THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

ReLOaD: Reinforcement Learning for Active Object Detection

Alex Nicholson

A thesis submitted for the degree of Bachelor of Engineering (Honours)

at

The University of Queensland in 2023

School of Electrical Engineering and Computer Science

Abstract

Robots are increasingly making their way out of the lab and controlled industrial settings into complex real-world environments where they can perform useful work. To do this they need high accuracy vision systems to make sense of the world around them and identify objects of interest. Existing computer vision algorithms are commonly used in robotic systems to perform common sensing tasks such as object detection or classification. But these systems can fail due to the presence of obstacles, occlusions or distance to the object causing robots to act in undesired ways or render them unable to progress. In this thesis, we propose a novel RL-based path planning algorithm for that provides navigational guidance to a mobile robot as to how it could move to provide better performance in collecting object detection data. The RL-based planner was trained in a simulated environment and was able to successfully navigate the robot and collect object detection data from the environment with a reasonable degree of efficiency, on par with a specifically hand-crafted solution. These findings show that RL-based approaches are capable of solving such path planning problems, however, they would likely be more beneficial in more complex environments where creating a competent hand-crafted solution would be completely infeasible.

All of the code for the training pipeline and simulator as well as the final trained model are available on GitHub at: `https://github.com/Anwealso/ReLOaD`.

Declaration by author

Acknowledgments

I would like to acknowledge the support from the following:

- Dr. Jen Jen Chung
  - for her assistance in the analysis and interpretation of results, feedback in design and testing of the observation space and reward function, and drafting assistance
- Dr. Liang Wang
  - for his original conception of the project, assistance in the analysis and interpretation of results, feedback in design and testing of the observation space and reward function, and drafting assistance
- Dr. Peter Böhm
  - for his sage advice in training debugging and moral encouragement

Financial support

No financial support was provided to fund this research.

Keywords

Reinforcement Learning, Mobile Robotics, Computer Vision, Informative Path Planning

Australian and New Zealand Standard Research Classifications (ANZSRC)

ANZSRC code: 461105, Reinforcement learning, 70%
ANZSRC code: 460304, Computer vision, 20%
ANZSRC code: 400711, Simulation, modelling, and programming of mechatronics systems, 10%

Fields of Research (FoR) Classification

FoR code: 4611, Machine Learning, 70%
FoR code: 4603, Computer Vision, 20%
FoR code: 4007, Control engineering, mechatronics and robotics, 10%

Contents

List of Figures

List of Abbreviations and Symbols

Abbreviations	
MLP	Multi-Layer Perceptron
RL	Reinforcement Learning
OD	Object Detection
IPP	Informative Path Planning
DQN	Deep Q Network
SAC	Soft Actor Critic
PPO	Proximal Policy Optimisation

Symbols	
E	Statistical entropy
IG	Information gain (difference in entropy)

Chapter 1

Introduction

This chapter, covers an introduction to thesis topic and motivation for the research, as well as an overview of the goals and scope covered by the project.

1.1 Reinforcement Learning for Active Object Detection

Vision systems are critical for a robots ability to perform useful work in the complex real-world environments in which we humans operate. Many robotics systems employ existing off-the-shelf computer vision algorithms to perform common sensing tasks such as object detection or classification. But these systems can often have robots halt their progress or proceed in an undesired way if they misidentify objects or cannot identify them with enough confidence to responsibly act, due to the presence of obstacles, occlusions or distance to the object. Additionally, although full-stack Reinforcement Learning (RL) behaviour have shown great success in robotic searching and identification tasks, the large amount of time and resources required to train models such as this makes them prohibitive for smaller labs and companies.

This thesis introduces an RL based navigation system that provides navigational guidance to a robot as to how it could move to provide better performance in collecting object detection data. By doing this, more performance can be squeezed out of off-the-shelf OD algorithms by better taking advantage of the mobility of a mobile robotics platform.

This task is formulated as an Informative Path Planning (IPP) problem, where the mobile robot agent must efficiently classify a set of known-position target objects in a 3D environment with the highest confidence possible using only 2D camera vision and given a limited movement budget. From this, the agent learns generalisable knowledge about how to find the most informative path(s) through the environment. The IPP task involves the composite task of selecting what nodes top go to as well as what order to visit them and is known to be NP Hard [1].

Due to this complexity, previous solutions to problems of this type have generally used heuristic-based traditional search algorithms such as greedy search [2] and evolutionary algorithms [1] [3], resulting in poor performance. These solutions relying on traditional search algorithms must also be recomputed for different budgets (which will always be changing in the real world due to differing time constraints, battery charge, etc.) making them impractical for adapting to changing task requirements. In order to address these issues of, some recent research [1] has successfully applied reinforcement learning to the IPP problem in the to learn a more generalisable policy about the structural composition of good informative paths. However, this work was still deals with quite a low dimensional state and action space and is limited by requiring an initial partially complete map of the informativeness distribution to aid with initial path planning.

Building upon previous approaches applied to similar situations, this proposed architecture uses the an OD model, to detect objects in the environment using a 2D camera. The classification confidences on the objects as well as their positions, the current position and orientation of the robot, the remaining budget are then fed into an RL-based deep neural network model to predict the next movement action. The reward function is given by total information gain over all of the targets in the environment to encourage the robot to detect and accurately classify the target objects with the highest confidence possible in the given time. The RL algorithm then outputs movement commands consisting of a desired twist (linear and rotational velocity) for the robot to carry out.

The RL model was trained within a custom simulator known as SimpleSim, with simulated object detection algorithm output vector and various experiments were conducted to evaluate the model's performance and investigate its function.

It is hoped that this research will show that RL can be an effective approach for improving the OD capabilities of a robot over existing drop-in OD tools, with reduced amount of time and training resources needed than for training end-to-end reinforcement learned behaviours of a robot encompassing raw image and multiple sensor inputs.

1.2 Goals

To fulfill the aims of this thesis, a number of goals were set out to guide the research. These goals are listed below:

- Develop an algorithm that uses RL to control the movements of a mobile robot to efficiently collect object detection data from targets.
- The algorithm must use outputs of object detection algorithm to guide its search
- Must be adaptable to variations in budget and number of targets to survey

The performance of the algorithm then needs to be evaluated and compared to previous non-AI based approaches to solving the problem, as well as suggestions given for possible avenues to address the strengths/weaknesses of the algorithm to improve it in the future.

1.3 Project Scope

In order to guide the research activities, the scope of the problem must be carefully defined. The following is an overview of what topics were considered in-scope for the project and therefore a focus of research and innovation and those that were considered out-of-scope (used off the shelf or not considered):

- In-Scope
  - State space, action space and reward function design, and tuning: The bulk focus of this work is in exploring the design of suitable state and action space as well as reward function for the RL agent to be able to learn the desired behaviours necessary to fulfil the thesis goals.
  - Environment and simulator creation: In order to create to train the agent on the chosen state/action/reward design, across a variety of environment configurations, creation of a customised simulator was a required in-scope element of the project.
  - RL algorithm selection and modification: The selection of a suitable RL learning algorithm as well as potential small selections of the internal functionality such as loss functions is in-scope.
  - Model training and hyperparameter tuning: Developing the training techniques and pipelines required to successfully train an RL model to solve the problem is within the scope of this work.
- Out-of-Scope
  - RL Learning Algorithm Design: This work focuses not on the design of efficient and general RL learning algorithms, but on the application of existing learning algorithms to a solve a new problem.
  - Creation of the object detection algorithm: This work aims to improve the overall effectiveness of using existing object detection algorithm on mobile robotics platforms, by adding an external robotic navigation system, not by to modifying the internal object detection model architecture itself.

Chapter 2

Literature Review

In this chapter, a summation of the existing approaches to solving informative planning problems is provided, as well an assessment of their limitations and areas which are targeted for improvement by this thesis.

2.1 Early Usage of RL in IPP

An early work in the active sensing / IPP space involved RL tackled the problem of scoring a goal in a RoboCup setting using a robot with limited field of view [4]. The robot was unable to observe both the ball and the goal at the same time, forcing it to choose between pointing its camera at the ball to minimize uncertainty in the ball's location, or point the camera in a direction that helps it minimize uncertainty in the relative location of the goal. The task of their active sensing system was to control the neck of the dog to gather the most valuable information for the given task, to update the environment positioning model so that a separate movement controller positioned the robot to kick the ball.

To frame the MDP for effective training, they also gave actions a cost proportional to the angle the neck of the dog moves, which provides a similar constraint to the energy budget restriction given to our robot. As a reward function, they also explicitly included the uncertainties of the estimated positions of the three objects (self, ball, goal) as state variables, much like in our application the inclusion of the object detection confidences of each object.

