Autonomous Navigation Using Feature-based Hierarchical Reinforcement Learning

Msc Thesis

Siddharth Unni

16-09-2016
Autonomous Navigation using Feature-based Hierarchical Reinforcement Learning

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering at Delft University of Technology

Siddharth Unni

16/09/2016

Faculty of Aerospace Engineering · Delft University of Technology
The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled “Autonomous Navigation using Feature-based Hierarchical Reinforcement Learning” by Siddharth Unni in partial fulfillment of the requirements for the degree of Master of Science.

Dated: 16/09/2016

Readers:

________________________________________________________
dr. Q. P. Chu

________________________________________________________
dr.ir. E. van Kampen

________________________________________________________
dr.ir. C. C. de Visser

________________________________________________________
dr.ir. W. van der Wal

________________________________________________________
Y. Zhou Msc.
Acknowledgements

This paper would not have been possible without the time, effort and invaluable input of my thesis supervisor, Dr. Erik-Jan van Kampen. I fully appreciate the freedom he gave me in choosing the direction in which to take this research, while being highly critical and involved through every development that was made. I am also thankful for all the time he made available for me, and for being extremely prompt when it came to clarifying any issues that arose throughout the research phase.

I would also like to thank my parents, Indira and Suneel for supporting me not just through this thesis, but through all aspects of my life for the last 23 years. I am grateful for all the values that they instilled in me and for teaching me everything I needed to know in order to complete this thesis. I would be remiss if I didn’t thank my brothers, Nikhil and Rahul, for challenging and pushing me over the years as only older brothers can.

I would also like to thank all my friends for providing me with the much needed escapes from the pressure and stress of completing a Masters.

Finally, I would like to thank my girlfriend, Emily, who has been a huge support throughout this entire process. She has continuously inspired, motivated and encouraged me to make this thesis the best it can be, and for that I am grateful. I also thank her for the many smiles she put on my face, especially when I needed them the most.
# Contents

Acknowledgements ................................................. i
Part I List of Figures ............................................. iv
Part I List of Tables ............................................... v
Part II List of Figures ............................................ vi
Part II List of Tables ............................................. vi
Part III List of Figures ........................................... vii
Part III List of Tables ........................................... vii
List of Symbols ................................................... viii
List of Abbreviations ............................................. viii
Introduction ......................................................... 1

## I Part I

1 Introduction ..................................................... 3

2 Reinforcement Learning Background .......................... 4

3 Comparison of Options, MAXQ and traditional Q-learning .. 7

4 Scalability of Learning using Options ......................... 13

5 Conclusion ....................................................... 25

6 Recommendations for Future Work .......................... 26

7 Bibliography ..................................................... 27

## II Part II

1 State of the art/ Literature review ............................ 28
   1.1 Fundamentals of Reinforcement Learning .................. 28
   1.2 Predominant Flat Reinforcement Learning Approaches .. 31
   1.3 Limitations of Flat Reinforcement Learning .............. 32
   1.4 Hierarchical Reinforcement Learning Methods .......... 33
   1.5 Discussion and Conclusions ................................ 39

2 Research Plan .................................................. 43
   2.1 Theoretical Content/ Methodology ......................... 43
   2.2 Experimental Set-up ....................................... 43
   2.3 Expected Results, Outcome and Results .................. 45
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Map used for initial investigation into performance of RL using absolute and relative states. S, A and B represent the start location and position of the targets, respectively. The black square indicates the agent’s position.</td>
</tr>
<tr>
<td>2</td>
<td>Number of steps required to complete an episode by passing over both targets as seen in Fig. 1 when using absolute states for Q-learning.</td>
</tr>
<tr>
<td>3</td>
<td>Number of steps required to complete an episode by passing over both targets as seen in Fig. 1 when using relative states for Q-learning.</td>
</tr>
<tr>
<td>4</td>
<td>Map used for preliminary comparisons between flat Q-learning, Options and MAXQ. S, A and B represent the start location and position of the targets, respectively. The target A is only accessible once target B has been visited. Therefore, the map initially looks like the figure on the left, and once B has been visited looks like the figure on the right.</td>
</tr>
<tr>
<td>5</td>
<td>Number of steps required to complete an episode by passing over both targets as seen in Fig. 4 using flat Q-learning.</td>
</tr>
<tr>
<td>6</td>
<td>Task used to train the agent to exit a room through wall following.</td>
</tr>
<tr>
<td>7</td>
<td>A near perfect policy for wall following, given the agent has an initial heading of 0° and is not adjacent to a wall.</td>
</tr>
<tr>
<td>8</td>
<td>Number of steps required to complete an episode by passing over target A as seen in Fig. 6 using flat Q-learning.</td>
</tr>
<tr>
<td>9</td>
<td>Number of steps required to complete an episode by passing over both targets as seen in Fig. 4 using Options.</td>
</tr>
<tr>
<td>10</td>
<td>MAXQ Task Graph for the target problem as seen in Fig. 4.</td>
</tr>
<tr>
<td>11</td>
<td>Number of steps required to complete an episode by passing over both targets as seen in Fig. 4 using MAXQ Decomposition.</td>
</tr>
<tr>
<td>12</td>
<td>Maps used for the two room, four room and six room problems respectively. The agent always begins at S and must visit states G1 to G6 (fewer if the two or four room cases are considered) in any order to complete the episode. The goal states are randomly generated within a room every episode while the door states are always located at the same locations.</td>
</tr>
<tr>
<td>13</td>
<td>Reward received per episode for completing the two room task as seen in Fig. 12 using Options with no prior knowledge of a high level policy.</td>
</tr>
<tr>
<td>14</td>
<td>Reward received per episode for completing the four room task as seen in Fig. 12 using Options with a high level policy learnt for a two room problem.</td>
</tr>
<tr>
<td>15</td>
<td>Reward received per episode for completing the four room task as seen in Fig. 12 using Options with a high level policy learnt for a four room problem.</td>
</tr>
<tr>
<td>16</td>
<td>Map used for the four room problem with battery element. The agent always begins at S and must visit states G1 to G4 in any order to complete the episode. The goal states are randomly generated within a room every episode while the door states are always located at the same locations. If the agent’s battery level is low, visiting states P1 to P4 replenishes the agent’s battery level.</td>
</tr>
<tr>
<td>17</td>
<td>Reward received per episode for completing the four room task as seen in Fig. 17 with an “infinite” battery level using Options with no prior high level knowledge.</td>
</tr>
<tr>
<td>18</td>
<td>Reward received per episode for completing the four room task as seen in Fig. 17 with a finite battery level using Options using prior high level knowledge from a case where the battery level is “infinite”.</td>
</tr>
<tr>
<td>19</td>
<td>Reward received per episode for completing the four room task as seen in Fig. 17 with a finite battery level using Options with no prior high level knowledge.</td>
</tr>
<tr>
<td>20</td>
<td>Reward received per episode for completing the four room task as seen in Fig. 17 with a finite battery level using Options with prior high level knowledge.</td>
</tr>
<tr>
<td>21</td>
<td>Reward received per episode for completing a two room variation of the task as seen in Fig. 17 with a high battery level using flat Q-learning (upper figure) and using options with a low battery level (lower figure).</td>
</tr>
<tr>
<td>22</td>
<td>Reward received per episode for completing a two room variation of the task as seen in Fig. 17 with a standard battery level (75) using flat Q-learning.</td>
</tr>
<tr>
<td>23</td>
<td>Reward received per episode for completing the four room task as seen in Fig. 17 using flat Q-learning with a decreasing $\epsilon$ value starting at 0.1 and decreasing until 0.05.</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>1</td>
<td>Comparison of performance between Options and MAXQ for the problem displayed in Fig. 4</td>
</tr>
<tr>
<td>2</td>
<td>Performance in terms of convergence rate, converged value and computation time for the two room four room and six room problem</td>
</tr>
<tr>
<td>3</td>
<td>Performance in terms of convergence rate, converged value and computation time for the four room with battery problem, evaluating the usage of past knowledge to accelerate learning</td>
</tr>
<tr>
<td>4</td>
<td>Performance comparison when solving the four room with battery problem for varying initial epsilon values as well as varying decay rates</td>
</tr>
<tr>
<td>5</td>
<td>Performance comparison when solving the two room with battery problem for varying epsilon decay rates</td>
</tr>
</tbody>
</table>

