

Final Report

|  |
| --- |
| School of Computing  Faculty of Engineering AND PHYSICAL SCIENCES |

Reinforcement Learning for Simulated Homing

Jamal Ahmed

Submitted in accordance with the requirements for the degree of  
**BSc Computer Science**

**2021/2022**

**COMP3931 Individual Project**

The candidate confirms that the following have been submitted*:*

|  |  |  |
| --- | --- | --- |
| **Items** | **Format** | **Recipient(s) and Date** |
| *Final Report* | *PDF file* | *Uploaded to Minerva (13/05/22)* |
| *Link to online code repository* | *URL* | *Sent to supervisor and assessor (13/05/22)* |
| *Trained models* | *.pth pytorch models* | *Sent to client and supervisor (13/05/22)* |
| Environments | .zip containing .sdf and .config files used by gazebo | *Sent to client and supervisor (13/05/22)* |

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) J.Ahmed

© 2022 The University of Leeds and Jamal Ahmed

# Summary

This paper explores the application of reinforcement learning techniques on the problem of autonomous navigation. The solution should be able to emulate an animals ability to find its way home regardless of the current position and especially when the target is out of sight.

The project starts by researching about the basics of machine learning and neural networks before further investigating into reinforcement learning. Research into similar approaches and their viability in solving the problem is also discussed before setting requirements and risk mitigation strategies when producing a solution.

We go on to discuss the construction of the environment and machine learning model then outline the various configurations and discuss their effects on the optimality of the converged solution. A final approach is then used to test and evaluate the effectiveness of the solution in navigating within different environments. Conclusions are then drawn about the behaviour of the model in relation to the configuration of the hyper-parameters before putting ideas forward for future work.

# Acknowledgements

*I would like to thank my supervisor Abdulrahman Altahan for giving me the freedom and guidance to explore this project.*

# Table of Contents

[Summary](#__RefHeading___Toc544_2999211779) III

[Acknowledgements](#__RefHeading___Toc546_2999211779) IV

[Table of Contents](#__RefHeading___Toc548_2999211779) V

[Chapter 1 Introduction and Background Research 7](#__RefHeading___Toc550_2999211779)

[1.1 Introduction 7](#__RefHeading___Toc552_2999211779)

[1.2 Machine Learning 7](#__RefHeading___Toc5361_1574822035)

[1.2.1 Convolutional Neural networks 7](#__RefHeading___Toc594_2999211779)

[1.3 Reinforcement Learning 8](#__RefHeading___Toc596_2999211779)

[1.3.1 Finite Markov Decision Process 8](#__RefHeading___Toc2632_2719817163)

[1.3.1 SARSA 9](#__RefHeading___Toc728_598456863)

[1.3.2 Q learning 10](#__RefHeading___Toc730_598456863)

[1.4 Deep Q Networks (DQN) 11](#__RefHeading___Toc749_598456863)

[1.5 Related work 12](#__RefHeading___Toc800_598456863)

[1.6 Requirements 14](#__RefHeading___Toc1774_1355306325)

[1.7 Risk mitigation 14](#__RefHeading___Toc5363_1574822035)

[Chapter 2 Methods 15](#__RefHeading___Toc556_2999211779)

[2.1 Environment 15](#__RefHeading___Toc2634_2719817163)

[2.1.1 The Robot 16](#__RefHeading___Toc879_598456863)

[2.1.2.1 Action space 16](#__RefHeading___Toc881_598456863)

[2.1.2 State representation 17](#__RefHeading___Toc598_2999211779)

[2.1.2.1 Image output 17](#__RefHeading___Toc883_598456863)

[2.1.2.2 Lidar Input 17](#__RefHeading___Toc814_598456863)

[2.1.3 Terminal States 17](#__RefHeading___Toc600_2999211779)

[2.1.4 Reward Systems 18](#__RefHeading___Toc834_598456863)

[2.1.4.1 Simple Reward System 18](#__RefHeading___Toc698_598456863)

[2.1.4.2 Dynamic Reward System 19](#__RefHeading___Toc700_598456863)

[2.1.4.3 In Sight Reward System 19](#__RefHeading___Toc702_598456863)

[2.1.4.4 Maximising Distance Reward System 20](#__RefHeading___Toc718_598456863)

[2.2 DQN 21](#__RefHeading___Toc560_2999211779)

[2.2.1 DQN Design 21](#__RefHeading___Toc885_598456863)

[2.2.2 DQN Parameters 21](#__RefHeading___Toc2636_2719817163)

[2.2.2.1 Mini batch and Replay memory size 21](#__RefHeading___Toc2638_2719817163)

[2.2.2.2 Learning rate 22](#__RefHeading___Toc2640_2719817163)

[2.2.2.3 Gamma Value 23](#__RefHeading___Toc2642_2719817163)

[2.2.2.4 Epsilon values 23](#__RefHeading___Toc2686_2719817163)

[Episodic decay 24](#__RefHeading___Toc690_598456863)

[Episodic and Action based decay 24](#__RefHeading___Toc692_598456863)

[Reward based epsilon 24](#__RefHeading___Toc694_598456863)

[2.3 Locations 25](#__RefHeading___Toc781_598456863)

[2.3.1 Randomising target location 25](#__RefHeading___Toc783_598456863)

[2.3.2 Randomising robot location 25](#__RefHeading___Toc785_598456863)

[Chapter 3 Results 27](#__RefHeading___Toc562_2999211779)

[3.1 Final approach 27](#__RefHeading___Toc606_2999211779)

[3.1.1 8m x 8m Room training and testing results 28](#__RefHeading___Toc734_598456863)

[3.1.2 10m x 10m Room with partitions 29](#__RefHeading___Toc736_598456863)

[30](#__RefHeading___Toc5997_1574822035)

[Chapter 4 Discussion 31](#__RefHeading___Toc564_2999211779)

[4.1 Conclusions 31](#__RefHeading___Toc566_2999211779)

[4.2 Ideas for future work 32](#__RefHeading___Toc1776_1355306325)

[List of References 33](#__RefHeading___Toc570_2999211779)

[Appendix A Self-appraisal 35](#__RefHeading___Toc572_2999211779)

[A.1 Critical self-evaluation 35](#__RefHeading___Toc574_2999211779)

[A.2 Personal reflection and lessons learned 35](#__RefHeading___Toc576_2999211779)

[A.3 Legal, ethical and professional issues 36](#__RefHeading___Toc578_2999211779)

[A.3.1 Legal issues 36](#__RefHeading___Toc580_2999211779)

[A.3.2 Ethical issues 36](#__RefHeading___Toc586_2999211779)

[A.3.3 Professional issues 36](#__RefHeading___Toc590_2999211779)

[Appendix B External Materials 37](#__RefHeading___Toc592_2999211779)

[B.1 DQN 37](#__RefHeading___Toc5452_1574822035)

[B.2 Environments 38](#__RefHeading___Toc5365_1574822035)

[B.2.1 8x8 Room 38](#__RefHeading___Toc5454_1574822035)

[B.2.2 10x10 Room with Partitions 38](#__RefHeading___Toc5456_1574822035)

[B.2.3 Turtlebot3 plaza 39](#__RefHeading___Toc5458_1574822035)

[B.2.4 Turtlebot3 house 39](#__RefHeading___Toc5460_1574822035)

# Chapter 1 Introduction and Background Research

## 1.1 Introduction

Homing is an animal’s ability to return to some initial starting position. Formulating models of this skill can act as a basis in providing solutions to the problem of autonomous navigation for robotics. To reduce failure costs when developing a solution, visualisation software can be utilised to simulate real world scenarios.

Reinforcement learning is a machine learning paradigm which imitates natural learning processes to produce solutions to highly complex/abstract problems. This is done through a repeated process of optimising an agent to produce actions which seek to cumulate some long term rewards determined by an environment.