Their results showed that their legged robot was able to learn a sensing strategy that provided significantly better performance in the real-world goal scoring task than both naïve sensor sweeping strategies and hand-tuned models. However, the complexity of this problem is certainly much less than our IPP problem as it is limited to a small action space discretised into 9 possible sensing actions, a very small state space and short time-horizon of only 9 sequential actions (cannot revisit past locations) across the episode for the overall task (simply get to the ball then execute pre-planned kick), so the ability to extend this RL approach to a much higher dimensional MDP is unclear.

2.2 Recurrent Neural Networks (RNN) RL Approach

Another more recent piece of research addressing the IPP in a manner very similar to our problem (excluding the vision element), focuses on the application of efficiently mapping Wi-Fi signal strength across an area using a mobile robot starting from a partial map of the signal strength of the area [1]. Although this research performs IPP on the significantly simpler information distribution of a 2D scalar data space (although with a longer time-horizon than the previously discussed paper), it provides a much more modern and advanced approach to solving IPP problems with RL in general by using deep neural networks. It improves upon existing traditional heuristics-based search solutions of greedy search [2] and evolutionary algorithms [1] [3], which need to re-compute solutions from scratch each time the budget changes and so cannot adapt their knowledge across different budgets.

They first discretise their continuous 2D state space and then use the initial pilot data to fit a Gaussian process regression model to the data to estimate the informativeness distribution. This initial bootstrapping approach to the solution is not possible with our problem as although the positions of our objects to identify are known, the orientation and surrounding occlusions are not known so it is not obvious to infer the informativeness distribution from them to use as pilot data.

They then apply an RL trained RNN network to the state to predict the best action to take next, which takes a record of the path traversed so far (partial path) as an input to help account for the fact that revisiting an area will provide reduced information gain compared to the first observation. This revisiting problem is central to our identification task as it is undesirable for the robot to do overlapping return trips to the same object, so this RNN architecture for the Q-function serves as a good potential approach to our problem.

2.3 RL-Search Hybrid Approach

In another recent research study applying RL to an IPP problem tackled the problem of ”UAV-based Active Sensing” [5]. The authors proposed a novel method for optimising the mapping of crop field temperature/humidity data using an MCTS (Monte Carlo Tree Search) with a CNN to predict the true temperature distribution of a 2D area [5]. Similar to our application, the agent starts with no knowledge about the information distribution over the area and only forms predictions about the informativeness of paths through sampling the environment. They utilise this offline-learned hybrid MCTS/CNN approach harnessing traditional search and RL system to search the current and past tree of observations to predict informative sensing actions [5] and thereby plan an informative path. This is a very interesting approach, and they claim that the method reduced runtime by “8-10 times” [5] compared to an evolutionary algorithm implementation and with similar accuracy. The method has some limitations. Firstly, it is unclear what value the past predictions have in this approach. Additionally, while the algorithm runs fast, it would require costly tree search re-computations if the exploration budget changes. Additionally, although his tackles a higher dimensional state and action space than the RoboCup research, it still uses a lower MDP dimensionality than that for our challenge using 2D camera vision, so the transferability of this approach may not be feasible with an even larger search tree due to our larger MDP dimensionality.

2.4 RL Usage in other Path Planning Applications

Although IPP is a somewhat smaller sub-field of path planning, across path planning more generally, RL has been used very widely to solve a large range problems with strong results.

One approach to a similar planar navigation problem sought to train a model to navigate to a target point in a 2D environment with obstacles, given a state space consisting of the relative angle and distance to the target, as well as a 2d lidar scan distances around the robot [6]. They also used an approach of starting the agent training in a 2D environment to reduce computational expense and then transferring to a 3D environment. Their results compared favourably, being on par with existing hand-crafted planning approaches for this type of problem such as RRT (rapidly-exploring random trees) [7] and PRM (probabilistic roadmap) [8]. Overall, although the point-to-point navigation problem they studied is somewhat simpler than the information based navigation problem attempted in this work, it demonstrates the success of RL applied to adjacent problems.

Another study [9] trained a model to do drive a robot in a planar 3D environment to reach a given goal position, using DQN. The agent was trained on a state space of raw RGB pixel input from the robots point of view, and with a discrete action state space (turn left, right, move forwards, back). They achieved strong navigation results in the 3D simulator, successfully being able to efficiently navigate to the target and avoid targets, and expressed optimism in being able to extend this approach to control of a real world robot.

2.5 RL Inside Object Detection Algorithms

Although our training problem can be framed as an IPP problem, the core application of the solution in this thesis is to create an integrated object detection vision system that exploits the ability of a mobile robotics platform to move around the environment to improve detection accuracy, thereby squeezing more performance out of the object detection network than the original off-the-shelf algorithm. This essentially means we are trying to improve the accuracy of an object detection system with an additional RL process. A wide range of research has been conducted aimed at similar goals in the computer vision space such as reducing compute usage and inference time or increasing accuracy on detection tasks – usually taking the form of region proposal [10] [11] or selection of greater resolution in areas [12].

All these approaches simply apply RL to within the image domain however, whereas we aim to exploit the meta-capabilities of a robot to modify the OD process at a higher logical level – in the actual 3D space sampling domain to increase the accuracy of the resultant samples by the object detection network down the line. None of these methods apply RL to this physical process of exploring the real-life 3D environment as a whole and temporally (since we need to explore both in space and time) - to find regions of interest for greater refinement. This research does, however, show some promise for the ability of our system to provide improvements to off-the-shelf OD models by performing a kind of region proposal in real-world 3D space - i.e. where should we move out robot body and head to get the best perception accuracies given a limited movement budget.

Chapter 3

Theory

This chapter covers the background theory content that is needed to understand the project, and which is used in the design of the solution.

3.1 Reinforcement learning

Reinforcement learning is a type of unsupervised machine learning, where an agent learns a desired behaviour, purely by interacting with its environment and receiving reward feedback for its actions. ’Good’ actions receive a positive reward and ’bad’ actions receive lower of negative reward. Over the course of training, the agent seeks to learn the answer to the question: ”For each state / point-in-time, what is the best action I can take?” (that not only maximises the reward received now but also leads to the most reward in the future). These learned mappings of states to their optimal action together form the overall policy of the agent.

3.1.1 Markov Decision Processes (MDPs)

Reinforcement learning works in a fundamental ‘see’, ‘think’, ‘act’ cycle. At each step, the agent observes the current state of the environment, and then must make its decision about which action to select based off of this information alone. This chosen action is then executed in the environment, and the environment reacts, moving us to our next state. Based upon what happened in the environment, i.e. the value of the new state we transitioned to, the reward is then calculated and sent back to the agent. This process can be formally represented as a Markov Decision Process (MDP) [13].

An MDP consists of the following parts:

- State space, S: The set of all possible states, s, that the environment can have.
- Action space, A: The set of all possible actions, a, the agent can take at any timestep.

- Transition function, $T(s,a)$: Defines what next state, s', the agent will end up in if it takes a certain action from a current state (this is often a stochastic function in that there could be multiple possible states the agent could end up in after performing its action).
- Reward function, $R(s,s')$: Defines what reward will be received for transitioning from one specific state to another.

Importantly, MDP problem definitions satisfy the Markovian property - that the future state of the environment can be fully determined from its current state and the action taken to progress it to the next state (i.e. environment has no hidden state or memory). This is critical for RL to be able to be applied to the problem, since at each step the agent has to make its action decision solely off the given current state vector fed to it, so it must assume that this is information sufficiently and full describes the environment for it to be able to model what the downstream consequences of its action might be in the future.

3.1.2 Reinforcement Learning

One of the earliest approaches to solving these MDP problems using reinforcement learning is Q-Learning, which attempts to estimate the value of every possible action at every possible state, and then selects the sequence of actions that have the highest possible value to form the final policy. It stores all of these values in a Q-Table, which will have dimensions of $|S|$ by $|A|$ (since it stores the value for every possible action at every state) [13]. This table can become very large for problems with large state spaces (large number of variables or possible range of values in state vector) or large action spaces (large number of possible actions / control inputs to environment) and obviously doesn't support continuous state or action spaces (as there would be infinite number of values to store).

To begin, the Q-Table values are all initialised to arbitrary fixed values. Then to train the model, at each timestep the agent selects an action, moving the environment to a new state and receives a reward value. The Q-Values for this state-action pair are then updated based on this new experience, using the Bellman Equation [13]:

$$Q^{new}(s_t,a_t) \leftarrow (1-\alpha)\cdot Q(s_t,a_t) + \alpha \cdot [r_t + \gamma \cdot \max_a Q(s_{t+1},a)] \tag{3.1}$$

This is essentially setting the new Q value to a weighted average of the old Q value and new target Q value, where:

- α is the learning rate (amount to move q value towards target value)
- r_t is the reward received after the agent's action
- $\max_a Q(s_{t+1},a)$ is the estimate of the optimal future value achievable from the next state the agent has ended up in
- γ is the a discount factor applied to the estimated downstream future value of the next state

By iterating this process many times, the Q-Values converge towards the true values of each state-action pair, and thereby the optimal policy for the environment is computed.

