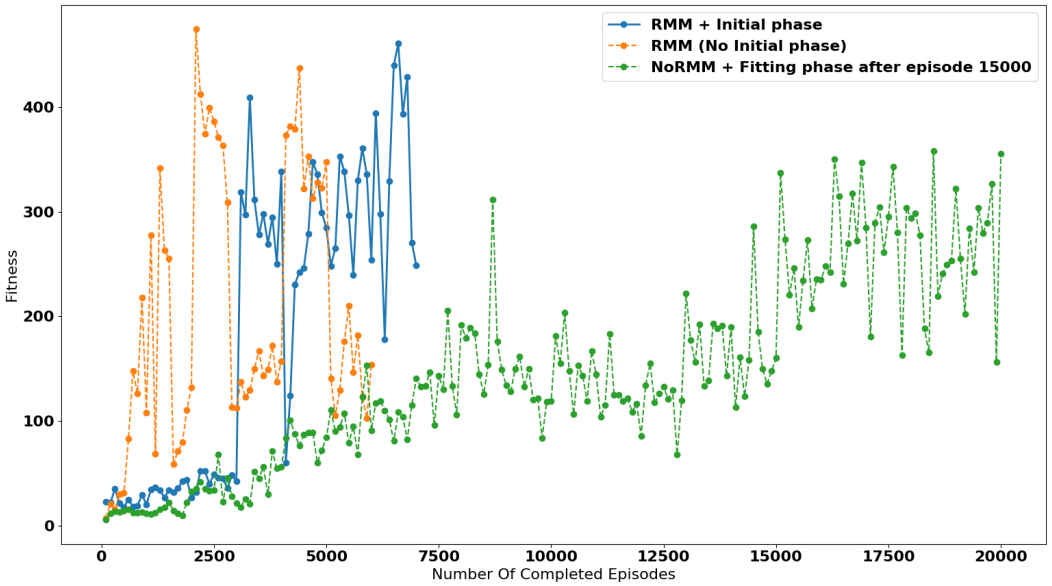


Figure 4.12: Plus experiment, fitness Comparison. The plot compares the fitness progresses for the different approaches averaged on every 100 episodes.

Number of completed episodes	Replay memory size	Completed actions
3000	50000	706653

Number of completed episodes in Minus corridor	Number of completed episodes in Plus corridor	Initial Hamming threshold
1500	1500	3%

Figure 4.13: Plus-Minus experiment, RMM, evaluation details.

4.4 Evaluation: Plus-Minus Experiment, Learning Two Corridors

In order to evaluate the RMM approach for learning multiple corridors, we move the agent to the Minus corridor. In Section 2.2.1, we present the navigation challenges of the corridor. There, the agent explores the new environment for 1500 episodes. Afterwards, we move the agent back to the original Plus corridor, where it completes additional 1500 episodes. The experiment settings are presented in Figure 4.13.

For the initialization process, we use the values that were achieved at the end of the Plus corridor experiment. We initialize the network by using the state of the network parameters from the end of the Plus experiment. We initialize the replay memory to the state after the episode 7000. Therefore, the replay memory initially contains only the samples from the Plus corridor. Even though it might be helpful to use much larger replay memory, in this experiment we set the size of the replay memory to 50000. Again, we follow the principle of training only by performing a fitting phase after every 100 episodes. The Hamming distance threshold reaches a maximum of 5% of the descriptor size during the experiment.

In Figure 4.14 we observe the fitness progress during the experiment. Even though the textures in the Minus corridor differ completely from the ones in the Plus corridor, already in the beginning of the experiment, the agent is able to use some of the knowledge from the Plus corridor and apply it to the Minus corridor. After a short amount of time, the agent learns to perform the U-turn in the Minus corridor and is able to reach high levels