Various abstractions of the learning process can be utilised in order to optimise the success, convergence and generalisation of the solution.

The implementation and modification of various learning techniques in order to train a convolutional neural network to simulate a robot with generalised navigational abilities to reach some predetermined destination are discussed in this paper.

## 1.2 Machine Learning

Machine learning is a subset of artificial intelligence aiming to create a model which imitates the brains ability to adapt to recognise patterns from some input data through a repeated process of fitting the model to the input. The momentum of updates can be varied based on the estimated error in the ability of our model to the optimum solution (Marsland 2015). This chapter explores different learning models that could be applicable in solving the problem of automatic navigation.

### 1.2.1 Convolutional Neural networks

Neural networks are a class of algorithms built to emulate the pattern recognition ability of the human brain. Networks are constructed from a number of artificial neurons each applying an updatable function to an input producing a value to estimate the presence of some pattern. Inputs can be data representing some information available to a particular problem or the output of another neuron. Utilising multiple layers allow a network to deduce more complex patterns of different features in a dataset(Marsland 2015). The training data available will affect what patterns the network learns to deduce and often the quality of the convergence. Training data that is more relevant to the problem we are trying to solve will allow for the network to better converge to a solution.

Updates to a network are calculated by estimating the error in classification of an input and updating the neurons functions to better fit to the intended output. Different types of optimisers can be used to better estimate the error such as Momentum and RMSP. When updating a network the amount of change will affect the stability of a model and a hyper-parameter called the learning rate can be used to control this. (Marsland 2015)

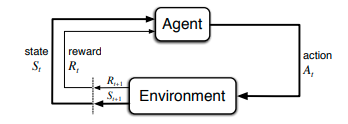
Convolutional layers can be utilised in order to learn feature detection in images and has a structure based on the visual cortex. They use a combinations of kernels of different sizes allowing them to utilise different amounts of the input to make a decision as well as stride length which determines the step size when moving across an image (Alejandro Escontrela 2018) . The number of input channels and the kernel configuration of a layer will determine the number of output channels to be used by the next layer. The configuration of the convolutional layers will directly affect the complexity and computing resources required to train a model.

## 1.3 Reinforcement Learning

Reinforcement learning is a class of machine learning algorithms in which we have an agent that’s primary goal is to interact with an environment and seek to maximise the cumulative reward from taking actions within that environment. The agent is not implicitly told what actions will lead to positive rewards however it must learn these through experience (Richard S. Sutton 2020 1.1). This is both an advantage and a disadvantage of reinforcement learning as it allows the agent to find novel solutions to problems which we may have not considered however the time to find such solutions is often time consuming.

### 1.3.1 Finite Markov Decision Process

This is a model used for reinforcement learning problems that describes a sequence of events where the probability of an event is dependent upon that preceding state. The model is comprised of an agent and an environment. The agent is the learner and decision maker and the environment is everything outside of that. The agent continually selects actions to take and the environment responds with new state information as well as some numerical reward that the agent seeks to maximise over time (Richard S. Sutton 2020 3.1).



The agent learns Q values of state actions pairs which are the estimates of the long term rewards of certain behaviours. These Q values are stored in a Q table and can be improved in different ways depending on how we choose to explore an environment and learn from it. This can be seen as a mapping of state action pairs to Q values (Richard S. Sutton 2020 3.1).

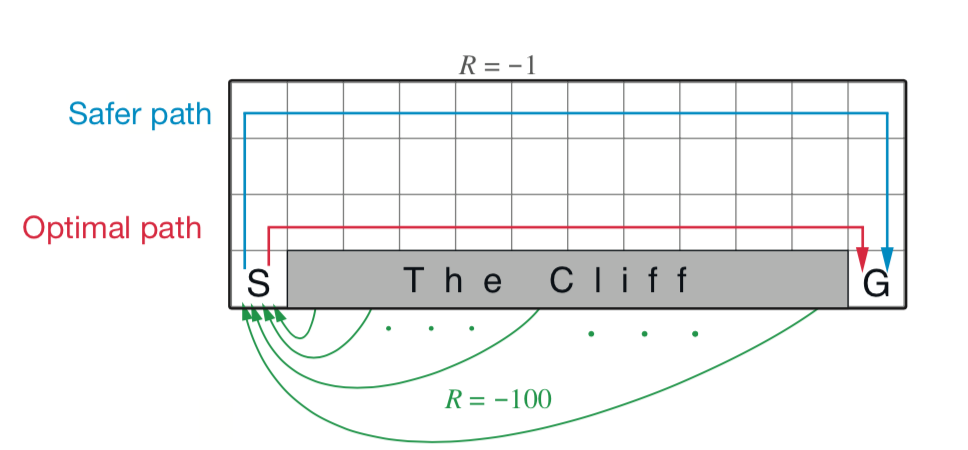
### 1.3.1 SARSA

SARSA is an on policy strategy where actions taken by the agent during training and testing are chosen based on the current policy learnt by the agent.

The Q value for an action are updated based on the reward for selecting that action as well as the reward from the following state assuming the current policy is used.

SARSA given enough training time will eventually converge to the optimal solution (Richard S. Sutton 2020) However it is more likely to converge to a solution that is less optimal at first as SARSA tends to be a more conservative training algorithm and learns to choose the safest path rather than the most optimal as can be seen in the cliff walking example.

This is preferred for when we want the agent to quickly learn about negative behaviour and states in the environment so that the robot doesn’t explore areas which lead to failure. In the context of mapless navigation this method of updating the Q Table would therefore be beneficial as there is a high cost for unwarranted behaviour such as collisions. However this depends on the environment in which the agent is trained in as if the path to the target destination is surrounded by negative states SARSA will learn to explore those states only after it has exhausted all other safer paths.



### 1.3.2 Q learning

Q learning is an off policy learning method. This means that it uses one strategy of choosing actions for training the model and another strategy for testing. During training an epsilon greedy policy is used where actions are either exploratory or chosen based on the currently learnt model(Richard S. Sutton 2020 3.1). In testing actions are solely based on the policy learnt by the agent.

Epsilon greedy exploration strategy is a commonly used method for dealing with the trade off between exploration and exploitation of the model. A random number between 0 and 1 is generated and if this value is less than the current value of epsilon the agent chooses a random action whereas for values larger than epsilon the agent chooses to exploit the information previously learnt from its experience. The value of epsilon will directly influence how we choose to explore and learn about the environment.

Q learning converges quicker to an optimal solution as it maximises the reward of the next state with greedy action selection(Richard S. Sutton 2020 3.1).

Q learning explores riskier states and can therefore find shorter more optimal paths than SARSA due to its exploratory nature during training however this does mean that Q learning methods may partake in negative behaviours more while converging to a solution.

## 1.4 Deep Q Networks (DQN)

This is an extension of reinforcement learning and deep learning where we use a neural network to estimate the Q value for different states rather than a Q table (Chris Yoon 2019). This allows an agent to map input states to action Q value pairs rather than mapping state actions pairs to Q values. This allows us to exploit higher dimensional input to learn about the optimal actions given an input state.

The structure of the neural network will affect the agents ability to recognise states and the number of output neurons correlate to the number of actions the robot can take within an environment.

The use of a replay memory and batch size are used to remove the correlation in consecutive data as not using this would only allow the agent to learn about experience that occurred sequentially within the environment. This would cause the model to be over fit to certain types of experience rather than generalising about what good behaviours are.

When training a DQN multiple hyper-parameters are used to influence certain behaviours within the final model. These factors affect how much the agent looks to seek long term reward as well as the magnitude of change to the neural network.