3.1.3 Deep Reinforcement Learning

One major drawback of Q-Learning, however, is that the Q-Table must hold a Q-Value for every state-action pair, which, for very large or high-dimensional discrete state and action spaces, or especially continuous state and action spaces, means that the size of the Q-Table can explode. This makes it both incredibly difficult to collect Q-Value experience for every single state as it may be only very infrequently visited, and the size of the Q-Table makes it very challenging to store. To address these limitations, Deep Q Networks (DQN) were introduced, which replaces the table of Q-Values with a neural network, which predicts the value for a given state action pair vector input. This allows Q-Learning to extend to higher dimensional state spaces.

3.1.4 Actor Critic Approaches and SAC

Actor-Critic Approaches represent another significant advancement in reinforcement learning, addressing several critical limitations inherent in DQN and other early methodologies. Prior approaches often struggled with problems such as sample efficiency, stability, and exploration. In essence, Actor-Critic approaches combine two essential components: the actor network, responsible for selecting actions, and the critic network, tasked with evaluating the chosen actions by estimating the expected cumulative rewards associated with them [13]. During training the actor strives to learn to generate the best actions, while the critic strives to learn the value of actions, evaluates these actions and guides the actor towards those that result in higher expected rewards. Essentially the actor function learns the optimal policy, while the critic network learns the value function or the value of state-action pairs. By working in tandem, these components improve learning efficiency and stability as the model moves towards convergence. Additionally, the actor network enables the learning of stochastic policies (unlike DQN, which simply outputs the one most-desired action), which is particularly relevant in scenarios where sensor noise (such as sensor noise form the object detection algorithm) introduces uncertainty in the environment's next state, allowing the agent to adapt to a range of possible outcomes.

Soft Actor-Critic (SAC), a prominent member of the Actor-Critic family, improves upon the concept by leveraging the concept of soft value functions / q-functions, employing a deep reinforcement learning framework to optimize the policy (actor) and value functions (critic) simultaneously. The use of soft value functions enhances the overall stability of the learning process and provides a graceful trade-off between exploration and exploitation. Additionally, SAC introduces an entropy regularization term, a penalty term added to the objective function that SAC seeks to maximize during training based on the entropy of the policy. This promotes better exploration, resistance to sensor noise and environment uncertainty, and mitigates the common issue of prematurely converging to sub-optimal solutions. During training, the policy gradient method is employed to update the actor, while the critic network is updated using value learning techniques [13]. This policy gradient method allows the actor to directly optimize continuous policies, providing smooth updates to the policy, improving its training on continuous action spaces. The combination of these elements, makes SAC much more powerful for training agents to perform more complex behaviours and in more complex, dynamic and noisy environments, than simpler approaches like DQN, enhancing their decision-making capabilities and overall performance.

3.2 Object Detection Algorithms

The task of object detection, a prominent subset of machine learning, entails the simultaneous localization and classification of objects within images or video frames. Object detection plays a crucial role in the field of mobile robotics, enabling robots to perceive and interact with their environments effectively. In the case of this project, the desire is to optimise the path planning of a mobile robot to efficiently survey and identify target objects using a camera equipped with an object detection algorithm.

One popular algorithm is YOLO, or 'You Only Look Once'. The YOLO algorithm operates by dividing an image into a grid of patches and employs a deep convolutional neural network (CNN) to perform classification and box regression on each patch. The model produces an output vector containing predictions of the likelihood of each patch containing an object, the class of the prospective object, and the precise specifications of its bounding box, including center coordinates, width, and height [14]. This output vector provides some of the primary information that is incorporated into the the state vector of the environment and used by the RL agent to make its navigation decisions. YOLO's notable advantages include its speed and lightweight compute requirements, enabling it to handle real-time object detection on commodity hardware. Its main downside, however is that its classification results are commonly inaccurate and un-confident (compared to other larger and more complex models), especially when used on complex scenes and backgrounds. By taking advantage of multiple observation and ability to move around for better views, the ReLOaD RL model aims to attenuate the downsides of algorithms such as these to achieve both real-time detection and high accuracy when integrating information from multiple observations.

3.3 Statistical Entropy and Information Gain

Entropy or 'statistical entropy' provides a metric of how little information is held within a distribution about / the amount of uncertainty of the outcome of sampling a random variable. If a variable has a uniformly distributed probability density function (PDF), then it has high entropy, as the distribution provides relatively little information about what the result is likely to be. Conversely, a distribution with a single strong peak in probability, with all other values much lower, has a very low entropy as it provides a lot of information about what the likely result will be. The entropy (designated $H(X)$) of the distribution of a discrete random variable, X, whose possible values are in the set χ , can be calculated using Shannon's Formula [15]:

$$H(X) = -\sum_{x \in \chi} p(x) \log p(x) \tag{3.2}$$

This is equivalent to the expected value of surprise for the distribution:

$$H(X) = \mathbb{E}[-\log p(x)] \tag{3.3}$$

Thus, the information gain provided by an observation can be computed as the difference in entropy between the prior and posterior distributions (X and X' respectively) across the same outcome set.

$$IG = H(X) - H(X') \tag{3.4}$$

This concept of information gain is the central component of the reward function used to train the ReLOaD RL agent.

Chapter 4

Methodology

This chapter details the methodologies used in the design and performance evaluation of the algorithm. This includes simulation of the object detection environment, learning algorithm selection, configuration of state and action spaces, and definition of the reward function.

4.1 Problem Definition

In order to achieve the aim of this thesis - creating an RL model that provides navigational guidance to a robot as to how it could move to provide better OD performance - we want the agent to learn to produce the most confident and correct object classifications. In order to solve this problem with RL, this desired behaviour first needed to be formulated into a concrete task definition that the agent can trained on. To achieve these goal behaviours, we can define task as an informative path planning (IPP) problem, where the agent must maximise the entropy reduction (or information gain (IG)) achieved over the target object classification distributions. The agent is given a limited time-budget in which to complete this task (limited episode length). By formulating the problem as such, the agent should learn a generalisable navigation policy that moves to collect OD information and how trades-off how long it should view each target given the time available. Formally this can be expressed as:

$$\begin{aligned}
&\text{The goal of the agent is to find an optimal policy, } a^* = argmax(IG_a(p,b)),\\
&\quad\text{where } p \text{ is the positions of the objects,}\\
&\quad b \text{ is the provided budget, and}\\
&\quad IG_a(p,b) \text{ is total information gain achieved by the policy over an episode,}\\
&\quad\text{given the target positions and the provided budget}
\end{aligned} \tag{4.1}$$

4.2 Simulator

In order to provide an environment for the agent to solve this problem, a custom object-detection simulator was required. The simulator must provide the critical role of simulating the object detection

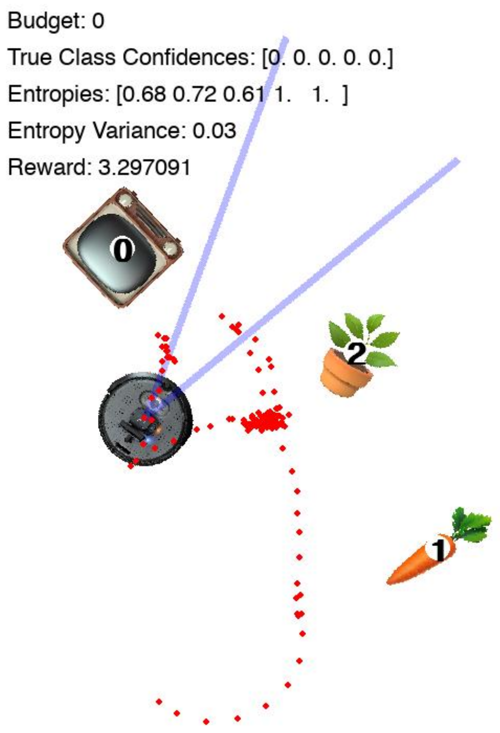
Budget: 0
True Class Confidences: [0. 0. 0. 0. 0.]
Entropies: [0.68 0.72 0.61 1. 1.]
Entropy Variance: 0.03
Reward: 3.297091
0
2
1

Figure 4.1: SimpleSim Simulator

input that the robot would receive from a camera viewing the real world. This could be achieved by simulating a robot in a 3D photo-realistic environment and taking a camera image from the point of view of the robot, then passing this image to an object detection algorithm like YOLO, and collecting the output vector. This approach was considered early in the project but proved to be very slow to run and sometimes unstable and incompatible across a variety of platforms.