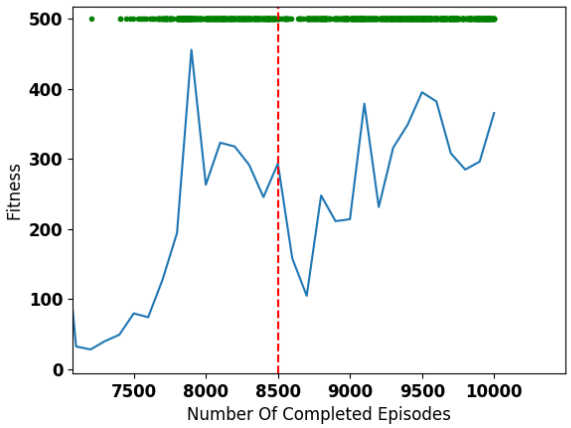


Figure 4.14: Plus-Minus experiment, fitness progress. The plot shows the fitness progress of the agent during the experiment averaged on 100 episodes. The experiment starts at episode 7000 when we move the agent from the Plus corridor to the Minus corridor. The red dashed line marks the episode in which we move the agent back to the Plus corridor.

of performance. After we move the agent back to the Plus corridor, we observe that the agent, to some extent, preserves the knowledge of the Plus corridor and is able to regain higher levels of fitness after a small number of episodes.

We compare the number of samples for each of the corridors in the replay memory after the 10000th episode in Figure 4.15. Even though, the agent has spent the last 1500 episodes in the Plus corridor, there are significantly more samples in the replay memory from the Minus corridor. This might imply, that the walls in the Minus corridor include more complex patterns in the textures than the walls in the Plus corridor resulting in higher Hamming distance. The results show that there is still a small number of samples tracing back to the Plus experiment. Such state of the replay memory indicates that there are some state-action pairs that the agent does not visit after learning to perform better in the corridor. This further implies that the sample selectivity for the replay memory is important for maintaining a good representation of the corridor environment.

In Figure 4.16, 4.17 we show the learned reward predictions after completing

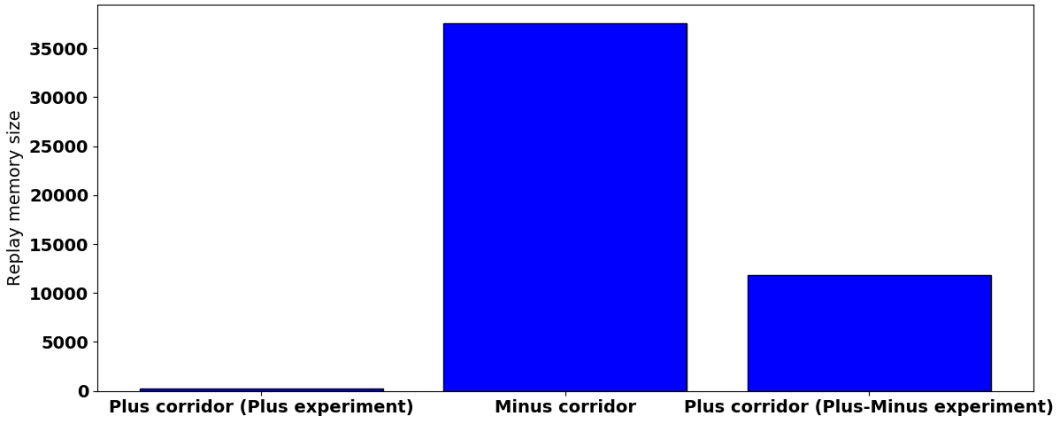


Figure 4.15: Plus-Minus experiment, number of samples in the replay memory for different corridors.

the Plus-Minus experiment. The main observation is that for the Plus corridor, the reward predictions do not show much variation when compared to the predictions from the previous experiment in the Plus corridor for the RMM approach. On the other hand, the agent is at the same time able to predict the appropriate rewards for the Minus corridor. We observe in Figure 4.17 that the predictions of the agent strongly favor a single action as it performs the U-turn. The only situations where the agent shows some uncertainty in predictions during the U-turn are the situations for which it is hard, even for a human eye, to predict the optimal action (e.g. when the agent is facing a wall at a certain distance).

Through this experiment we have shown that the agent is capable of learning two corridor environments simultaneously. While the performance may still vary throughout the episodes for the individual corridors, the agent is able to predict the appropriate rewards for both corridors for the most of the situations. We perform the final experiment and evaluate the ability of the agent to learn a complex corridor environment in Section 4.5.

