

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*:*

*<As an example>*

|  |  |  |
| --- | --- | --- |
| **Items** | **Format** | **Recipient(s) and Date** |
| *Final Report* | *PDF file* | *Uploaded to Minerva (DD/MM/YY)* |
| *Link to online code repository* | *URL* | *Sent to supervisor and assessor (DD/MM/YY)* |
| *User manuals* | *PDF* | *Sent to client and supervisor (DD/MM/YY)* |

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)

© 2022 The University of Leeds and <full name of candidate>

# Summary

*<Concise statement of the problem you intended to solve and main achievements (no more than one A4 page)>*

Reinforcement learning has been used

# Acknowledgements

*<This page should contain any acknowledgements to those who have assisted with your work. Where you have worked as part of a team, you should, where appropriate, reference to any contribution made by others to the project.>*

*Note that it is not acceptable to solicit assistance on ‘proof reading’ which is defined as “the systematic checking and identification of errors in spelling, punctuation, grammar and sentence construction, formatting and layout in the text”; see*

https:://www.leeds.ac.uk/secretariat/documents/proof\_reading\_policy.pdf

# Table of Contents

[Summary 3](#__RefHeading___Toc544_2999211779)

[Acknowledgements 4](#__RefHeading___Toc546_2999211779)

[Table of Contents 5](#__RefHeading___Toc548_2999211779)

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

[1.1 Introduction 7](#__RefHeading___Toc552_2999211779)

[1.2 Machine Learning 7](#__RefHeading___Toc554_2999211779)

[1.2.1 Neural networks 7](#__RefHeading___Toc594_2999211779)

[1.2.2 Convolutional Neural Networks 7](#__RefHeading___Toc716_598456863)

[1.3 Reinforcement Learning 7](#__RefHeading___Toc596_2999211779)

[1.3.1 SARSA 7](#__RefHeading___Toc728_598456863)

[1.3.2 Q learning 7](#__RefHeading___Toc730_598456863)

[1.3.2 Exploration vs Exploitation 7](#__RefHeading___Toc732_598456863)

[1.4 Deep Q Networks 7](#__RefHeading___Toc749_598456863)

[1.5 Related work 7](#__RefHeading___Toc800_598456863)

[Chapter 2 Methods 8](#__RefHeading___Toc556_2999211779)

[2.1 Environment 8](#__RefHeading___Toc558_2999211779)

[2.1.1 The Robot 8](#__RefHeading___Toc879_598456863)

[2.1.2.1 Action space 8](#__RefHeading___Toc881_598456863)

[2.1.2 State input 8](#__RefHeading___Toc598_2999211779)

[2.1.2.1 Image input 8](#__RefHeading___Toc883_598456863)

[2.1.2.2 Lidar Input 9](#__RefHeading___Toc814_598456863)

[2.1.3 Terminal States 9](#__RefHeading___Toc600_2999211779)

[2.1.4 Reward Systems 9](#__RefHeading___Toc834_598456863)

[2.1.4.1 Simple Reward System 10](#__RefHeading___Toc698_598456863)

[2.1.4.2 Dynamic Reward System 10](#__RefHeading___Toc700_598456863)

[2.1.4.3 In Sight Reward System 10](#__RefHeading___Toc702_598456863)

[2.1.4.4 Maximising Distance Reward System 10](#__RefHeading___Toc718_598456863)

[2.2 DQN 11](#__RefHeading___Toc560_2999211779)

[2.2.1 DQN Design 11](#__RefHeading___Toc885_598456863)

[2.2.1 Episodic decay 11](#__RefHeading___Toc690_598456863)

[2.2.2 Episodic and Action based decay 11](#__RefHeading___Toc692_598456863)

[2.2.3 Reward based epsilon 12](#__RefHeading___Toc694_598456863)

[2.4 Generalisation methods 12](#__RefHeading___Toc781_598456863)