Instead, seeing as the agent only sees the output detection vector from the object detection algorithm (confidences and bounding boxes), it was proposed that it would be sufficient to, instead, simulate just this output vector. To do this, a simplified 2D simulator was developed, called SimpleSim (see figure 4.1). Since the simulator only has to simulate the confidences and positional information, it allows it to run much faster, not needing to render a 3D scene and viewpoint images. When a target comes within the agent's restricted 30 deg field of view (FOV) cone object detection confidence on the true class is simulated based on a weighted average (4:1) of how far the object is away from the robot (closer = more confident), and how off-angle the target is being viewed from (viewing from the front = more confident, viewing from the side or back = less confident). Gaussian noise with a mean of zero and a standard deviation of 0.1 is then added to the confidence on the true class to simulate sensor noise. The the remaining probability (1 - the probability of the true class is randomly split between all of the remaining false classes to form the final confidence vector. Due to this noise, the highest achievable confidence (averaged over a large number of samples) is 93.25%, making the lowest achievable entropy on a target 0.1717.

For the true class of the target (class $n_{\text{true\_class}}$), the confidence is given by:

$$
\begin{aligned}
\text{confidence}_n =&(\text{distance\_factor} \times \text{distance\_weighting}) \\
&+ (\text{orientation\_factor} \times \text{orientation\_weighting}),\ \text{for } n = n_{\text{true\_class}} \\
\text{where distance\_factor} =&\sqrt{(x_{target} - x_{robot})^2 + (y_{target} - y_{robot})^2}\ , \\
\text{orientation\_factor} =&|\text{heading angle of the target} - \text{heading angle from target to robot}|, \\
\text{distance\_weighting} =&4, \\
\text{orientation\_weighting} =&1
\end{aligned}
\tag{4.2}
$$

And for all the other false classes of the target, the confidence is given by:

$$
\begin{aligned}
\text{confidence}_n =&\frac{s_n}{\sum_{n=1}^{n=n_{\text{true\_class}}-1} s_n + \sum_{n=n+\text{true\_class}+1}^{n=N-1} s_n} \cdot (1 - \text{true\_class\_confidence}), \\
&\forall n \in \{N \setminus n_{\text{true\_class}}\} \\
&\text{where } n \in \mathbb{N}^N \text{are the possible detection classes (1, 2, 3, ...)}, \\
&\text{where } N = \text{number of object detection classes}, \\
&s_n = \text{a sample drawn from the uniform distribution } S \sim \mathcal{U}[0,1]
\end{aligned}
\tag{4.3}
$$

The code implementation of this confidence calculation can be found in supporting materials. This method introduces sensor noise to the confidence of the true class, as well as adding simulated confusion between the true class and other classes, similar to the confusion that can occur between classes in real object detection algorithms. In this way is it possible for the agent to initially observe a top 1 class classification that is not of the true class, however it is highly likely that by moving closer and/or taking multiple successive observations the integrated observations will converge to having the true class as the highest confidence class.

4.3 ReLOaD RL Agent

4.4 State Space Design

In the state space observed by the agent, in addition to the object positions and current timestep confidences vector, the agent also receives some additional properties that describe the past history of observations in the environment and the amount of work that the agent has left to go to guide its decision making and satisfy the Markovian property. The full state vector received by the agent is as follows:

- Number of targets [integer]
  - This allows the agent to know how many values of the target positions vector are actually in use this episode
- Remaining budget [integer]

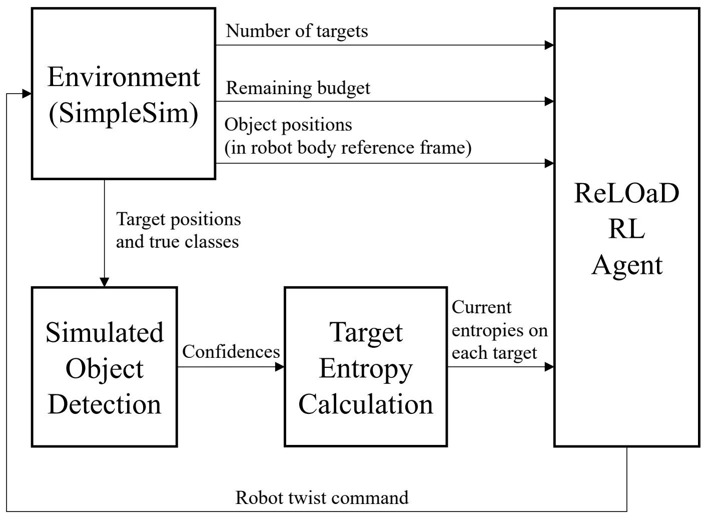
Environment (SimpleSim)
Number of targets
Remaining budget
Object positions (in robot body reference frame)
ReLOaD RL Agent
Target positions and true classes
Simulated Object Detection
Confidences
Target Entropy Calculation
Current entropies on each target
Robot twist command

Figure 4.2: ReLOaD System Architecture

- Object positions (in robot body reference frame) [float]
  - This vector has a fixed length, but each episode a variable number of targets can be given to the model (from one to the max value), so vector is simply filled with the number of targets provided and the rest of the positions in this input vector are simply padded out with zeros
- Current entropy on each target [float]
  - Again, this vector has a fixed length, but is simply padded with zeros for unfilled values to account for the variable number of targets.

4.4.1 Action Space Design

In terms of the action space design, a continuous action space was chosen, where the action is a robot twist movement command, $[v, \omega]$ (desired linear and angular velocity). Utilizing a continuous state space offers distinct advantages for the fundamentally continuous control problem of controlling the robot's movement. Compared to using a discrete move forward/back or turn left/right one unit movement command or a discretised twist, representing the movement actions as a continuous twist, provides finer control granularity, allowing agents to adapt precisely to the layout of the environment. Additionally, when pairing this with SAC, the continuous state spaces enable smoother policy updates, reducing abrupt changes in the policy / action selection. This smooth transition not only enhances the overall learning process but also mitigates the undesirable consequences of sudden policy shifts in specific states, resulting in more efficient control outputs from the agent.

4.4.2 Learning Algorithm and Training

For the final ReLOaD model, SAC was chosen as the learning algorithm (out of a number considered, including DQN and PPO) as it provided the best training performance on the final state and action space definitions. Testing was conducted on custom built models using TensorFlow TF-Agents [16] and base PyTorch [17], but for the final SAC, DQN, and PPO models used in performance comparison the standard implementations provided by Stable Baselines 3 [18] were used due to their ease of usability. The model was trained without the use of any curriculum learning, as it was not found to be required with the design of the reward as detailed in the next section.

4.4.3 Reward Function

The reward received by the agent, R_s is a scaled version of the the total information gain IG made over the episode so far. Specifically, the reward function is defined as so:

$$\begin{aligned} R_s &= 10 \cdot \frac{1}{M} \sum_{m=0}^{M-1} IG_m \\ IG_m &= \frac{1}{T_m} \sum_{t=0}^{T_m-1} (1 - \sum_{n=0}^{N-1} -p_{mn} \log_N p_{mn}) \\ \text{where } M &= \text{number of targets,} \\ T_m &= \text{number of observations on target } m \text{ so far,} \\ N &= \text{number of object detection classes} \\ p_{mn} &= \text{the predicted probability of target number } m \text{ being of class ID } n \end{aligned} \tag{4.4}$$

Since the object detection output vector for each target provides a distribution of the confidences across every class (confidence that the object is of that class, as in figure 4.3), the information gain in this distribution represents the amount of information collected about class of that object. As the agent observes the target, over multiple observations, a peak should form in the confidence distribution for the true class value of the target. This results in a reduction in the entropy and information gain on the target from the original state of no-knowledge of the targets class. Importantly, the information gain does not distinguish between the distribution peaking about the the true class or some other false class, it simply measures that the distribution is peaked (it is assumed that the classification result will converge to the true class when integrating the object detection classification output over multiple observations). Since it does not require knowledge of the true class of the, this information can provided to the agent both during training and execution time.

The first couple of observations result in a relatively large drop in entropy, as the distribution quickly develops a peak from a uniform state, however, once a significant peak has been developed, subsequent observations create a relatively small reduction in entropy. As a result, this provides the agent with diminishing returns for many consecutive observations of the same target as the capacity to increase the reward additionally decreases. This, in addition to averaging the information gain across the targets encourages the agent to view each target only as much as is necessary to get a good

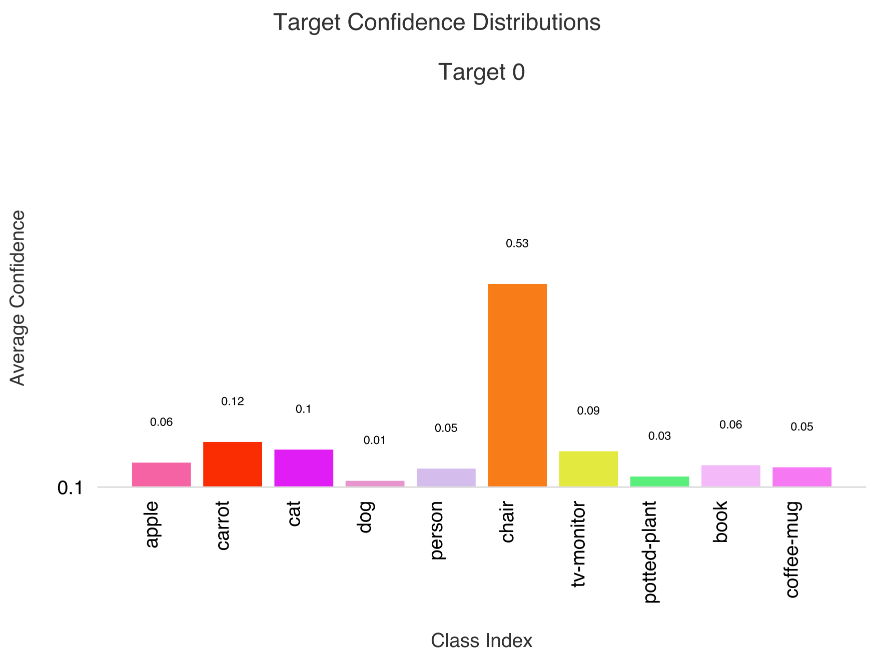

Figure 4.3: Example OD Confidence Histogram

classification result, and then move on to other targets to balance the information collected on each target.