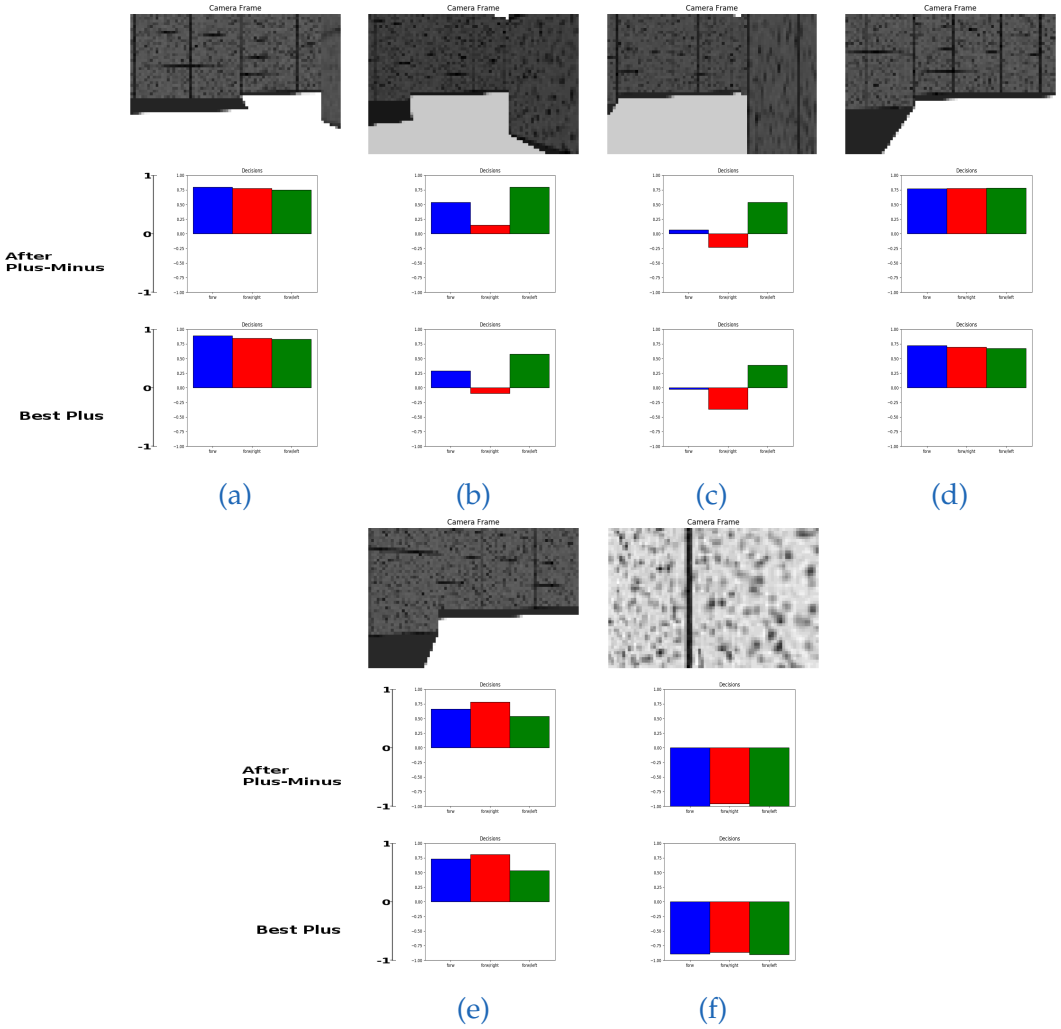


Figure 4.16: Plus-Minus experiment, reward prediction examples for the Plus corridor. Each of the image triples a-f shows in the upper image the observation state. In the middle image we show the reward predictions for each of the actions for RMM approach after the Plus-Minus experiment. In the lower image we show the reward predictions for each of the actions for the state of the network where the agent reached the best performance in the Plus experiment. We observe similar predictions with only slight variations.