[2.4.1 Randomising target location 12](#__RefHeading___Toc783_598456863)

[2.4.2 Randomising robot location 12](#__RefHeading___Toc785_598456863)

[Chapter 3 Results 13](#__RefHeading___Toc562_2999211779)

[3.1 Training Results 13](#__RefHeading___Toc606_2999211779)

[3.1.1 4x4 Room 13](#__RefHeading___Toc734_598456863)

[3.1.2 10x10 Room with partitions 14](#__RefHeading___Toc736_598456863)

[3.2 Generalisation and Consistency 14](#__RefHeading___Toc738_598456863)

[3.2.1 One successful model 14](#__RefHeading___Toc740_598456863)

[3.2.1 A more successful model 14](#__RefHeading___Toc742_598456863)

[Chapter 4 Discussion 15](#__RefHeading___Toc564_2999211779)

[4.1 Conclusions 15](#__RefHeading___Toc566_2999211779)

[4.2 Ideas for future work 15](#__RefHeading___Toc568_2999211779)

[4.2.1 Input data and computing resources 15](#__RefHeading___Toc612_2999211779)

[4.2.2 Machine learning methods 15](#__RefHeading___Toc614_2999211779)

[List of References 16](#__RefHeading___Toc570_2999211779)

[Appendix A Self-appraisal 17](#__RefHeading___Toc572_2999211779)

[A.1 Critical self-evaluation 17](#__RefHeading___Toc574_2999211779)

[A.2 Personal reﬂection and lessons learned 17](#__RefHeading___Toc576_2999211779)

[A.3 Legal, social, ethical and professional issues 17](#__RefHeading___Toc578_2999211779)

[A.3.1 Legal issues 17](#__RefHeading___Toc580_2999211779)

[A.3.2 Social issues 17](#__RefHeading___Toc582_2999211779)

[<Discussion of social issues> 17](#__RefHeading___Toc584_2999211779)

[A.3.3 Ethical issues 17](#__RefHeading___Toc586_2999211779)

[<Discussion of ethical issues> 17](#__RefHeading___Toc588_2999211779)

[A.3.4 Professional issues 17](#__RefHeading___Toc590_2999211779)

[Appendix B External Materials 18](#__RefHeading___Toc592_2999211779)

# Chapter 1 Introduction and Background Research

## 1.1 Introduction

Reinforcement learning is a machine learning paradigm in which there is an agent interacting within an environment. The agent learns about the environment over time as it continues to take actions within this environment and then assesses the outcome of these actions based on a reward signal sent from the environment.

Simulated homing is where we use visualisation software in order to simulate the navigation of a robot from some arbitrary starting position to a target destination.

This project will explore the implementation and modification of a deep q network to solve the problem of navigating a robot in an environment to a target destination particularly when the home is hidden.

## 1.2 Machine Learning

<This section heading is purely a suggestion -- you should subdivide this chapter in whatever manner you think makes most sense for your project. It may also make sense to spread the `Background Research' over more than one chapter, in which case they should be named sensibly.>

### 1.2.1 Neural networks

### 1.2.2 Convolutional Neural Networks

## 1.3 Reinforcement Learning

### 1.3.1 SARSA

### 1.3.2 Q learning

### 1.3.2 Exploration vs Exploitation

## 1.4 Deep Q Networks

## 1.5 Related work

# 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 model require the agent to be retrained in the environment which is time costly. To maximise the chance of success, reinforcement learning techniques which have been successfully implemented for other applications will be used at first before suggesting further improvements to the model to better fit the intended use case. Models in this section were trained partially to test the feasibility of different training methods

~~However due to the nature of reinforcement learning we know that given any policy and enough iteration we will eventually converge to some solution I think I read this somewhere.~~

## 2.1 **Environment**

The environment was constructed in gazebo using the building editor and the target was made using the model editor provided. Targets are dynamically added and removed using the gazebo messaging service.