This reward function also provides a smooth and consistent positive reward gradient moving in towards the target (as closer observations will reduce the entropy). This incentivises the agent to come closer and collect high-quality observations instead of collecting inaccurate classification readings from a poor position. The smoothness of the reward gradient along with reward being distributed widely across the environment (any state that can observe at least one target) also means that from the beginning of training, even with a random policy, the agent is likely to quickly acquire reward feedback, helping assist in early policy development.

Chapter 5

Results and Discussion

In this chapter, the final model performance results are presented, as well as a comparison across a number of popular learning algorithms training on the final MDP configuration. Following this, an ablation study is performed on the final model and environment architecture in order to better demonstrate the major elements of the system and their effects on overall performance.

5.1 Final Policy Performance

5.1.1 Performance Benchmark against Baselines

The performance results of the final model are provided in figure 5.2 compared to the average maximum available reward, along with a comparison to a random policy and hand-crafted search strategy. This hand-crafted search strategy consists of recursively navigating the robot to the closest target, viewing it for an even percentage of the available budget, and then moving on to the next target until all have been visited and the episode is over (full search logic can be found in codebase provided as supporting material).

Note that due to the Gaussian noise introduced into the confidence vector, the expected lowest possible entropy achievable on a target on a consistent basis is 0.1717. As such the maximum reward available in the environment is 1656.6. This maximum available reward in the environment is not actually achievable by the agent as it assumes that all available information is gained instantaneously, which is not feasible since it takes time for the agent to travel to the targets and multiple observations to bring down the entropy.

The policy was trained for four million iterations on the SimpleSim environment (approximately eight hours on an NVIDIA GeForce RTX 3080), and then tested in an evaluation environment for 1000 episodes each. The results show our solution reached a reward performance of 849.21 with a standard deviation of 643.68 for the trained final design ReLOaD model (highest-reward model of the training run), collecting 51.26% of the average available reward in the environment, versus an average reward of 906.98 with a standard deviation of 637.06 (54.75% of available) for the hand-crafted search

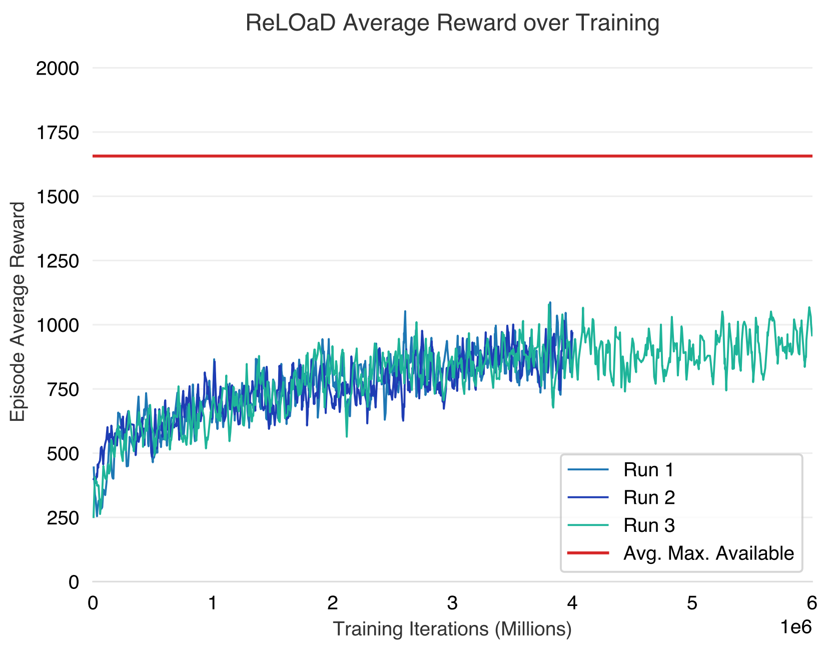

Figure 5.1: ReLOaD Average Reward over Training (3 trials)

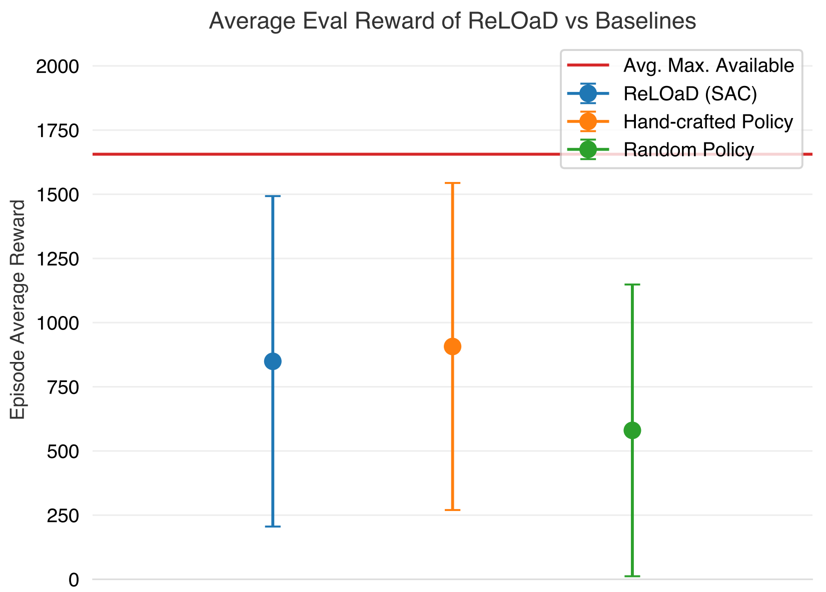

Figure 5.2: Comparison of Final Design Against Handcrafted Policy and Random Policy (error bars represent standard deviation)

implementation, and 580.21 with a standard deviation of 568.24 (35.02% of available) for the random policy.

5.1.2 Policy Behaviour Characteristics

This final policy displays reasonable level of proficiency in solving the IPP problem, successfully executing the three main sub-task behaviours needed (see examples of completed episodes in figures 5.3 and 5.4) (videos of the final policy available via supporting material). As previously stated, these

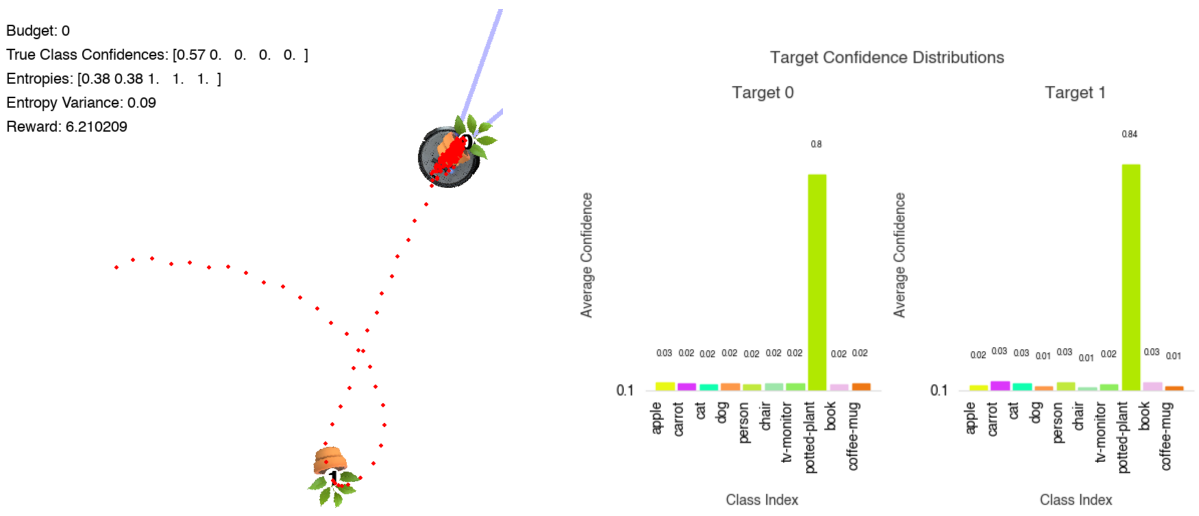
Budget: 0
True Class Confidences: [0.57 0.   0.   0.   0.]
Entropies: [0.38 0.38 1.   1.   1.]
Entropy Variance: 0.09
Reward: 6.210209