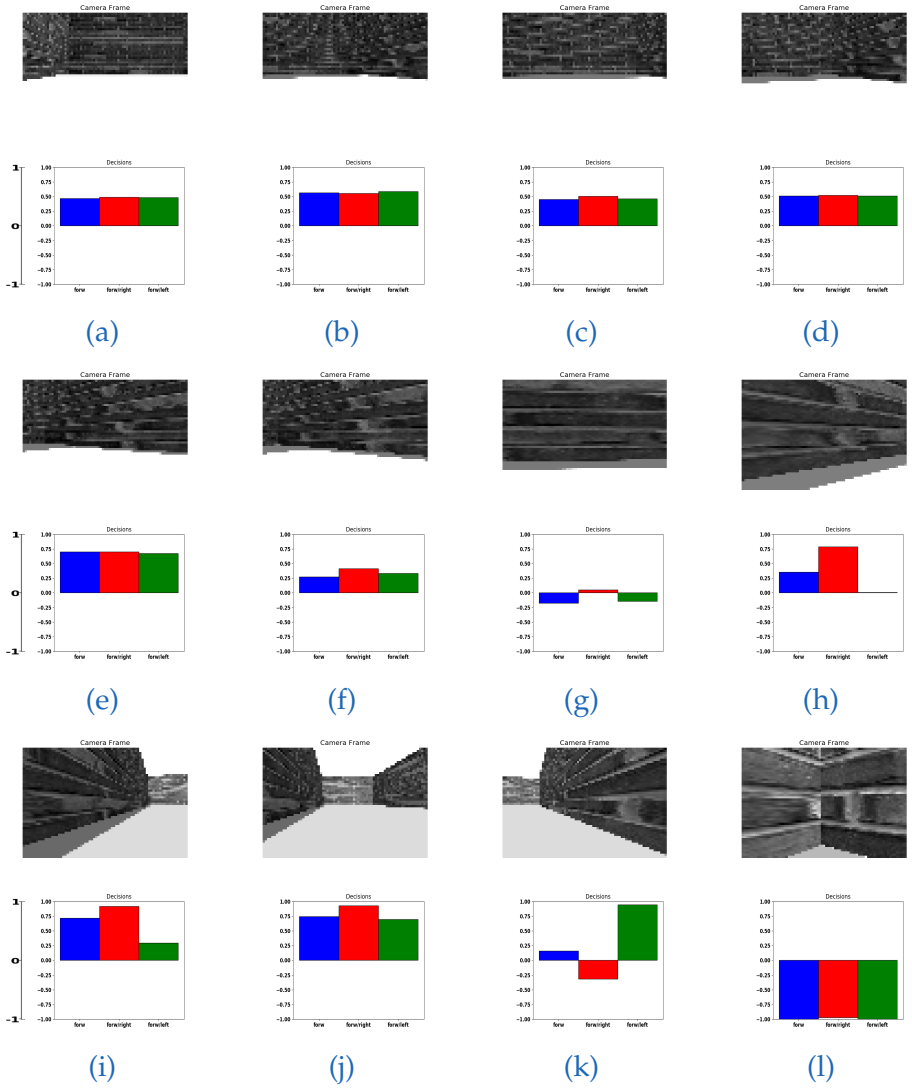


Figure 4.17: Plus-Minus experiment, reward prediction examples for the Minus corridor. Each of the image pairs a-l shows in the upper image the observation state. In the lower image we show the reward predictions for each of the actions for RMM approach after the Plus-Minus experiment. In examples a-j we observe the reward predictions as the agent completes the U-turn in the corridor. The rewards are balanced up to the first critical observation state, example f, where the agent starts predicting much larger rewards for the action leading to the safe state. In example g), we note that there is a lot of uncertainty as the agent is facing the wall and it is hard, even for a human eye, to predict the optimal action. Examples k) and l) show the predictions for the additional situations in the corridor.

Number of completed episodes	Max. Replay memory size	Completed actions
3000	90000	273965

Figure 4.18: I16c, RMM, evaluation details.

4.5 Evaluation: I16c Experiment, Learning A Complex Corridor

In this scenario, the agent learns a single complex corridor environment (I16c) for 3000 episodes. The Section 2.2.1 presents the challenges for the navigation in the I16c corridor. The details of the evaluation are shown in Figure 4.18. Because of the long and narrow parts of the corridors, the agent prefers to roam through a single area of the corridor. For this reason, we interact with the corridor environment by placing and removing obstacles in the corridor and force the agent to move into the different sections of the corridor. We refer to this experiment as the I16c experiment.