PART I List of Tables
PART II List of Figures

1.1 Average performance of the $\epsilon$-greedy action value methods on the 10-armed testbed for varying values of $\epsilon$ (Sutton and Barto [1998]) ................................................. 30
1.2 Comparison between the state trajectory of a MDP and a SMDP and their relation to the options framework Sutton et al. [1999] ................................................................. 34
1.3 Required computational time comparison of HAM and flat RL method for Parr’s maze (Parr and Russell [1998]) ................................................................. 36
1.4 Task graph for the Taxi problem Dietterich [2000] ................................................................. 37
1.5 Comparison of average reward obtained through Bayesian-MAXQ, MAXQ, Bayesian and Q-learning for the Taxi problem Dai et al. [2010] ................................................................. 38
1.6 Task graph for the taxi problem as per the OMQ approach Shen et al. [2006] ................................. 39
1.7 Required iterations for convergence comparison of OMQ with MAXQ and Option methods for the Taxi problem Shen et al. [2006] ................................................................. 39
1.8 Representation of recursively optimal policy. The shaded cells highlight where the locally optimal policy diverges from the global optimal. (Dietterich [2000]) ................................................. 41

3.1 Optimal state value function and optimal policy derived using Monte Carlo Exploring Starts 47
3.2 Comparison of the average reward per episode for the racetrack problem using a flat and hierarchical approach ................................................................. 49
3.3 Route followed by the agent in an optimal run after 1000 training episodes for a flat RL implementation ................................................................. 50
3.4 Route followed by the agent in an optimal run after 1000 training episodes for a HRL implementation ................................................................. 50

PART II List of Tables

3.1 Change in average reward for different Epsilon values with $P_{\text{initial}} = \text{random}, Q_{\text{initial}} = \text{zeros}$ 48
PART III List of Figures

1. Learning curve demonstrating a near perfect policy for the navigation option, where the agent always chooses and action that maintains it’s current heading when not faced with a wall . . . 52
2. Learning curve demonstrating a near perfect policy for the exit room option . . . . . . . . . . 52
3. Learning curve demonstrating a near perfect policy for the staying in room option . . . . . . 53
4. Learning curve demonstrating a near perfect policy for the charging option . . . . . . . . . . 53
5. Reward received per episode for completing the six room task without battery with one ran-
domly generated goal per room using Options without any prior high level knowledge . . . . . . 54
6. Reward received per episode for completing the task as seen in Fig. 18 of Part I with an initial
Epsilon value of 0.1 and a decay rate of 0.9975 . . . . . . . . . . . . . . . . . . . . . . . . . . 54
7. Reward received per episode for completing the task as seen in Fig. 18 of Part I with an initial
Epsilon value of 0.05 and a decay rate of 0.9955 . . . . . . . . . . . . . . . . . . . . . . . . . . 55
8. Reward received per episode for completing the task as seen in Fig. 18 of Part I with an initial
Epsilon value of 0.05 and a decay rate of 0.9975 . . . . . . . . . . . . . . . . . . . . . . . . . . 55
9. Reward received per episode for completing the task as seen in Fig. 18 of Part I with an initial
Epsilon value of 0.05 and a decay rate of 0.999 . . . . . . . . . . . . . . . . . . . . . . . . . . 55
10. Reward received per episode for completing the four room task with two targets and battery
element using Options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
11. Reward received per episode for completing the four room and battery task with one randomly
generated goal using Options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
12. Reward received per episode for completing the four room and battery task with one randomly
generated goal using Options with a high reward for finding the goal . . . . . . . . . . . . . . . 57
13. Repeated trial showing reward received per episode for completing the four room, four goal
and battery task using Options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
14. Second repeated trial showing reward received per episode for completing the four room, four
goal and battery task using Options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58

PART III List of Tables

1. Performance of learnt sub policies in terms of iterations for convergence and computation time 53
2. Comparison of performance of the six room problem using no prior knowledge with a case
where prior knowledge is used . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
3. Comparison of performance for the four room problem with four goals and battery element
using the same setting so as to demonstrate variation between trials . . . . . . . . . . . . . . . 58
List of Symbols

\(\alpha\) Learning rate
\(\epsilon\) Epsilon value
\(\gamma\) Discount factor
\(\pi\) Policy
\(\tau\) Number of time steps within an option or temperature parameter
\(A\) Action set
\(a\) Action
\(C\) Completion function
\(i\) Subtask in hierarchy
\(N\) Number of steps for which a subtask is executed
\(o\) Current option
\(P\) Transition probability
\(Q\) State-action or state-option matrix
\(r\) Reward per action
\(R\) Reward per episode
\(S\) State set
\(s\) Current state
\(s'\) Next state
\(V\) Value function

Subscripts
\(k\) Current value
\(k + 1\) Updates value
\(t\) Current time step
\(t + 1\) Next time step

List of Abbreviations

HAMS  Hierarchical Abstract Machines
HRL  Hierarchical Reinforcement Learning
MDP  Markov Decision Process
OMQ  Options and MaxQ
RL  Reinforcement Learning
SLAM  Simultaneous Localisation and Mapping
SMDP  Semi Markov Decision Process
UAV  Unmanned Aerial Vehicle
Introduction

In recent decades, the autonomous navigation of UAVs has become very much a reality, and is currently being used in limited capacities for various kinds of tasks. These range from search and rescue missions to assistance with household activities. In any event, regardless of the task being tackled, new and novel ways to introduce safety and improve performance for autonomous navigation are being continually explored.