## 1.5 Related work

Reinforcement learning has also been combined with deep learning techniques so that the agent can learn from high dimensional input such as video and images. (Volodymyr Mnih 2013) is one of the most renowned examples of this where an agent was trained using a deep Q network to learn state of the art policies to play six out of seven Atari 2600 games using the game pixels as input. The RGB pixels were used as input to a convolutional neural network which allowed the algorithm to optimise the filters for feature selection rather than the filters having to be engineered using prior knowledge. This independence allows the algorithm to hone in on patterns in the input that humans may not be able to discover or perceive. The deep q network used here would be classed as end to end as the input and reward system are interlinked.

Others have successfully made models that can achieve the task of homing and autonomous navigation however most if not all these models rely on some form of mapping to allow the robot to navigate the environment. Examples of this would be self-driving vacuum cleaners. These devices use sensors to be able to detect objects and cliffs as well as estimate the robot’s current position in a space and use this to create a virtual map of the environment. This technique is known as SLAM (simultaneous localisation and mapping). This approach while appropriate in most environments might not be suitable for dynamic environments in which obstacles may move seemingly random to the device. An example of such an environment may be shop floors or restaurants that may change seating areas depending on reservations as group sizes may vary.

The work proposed by Pengpeng Zhang (2019) shows an autonomous robot equipped with deep reinforcement learning techniques that allow for mapless navigation however this approach uses 10 dimensional laser readings as their sensory input rather than visual data. This method while allowing for autonomous navigation doesn’t teach the agent about how to navigate an environment using visual imagery. A combination of visual imagery and laser sensor data could be used to create a model that would port over to a real world environment more effectively as images alone may not generalise well to object avoidance however laser sensor data would.

Liulong Ma, Yanjie Liu and Jiao Chen (2019) propose a model for maples robot navigation using RGB images as input. Their approach used 3 convolution layers to extract features from the input as well as two fully connected dense layers to determine Q values for the discrete action space. This was further improved by decoupling the visual feature model from the Deep reinforcement learning network which allows for easier migration to new environments according to the authors.

The model learned quickly and efficiently in simulated environments however this is not a strong indication of how well the agent will perform in the real world. The decoupling of the visual network and the reinforcement learning algorithm improved the sample efficiency and generalisation of the agent to new environments. This methodology of separating representation learning, and policy learning is relatively new and has been explored by (Adam Stooke 2021) where they show an unsupervised learning model which trains a convolutional encoder to associate pairs of observations within a short time span. The testing shows that the use of the encoder matches or outperforms end to end reinforcement learning models in most environments however the flexibility and improvement gained from reward free representational learning will likely add more complexity to the solution than it is worth especially for real world environments.

Using a simulation to train the model allows for faster training as the environments can be reset and changed quicker which in turn allows us to generate large amounts of training data for the agent to learn from with ease. We can also simulate environments that would be hard to train an agent in such as space or roads as there is a high cost of failure and safety issues. However, by using a simulation this also means that we have a potential for over fitting to the simulation and creating a model that would not generalise well to the real world. Simulating complex environments also increases the computational cost required to train the model which may hinder the performance of the robot as the agent may not be able to process the state changes fast enough due to the extra computation required to simulate the environment.

The efficacy of using a simulation to train a reinforcement learning agent that will perform in the real world has been explored by Blazej Osinki(2020) where they propose the idea that good performance in a simulation doesn’t equate to proper function in the real world. They show that discrete action spaces generally lead to weak performance in the real world and that regularisation and model-based approaches had the best operation. The models used by Blazej Osinki(2020) were for real world autonomous driving which while relevant are subject to a lot of external factors that would not be applicable to the scope of this paper. For example, a simulation of city roads could not accurately factor in all aspects of the real world however a simulation modelling a home environment or restaurant is a lot more feasible as there are less factors at play.

## 1.6 Requirements

Implementation of a reinforcement learning methodology to simulate human learning to train a model to control a robot within some simulated environment.

The agent should have the ability to recognise some object or target destination within the environment and navigate towards it.

The robot should be able to avoid obstacles and partitions whilst moving to find the target which can be within sight or hidden from the agent at any given time.

## 1.7 Risk mitigation

To minimise risk existing methodologies can be utilised before modification to fit the intended purpose

Limited computing power will affect the quality of the solution thus we can minimise the complexity of the problem and input to improve performance through various means:

The environment can be simplified at first before trying to increase realism so that computing resources are focused on the agents ability in learning rather than the simulation. The simulation can also be run headless meaning no graphical output is displayed to the user while training which will further reduce the computing power needed.

The state input can be reduced and abstracted to contain the most significant information the agent would need to make a decision. This can be achieved by reducing the dimensionality of the image.

# Chapter 2 Methods

Due to the nature of reinforcement learning a models efficacy can only be determined once the model has been trained a sufficient amount of times within the environment. This means any changes made to the training algorithm require a new agent to be trained which is time costly. To maximise the chance of success, reinforcement learning techniques which have been successfully implemented for other applications will be applied before suggesting further improvements to fit the intended use case. Models in this section were trained for 100 episodes to test the feasibility of different training methods before concluding to a final approach.

At first the weighting of the DQN can be initialised randomly. Once a better model is produced it can be used for initialisation allowing for retraining which reduces training time and allows for fine tuning of a solution by using different hyper-parameters than the ones initially used to train.

## 2.1 **Environment**

The environment is an important factor when training a machine learning model as this will determine what and how the agent learns .The environment is the surroundings of the robot such as the target position and obstacle information but also consists of the agents actions as well as a representation of the state of the environment which includes the agents orientation and position. The reward system also falls under the environment component as this aspect will take the agents action and return numerical information about the value of that action given that state.

Environments can be simulated using gazebo which is a visualisation software that provides a building editor to create partitions and rooms within a simulation. The target object was designed using the model editor and given the colour green to avoid compatibility issues with colour representations used between the Robot Operating System(ROS) and OpenCV. ROS is an open source software package that contains libraries to aid in the creation of a roboticised system (Robotis 2022). OpenCV is a python library used for real time computer vision and converts the images received from ROS into a format which is python compatible

The gazebo messaging service can then be used to dynamically add and remove targets inside the simulation (Open Robotics 2022).

### 2.1.1 The Robot

The TurtleBot3 waffle\_pi was chosen as it has an RGB camera as well as laser sensory information allowing the robot to detect the distance between itself and various objects. The robot has a max linear speed of around 0.3m/s and an angular speed of 1.8rad/s. The documentation provided by turtle bot can be used to setup gazebo and ROS to allow communication between the simulation and the DQN (Robotis 2022).

Automatic Addison (2020) provides further details on how to configure and run turtlebot models within gazebo.

#### 2.1.2.1 Action space

The action space is the actions that the agent will be allowed to perform in the environment. The actions that an agent takes directly influences the states it will explore therefore it is important to test multiple different action spaces to see which actions allow the agent to explore its environment optimally.

Initially an action space where the agent could move forwards backwards,rotate and stop was used. This was unsatisfactory as when the agent was moving backwards the camera image it would receive from the environment would be the same as if it was moving forwards. The agent also learnt behaviours such as rotating endlessly as this would lead to the least collisions. This could be accounted for by further adding more conditions to the reward system however this would increase the complexity and noise within the training data.

The action space was then changed so that the robot had a constant linear velocity and could alter its angular velocity by some value. This prevented the agent from learning indefinitely rotating models however it increased the complexity of the solution as in some states the agent could not do anything to avoid collision as it could only take 3 actions. Various angular velocities were then used so that the agent could learn to use more angular velocity when needed. This however posed another problem in that the agent could not process the states quick enough for the action that it chose to be relevant to the current state of the environment. To fix this the linear speed of the robot was reduced to 0.15 and the angular velocity was either +-0.3 or +-0.15 or 0. This setup was the best compromise between speed of the robot and the training time of the agent.