Figure 5.3: Example of ReLOaD Finished Episode Path with 3 Targets

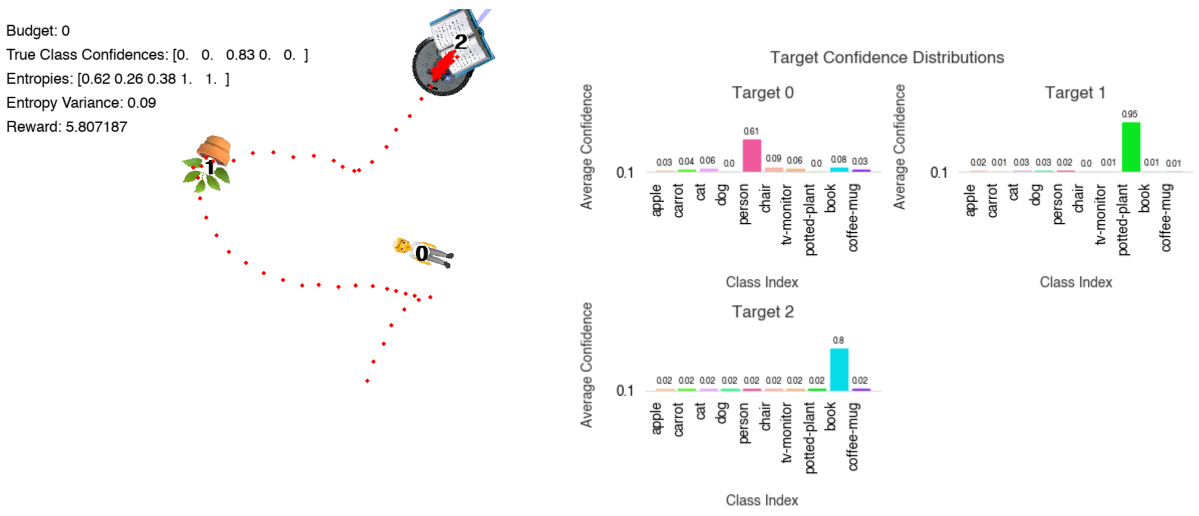
Budget: 0
True Class Confidences: [0.   0.   0.83 0.   0.]
Entropies: [0.62 0.26 0.38 1.   1.]
Entropy Variance: 0.09
Reward: 5.807187

Figure 5.4: Example of ReLOaD Finished Episode Path with 2 Targets

are: effectively manoeuvring to inspect targets (manoeuvring to information-dense areas of interest), trading-off how long to spend at a target and if/when it should move on to another, and efficient path planning / routing between those areas of interest.

Generally the ReLOaD policy performs better in episodes where there are a fewer number of targets (as these are simpler cases), with its average reward generally going down as the number of targets increases as can be seen in figure 5.5. This same trend is also present in the handcrafted policy, as fundamentally, the information and therefore reward density of the environment is lesser when the reward is divided up amongst a number of targets .

Sub-Task 1: Manoeuvring to Inspect Targets

In terms of maneuvering to inspect targets (sub-task 1), the evaluation yielded several noteworthy behavioural observations. First and most basically, our reinforcement learning algorithm effectively controlled the vehicle's movement, allowing it to drive to and between the targets, and exhibited proficiency in both forward and backward movement using the holonomic drive. Although the

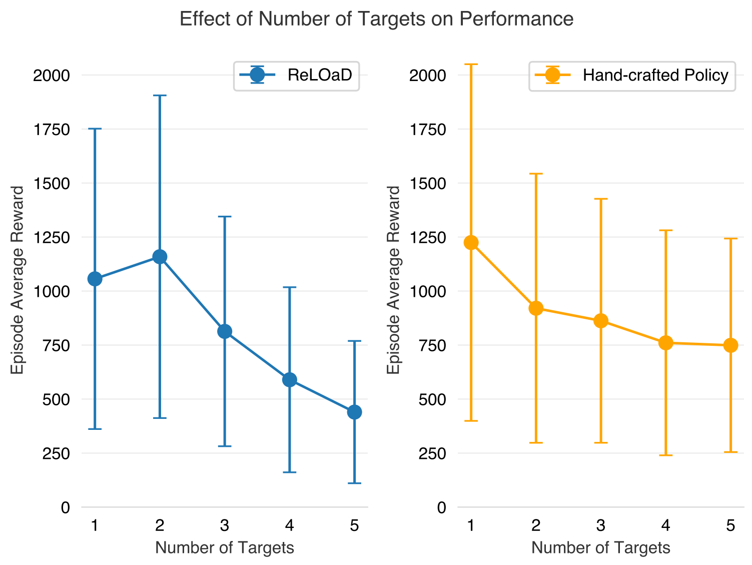

Figure 5.5: Comparison of Performance of ReLOaD vs Handcrafted Policy Across Different Numbers of Targets (error bars represent standard deviation)

vehicle displayed a peculiar oscillatory behavior and jerky movement back and forth occasionally. These idiosyncrasies did not substantially impact the information gathering performance (and are not surprising, seeing as the agent was never given any explicit penalties based on the characteristic its motion itself), though they could create potential issues on a real-world robot implementation. The algorithm also showcased the capability to approach targets by driving towards them to obtain a better view and displayed an inclination to maintain targets in its field of view, even occasionally driving backward to achieve this objective. Moreover, the algorithm demonstrated a strategic inclination to keep multiple targets within its visual range, a valuable strategy that provides increased information gain. Surprisingly, it displayed relatively little tendency to spend time precisely adjusting its viewing angle on the targets, heavily prioritising distance over angle in its decision-making process. This was expected based on our experimental design, however, as the OD confidence is calculated based on a 4:1 weighting of distance to angle offset, and the model lacked explicit information about the targets' orientation angles in its state vector.

Sub-Task 2: Time-Management Between Targets

The RL-based ReLOaD policy tends to do much fewer observations of the objects than the hand-crafted on average to achieve the same average reward 5.6 - much less of the episode is spent on average looking at any individual target. Comparatively, the hand-crafted policy obviously splits its time relatively evenly amongst the number of targets as designed, with means fractions of the episode spent inspecting each target very close to an even distribution (1, 0.5, 0.33, 0.25, 0.2 for number of targets from 1 to 5 respectively). On average it spends the same amount of time looking on each target, irrespective of the number of targets, mostly because it views them all relatively few times anyways. From the third and fourth quartile bars, however, we can see that the amount of time spent on each target does trend downwards (the distribution skews noticeably lower) the more targets that are added,

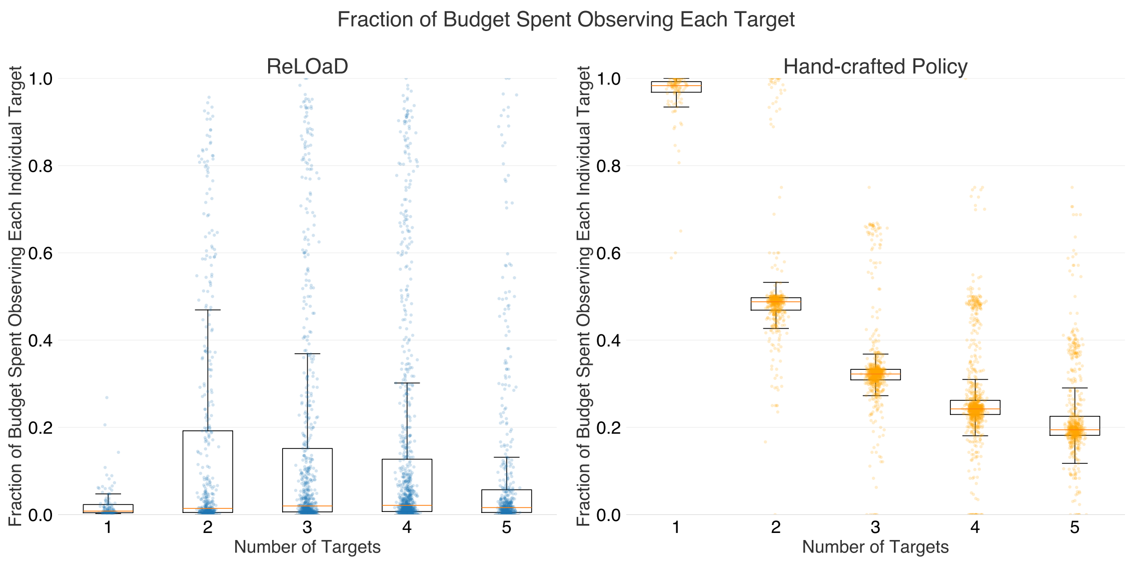

Figure 5.6: Fraction of Episode Spent Looking at any Individual Target (quartiles shown)

indicating that there is still trading-off occurring, due to the greater number of targets to visit with the same average budget. This can bee seen in the policy with the agent sometimes driving towards a target backwards, an undesired behavioural side-affect of using an entropy based reward. This strategy allows the agent to not 'pollute' its observations by with a low-confidence observation from far away (as this will pull down the average entropy, taking a large number of good observations to reverse the effect) but instead get close before observing the target resulting in much more favourable average entropy in a shorter time-period.