Despite there being several approaches to performing autonomous indoor navigation, this thesis is concerned with a machine learning approach known as Reinforcement Learning (RL). RL is a fundamental learning approach, wherein the agent is not explicitly told what to do, but rather must discover the best action to take by understanding which action yields the greatest reward when in a certain state. Therefore, RL aims to mimic the trial and error way in which a human being might learn several different skills. This simple principle underlying RL makes it a very powerful tool as these strategies can be applied to a very broad spectrum of problems. Despite its strengths however, RL struggles when faced with complex problems and large problem spaces, as the ability to learn effectively is adversely affected as the problem size grows. This limitation has given rise to Hierarchical Reinforcement Learning (HRL), and extension to traditional RL, wherein learning takes place on different levels, with decision making being done with varying levels of detail. This is known to accelerate learning and further represents the way in which human beings go about performing complex tasks.

The primary aim of this report i to investigate the potential of HRL in terms of enabling UAVs to successfully perform autonomous navigation of a previously unexplored area. Further, the limitations of flat RL at solving complex and large problems is demonstrated and the need for HRL is made explicit. To that end, a thorough assessment of existing approaches and standard practices is carried out, as presented in Part II of this report. Ultimately a decision is made to evaluate in some detail the Options and MAXQ methods and therefore these methods are used to solve simple navigation problems. Subsequently, the Options method is explored in significant detail and implemented on a complex navigation problem, with a focus placed on the ability to reuse prior knowledge as a) the size of the problem increases and b) the complexity of the problem increases.

Research Questions

This section will state the primary research question that the thesis aims to answer and will subsequently list sub research questions which when answered will together answer the primary research question.

Primary Research Question

"How can a quadrotor exploit hierarchical reinforcement learning methods to optimally* navigate an area and locate points of interest?"

* Optimally here is defined in terms of a) the size of the problem space that can tackled, b) the number of iterations until convergence, c) the final value of the converged value function d) the computational time required for convergence

Sub-Research Questions

1. "To what extent can navigational problems be solved by HRL?"
• "To what extent are RL methods in general suitable for solving navigational problems?"
• "How does the introduction of a hierarchy influence performance for the given task?"

2. "What benefits and limitations does HRL possess as opposed to flat RL methods?"
• "What are the various HRL methods that could be used to solve navigational problems?"
• "What are the challenges/difficulties with implementing HRL?"
• "What is the effect of HRL on computation time, number of actions required for convergence and final value upon convergence with respect to flat RL?"

3. "What are the benefits and limitations of the Options HRL approach with respect to the MAXQ HRL framework?"
• "What are the challenges/difficulties with implementing the different HRL methods?"
• "How do the aforementioned HRL methods compare in terms of computational time, number of actions until convergence and final value upon convergence?"

4. "What is the effect of scaling up the problem space on performance of the HRL method in question?"
• "How does performance change when the problem size is increased?"
• "How does performance change when the complexity of the problem is increased, for instance through the addition of more tasks?"
• "To what extent is the transfer of learning possibly when the problem is scaled up, in either size or complexity?"
• "If transfer of learning is possible, to what extent does it influence performance for a particular HRL method?"
• "What is the limit, if any, until which the implemented HRL method is still functional?"
• "What are the difficulties involved with making the HRL method functional for a larger problem space?"

Report Layout

This report consists of three parts. Part I is a standalone paper highlighting the final results of this thesis. This paper deals with a complex indoor navigation problem and demonstrates the ability of the Options HRL method to effectively solve such a problem in an efficient manner. It further demonstrates how the use of prior high level knowledge has a significant and positive impact on the learning rate as a) the problem size scales up and b) the problem complexity scales up. This paper also highlights the ineffectiveness of traditional flat Q-learning at solving such a complex problem. Prior to dealing with a complex navigation problem a preliminary investigation into the MAXQ and Options method as applied to a simple navigation problem is also explored. Part II of this report consists of a preliminary report. This report serves as a literature study and presents the motivation for the research presented in the paper. Therefore, it provides an in depth study of existing work in the field while providing the theoretical foundation for the results presented in Part I. Furthermore, Part II also presents some preliminary results showing the promise and feasibility of using HRL in navigation. Part III of of this report presents additional results that could not be included in Part I yet still have relevance and lead to meaningful insights. Following part III, a conclusion to the entire report is presented.
Part I
Autonomous Navigation using Feature-based Hierarchical Reinforcement Learning

The ease of availability of low cost aerial platforms has given rise to extensive research in the field of autonomous navigation. There are strong indications in existing research that UAV autonomy leads to significant gains in terms of safety as well as performance in a number of scenarios, including but not limited to search and rescue missions in disaster areas. This paper tackles the problem of autonomous indoor navigation by applying reinforcement learning. In specific terms, this paper employs hierarchical reinforcement learning methods in order to overcome the challenges posed by the complexity and size of the problem space when dealing with navigational problems. In addition, a feature-based relative state is used so as to contain the number of states to a manageable level, while ensuring that the learnt policies are perfectly valid, independent of the size of the problem. The findings of this paper successfully demonstrate the ability of the Options method to solve a complex navigation problem involving goal finding, room exiting and battery recharging in an efficient manner, and further to solve a problem that cannot be solved using traditional flat Q-learning. In addition, this paper provides evidence of the positive influence of prior high level knowledge on the learning rate for a navigation problem as a) the problem size scales up and b) the problem complexity scales up.