It was further found that an optimal solution with faster speeds could be achieved in a feasible amount of time by reducing the graphical output of the simulation. Convergence for faster speeds was still slower however by retraining models using slower speeds with faster ones the model converged much quicker. This is due to the agent already having learnt what actions cause the environment to react in a certain way however now the magnitude of change is greater and has to be adjusted for.

### 2.1.2 State representation

#### 2.1.2.1 Image output

The turtlebot3 outputs a 640x480 BGR image which was used as a representation of the environment for the agent however this input was too large to run effectively on the hardware available so this was reduced to an 84x84 image and then converted to greyscale reducing the dimensionality to one simplifying the input and reducing computation time whilst retaining significant information about the current state. The input was further modified to convert the target pixels to 255 to increase the contrast between other objects and the target. This was achieved by iterating over the pixel information and comparing the values to the colour of the target and overwriting those values.

#### 2.1.2.2 Lidar Input

The turtlebot3 waffle\_pi also includes lidar data which gives distance information by firing lasers and measuring the time it takes for the laser to return to the robot. The system can measure surfaces up to 3m away in the simulation and returns np.inf for values further than that as the distance could not be accurately measured. np.inf cannot be used as an input for the DQN so this is replaced with an upper bound of 10. According to the documentation provided by turtle bot 24 laser data points is sufficient for an agent to learn about an environment but in this case 84 data points were used as this was equal to the width of the image input. The array can then be appended to the 84x84 image matrix to form an 85x84 matrix which is converted to a tensor using the pytorch framework before being given to the DQN.

### 2.1.3 Terminal States

Terminal states are used as an indication to the agent that the actions leading up to the current state have caused the agent to end the experience. This tells the agent that the task or goal has been completed and the environment can be reset and a new episode can take place for the agent to learn from. Using only one terminal for when the agent successfully found the target led the agent to spend a lot of training time exploring states that were unsatisfactory such as colliding with objects and getting stuck at walls. To combat this collisions were set to be a terminal state which would encourage the agent to not approach states that would cause it to collide with objects as this would result in a negative episode. Upon colliding the robot is reset to its initial position within the environment. However by considering both negative and positive terminal states the influence of training iterations changes as when training an agent we generally want to train an agent for a certain number of successful episodes. This is factored for in the training loop of the DQN so that episodes are incremented only for positive terminal states. This condition allows for better replication of training results as an agents behaviour will vary widely across training sessions and may collide more often initially due to the weight initialisation and exploration of the environment.

It was also considered to not have collisions as a terminal state but rather assign a negative value and reset the robot to its initial position however this caused the agent to learn that if it is in an unsatisfactory state it can reset its position by crashing which is unwanted behaviour.

A third terminal state can be implemented which is returned to the agent when a certain number of actions have been taken. This is used to discourage the agent from using training time exploring paths in the environment that would be considered safe and good behaviour but are not optimal paths to the target destination.

### 2.1.4 Reward Systems

The reward system is a part of the environment system. The reward function receives an action taken by the agent and returns information about how good that action was as well as the representation of the environment and if the current state is terminal. Therefore it is important to choose a reward system that will allow for optimal learning and generalisation of the environment.

Another factor to note is that the gamma factor used for training the agent will affect how successful a reward system will be as this parameter controls how much the agent seeks future reward. Generally values are set between 0.9-0.99.

#### 2.1.4.1 Simple Reward System

This reward system is where we set every action taken by the agent to be a value of -1. Successful terminal states will have values of +100 and -100 for unsuccessful states. As the agent explores the environment every action has the same value so long as the agent isn’t stuck. As the agent completes more training episodes states that lead up to the target will have larger Q values due to the gamma factor which controls how the agent seeks long term reward. States that lead to collision will therefore have lower Q values. This reward system given enough training time will converge to a solution however an assumption that all states that are not the target or a collision are equal is made which is an abstraction. This means that to learn a representation of the utility of a state the agent has to reach a terminal state. This model heavily relies on experience and this can be seen in the time it takes for the agent to converge to a solution.

#### 2.1.4.2 Dynamic Reward System

This reward system had been previously used for similar tasks using only laser sensory data and was shown to be effective (Robotis 2022) . The reward returned to the agent is calculated based on the distance and angle to the target square. This takes into account that all states that are not terminal are not equal and have varying amounts of utility. An equal weighting between distance and angle was set with a max distance of 5M and max angle of 180 and the rewards were mapped between 0 and -1.. This model was able to train effectively in a simple 4x4 room however when moving to more complex environments the agent started to act inappropriately especially when the target was out of sight. This is due to the same state information having varying rewards. For example when the agent is approaching a wall in one episode the wall may have a reward of -0.5 and in another -0.1 due to varying target positions. The same state has different utilities across episodes which is true however this fluctuation can not be utilised effectively by the DQN as it causes the model to over fit to individual episodes.