Interestingly, when we look at the distribution of the observations that were taken amongst the targets (i.e., how many observations were taken on each individual target as a fraction of the total number of observations across all targets in an episode - see figure 5.7), it is clear that of the observations the agent is making, they are actually relatively evenly distributed across each of the targets (similarly to the hand-crafted policy), with the mean fractions of the episode spent on each target being approximately 1, 0.5, 0.16, 0.12, and 0.08 for numbers of targets from 1 to 5 respectively. This just speaks to reaffirm the fact that the approach of the agent is still to inspect each of the targets, just with a preference towards a lesser number of high-quality, high-confidence observations over a greater number of low- or medium-confidence observations. The mean fractions are still lower at all numbers of targets than an even split, with clear clusters around zero, however, due to the agent sometimes choosing not to observe some targets, or to far prioritise some targets over others (resulting in clusters also around 1). It is difficult to say for sure if this is an undesirable behaviour, however, as this effect could be the result of negligence of certain targets, or simply attention being given more to the targets that need it more (could produce the most reward through greater observation), or simply using up unneeded time by observing the last target after successfully identifying all targets. But this uneven prioritisation behaviour does certainly result in more inconsistency in its inspection, as is reflected in the much higher variances for ReLOaD in figure 5.7.

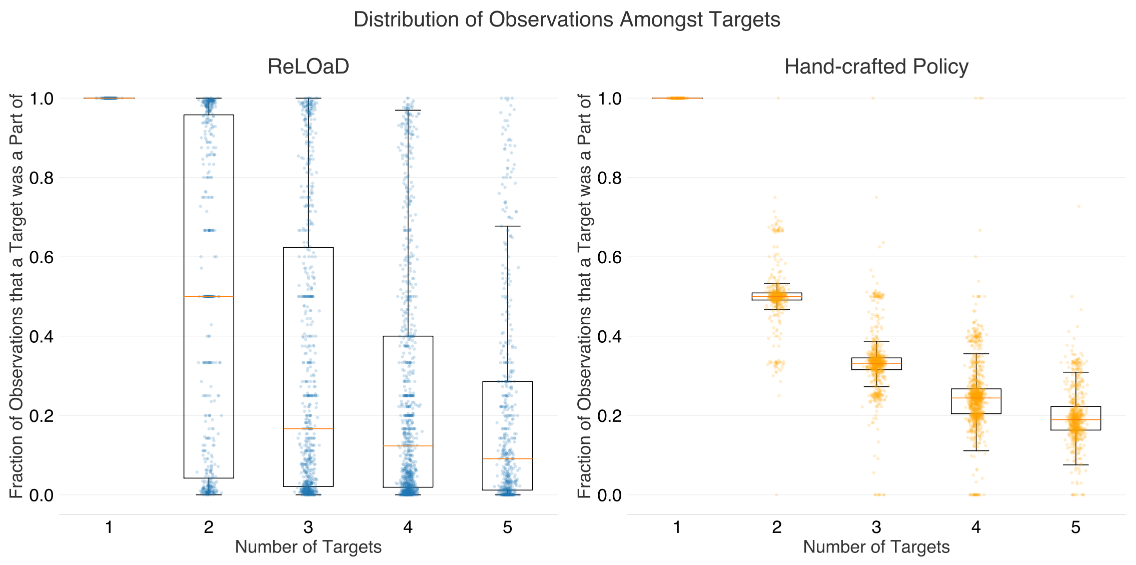

Figure 5.7: Fraction of Total Observations that any Individual Target was Involved in (quartiles shown)

Sub-Task 3: Global Path Planning

The path that the agent takes between the targets was quite variable across the many environmental conditions, it did generally follow the pattern of going between the regions close-by each target (as seen in figure 5.4) but also presented abnormal behaviours, such as performing a turn movement to sweep its gaze over a number of targets to quickly observe them at once rather than drive to them individually. Additionally, the agent would often moving on from a target before fully reaching it (likely due to the diminishing returns of needing to observe multiple times to pull the average entropy up from a poor starting observation). And finally, as previously discussed, the uneven prioritisation of targets can often lead the agent to not visit near a particular target at all.

5.2 Ablation Studies

5.2.1 Comparison of Different Learning Algorithms

The performance of the final SAC model was also compared against models trained on the same final environment definition but instead using the DQN and PPO (with continuous action space) algorithms, with all algorithms trained to convergence (videos of policies provided in supplemental material). As can be seen in figure 5.8, in this experiment, the maximum average reward achieved was 880 for SAC, 440 for PPO (with continuous action space), and 270 for DQN. The gap in performance is also visible in the behaviours developed by the agents. DQN had difficulty learning almost any strategy of value, only bing able to learn very simple behaviours such as spinning around on the spot (guaranteed for targets to periodically pass into view) or sometimes being able to drive directly to the nearest object when the robot spawned nearby a target. PPO does somewhat better, ability to go towards the targets and move itself around to view multiple different targets, but shows a policy of much less complexity. The PPO policy produced more of a ”sweep generally across all the targets” strategy, however, rather

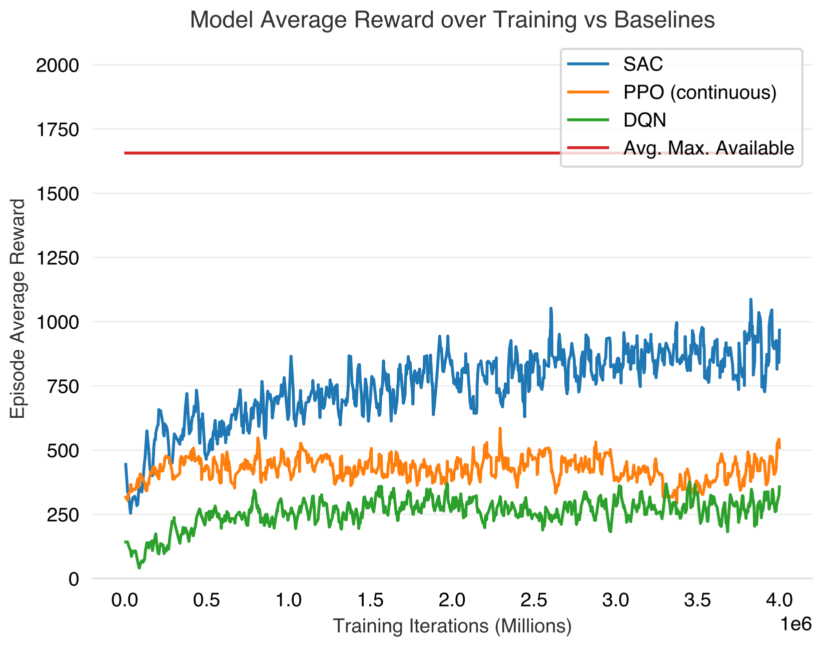

Figure 5.8: Comparison of Performance Achieved with Popular Learning Algorithms

than trying to inspect any particular one specifically, struggling with the multiple step forward planning of how to inspect the targets methodically. It also often became ’stuck’ at the first target it viewed, not successfully managing to leave it’s ’reward zone’ to trade-off and inspect another target. These results suggest that SAC has a better capability than PPO (and discrete DQN showed complete inability) for learning complex behaviours in fundamentally continuous environments such as this one.