Nomenclature

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>Epsilon value</td>
</tr>
<tr>
<td>$Q$</td>
<td>State-Action or State-Option Matrix</td>
</tr>
<tr>
<td>$s$</td>
<td>Current state</td>
</tr>
<tr>
<td>$s'$</td>
<td>Next state</td>
</tr>
<tr>
<td>$N$</td>
<td>Number of steps for which a subtask is executed</td>
</tr>
<tr>
<td>$o$</td>
<td>Current Option</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Number of time steps within an option</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Discount factor</td>
</tr>
<tr>
<td>$t + 1$</td>
<td>Next time step</td>
</tr>
<tr>
<td>$k + 1$</td>
<td>Updated value</td>
</tr>
<tr>
<td>$\pi$</td>
<td>Policy</td>
</tr>
<tr>
<td>$V$</td>
<td>Value function</td>
</tr>
<tr>
<td>$C$</td>
<td>Completion function</td>
</tr>
<tr>
<td>$i$</td>
<td>Subtask in hierarchy</td>
</tr>
<tr>
<td>$a$</td>
<td>Action</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Learning rate</td>
</tr>
<tr>
<td>$r$</td>
<td>Reward</td>
</tr>
<tr>
<td>$t$</td>
<td>Current time step</td>
</tr>
<tr>
<td>$k$</td>
<td>Current value</td>
</tr>
</tbody>
</table>

I. Introduction

The autonomous control of an agent, such as an Unmanned Aerial Vehicle (UAV), for navigation tasks is known to have several advantages and applications. When dealing with disaster stricken situations for instance, there are explicitly apparent benefits in terms of safety, as humans beings do not need to put themselves in harms way. In addition to that, there is evidence to suggest that autonomous control could lead to improved performance, especially when considering the use of swarms for search and rescue missions, as well as their ability to explore areas that human beings may physically be incapable of accessing. Therefore, in recent years, extensive research has been carried out on developing and applying strategies so as to enable agents to make decisions without any form of human intervention, such that they can interact with unknown
environments and function at a high level. This recent interest has been further propelled by the ease of availability of low cost yet highly function small to medium sized aerial platforms.

Researchers have attempted to solve the navigation problem, using amongst other methods, probabilistic motion models with particle or Kalman filters. These methods attempt to generate a map of the domain being explored in somewhat real time, through a process known as simultaneous localisation and mapping. In addition, machine learning approaches are also commonly used for navigation tasks, with a focus traditionally being on supervisory learning. While all the aforementioned methods are promising and present their own unique strengths and weaknesses, when a new and dynamic environment is encountered, all these methods display serious limitations as they are unable to learn for themselves and are therefore not very good at adapting to changes in the system.

An alternative machine learning method that has become popular in recent years is Reinforcement Learning (RL). RL is arguably the most intuitive form of learning, and is predicated on an agent’s ability to associate a reward with an action, given a particular state. Therefore, unlike supervisory learning, the agent has no explicit instructor and is never told what to do, but determines the optimal action based on feedback received from the environment. The fundamental simplicity of RL makes it an extremely powerful method that can be applied to a wide range of problems. In the context of a navigational task, RL has the potential to exploit knowledge learnt from different yet similar situations, thereby allowing the agent to effectively navigate new and unknown environments much like a human being might. Despite all of its strengths, RL has long suffered from the curse of dimensionality, wherein as the size and complexity of the target problem increases, the agent’s ability to effectively learn at a feasible rate greatly decreases. This has given rise to an extension to classical RL, referred to as Hierarchical Reinforcement Learning (HRL). As the name suggests, HRL attempts to exploit decision evaluation at multiple levels, and in doing so aims to drastically decrease the number iterations required for learning to take place.

The primary aim of this paper is to solve complex navigational problems, specifically one in which an agent must autonomously perform indoor navigation and locate points of interest, while having to periodically perform a sequential task such as recharging its battery. In realising that goal, RL and HRL will be used. This paper presents decision making that is based in the relative state and therefore makes use of perceived features in order to determine actions. In solving the primary target problem, an attempt is made to investigate the effect of prior knowledge on learning, as a) the size of a problem is increased and b) the complexity of a problem is increased.

Section II of this paper aims to outline the theoretical fundamentals of the RL and HRL concepts that are applied in this paper. Section III presents the preliminary simulation results obtained by applying flat Q-learning, Options and MAXQ methods to a simplified version of the target task. Section IV presents the final simulation results obtained by applying Options and flat Q-learning to the complex target task. Finally, Sections V and VI highlight the relevant conclusions and discuss areas for further research, respectively.

II. Reinforcement Learning Background

This section will provide the relevant background information on RL so as to clarify the theoretical approach used when implementing the various methods. To that end, the fundamentals of RL will be presented, followed by a more detailed description of the specific flat and hierarchical algorithms used.

A. RL Preliminaries

RL, in its general sense, is composed of three elements, namely states, actions and rewards. These three elements come together to form the concept of a policy, which is defined as "a mapping from states to actions" (Watkins 1989). As per Watkins "the aim of the learner will be to construct a policy that is optimal in the sense that, starting from any state, following the policy yields the maximum possible expected return that can be achieved starting from that state" (Watkins 1989). Ultimately, the goal of all RL is to determine a policy that results in the optimal value function.

All decision making considered in this paper is bound to either the Markov Decision Process (MDP) or the Semi Markov Decision Process (SMDP). As per the MDP framework, all information required for the agent to make subsequent decisions is contained in its current state, and therefore no prior knowledge of
previous states is required. While flat RL problems fit within the MDP framework, HRL makes use of a generalisation of the MDP known as a SMDP. The primary difference between the two is that an action in SMDPs can last multiple time steps. In other words, while a flat RL approach consists only of primitive actions (executed over one time step), an SMDP consists of actions that are either primitive or temporally extended.

In terms of action selection given a certain policy, there are several different approaches that can be applied. This paper however is only concerned with the $\epsilon$-greedy action selection approach. An $\epsilon$-greedy policy simply means that the non greedy action is chosen with a probability of $\epsilon$ and the greedy action is chosen with a $1 - \epsilon$ probability.

B. Flat Reinforcement Learning

In terms of flat RL, this paper makes use of Q-learning, arguably the most extensively used classical RL method. The model free nature of this method makes it a suitable choice when considering the complexity of the problem to be solved. As per this approach, the value function is updated at the end of every time step using temporal errors and not at the end of each episode, which consequently implies that learning can take place within an episode (Kober et al. [2013]). The Q-learning algorithm, as seen in Eq. 1 (Sutton and Barto [1998]) approximates the optimal state-action function ($Q^*$) at each time step. Given that all state action pairs are continually updated, the state value function is guaranteed to converge at the optimal state-value function with a probability of 1 (Sutton and Barto [1998]). The step size parameter, $\alpha$ and discount rate $\gamma$ in Eq. 1 must be tuned for the task at hand, as they influence the learning rate. As can be seen, the state-action estimate $Q(s_t, a_t)$ is updated using the current state-action function, the reward $r$ at time step $t+1$ (the observed reward), the $\max Q$ value of the next state $s_{t+1}$ given a certain action $a$ has been executed and the previously described parameters.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max Q(s_{t+1}, a) - Q(s_t, a_t)]$$  \hspace{1cm} (1)

C. Hierarchical Reinforcement Learning

Flat RL methods limit the designer’s ability to provide additional knowledge to the agent, such that it has a better understanding of how a complex target task may be executed optimally. It further limits the agent’s ability to deal with different subtasks using different approaches, such as having different action selection methods. This ties in in some ways to the fact that RL methods scale very poorly, both in terms of the aforementioned complexity of the problem but also in terms of the size of the problem space that needs to be dealt with. This problem is further extended by the fact that flat RL is bound by the MDP which is limited to discrete-time models, thereby adding extra computational burdens on the agent. HRL methods on the other hand are free to exploit the SMDP framework wherein computations can be performed within the continuous time domain. In addition, the use of subtasks and subgoals in HRL implies that learnt policies for subtasks can be transferred to different problems with the same or similar subtasks, and by doing so can lead to accelerated learning. The complex nature of the problem being considered in this paper results in classical reinforcement learning not being a suitable fit to find the optimal solution, thereby creating the need for further investigation into hierarchical approaches.