#### 2.1.4.3 **In Sight** Reward System

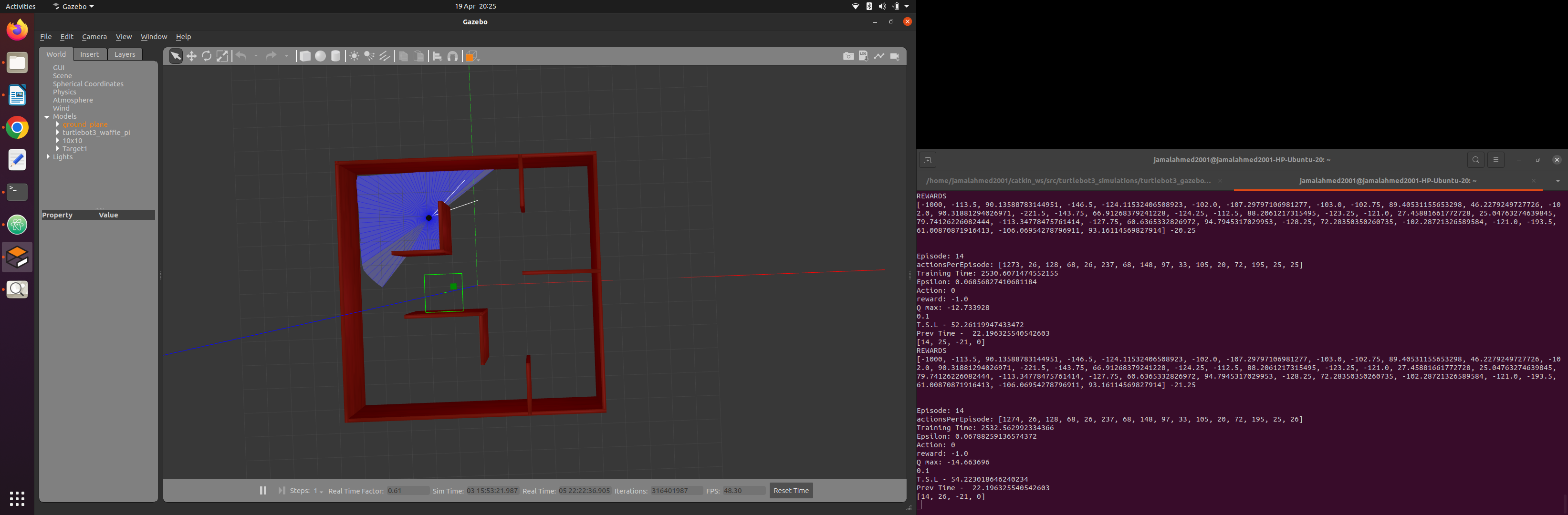
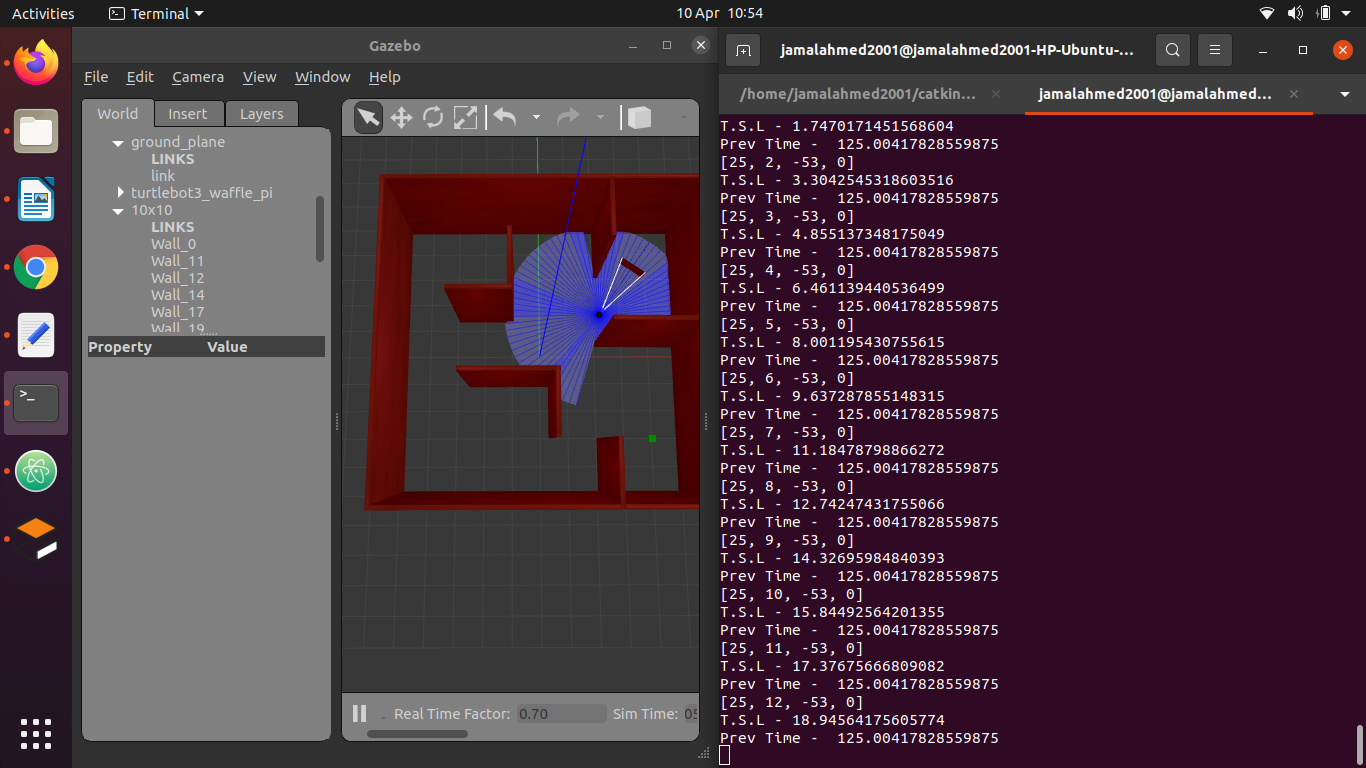
To combat the shortcomings of the dynamic reward system whilst also taking its strengths the system was modified so that if the image state contained the target the dynamic reward system is used and the reward is calculated based on distance and angle. However if the target is not within sight of the agent then a reward of -1 is returned. This keeps the reward system consistent as only images with the target have varying rewards and the rewards correspond to the state returned. All other states are equally bad and so the agent learns to find the states where it can see the target and then learn to approach the target. This encourages behaviour that moves the agent closer to the target only when the agent can see it.

#### 2.1.4.4 Maximising Distance Reward System

To optimise how the agent should explore the environment when the target is not within sight a condition was added such that if the distance in front of the robot is larger than all other distances received the reward returned to the agent was -0.75. This tells the agent that states with more distance in front of the robot have better utility than states with less distance to an obstacle. It can be seen in the images below that paths following the light blue which indicate areas of maximum distance will tend to lead to optimal behaviour. Once the target is within sight the dynamic reward signal is used and mapped to return values of 0 to -0.5.

|  |  |
| --- | --- |
| **Action** | **Reward** |
| Target found | +100 |
| Collision | -100 |
| Target in sight | -0.5 to 0 |
| Maximising distance | -0.75 |
| Not maximising distance | -1 |

Taking this further another condition was added to the reward system where if the agent was maximising the distance in front of the robot i.e. the distance in front of the robot was greater than the distances returned by the other lasers the reward returned was -0.75 and any state where the agent is not maximising the distance the reward was -1 unless the target was in sight in which case the dynamic reward system was used. This implementation allowed for the agent to explore the environment more effectively when the target was not in sight. This can be seen in the screenshots below where the light blue indicates maximal distances.



## 2.2 **DQN**

### 2.2.1 DQN Design

The construction of the DQN was implemented using PyTorch. The DQN was composed of 3 convolutional layers and 2 dense layers. The first layer took an input of 4 frames of 85x84 matrices and used an 8x8 kernel with stride 4 and outputted 16 channels. The 16 channels were inputted into the second layer which used a 4x4 filter with stride 2 and outputted 32 channels. This was then inputted into the final convolution layer where a 3x3 kernel with stride 1 was applied and outputted 64 channels.

The first dense layer had 3136 inputs and 512 outputs which were inputted into the second fully connected layer which outputs to the number of actions the agent can take in the environment.

This implementation was chosen as it has been effective in solving other reinforcement learning problems such as playing Atari games shown by Volodymyr Mnih and digit recognition demonstrated by Alejandro Escontrela.

### 2.2.2 DQN Parameters

When using a DQN there are multiple parameters that can be changed that will affect the learning of an agent. To optimise the computational speed and convergence of the solution multiple values should be tested. .

#### 2.2.2.1 Mini batch and Replay memory size

Replay memory size sets the length of the array used to store the experience of the agent as it explores the environment. Using a large replay memory is not storage efficient and slows down learning due to experience of unsatisfactory actions being sampled during the update of the network repeatedly in place of better actions that have been taken. In contrast small values tend to get stuck in local minima due to the memory only consisting of experience produced by the currently learnt model.

The mini batch size determines the amount of data that will be used from replay memory to compute the gradient during an episode. Mini batching allows for the training of weights to be updated multiple times over an episode. Using a mini batch size equal to the episode length is known as full batch learning and often produces models that perform well in the training environment but lack generalisation. Online learning can be achieved using a batch size of 1 which tends to produce models with better generalisation but with longer training times and less optimal solutions.

Large replay memory can be offset with small batch sizes as this will cause the network to be updated repeatedly over the course of an episode hence increasing the probability of beneficial decision making can be included in the update of the network. The opposite also holds true given that the length of memory is at least equal to the length of an episode.

Values for both the replay memory and batch size that have previously yielded optimal models for other problems were tested before settling on a range of values that produced satisfactory results for the convergence of the solution as well as the optimality of navigation and obstacle avoidance. Models were given a gamma value of 0.99 and epsilon range of 1 to 0.05 and trained for 100 episodes in environment B.2.2.

|  |  |  |
| --- | --- | --- |
| Batch size | Success:Unsuccessful | Observations |
| 2 | 26:45 | Agent drove erratically and aimlessly |
| 8 | 24:37 | Model seemed to drive aimlessly until the target was in sight however a preference for obstacle avoidance was shown |
| 10 | 23:50 | The model behaved similarly to batch size of 8 however the obstacle avoidance seemed to be better |
| 16 | 12:32 | Agent constantly oscillated while driving |
| 32 | 30:15 | Model drove smoothly and effectively |
| 64 | 50:50 | Given good starting positions the model would find the target effectively. However in low value states the model failed to navigate effectively |
| 128 | 32:25 | The model took longer to train however the agent seemed to explore purposeful states within the environment |

#### 2.2.2.2 Learning rate

This learning rate influences the effect that an update has on the networks weights. Values closer to 1 causes the weightings to fluctuate more and tend to over fit to the current state of the environment thus failing to converge to a solution. Smaller step sizes will converge to an optimal solution at the expense of training time.