5.2.2 State Space: Absolute vs Relative Positions

An experiment was also conducted training SAC on a modified version of the final state space using as absolute rather than relative position format for the positions of targets. In the final ReLOaD model, the state space uses represents the positions of the targets to the agent as the relative positions of / relative distances to each target from the robot. This means that the learning algorithm must do less internal modelling to determine what action to take - since if the relative x/sideways distance is zero, and the y/forwards distance is positive, then the object must be directly in front, so the agent should select to drive forwards. In comparison, if the positions of the robot and targets are provided to the agent simply as Cartesian (x,y) coordinates (along with the angle heading of the robot being provided), then the agent must first determine the angle difference between the current heading of the robot and the vector from the robot to the target, and then compute the Cartesian distance between to the target, in order to then decide what twist action it should take to get there. Although it is still entirely possible to determine the relative position of each target and decide what action to take using this process from absolute information, this added complexity had a disastrous affect on the models performance, as can be seen in figure 5.9. It is likely that the difficulty of having to first learn to model this mathematical process internally (with no specific reward being allocated to encourage the models ability to do this), before being able even start learning how to then make rational path planning decisions from these

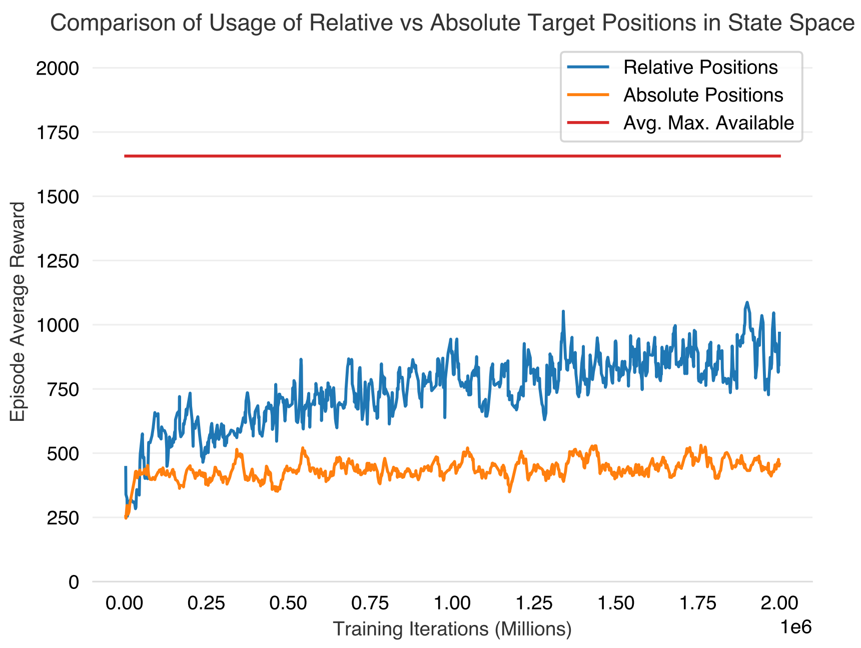

Figure 5.9: Comparison of SAC Model Trained on State Space with Absolute vs Relative Distances

derived results provided too much of an upfront learning curve for the agent to be able to get a start on the right track and then efficiently explore to develop its policy. This emphasises the need for careful definition of the MDP format for training RL algorithms, as the learning process can be brittle to initially seemingly innocuous changes such as this.

Chapter 6

Conclusion

This chapter provides a summary of the solution design and results produced as well as the findings that can be drawn from the project. The chapter also covers some limitations present within the design and potential topics of future work to address these.

6.1 Summary and Conclusions

This thesis was to develop an RL-based algorithm to control the movements of a mobile robot to efficiently collect object detection data from targets. By doing this, more performance can be squeezed out of off-the-shelf OD algorithms by better taking advantage of the mobility of a mobile robotics platform.

We did succeed in training an RL agent that can navigate the robot and collect object detection from the environment. The policy does so with reasonable efficiency, performing approximately on par with a handcrafted policy (although utilising a considerably different strategy) made to solve the same problem, but with less consistent results, especially struggling with longer-horizon planning on cases with larger number of targets.

These findings show that RL is certainly feasible for solving such path planning problems (especially with more refinement of the problem definition) however, it is possibly more suitable for other types of environments. For this problem, “information distribution” (distribution of information across the multidimensional state space) is relatively simple - since the primary pattern is information density is high closest to the targets. And since all of the information needed to determine this information density is also readily given, it is relatively simple to calculate an effective handcrafted policy.

As such, in light of their difficulty in design and tuning, RL-based methods would likely be more beneficial for environments with more complex “information distribution” pattern/dynamics such that a policy could not be as easily hand-crafted to solve the problem (better take advantage of the ability of RL to model complex patterns). That is to say that reinforcement learning approaches do show promise in solving IPP problems, as well as path planning in general, but that the more complex a problem is (ideally with hand-crafting any competent solution being completely unfeasible), the greater potential to exploit the strengths of RL-based approaches and overcome their drawbacks.

6.2 Limitations

This work also presently has a number of limitations that hold it back from practical deployment as a planner on a real-world robot. Primarily, is the inclusion of the target positions in the state vector, meaning that the positions of the objects must be known before the start of the surveying, which is possible in some applications but not in applications involving exploration in uncertain terrain. Secondly, the model was only trained on environments free of obstacles. This is due to the difficulty of incorporating variable number of variable sized obstacles as numerical coordinates into the current state vector without making the state space too bloated for the agent to make sense of. Finally, the agent was trained in with simulated object detection output vector rather than using a real object detection algorithm on viewpoint images. As such moving towards real object detection in a photo-realistic simulator such as iGibson or the real world, may present some more complicated dynamics or noise patterns that the agent might have difficulty to adapting to and need retraining for this new input vector.

Another limitation, which is common to many uses of machine learning in robotics is the general lack of explainability and guarantees that an RL-based path planning algorithm brings. Even when an RL policy can provide improved performance, often the deterministic nature of a handcrafted solution can provide greater safety and reliability that it will provide at least relatively good path every time - something that can be even more valuable in service. As a result, this RL policy would likely need to be surrounded by some hand-crafted guardrails to bound its behaviour within a zone that is safe for the environment and the robot itself.

6.3 Future Work

In order to improve the performance and address some of the limitations discussed previously there a number of future avenues of research that could be continued in the future. firstly among these is moving from an object-coordinates based state space vector for the agent to using an array of RGB pixel values as input (paired with the existing OD confidence vectors). This would allow for the agent to be trained with obstacles by providing a simple way of representing them in the state space (just as black pixels), as well as make the agent inherently flexible to different numbers of targets (as they could just appear as colored pixels in the environment). This would, however, take far more computing resources and training time to be able to effectively run the development cycles required to develop this design. Additional goals would be to test the models adaptability in moving into using OD confidences generated from camera images using a photo-realistic simulator such as iGibson for training.

Bibliography

[1] Y. Wei, R. Zheng, A reinforcement learning framework for efficient informative sensing, IEEE Transactions on Mobile Computing (2020) 1–1doi:10.1109/TMC.2020.3040945.

[2] S. Yu, J. Hao, B. Zhang, C. Li, Informative mobility scheduling for mobile data collector in wireless sensor networks, IEEE, 2014, pp. 5002–5007. doi:10.1109/GLOCOM.2014.7037598.

[3] A. Viseras, R. O. Losada, L. Merino, Planning with ants, International Journal of Advanced Robotic Systems 13 (2016) 172988141666407. doi:10.1177/1729881416664078.

[4] C. Kwok, D. Fox, Reinforcement learning for sensing strategies, IEEE, 2004, pp. 3158–3163. doi:10.1109/IROS.2004.1389903.

[5] J. Ruckin, L. Jin, M. Popovic, Adaptive informative path planning using deep reinforcement learning for uav-based active sensing, IEEE, 2022, pp. 4473–4479. doi:10.1109/ICRA46639.2022.9812025.

[6] J. Gao, W. Ye, J. Guo, Z. Li, Deep reinforcement learning for indoor mobile robot path planning, Sensors 20 (19) (2020). doi:10.3390/s20195493.
URL https://www.mdpi.com/1424-8220/20/19/5493

[7] S. LaValle, Rapidly-exploring random trees: A new tool for path planning, Research Report 9811 (1998).

[8] L. E. K. J.-C. Latombe, Probabilistic roadmaps for robot path planning, Pratical motion planning in robotics: current aproaches and future challenges (1998) 33–53.

[9] J. Xin, H. Zhao, D. Liu, M. Li, Application of deep reinforcement learning in mobile robot path planning, in: 2017 Chinese Automation Congress (CAC), 2017, pp. 7112–7116. doi:10.1109/CAC.2017.8244061.

[10] A. Pirinen, C. Sminchisescu, Deep reinforcement learning of region proposal networks for object detection, IEEE, 2018, pp. 6945–6954. doi:10.1109/CVPR.2018.00726.

[11] S. Mathe, A. Pirinen, C. Sminchisescu, Reinforcement learning for visual object detection, IEEE, 2016, pp. 2894–2902. doi:10.1109/CVPR.2016.316.

[12] B. Uzkent, C. Yeh, S. Ermon, Efficient object detection in large images using deep reinforcement learning, in: Proceedings of the IEEE/CVF winter conference on applications of computer vision, 2020, pp. 1824–1833.

[13] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, A Bradford Book, 2018.

[14] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, IEEE, 2016, pp. 779–788. doi:10.1109/CVPR.2016.91.

[15] C. E. Shannon, A mathematical theory of communication, The Bell system technical journal 27 (3) (1948) 379–423.

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/

[17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in pytorch (2017).

[18] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann, Stable-baselines3: Reliable reinforcement learning implementations, Journal of Machine Learning Research 22 (268) (2021) 1–8.
URL http://jmlr.org/papers/v22/20-1364.html