We initialize the agent to the state after the Plus-Minus experiment. Additionally, we mark all samples in the replay memory as the discard candidates as, in this experiment, we are only interested in evaluating the aspect of the agent learning one large complex environment. In this experiment, as the agent is not able to handle the complex corridor with the replay memory of size 50000, we gradually increase the limit by 5000 each time the agent is not able to mark more than 2000 samples (as opposed to the approach in Section 3.5.1 where we always increase the Hamming distance threshold). We continue increasing the limit until it reaches the maximum limit of 90000. Afterwards, we continue the original approach and gradually increase the Hamming distance threshold instead. The threshold is initially 5% and reaches the 7% of the descriptor size at its peak. We follow the principles of training only through the fitting phase every 100 episodes.

We do not show the fitness progress for this experiment, as the values do not contribute to the evaluation factor due to the properties of the corridor and the human interaction.

We observe the behavior during the learning process. At the beginning, the

agent seems to perform well when encountered with the new wall textures. However, the agent soon arrives at the windows section and faces a hard task to solve. However, during the training, the agent is able to perfectly solve this obstacle. We observe the similar behavior for the bridge sections that connect the two sides of the corridor.

We observe the learned reward predictions for the corridor in Figure 4.19. Even though the I16c corridor appears more complex in comparison to the Plus and Minus corridor, the agent is able to successfully learn this environment. We observe that the agent is able to learn good reward predictions for the difficult situations in the corridor including different wall textures, window patterns and obstacles. We further evaluate the agent by performing a test run in Section 4.5.1.

4.5.1 Evaluation: I16c Corridor, Test Phase

In this section, we present the results at the test time. After completing the I16c experiment, we evaluate the ability of the agent to move through the I16c corridor using the previously learned knowledge.

As the corridor includes dead ends where the agent has no other choice but to bump, we modify the action policy to include the rotation in place. If the reward prediction for the action chosen by the action policy is negative, the agent does not complete the action but instead rotates based on this action. In this case, if the action policy suggests the forward/left action, the agent rotates to the left instead and otherwise rotates to the right. The agent keeps rotating in the same direction and observing the new states until the action policy suggests taking an action with positive reward.

Because of its reward prediction, the agent naturally follows the long straight part of the corridor. Therefore, as the agent gets closer to the last intersection point, we place one obstacle in such a way to make a turn instead. We show the recorded video of the test run in our online repository, see Section 2.4. We observe a rough sketch of the route that the agent traverses during the test in Figure 4.20. The agent is able to pass the straight section of the corridor that includes windows without any issues. After reaching the new obstacle, the agent rotates in place and continues its route over the bridge