The Adam optimiser is an optimisation algorithm that incorporates the use of momentum and Root Mean Square Propagation to calculate gradient descent. This allows for the optimisation to pass over local minima more effectively whilst reducing variance in a global minimum (Marsland 2015).

To contextualise the learning rate optimisation to this problem the optimiser can be re-initialised upon every terminal state with a learning rate mapped between a suitable range such as 1x10-5 and 1x10-10. The value is mapped based on the agents success during an episode which can be quantified by the number of collisions(negative terminal states) that the agent executes during an episode. Smaller learning rates were used to prevent divergence to the optimal solution.

#### 2.2.2.3 Gamma Value

The discount factor governs the importance of future rewards in comparison to immediate reward. By considering how far in the future and to what extent a states reward should influence the current state we can approximate the discount factor to use. For example if we want the agent to consider 10 actions into the future and assign 50% of the current states value to the 10th state we would use a discount factor of around 0.933 = 0.51/10.

weighting1/number of future states

This approximation is dependent upon the type of reward system and is tailored to simpler reward systems but does provide values that work for complex reward systems. This is intuitive as using values above 0.9 considers 50% of the value of the 7th proceeding state and given sufficient training time the positive rewards of the target states will start to bear more weighting. Smaller values of gamma can be used for reward systems that better represent the utility of a state as the agent can rely less on the value of future rewards to determine the current states value. This should improve the generalisation of the navigation.

With simple reward systems small values of gamma train a model that moves erratically and navigates within the environment aimlessly. Values between 0.9 and 0.99 converged to solutions which could effectively navigate the environment.

#### 2.2.2.4 Epsilon values

The epsilon value determines the frequency of committing random actions within the environment, This value ranges from 0 to 1 and is decremented over training in a multitude of ways. The method of decrement will influence the choice of initial epsilon. When using simpler decay systems large initial randomness while beneficial in terms of experience may not be exploited effectively resulting in little to no convergence Using too small a value however exposes the agent to getting stuck in local minima due to lack of exploration. The training environment will also affect the choice of epsilon as simpler environments can start with higher values of randomness due to the higher probability of successful episodes.

#### Episodic decay

Episodic decay is where the value of epsilon decreases with every successful episode the agent manages to complete in the environment. The epsilon value for an episode is calculated using np.linspace which returns an evenly spaced sequence of numbers starting at the models initial epsilon value and ending at the models final epsilon value. The length of the returned array is equal to the number of training episodes. The success of this strategy therefore depends on the values set for the initial and final epsilon as well as the number of training episodes as the interval between epsilon decrements will affect how quickly the model learns.

This epsilon strategy has been used in simple implementations of a DQN because the more the agent finds the target the more successful experience the agent has learnt about the environment and thus we are able to rely on the trained model to help the agent explore more successful states.

#### Episodic and Action based decay

Action based decay is where the value of epsilon for an episode decays by a discount factor for every action the agent takes in the environment. This means that as the agent takes an action in the environment it will start to make less random actions. This should reduce the number of redundant states the agent has to explore as this will allow for a large initial epsilon to be used letting the agent explore a lot of states early on but then exploiting the information learnt a lot sooner. This means that the agent gets to explore the environment for a given episode before it starts to exploit the information learnt from that as well as previous episodes.

#### Reward based epsilon

Reward based epsilon is where the value of epsilon for a given action is set based on the ratio between the current episode reward and the previous episode reward. This method worked better for more complex reward systems as for some episodes in simpler reward systems the previous episode reward would be positive due to the agent finding the target very quickly whilst the current episode reward would always be negative until the agent found the target. This then caused the epsilon value at the start of the episode to be set to the models final epsilon value and increase randomness as the agent took more actions rather than starting at the value set by the episodic decay array and decreasing randomness as the agent took more actions and then increasing randomness once the agent exceeded the previous episode reward. In more complex reward systems the rewards were lower meaning the episode rewards never exceeded 0 allowing for the epsilon strategy to work effectively.

## 2.3 **Locations**

### 2.3.1 Randomising target location

Randomising target locations exposes the neural network to multiple paths within the environment ensuring the solution is derived from various experiences to prevent over fitting to specific episodes. The model is more likely to converge to a solution that explores all areas of the environment in order to reach states where the target is within sight Training episodes are of varying difficulty and length allowing the agent to build solutions to simpler problems within the environment before proceeding to paths where the target is out of sight. Arbitrary locations can hinder the performance of a model as randomness may lead to repeated exposure to episodes of either simple or complex solutions so by choosing specific points within the environment and randomising between those spots we can have more control over the exploration and solution. Large samples of target locations converge to optimal solutions but are subject to slow convergence. In contrast small sample sizes provide insufficient exploration of the environment. When choosing the sample size, distinct points were selected to ensure that the target would be in all areas of the environment.

### 2.3.2 Randomising robot location

To further improve generalisation the robot starting position was also randomised for every episode. The initial position can be selected from the same points as the target locations so long as the current target and robot are not spawned in the same location.

This allows for the agent to explore more of its environment in a shorter number of episodes and learns to find the target from all areas of the map. Successive episodes will therefore converge quicker as the agent will have been exposed to a larger number of optimal paths and will be able to recognise more valuable states quicker. Assuming there are n randomised locations selected for the environment then there will be n2 different paths that the agent will experience. The number of paths can be further increased by randomising the orientation of the robot which can prevent the agent from getting stuck at unsuccessful models.

# Chapter 3 Results

## 3.1 **Final approach**

An agent that effectively navigates an environment with repeatable results can be consistently obtained using the following configuration

An environment with partitions can be utilised for the agent to learn general navigation ability in order to reach the destination. A Robot with a consistent linear velocity of 0.3m/s prevents the agent from exploiting inaccuracies within the reward system and spending training time producing futile results. The agent can choose between 5 discretised actions within the environment to change the angular velocity of the robot. The robot can move diagonally both left and right by using an angular velocity of +-0.3m/s. An action setting the angular velocity to 0 permits linear navigation of the robot. 2 further actions with angular velocities of +-0.6 allow for turning that is twice as fast as the linear movement of the turtle bot so that the agent can steer more effectively in difficult scenarios.

State information is provided through the use of an image from the turtle bot camera and 84 distance points that indicate the distance surrounding the agent. The final state input constitutes of an 84x84 greyscale adjusted image with an appended row containing laser sensory data. Colour information ranges from 0 to 1 excluding pixels with the colour matching the target which are assigned values of 255. Distance measurements are rounded to 1 decimal point and lengths that are too large to estimate are assigned a value of 10.

A dynamic maximising distance reward system can be configured with the following state representations

|  |  |
| --- | --- |
| State | Reward |
| Target found(Terminal) | +100 |
| Target in sight | Linear mapping ranging from -0.5 to 0 calculated on angle and distance of agent to destination. |
| Maximising distance ahead | -0.75 |
| Collision(Terminal) | -100 |
| Other | -1 |

An adaptive learning style was used which increases the learning rate if the agent starts to exhibit more negative actions over an episode. The scale was configured to vary between 0.0003125 and 0.00003125. This range was used because the weights were randomly initialised with values from -0.01 to 0.01 and the most successful batch size was 32. The magnitude of the initial weights divided by the update size gave the upper bound for the learning rate. The lower bound was the upper bound divided by 10.