This paper will evaluate in varying levels of detail the Options framework as well as the MAXQ framework.

1. Options

The term options is defined as a "generalization of primitive actions to include temporally extended courses of action" (Mcgovern et al. [1998]). An option is composed of three elements, namely an input set, a policy and a termination condition. An option can be chosen if and only if the the state it is in at a certain point in time is a subset of the input set that constitutes the option. Once an option is chosen, the policy of that option is followed until the termination condition is met. Once an option has terminated, a higher level controller determines the next course of action, which could include selecting another option.

The input set and termination condition of an option allow for the range over which the option must be defined to be limited in a useful manner (Sutton et al. [1999]). In effect, since policies in the options method need to be learned independently of the macro task being performed (either pre-programmed or learnt through some other training method), it is advantageous if the policy only needs to be defined for an
input set and not all the states of the entire macro problem. Therefore, in terms of policy generation, the input set and termination condition apply a very useful constraint to the subtask.

Equation 2 (Barto and Mahadevan [2003]) shows the Q-learning algorithm to be followed for updates of the state-option matrix, $Q$. As can be seen, this equation is analogous to Eq. 1 with the exception of two elements. Firstly, an action $a$ is replaced by an option $o$. In addition, the variable $\tau$ represents the number of time steps for which an option is executed for a given option. $Q_{k+1}$ refers to the updated $Q_k$ value. Much like with the flat approach, the state-option matrix is updated once an option is terminated. As can be deduced, if all the options $o$ are constituted of primitive actions, Eq. 2 is the same as Eq. 1. While this could be considered a strength in terms of implementation (due to its similarity to flat Q learning), there are also drawbacks due to the lack of updates until an option has terminated [Precup 2000]. In effect, since updates are only received upon termination of an option, non terminating options are never updated, and furthermore no useful information can be extraction during the execution of an option. This therefore implies that the termination condition of options must be carefully evaluated such that updating can take place frequently enough.

$$Q_{k+1}(s,o) = (1 - \alpha_k)Q_k(s,o) + \alpha_k [r + \gamma \max Q_k(s',o)]$$  \hspace{1cm} (2)

In terms of implementation of the options method, a three step method that can easily be applied to the navigation task at hand is considered. The first step is to determine a set of local subgoals from the primary target task. This means breaking down the primary target task into subtasks. Once this is done, options are determined for each of the subtasks as far as possible. Finally, the options are framed such that they can be reused for subtasks that resemble the subtask they are devised for, and in doing so can accelerate learning for the primary target task.

2. MAXQ

The MAXQ (Dietterich [2000]) approach works by decomposing the value function of the primary target task into a group of value functions for subtasks. Therefore, integral to MAXQ is the ability to decompose a value function into a set of value functions. As with options, MAXQ also relies on the SMDP framework, and solves problems by creating a hierarchy of SMDPs that are processed simultaneously to solve the primary target task. However, unlike the aforementioned methods, the MAXQ method does not aim to directly reduce the target task to a single SMDP, but rather creates a hierarchy of SMDPs that are solved simultaneously.

Each subtask within the MAXQ framework is composed of three elements, namely a subtask policy, a termination condition and a pseudo reward function per subtask (Barto and Mahadevan [2003]). A subtask is similar to a hierarchical option, and therefore a subtask policy executes actions or selects other subtasks that fall under it in the task graph. These policies are considered to be deterministic and therefore need to be learned by the agent or pre-programmed. The termination policy allows for the division of the target MDP into subtasks of sub MDPs (or SMDPs) as it determines when a certain subtask is completed and a transition to the next one can be made. The pseudo reward function is not integral to execution of the target task, but is something that is required during learning of policies for subtasks.

The hierarchical structure of the MAXQ method implies that the $Q$ matrix is composed of $i$, $s$ and $a$, where $i$ refers to the subtask in the hierarchy being evaluated, $s$ refers to the state and $a$ the action. For all subtasks that are not primitive actions, $Q(i,s,a)$ is computed as shown in Eq. 3 where $\pi$ refers to the policy being followed, $V^\pi(a,s)$ is the projected value function for the child task and $C^\pi(i,s,a)$ is the completion function and represents the discounted cumulative reward for completing the subtask in question. $V^\pi(a,s)$ and $C^\pi(i,s,a)$ are elaborated in Eqs. 4 and 5 respectively. $V_{t+1}$ and $C_{t+1}$ represent the updated values for $V_t$ and $C_t$ respectively, $N$ is the number of steps for which a particular subtask is executed and $s'$ is the observed resultant state.

$$Q^\pi(i,s,a) = V^\pi(a,s) + C^\pi(i,s,a)$$ \hspace{1cm} (3)

$$V_{t+1}(i,s,a) = (1 - \alpha(i)) \cdot V_t(i,s) + \alpha(i) \cdot r_t$$ \hspace{1cm} (4)

$$C_{t+1}(i,s,a) = (1 - \alpha(i)) \cdot C_t(i,s,a) + \alpha(i) \cdot \gamma N V_t(i,s')$$ \hspace{1cm} (5)
III. Comparison of Options, MAXQ and traditional Q-learning

Simulation results in this paper will be presented in two distinct sections. This section will deal with the preliminary simulations used to explore the Options and MAXQ frameworks in a relatively simple scenario, such that an evaluation can be made on which method might be more suitable for a more complex version of the same problem. In addition, a justification for the use of relative feature-based states will also be provided.

A. Problem Definition and Implementation

The problem is set up as a gridworld wherein the agent must locate a point of interest and is only then capable of leaving an enclosed area (the equivalent of a room). This could have several applications, such as inspecting a house after a disaster and looking for victims or other objects of interest. All policies and decision making is done through the use of RL algorithms that abide by the MDP or SMDP framework.