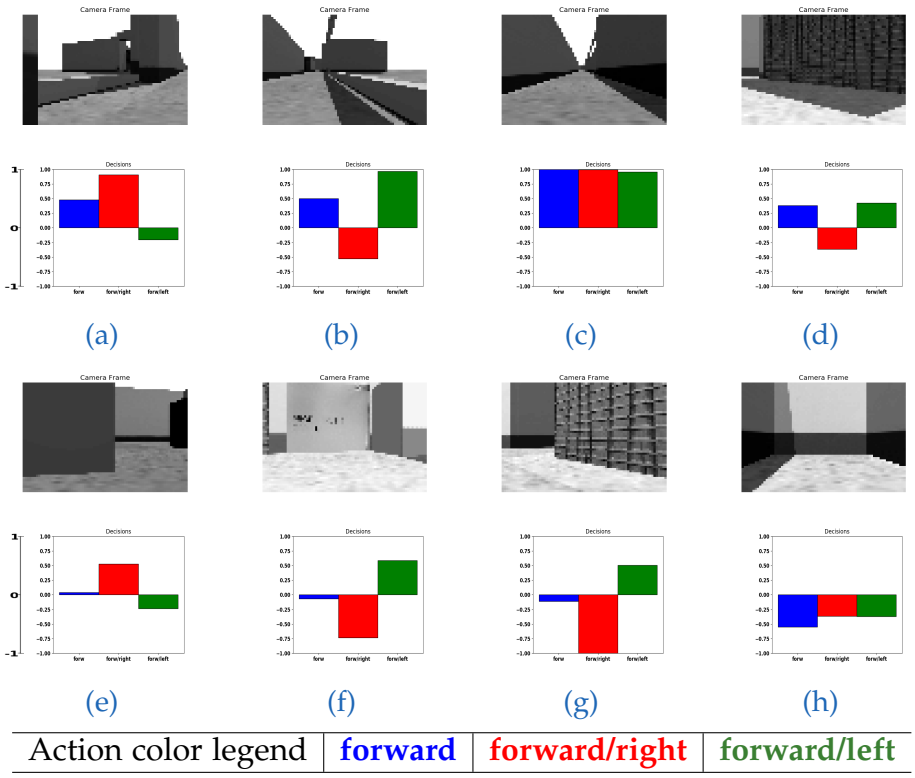


Figure 4.19: I16c experiment, reward prediction examples for the I16c corridor. Each of the image pairs a-l shows in the upper image the observation state. In the lower image we show the reward predictions for each of the actions for RMM approach after the I16c experiment. a,b) The examples containing windows. c) The agents sees a long straight corridor. d) The agent is making a turn at the bridge area. e) There is an obstacle on the left side of the agent. f) The agent is approaching Prof. Lepetit's office. g) The agent is at the bridge section. h) Agent sees a dead end.

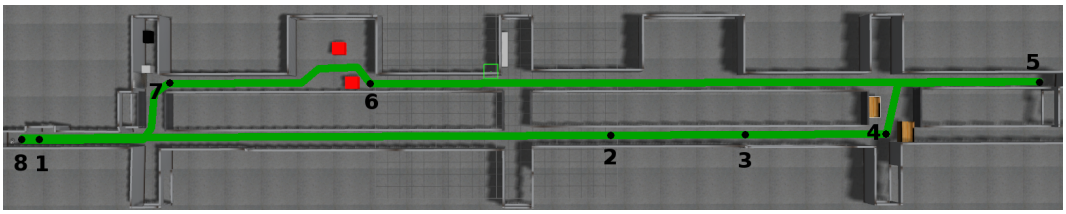


Figure 4.20: I16c corridor, test run. 1) Right at beginning, the agent passes near a window area without any issues. 2) The agent overestimates the motion to the left and is facing the wall. It predicts negative rewards and therefore rotates to the right. 3) The agent continues through a long window area. 4) We place an obstacle in front of the agent. The agent approaches it, stops, rotates in place to the left and continues to navigate without bumping. 5) The agent goes right at the next intersection not knowing that it will reach a dead end. As the agent approaches the dead end it rotates in place for 180 degrees and continues to navigate. 6) The agent passes by a set of obstacles. 7) The agent turns left after seeing Prof. Lepetit's office, as to the right, there are some obstacles which often resulted in a bump during the learning phase. 8) The agent reaches the dead end, rotates for 180 in place and finishes the run without a single bump.

and into a different part of the corridor. The agent continues to avoid the obstacles in the way, crosses the bridge and eventually reaches its starting point. As the agent is now looking at a dead end, it rotates in place to complete the test run.

By completing the test run we conclude that our agent is fit to learn larger and more complex environments. The current reward definition makes the agent favor the long straight parts of the corridor. It might imply that using the traveled distance may not be enough for the reward signal on its own. However, the final experiment shows that the agent is able to learn the parts of the corridor which it does not visit often pointing to the importance of the replay memory management. Through this, we conclude the evaluation chapter.

5 Discussion

The goal for this thesis was to investigate the reinforcement learning method for the task of navigation in corridor environments using monocular images. We have focused on the utilization of the experience replay for decorrelating the data during the training. Herein, we have introduced, what might be an important factor for learning more complex environments, the concept of sample selectivity in the replay memory and pointed out, through multiple evaluations, the possible advantages of further pursuing such approaches.

5.1 Limitations

During the experiments, we have noticed a certain level of variation in the agent's performance. One of the factors that influences such behavior is the narrow field of view which sometimes leads to bumping into the objects in the immediate surrounding. The agent is not able to associate negative rewards when seeing long straight corridors as such samples lead to the highest rewards most of the time. This is a direct constraint of our approach and we discuss the possible ways to make the agent more robust to such situations in Section 5.3.

The obvious drawback of the experience replay approaches are the high hardware requirements for maintaining the replay memory. For our approach, we were satisfied with the results obtained by keeping 90000 samples in the memory but this number may scale with the complexity of the environment. In comparison, the DQN approach [13, 14] maintained 1000000 samples in the replay memory to reach such high level of performance. Therefore, it remains unclear to what extent we can expand the replay memory size and how complex environments our agent could eventually learn.