An initial epsilon of 1 and final value of 0.05 using a linear episodic and exponential action based decay was used. The current epsilon value was multiplied by 0.99 for every action taken by the agent however when the agent exhibited more collisions this factor was incremented up to 0.999.

The agent while training, takes up to 2 seconds to process a state-action pair whereas in testing, the agent performs one action per second on average. This information can be used to estimate the number of actions required for the agent to find the target.

To test the generalisation capability of the models, the agents were tested in environments that they had not been trained in before. The model trained in the 8x8 room was tested in the 10x10 room with partitions and vice versa. The 10x10 room model was also tested in a simulated flat environment.

### 3.1.1 8m x 8m Room training and testing results

The longest optimal solution in this environment would be a straight line from corner to corner which would be 8m and would require the agent to take around 27 actions to travel that distance. Using an upper limit that is too close to the worst case scenario for the optimal solution would cause over-fitting to the current environment so a limit of 60 was chosen. This allows the robot to travel up to 18m to find the target in an environment with an area of 64m2.

|  |  |
| --- | --- |
| **Hyper-parameter** | **Value** |
| Gamma | 0.975 |
| Initial epsilon | 0.5 |
| Final Epsilon | 0.05 |
| Replay memory size | 10000 |
| Mini Batch size | 32 |
| Training episodes | 50 |

Tab*le 1: 8*x8 Parameters

This model was tested for 100 episodes and found the target within the specified action limit on all occasions. Anecdotally the agent learnt to rotate within the environment at the higher speed until the target was within sight and then slowed down the rotation and move towards the goal which can be seen as an optimal solution to the context of this environment.

When tested in an environment where the target could be hidden behind a partition this model was able to find the goal 52 times out of 100 testing episodes.

### 3.1.2 10m x 10m Room with partitions

Using the same configuration as the 8x8 room resulted in a less than optimal solution as the agent could not effectively navigate the environment to find the target when hidden behind obstacles.

The longest shortest path in this environment is more difficult to calculate so an estimate of 15m was used to set an episode length. The limit was increased to 100 actions allowing the agent to travel up to 30m to find the destination.

Increasing the number of training episodes to 300 and using an initial randomness of 1 and batch size of 64 resulted in a model that could find the target given a starting orientation that was not facing a wall. When the starting position was too close to an obstacle the agent seemed to not react quickly enough to be avoid collisions.

The pth file created from training in the 8x8 room was then used to initialise the model when training in the room with partitions. This was done as the model would already have some understanding of the target and obstacle avoidance and would only have to learn how to navigate through the environment to find the destination. This shortened the training time required and the randomness needed for the agent to converge to an optimal solution and the following parameters were used to retrain the model.

|  |  |
| --- | --- |
| **Hyper-parameter** | **Value** |
| Gamma | 0.99 |
| Initial epsilon | 0.1 |
| Final Epsilon | 0.01 |
| Replay memory size | 100000 |
| Mini Batch size | 64 |
| Training episodes | 200 |

Tab*le 2: 10*x10 Retrain Parameters

The best model that was produced was created using the hyper-parameters and initialisation outlined above. When tested for 100 episodes the agent found the target 63% of the time. Given more training time and a smaller learning rate the agent could learn a more efficient model, however due to lack of time this could not be tested.

Applying this model to the 8x8 room showed that the agent could always find the target with randomised starting and destination points. When testing in the flat environment (B2.4) the agent could not accurately distinguish between green obstacles and the target, and would approach these items thus ending in a collision.

### 

# Chapter 4 Discussion

## 4.1 Conclusions

An optimal model can be achieved given appropriate hyper-parameters. Parameters can be fine tuned through a process of experimentation and observation of the success and adaptability of the produced models. The solution created can effectively train a model to simulate navigational abilities to some predetermined destination. The environments used to show this were of a simplistic nature however this was due to limited computing resources. With more powerful and efficient computing solutions there is no reason as to why this could not be implemented in a more complex and realistic environment.

The training algorithm abstracts how humans learn to produce some end result. The process assumes that at first we have no understanding of the problem and we explore the environment to gain specific understandings about the current experience. As the agent gains experience it learns a better representation of beneficial and inhibitory actions in terms of achieving the specified goal. The need for exploration decreases, and generalisation about what has been experienced can be done to grasp an overall understanding of positive actions. If the performance starts to deteriorate then using more random actions rather than relying on the learnt model allows for the agent to experience different choices which may be better than what it has learnt previously.

The batch size determines what areas of our experience we choose to learn from, and it is important that a size that allows for the model to learn from all areas is used, so that none of the training data is wasted. How the agent chooses to navigate the environment when training directly influences the experience available in memory. The convergence to a solution is then dependent upon how the agent chooses to utilise this training data in order to better learn about its goal. Influencing what is available in memory and how much the agent uses from memory to update the network will influence the stability of the model as it converges to a solution. This is done by having a custom variable (EpisodeSuccess) that indicates a notion of the success of a model during learning based on the number of negative terminal states to the number of successful episodes.

As the model trains the estimations of long term rewards improves as the estimated Q value is dependent upon successive states. This imitates the aspect of human behaviour where as infants we learn about short term survivalist behaviours before understanding the importance of long term goals.

## 4.2 Ideas for future work

Modelling the variance of the hyper-parameters to allow for contextual learning of an agent and its exploration within an environment can be done to speed up the convergence. This can be based on how behaviour and environmental factors produce better mindsets within humans and comparing these to the different configurations of hyper-parameters.

For example, in periods of increasingly negative behaviour, the agent can utilise a larger memory and batch size whilst increasing the learning rate to increase change to the neural network. Increasing the use of random actions can further help the agent explore actions with better results. When the model shows signs of improvement, larger batch sizes and a smaller learning rate will stabilise the changes made to the network, as updates will occur less often and will have less of an effect to the model.

Improving the representation of the success of the model whilst training will allow for better influence over the hyper-parameters and therefore the learning styles of the model. Calculating the shortest path to the destination and comparing this to the agent’s solution could be useful in improving the optimality of the final model. Manipulating the experience in replay memory so that positive episodes are kept longer than negative episodes will mean the algorithm samples more favourable actions and reinforces them.

Utilising cloud computing would allow for faster training times due to computation being completed on a GPU rather than a CPU which is better optimised for mathematical computation.

# List of References

Richard S. Sutton, Andrew G. Barto (2020). “Reinforcement Learning an Introduction second edition”

Volodymyr Mnih (2013). “Playing Atari with Deep Reinforcement Learning”

Linhai Xie (2019). Reinforcement Learning Based Mapless Robot Navigation

Liulong Ma, Yanjie Liu\* and Jiao Chen (2019). “Using RGB Image as Visual Input for Mapless Robot Navigation”

Adam Stooke (2021). “Decoupling Representation Learning from Reinforcement Learning”

Pengpeng Zhang∗, Changyun Wei∗, Boliang Cai∗ ,Yongping Ouyang∗(2019) “Mapless Navigation for Autonomous Robots: A Deep Reinforcement Learning Approach”

Bła˙zej Osi´nski1,3 , Adam Jakubowski1 , Paweł Zie˛cina1 , Piotr Miło´s1,5 , Christopher Galias1,4 , Silviu Homoceanu2 and Henryk Michalewski3 (2020) “Simulation-Based Reinforcement Learning for Real-World Autonomous Driving”

Marsland. (2015). “Machine learning : an algorithmic perspective (Second edition.)”. CRC Press.