The start position of the agent at the beginning of every episode is random, however the point of interest is always located at the same point. This therefore implies that the agent must be capable of finding the target from any position on the map. The exit from the room can be located in one of four locations (one on each of the four walls), and is randomly selected at the start of each episode. The problem setup will be further described in this Section.

1. Absolute State vs Relative State

An initial attempt was made to apply RL algorithms in the absolute state. In this scenario, the state is composed of the agent’s x and y coordinates in the gridworld, and therefore decision making is done based on the agent’s absolute location. This is comparable to using a GPS wherein the location of the agent is known, and based on that knowledge a particular action is chosen. There is extensive evidence in literature to suggest that promising results can be obtained for navigational tasks using the absolute state (Harris et al. [2015], McGovern [2002], Wang et al. [2014] and Shen et al. [2006]). Figure 1 shows the map wherein initial simulations were carried out. In this task, the agent begins at S and must pass over points A and B in any order in order to complete the episode. The start position and target positions are the same every episode. Fig. 2 shows the number of steps required per episode in order to complete the aforementioned task. The Q-learning algorithm shown in Eq. 1 is used. As can be seen, even after a 100 episodes there is no sign of convergence for what is a relatively easy problem. This can be attributed to the fact that there are multiple positions that need to be visited in order to complete the episode. Using a more aggressive exploration policy initially could help overcome this problem, however it is not hard to see how the absolute state could have serious limitations when dealing with multiple targets which possibly change location frequently. Therefore, use of an alternative state definition, namely the relative state was considered. The relative state is defined not in terms of location, but in terms of the features perceived by the agent when it is in a particular location. This is comparable to using cameras or one’s eyes to make a decision based on what can be seen, thereby enabling learning for a more general and dynamic scenario. Not only does the relative state show promise in terms of dealing with multiple targets, but it also means that the number of states is fixed, regardless of how big the size of the problem space is. The performance of the relative state can be seen in Fig. 3. As can be seen, convergence takes place in less than 10 episodes, with the exception of a few exploratory outliers. Therefore, the relative state will be used in this paper. More information about the states, actions and rewards will be provided in the next subsubsection.

Beyond this demonstration in terms of dealing with multi-target problems, the relative state also has apparent benefits over absolute state use in terms of containing the total number of states. As is described in Subsection 2 in terms of pure navigation, the total number of states in the feature-based approach is contained to 108 regardless of how big the problem size becomes. Therefore, the moment we deal with a grid that larger than 10x10, using the absolute state becomes sub optimal. Since the broader aim is to develop methods that are applicable for large problem spaces, it is clear to see how the relative state lends itself nicely to scaling learnt policies. In terms of practical applications, using feature-based states can be advantageous when dealing with indoor areas where GPS and therefore absolute location might be unavailable or unreliable.
2. States, actions, rewards

After a preliminary investigation into how the states should be defined, the final version of the preliminary problem is set up. The map showing the task to be completed can be seen in Fig. 1. As per this problem, the agent must first visit location B which is at a fixed location, and can then proceed to exit the room from A. Once the agent reaches A the episode is over. The start position S is randomly generated within the room at the start of each episode, and the exit A is randomly generated in one of four locations which could be on any wall. The area the agent can explore consists of 11x11 discrete locations, including the surrounding wall.

The agent’s state is composed of what it can see one step ahead, one step to the right, one step to the left, it’s heading and whether or not it has already visited state B. If there is no obstacle or goal state, the agent perceives a 0, if there is an obstacle (such as the wall), the agent perceives a 1 and if there is a goal state, the agent perceives a 2. The agent’s heading can be either 0 (East), 90 (North), 180 (West) or 270 (South). If the agent has visited the target inside the room, the last element of its state changes to a 1, until which time it is 0. This gives rise to a total number of states of 216 (3 × 3 × 3 × 4 × 2).

In terms of actions, the agent is able to move one step forward (continue in the same heading) or turn either to the right or the left (change heading by 90° in either direction). Therefore, the Q matrix being considered is 216 x 3, regardless of the size of the problem being considered. However, changing complexity, for instance by adding a charging station and battery level, would influence the size of the Q matrix.

The rewarding strategy used is rather straightforward. Each action that the agent takes, whether it be moving forward or turning, results in a reward of -1. Colliding with an obstacle or going out of bounds incurs the standard reward of -1 for the action taken, and the agent is initialised at it’s previous valid position. If the agent reaches a goal state, a reward of +5 is received. A reward at a given goal state can only be picked up once per episode. This prevents the agent from repeatedly going to the same state to collect rewards within an episode. It should also be noted that once a goal state has been visited and the reward has been collected, the state perceived by the agent at the goal state changes from a 2 to a 0. In other words, it no longer sees a goal state where there earlier was one.
Figure 3. Number of steps required to complete an episode by passing over both targets as seen in Fig. 1 when using relative states for Q-learning.

Figure 4. Map used for preliminary comparisons between flat Q-learning, Options and MAXQ. S, A and B represent the start location and position of the targets, respectively. The target A is only accessible once target B has been visited. Therefore, the map initially looks like the figure on the left, and once B has been visited looks like the figure on the right.

B. Preliminary Simulation Results

This section will present the results for the three applied methods independently.

1. **FLAT RL**

Q-learning, as seen in Eq. 1, was applied to solve the above mentioned problem, as seen in Fig. 4. The agent is allowed to take up to 5000 steps to complete an episode, and if it does not complete the episode in this amount of time, the current episode is terminated and the next one is begun. The learning rate $\alpha$ was kept constant at 0.1 and the discount factor $\gamma$ was set to 0.9. In terms of action selection, an $\epsilon$-greedy policy was implemented, with an initial $\epsilon$ value of 0.1. This value was decreased by a factor of 0.999 every episode, so that less exploration took place as the agent learnt better policies. When applying a flat RL method to a target task with distinct subtasks, the ability to have only one set of parameters and only one action selection strategy is a huge limitation. In effect, the open space navigation component of finding goal B requires a high level of exploration as the path to be taken is different for every run, and all locations that are not adjacent to the wall or near a goal state look the same to the agent. That being said, in terms of finding the exit, it is clear that the agent should adopt a wall following policy and does not need much exploration once the policy is learnt. The inability to distinguish between these two elements is largely responsible for the poor results seen in Fig 5. As can be seen, the results are extremely random and even after 2000 episodes there is no sign of convergence at all.

2. **Options**

In implementing the Options framework, the target task was divided into two subtasks, namely open space navigation to find goal B and wall following to exit the room through goal A. Given that the agent is aware that it has visited goal state B, a designer could easily deterministically specify which option to follow given the state the agent is in. However, the purpose of this simulation was to determine how effectively an agent can automatically discover the global hierarchy given a set of learnt policies. To that end, the agent
was trained independently of the target task for open space navigation and wall following, using standard Q-learning and was allowed to discover the hierarchy through training.