5.2 Learning A Real Corridor

As presented in Section 2.1, there are many difficulties for evaluating the learning process in real corridors. Therefore, we do not mention any results for such environments in Chapter 4. However, the early experiments did show the notion of learning in the real corridor. In these experiments, the agent was placed in a real corridor, similar to the I16c corridor from Section 2.2.1. By using a replay memory of size 1000 and the NoRMM approach, the agent was able to learn small parts of the corridor, including a windowed area. Hence, as this is of direct interest for the future work, we provide a video showing the behavior of the agent in the learned part of the corridor in our online repository, see Section 2.4.

5.3 Possible Improvements

5.3.1 Continuous Action Space

So far, we have considered only a simple discrete action space in our experiments. However, to utilize the robot to its full extent, it would be useful to consider the continuous action space instead. One way of extending our approach to the continuous action space is tightly related to the one that was used for learning robotic grasping from the monocular images in the work of Levine et al. [10]. It utilizes the Cross-Entropy Method(CEM) [16], a derivative-free optimization technique, to choose the optimal motions of the robotic arm. The algorithm samples a batch of possible actions from the initial Gaussian distribution, then fits a new Gaussian distribution to the best samples and repeats the process iteratively. Finally, the algorithm chooses the best action from the sampled sets and performs a grasp attempt. We can utilize the similar approach for the task of navigation in the corridor environment. The current neural network architecture takes a vector of size 3 for the action input but it can be easily adapted to fit the eventually different requirements for the continuous action space. As the RMM approach for managing the replay memory also considers the actions when looking for

the similar samples, the RMM method would need to be appropriately adjusted to fit the action space.

5.3.2 Using Pre-Trained CNN For Image Preprocessing

Instead of learning the convolutional layers for processing the image input, it is possible to use previously trained state of the art neural networks. As we are not interested into the outputs of the networks, but rather the inner feature representations, different architectures come to mind [18, 4, 23]. Therefore, the number of the learning parameters would be reduced to the parameters for learning the processing of the command input and correlating the preprocessed image and action data to predict the reward. This could have a significant effect on the learning time and the robustness of the algorithm. The other possible advantage is that we could reduce the hardware requirements for maintaining the replay memory, assuming that the extracted features have lower dimensionality than the original input images. If we fix the pre-trained CNN and use it for preprocessing only, we could use the data of much lower dimensionality for the representations in the replay memory.

5.3.3 Asynchronous Learning

Although such approach would be inconvenient in the real world environments, it is possible to reach high performance without experience replay by utilizing the multi-agent learning. For the task of playing ATARI 2600 games, Mnih et al. in 2016 [12] suggested using multiple agents for the learning task and removed the need of the replay memory in the learning process. Additionally, the results from the paper show much faster convergence to the optimal behavior of the agent compared to the single-agent DQN method. The application for our approach could also include the possibility of placing multiple agents into different corridors and investigating if the agents are able to learn multiple corridors by cooperating with each other. However, the downside of such approach for its utilization in the real

world is the difficulty of creating suitable environments and acquiring the necessary number of robots to explore those environments.

5.3.4 Expanding The Observation State

In order to improve the robustness to the nearby obstacles outside of the agent's field of view, it would be helpful to consider multiple frames for the observation state. We could incorporate this aspect by adapting the network architecture to take the tiled sequential frames as input. However, this would in return imply much higher hardware requirements for the replay memory as well.

Furthermore, we could incorporate the recurrent neural networks (RNN) to process the sequential data. The Long Short-Term Memory (LSTM) [8] was shown useful for the task of playing ATARI 2600 games by Mnih et al. in [12]. The previously learned CNN can be used as a preprocessing step for the RNN as sketched in Figure 5.1. However, the requirements of keeping the data sequential partially contradicts the RMM idea of lowering the correlation of the data during the training phase. Therefore, the RMM would require additional adjustments to fit such approach.