ROBOTIS 2022 [Online] [Accessed 16/12/21] Available from: <https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/>

Open Robotics 2022 [Online] [Accessed 16/12/21] Available from: https://gazebosim.org/

Dabit Industries [Online] [Accessed 27/12/21] Available from: https://dabit-industries.github.io/turtlebot2-tutorials/14b-OpenCV2\_Python.html

Alejandro Escontrela 2018 [Online] [Acces 29/12/21] Available from: https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1

Automatic Addison [Online] [Accessed 17/12/21] Available from: https://automaticaddison.com/how-to-launch-the-turtlebot3-simulation-with-ros/

Pytorch [Online] [Accessed 3/1/22] Available from: https://pytorch.org/tutorials/intermediate/reinforcement\_q\_learning.html

Chris Yoon 2019 [Online] [Accessed 10/1/22] Available from: https://towardsdatascience.com/dqn-part-1-vanilla-deep-q-networks-6eb4a00febfb

# Appendix A Self-appraisal

## A.1 Critical self-evaluation

The implementation of the solution was completed in good time however when writing the report this was not managed effectively due to the lack of structure in my methods of learning about different reinforcement learning techniques. The research only explored a limited number of reinforcement methods and once a potential solution was found I chose to experiment with different configurations to better understand the functionality of the model. This gave me a practical understanding of the parameters however the final solution was simple in nature due to more time spent on experimentation.

The coding could have been improved through the use of better commenting and better application of git. Version control was used however in the form of creating copies of the currently working model and making changes to the copy.

The optimisation and minimisation of state information was done effectively to allow for the algorithm to run efficiently on limited computing resources.

## A.2 **Personal reflection an**d lessons learned

A structured approach to this project would have allowed for a better outcome of the final model however due to my inexperience with machine learning and robotics a lot of time was spent learning the processes on how to get different components to interact with one another such as Gazebo, ROS and python. This allowed me to better understand the various mechanisms that go into producing and controlling a robot.

In better understanding reinforcement learning and how to optimise the learning of an agent through various parameters and how these can relate to my own mindsets I better understood how my own experiences and current environment and choices influence my ability to achieve my own long term goals and how I can optimise my own thinking to better my exploration of life to improve my mindset.

## A.3 Legal, ethical and professional issues

### A.3.1 Legal issues

All the software used was open source. If the system were to be used in a public setting then because of the constant capturing of images we would have to account for data protection laws.

### A.3.2 Ethical issues

Due to the unpredictable nature of the models decision making we cannot ensure the model will take the ethical choice in unknown scenarios. For example the agent may be in a situation where it will collide with one of 2 obstacles, a wall and a person. The agent should choose to collide with the wall but certifying that the model can distinguish between these obstacles is not possible.

The rise of Artificial Intelligence also leads to an increase in unemployment as more jobs that used to require human interaction can be replaced with more productive and efficient machines.

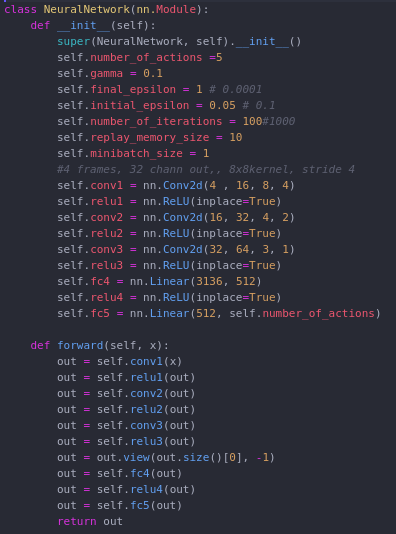
### A.3.3 Professional issues

This project could be used for small scale robots to navigate within controlled environments. This would ensure that if the model was to misbehave the consequences would be manageable. Such an example would be autonomous vacuum cleaners as a collision would not result in catastrophic damage to the robot or the surroundings. Using this approach for a more complex navigational problem such as self driving cars would not be appropriate as the solution does not account for highly dynamic environments. This is not to say that the reward system and network configurations could not be adapted to such a task however the computing resources needed are beyond this scope.

# Appendix B External Materials

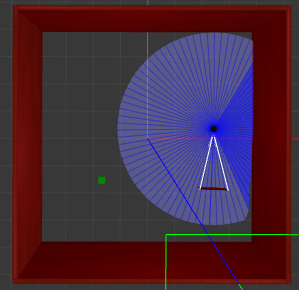
## B.1 DQN

The neural network structure has been made available by external users completing similar projects (Volodymyr Mnih 2013 and Alejandro Escontrela 2018).

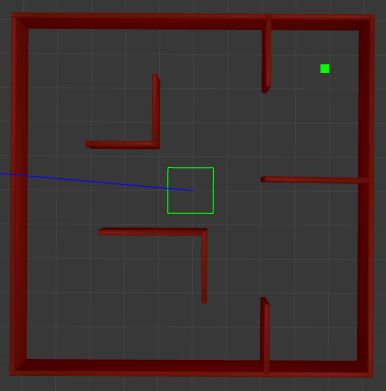


## B.2 Environments

### B.2.1 8x8 Room

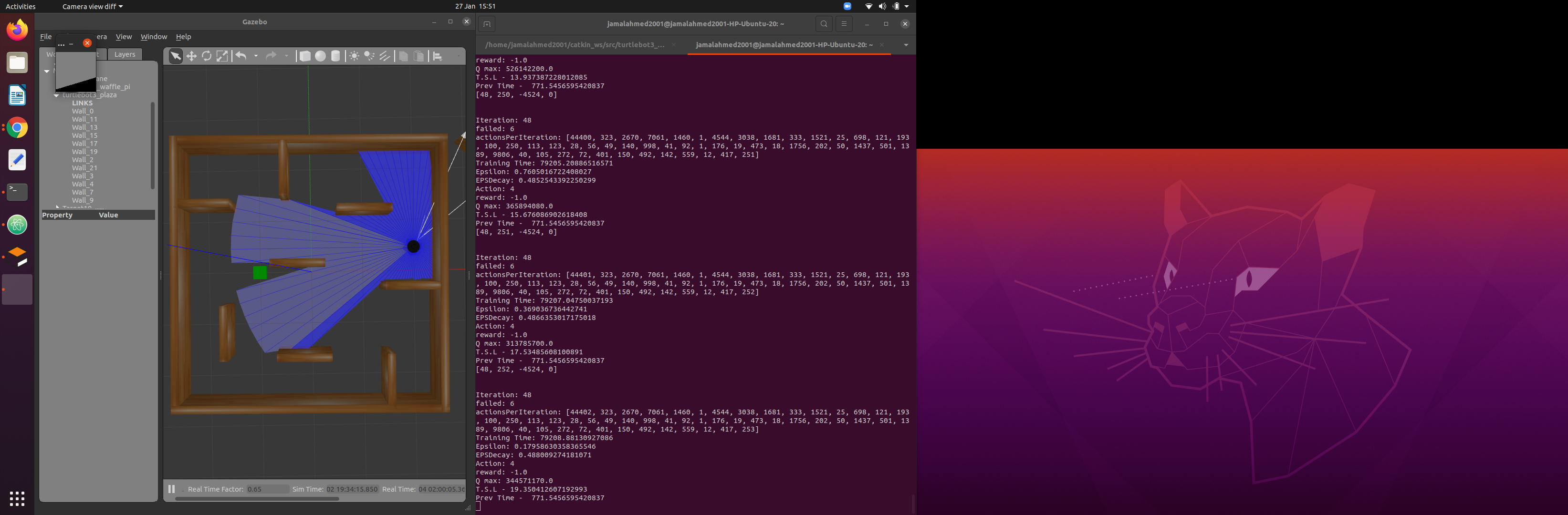


### B.2.2 10x10 Room with Partitions



### B.2.3 Turtlebot3 plaza

Provided by Robotis



### B.2.4 Turtlebot3 house

Provided by Robotis