An example of subtask learning can be seen in Figs. 6, 7 and 8. Figure 6 shows the problem used to train the agent to exit a room. The agent spawns at a randomly generated location S and must find its way out of the room through a randomly generated goal A. It should also be noted that training for the optimal policy is done on a smaller problem space than that used for the target problem as seen in Fig. 4. This is a huge advantage of using the options method in the relative state, as sub policies can be learned extremely quickly and easily. Figure 8 shows that the agent is able to learn the optimal policy in under 60 episodes. The near perfect policy learnt can be seen in Fig. 7. The policy shown here is for a case where the agent is initially in a state where its heading is $0^\circ$ and it is not adjacent to a wall or a goal state. As can be seen, the policy is not necessarily optimal for every possible scenario, however when tackling such a problem in a generalised way, the policy is essentially optimal.

Figure 6. Task used to train the agent to exit a room through wall following

Figure 7. A near perfect policy for wall following, given the agent has an initial heading of $0^\circ$ and is not adjacent to a wall.

Figure 8. Number of steps required to complete an episode by passing over target A as seen in Fig. 6 using flat Q-learning
Having derived policies for the subtasks, Eq. 2 was used to update the global policy to be followed. In defining the states over which each of the options is valid, a certain degree of determinism is used. This is in keeping with how the Options framework should be implemented, as limiting the range over which an Option is valid is critical to facilitating rapid learning in a complex environment. To that end, the general navigation option is available over all possible states, as the agent could always have need for this. The room exiting option however is only valid for states where state B has already been visited, as it is only then the agent needs to leave the room.

\( \alpha \) for the global Q function is set to 0.1 and the discount factor \( \gamma \) is 0.9. An \( \epsilon \)-greedy policy is used to select the option, and is initially set to 0.1, but is decreased by a factor of 0.995 per episode. The general navigation policy has an \( \epsilon \) value set at 0.25, implying that the agent will turn every one in four steps. This action selection approach was found to provide the best results when navigating through an area where the majority of states are identical. The room exit option uses a greedy policy, as the learnt policy is optimal and no exploration is required. The results obtained using the Options method can be seen in Fig. 9.

\[\text{Figure 9. Number of steps required to complete an episode by passing over both targets as seen in Fig. 4 using Options}\]

3. MAXQ

The primary target is divided into the same subtasks as with Options, namely a general navigation task and a room exiting task. Unlike Options, the MAXQ method requires the designer to explicitly state the hierarchy, and there is no way for the agent to learn the hierarchy on its own. The hierarchy employed in solving the target task is shown in Fig. 10. As can be seen, the task graph contains Max-nodes and Q-nodes, and while not strictly necessary for implementation of the MAXQ method, these provide clarity when computing Q values. In effect, the Max-nodes correspond to primitive actions and subtasks and represent the value computed using Eq. 4 whereas the Q-nodes correspond to the actions available for a particular subtask and store the values computed using Eq. 5. The policies used for completing the subtasks are the same as for those used for the previously mentioned Options method. In addition, the parameter setting and \( \epsilon \) values for the subtasks are also the same as those used for the Options method.

The MAXQ-0 approach, as seen in Algorithm 1, was used in solving the target problem at hand. Although not as widely applicable as the more general variant, MAXQ-Q, for a case where all pseudo reward functions are set to 0, MAXQ-0 is a perfectly valid approach which guarantees convergence [Dietterich 2000]. The approach follows a recursive process with a top-down decision making structure, with higher level decisions only being re-evaluated when lower level subtasks reach their termination condition. For the navigation process to find B, the termination condition was the same as the goal state. In terms of exiting the room, to avoid determinism, the terminal states involved not only the goal state A but also being in locations that are not adjacent to a wall. While the agent remained near the wall, the wall following policy was adhered to, if that was the chosen subtask.

As can be seen, the MAXQ method converges at around 35 runs, with the exception of an outlier. Given the simplicity of the problem it is surprising that so many iterations are required for the agent to learn the optimal hierarchical policy. This slow learning can be attributed to a poor definition of the termination predicates.
C. Synthesis

Having implemented flat RL, Options and MAXQ to solve the problem depicted in Fig. 4, some conclusions can be drawn with respect to performance and ease of implementation. Firstly, it can be easily concluded that Q-learning without any hierarchy is unable to solve what is effectively a rather simple problem. Figure 5 shows that even after 2000 episodes, no learning takes place. This can be attributed in large part to the different nature of the two subtasks being considered, the different start position and the use of relative states.

The Options and MAXQ methods both show great promise and learning takes place in a few runs as seen in Figs. 9 and 11. It should however be noted that perhaps the problem being considered here was too simplistic and therefore did not leave enough room for the complete strengths and weaknesses of the two methods to be fully visible. However, it did provide sufficient insight to make an informed decision on which method would be a better fit when dealing with a more complex problem of this nature. From a purely quantitative viewpoint, Table 1 shows how the methods compare. As can be seen, the Options method performs better than the MAXQ method for all considered criteria. It should be noted that these results do not speak to the overall performance of the two methods with respect to each other but is only an indication of their performance given the problem being considered and the way in which they are implemented.
Algorithm 1 Pseudo code for the MAXQ-0 learning algorithm (Dietterich [2000])

1: if \( i \) is a primitive MaxNode then
2: \( \text{execute } i, \text{ receive } r, \text{ and observe result state } s' \)
3: \( V_{t+1}(i,s) := (1 - \alpha_t(i)) \cdot V_t(i,s) + \alpha_t(i) \cdot r_t \)
4: \( \text{return } 1 \)
5: else
6: let count = 0
7: while \( T_i \) is false do
8: Choose an action \( a \) according to the current exploration policy \( \pi_x(i,s) \)
9: let \( N=\text{MAXQ-0}(a,s) \) (recursive call)
10: \( C_{t+1}(i,s,a) := (1 - \alpha_t(i)) \cdot C_t(i,s,a) + \alpha_t(i) \cdot \gamma^N V_i(i,s') \)
11: count:=count+N
12: \( s := s' \)
13: end
14: return count
15: end MAXQ-0
16: // Main Program
17: initialise \( V(i,s) \) and \( C(i,s,j) \) arbitrarily
18: MAXQ-0 (root node 0, starting state \( s_0 \))

Table 1. Comparison of performance between Options and MAXQ for the problem displayed in Fig. 4

<table>
<thead>
<tr>
<th></th>
<th>Options</th>
<th>MAXQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations until convergence</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>Converged value [steps per episode]</td>
<td>69.1</td>
<td>85.2</td>
</tr>
<tr>
<td>Computation time for 100 iterations [seconds]</td>
<td>12.5</td>
<td>14.2</td>
</tr>
</tbody>
</table>

In terms of implementation, the lack of a termination predicate and the relatively flat method in which Options can be used, makes it a better fit for the problem being considered. The nature of the problem means that there is no apparent need for execution of the policies to take place in a hierarchical manner, which is the primary strength of using MAXQ. Therefore, it is concluded that Options will be used to tackle a more complex variant of the already analysed task, the results of which will be presented in the subsequent section.