### 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 0.3m/s and angular speed of 1.

#### 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 state it was receiving from the environment would be the same state that it would receive from going forwards. The agent also learnt behaviours such as rotating endlessly as this would lead to the least collisions.

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 slower solutions with faster speeds we could reach convergence 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 input

#### 2.1.2.1 Image input

Initially the input used for the DQN was a 640x480 BGR image that came from the camera mounted on the turtlebot3 waffle\_pi. 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 which would reduce computation time. This reduces the amount of information the agent has to learn about which may have improved the model however the reduction in computation time was preferred as the agent could not respond to environment changes effectively The input was further modified to convert the target pixels to 255 changing the colour to white which increases the contrast between other objects and the target.

#### 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. To match the image input size the number of lasers used was 84. The array was then appended to the 84x84 image matrix to form an 85x84 matrix which is converted to a tensor before being given to the DQN.

### 2.1.3 Terminal States

At first only one terminal state was used which was when the agent successfully found the target. However this lead to the agent spending a lot of training time exploring states that were unsatisfactory such as colliding with objects and getting stuck at walls. To combat thiis 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 bad episode.

The training loop was updated to only consider positive terminal states as episodes as we want to train the agent for a certain number of successful episodes not just episodes as a whole as the agent may collide a different amount of times every time it is trained meaning to replication of results would be difficult as total number of episodes would vary.

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.

### 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 next state and if that 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. This reward system works because as the agent explores the environment it updates the models estimated value of that state with the reward returned with that state as well as a certain number of successive states depending on the gamma value set. This means that states that lead up to the target state will be classed as better than states that lead up to an unsuccessful state.

#### 2.1.4.2 Dynamic Reward System

This reward system had been previously used for similar tasks but using only laser sensory data and was shown to be effective. This is where the reward returned to the agent is calculated based on the distance and angle to the target square. This model was able to train effectively in a simple 4x4 room however when moving to more complex environments the model started to act inappropriately especially when the target was out of sight. This makes sense as when the target is out of sight states will have varying rewards based on the location of the target and the robot. The rewards for a given state were therefore not consistent and the agent was unable to learn an appropriate model.

#### 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.

#### 2.1.4.4 Maximising Distance Reward System

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 returned -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.

## 2.2 **DQN**

### 2.2.1 DQN Design

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 the 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.

Talk about what the different kernel and stride combinations do. Something about high level features and low level ones.

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 shown to converge for many different solutions. Reference

### 2.2.1 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 and shown to work…. This makes sense as the more the agent finds the target the more successful experience the agent has learnt about the environment and thus we should be able to rely on the trained model to help the agent explore more successful states.

### 2.2.2 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.

### 2.2.3 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.

Figures can be added using the Illustrations section of the Insert tab.

## 2.4 Generalisation methods

### 2.4.1 Randomising target location

By randomising the target location over successive episodes and keeping the initial position of the robot constant the agent will learn a model that allows the robot to explore the entire environment in an optimal way as each episode will be a different optimal route to the target state. By setting the number of target locations we are also selecting the number of optimal routes the agent will have to learn in that environment. A small number of locations leads to a successful model but with poor generalisation of its environment as well as other environments. Using a large number of target locations leads to a model that has poor generalisation in environments other than that which the agent was trained in.

### 2.4.2 Randomising robot location

To further improve generalisation the robot starting position was also randomised for every episode. This allows for the agent to explore more of its environment in a shorter number of episodes. This is beneficial as then the agent will learn the optimal paths from different states much 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.