5.3.5 Measuring The Sample Similarity

In this thesis, we have presented a basic method for measuring the similarity between the samples in order to evaluate the importance of the sample selectivity for the experience replay approaches. In the Plus-Minus experiment, we have observed that, after being moved to a different corridor, the agent is able to preserve the samples from the previous corridor after a longer period in time. It would be of interest to perform the true evaluations of the similarity measurements and to test the approach for the features extracted from the different layers of our network. In particular, intriguing is the fully connected layer that processes the image input (*FConn1*) as it is of much lower dimensionality compared to the *Max Pool2* layer. On the other hand,

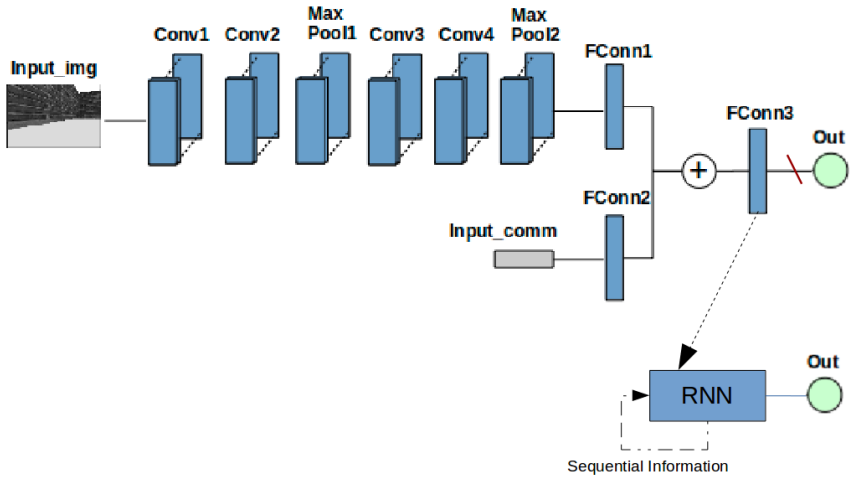


Figure 5.1: The RNN network sketch, sequential processing. Use the learned CNN as a preprocessed input for the RNN.

we could also consider the fully connected layer that combines the processed image and action inputs (*FConn3*) and intuitively removes the need of checking whether the action is the same for the compared samples.

5.4 Conclusion

Through this, we conclude our findings. We have shown that the reinforcement learning on its own can be a good strategy for the domain of autonomous corridor navigation. Without any image preprocessing techniques that might be helpful for this task, we have seen that the agent was able to perform well for the learned corridor. We have exposed some of the issues for the experience replay and presented the RMM approach that considers the sample similarity for the replay memory to circumvent those issues. We have evaluated our approach on multiple challenges and in the end shown that the agent is able to perform well in the complex corridor environments. Even though our approach still offers improvement opportunities, it shows that, with basic reinforcement learning concepts, we

can already achieve good results that motivate us to further pursue such approaches in the field of robotics.

Bibliography

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. In *J. Artif. Intell. Res.*, 2013.

[2] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF: Binary Robust Independent Elementary Features. In *Proceedings of the 11th European Conference on Computer Vision*, 2010.

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[5] Richard Held and Alan Hein. Movement-produced stimulation in the development of visually guided behavior. In *J Comp Physiol Psychol*, 1963.

[6] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *The 12th International Symposium on Experimental Robotics (ISER)*, 2010.

[7] Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. In *Proceedings of the International Conference on Machine Learning*, 2010.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. In *Neural Comput.*, 1997.

[9] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998.

[10] Sergey Levine, Peter Pastor Sampedro, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. In *International Symposium on Experimental Robotics*, 2017.

[11] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian P. Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. In *International Conference on Robotics and Automation*, 2010.

[12] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

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

[15] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[16] Reuven Y. Rubinstein and Dirk P. Kroese. *The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-carlo Simulation (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2004.

[17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. In *Int. J. Comput. Vision*, 2015.

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. International Conference on Learning Representations*, 2014.

[19] Dirk Pensky Stefan Schäberle. *Robotino Manual*. FESTO.

[20] Sebastian Thrun and John J. Leonard. *Springer Handbook of Robotics*, pages 871–889. Springer Berlin Heidelberg, 2008.

[21] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[22] Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[23] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *Advances in Neural Information Processing Systems*, 2014.

[24] Frederik Zwilling, Tim Niemueller, and Gerhard Lakemeyer. Simulation for the robocup logistics league with real-world environment agency and multi-level abstraction. In *RoboCup Symposium*, 2014.