# Chapter 3 Results

<Results, evaluation (including user evaluation) *etc*. should be described in one or more chapters. See the `Results and Discussion' criterion in the mark scheme for the sorts of material that may be included here.>

## 3.1 Training Results

### 3.1.1 4x4 Room

The best model achieved for this environment setup was trained using the maximising distance reward system as well as action and episodic based decay. Gamma was set to a value of 0.85 and initial epsilon value was 0.8 and final epsilon was 0.05. The agent was trained for 195 episodes however examining the episode rewards and number of actions showed that the model may have converged to a solution at around 150 episodes. The learning rate was set to 0.000025. To prevent the agent from learning inefficient models such as endlessly rotating a limit of 60 actions per episode was used. This forces the agent to learn more efficient solutions as it has to learn to achieve the best cumulative reward over 60 actions.

This model was tested for 100 episodes and found the target within 60 actions for each episode and never collided with the walls.

This model was then tested on the 10x10 room with partitions and managed to achieve better generalisation than other reward models from this environment. The ratio of successful to unsuccessful episodes when tested was 0.5 whilst other models achieved ratios of around 0.1. This ratio indicates how often the robot collides with objects in comparison to achieving the goal of reaching the target.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Gamma | Initial Epsilon | Training episodes | Training Time | S/U Ratio | Average actions per episode Training /Testing |
| 0.95 | 0.8 | 100 |  | 1 |  |
| 0.75 | 0.8 | 100 |  | 1 |  |
| 0.85 | 0.8 | 100 | 25920 | 1 |  |

**Table 2.1** This is the table description in the ‘table description’ style.

### 3.1.2 10x10 Room with partitions

Using the same parameters that achieved model A in this environment showed that the agent could learn optimal paths however the training episodes were much longer than in the 4x4 room and to find 27 targets took the agent 23200 actions which is 860 actions on average per episode.

Using the model trained in the 4x4 room and retraining with the same parameters but an initial epsilon of 0.5 shortened the duration of training episodes on average by 500 actions. This value was chosen as the ratio between successful and unsuccessful states this model achieved when tested in this environment was 0.5 so the agent should make random actions 50 % of the time. To find 27 targets the agent took 4237 actions which is an average of 156 actions per episode. Including the time taken to train the initial weights the total time was 33000 seconds in comparison to 44100 seconds when trained from some random initialisation. When tested the model achieved 0.4 ratio

## 3.2 Generalisation and Consistency

### 3.2.1 One successful model

### 3.2.1 **A more** successful model

# Chapter 4 Discussion

<Everything that comes under the `Results and Discussion' criterion in the mark scheme that has not been addressed in an earlier chapter should be included in this final chapter. The following section headings are suggestions only.>

## 4.1 Conclusions

<Text in 11-point size and 1.5 line spacing.>

## 4.2 Ideas for future work

<Text in 11-point size and 1.5 line spacing.>

### 4.2.1 Input data and computing resources

### 4.2.2 Machine learning methods

# List of References

*<It is expected that the list would reflect the breadth and depth of scholarly research undertaken by the student during the course of the project.>*

# Appendix A Self-appraisal

<This appendix must contain everything covered under the ’self-appraisal’ criterion in the mark scheme. Although there is no length limit for this section, 2-4 pages will normally be suﬃcient. The format of this section is not prescribed, but you may like to consider the following sections and subsections.>

## A.1 Critical self-evaluation

## A.2 Personal reﬂection and lessons learned

## A.3 Legal, social, ethical and professional issues

<Refer to each of these issues in turn. If one or more is not relevant to your project, you should still explain *why* you think it was not relevant.>

### A.3.1 Legal issues

<Discussion of legal issues>

### A.3.2 Social issues

### <Discussion of social issues>

### A.3.3 Ethical issues

### <Discussion of ethical issues>

### A.3.4 Professional issues

<Discussion of professional Issues>

# Appendix B External Materials

<This appendix should provide a brief record of materials used in the solution that are not the student's own work. Such materials might be pieces of codes made available from a research group/company or from the internet, datasets prepared by external users or any preliminary materials/drafts/notes provided by a supervisor. It should be clear what was used as ready-made components and what was developed as part of the project. This appendix should be included even if no external materials were used, in which case a statement to that effect is all that is required.>