IV. Scalability of Learning using Options

This section will describe the implementation and results for complex navigational problems, solved using both the Options HRL method as well as traditional flat Q-Learning. The aim of this section is to show the ability of the Options framework to solve complex navigation problems, as well as demonstrate how a hierarchical framework defined in the relative state is capable of accelerated learning as the size and complexity of the problem space increases. These results will be compared to the flat Q-learning to further accentuate the advantages and strengths of using HRL methods. Subsequently a sensitivity analysis as well as a verification of the presented results is also provided.

A. Options Simulation Results

This subsection presents results demonstrating learning on an indoor navigation problem using Options as the size of the problem scales up as well as the complexity of the problem scales up.

1. Learning as problem size scales up

The first problem being tackled is the agent’s ability to apply knowledge learnt on a small scale to a larger problem. To that end, a multiple room problem is considered, with one goal in each room. An episode is completed once an agent has visited all the available goals. A two room, four room and six room problem
as shown in Fig. 12 are considered, such that the transfer of learnt policies can be effectively compared and evaluated. To that end, all three scenarios are solved from scratch with no higher level policies being known to the agent. This is then compared to the agent’s performance when the two room policy is used to solve the four room case, and the four room policy is used to solve the six room case. The agent is given a maximum of 500 actions to complete an episode.

Figure 12. Maps used for the two room, four room and six room problems respectively. The agent always begins at S and must visit states G1 to G6 (fewer if the two or four room cases are considered) in any order to complete the episode. The goal states are randomly generated within a room every episode while the door states are always located at the same locations.

a. States, actions, options and rewards

The agent’s state is defined in the same way as described in Subsection 2 of Subsection A of Section III. The only addition is the agent’s ability to perceive a door state, thereby increasing the total number of states to 512 (4 * 4 * 4 * 4 * 2). Given the agent is unable to see diagonally ahead of it, without a unique door state, if an agent is facing a door it will perceive [0 0 0] which provides no indication that it is facing a door. In terms of the size of the problem space, the agent can be in one of 60, 146 or 213 unique locations, depending on whether the two room, four room or six room scenario is considered respectively. The actions, as mentioned in Subsection 2 of Subsection A of Section III, consist of the primitive actions of moving forward, turning right or turning left.

In solving this problem, the agent is free to choose from three different options. The first is a general navigation option that consists of single step primitive actions used to explore within a room. The second is a room exiting option, and is composed of multistep primitive actions and is used to move from one room to another. The third is an option to stay in a particular room and is composed of a 2-step primitive action and effectively changes the heading of the agent by 180°. Therefore, if an agent is facing a door state, it can actively choose not to leave the room by invoking this option. In terms of option selection, Eq. 2 is applied using an \(\epsilon\)-greedy policy. The \(\epsilon\) value starts at 0.1 and is decreased by a factor of 0.999, and is eventually set to 0. The agent is free to choose between the general navigation and room exiting options at all times, however can only invoke the staying in room option when it is faced with a door. This level of abstraction is valid as it does not limit the applicability of the learnt policies as the nature of the problem changes.

In terms of sub policies, the learnt policies are assumed to be near perfect. Therefore, the room exiting and staying in the room options are implemented in a greedy manner. The navigation option is trained such that if the greedy choice is made, the agent will always keep its current heading, unless near a wall in which case it will turn around. In training, it was also noted that the agent’s behaviour in corners is somewhat unpredictable and would benefit from better corner behaviour. The navigation option therefore has a constant \(\epsilon\) value of 0.25 such than on average one in four moves is random and leads to a change in direction. This \(\epsilon\) value is found to encourage the right amount of exploration, and is somewhat deterministic in its nature as it is set based on the size of the rooms. Should the rooms be larger, it might be advisable to lower the probability with which random actions are chosen as it could be beneficial to allow the agent to travel in straight lines for longer periods.

Due to the specific setup of the problem with a goal in each room, the rewarding strategy is designed to encourage a specific behaviour. This behaviour is that the agent should stay in a room and look for a goal, and only once a goal has been found should it move to the next room. Therefore, if an agent leaves a room without finding the target in that room, a negative reward of -50 is incurred. However if an agent visits the
goal state and then leaves the room, a reward of +50 is received upon entering the door state. Finally, a reward of +10 is received when an agent visits the goal state. It should be noted that these rewards can only be received once per episode per goal-door combination, and therefore repeatedly visiting the same door state does not result in repeated rewards.

b. Results

An initial attempt to solve the two room problem as depicted in Fig. 12 resulted in the learning curve seen in Fig. 13. As can be seen, the value function converges at approximately 350 episodes, which is what one might expect considering that the sub policies are perfectly trained, and therefore the agent essentially has to learn a policy for a Q matrix that is 512x3. The policy can be expected to converge as the agent must essentially learn that it should not leave a room until a target has first been found within said room. It is further interesting to assess how an agent might perform in a four room problem, when having to learn a high level policy from scratch, under the same conditions as the two room problem. The results of learning for a 4 room task can be seen in Fig. 14, with convergence occurring at approximately 400 episodes. The similarity in terms of convergence rate between the two and four room problem is not particularly surprising as in essence, the two room problem is identical to the four room problem in that the agent must determine when to stay in a room and search for a goal, and when to leave it. The only explicitly unexplored state for the four room problem is the transition through a door with a heading of $90^\circ$ and $270^\circ$. The other element that could require learning is understanding when to adopt the room exiting option, as the presence of multiple doors in a room (which is not experienced in the two room case), could lead to sub-optimality if the incorrect door is repeatedly exited from. In looking at these figures, as well as all other figures in this section, it should be noted that x axis scales are not always the same, and therefore even if two figures seem similar at first glance, they might not necessarily be that similar.

Figure 13. Reward received per episode for completing the two room task as seen in Fig. 12 using Options

Figure 14. Reward received per episode for completing the four room task as seen in Fig. 12 using Options with no prior knowledge of a high level policy

Having assessed the learning rate for the two and four room problems without any prior knowledge apart from the sub policies, it is now worth investigating how knowledge of the two room problem can be used to
accelerate learning for the four room problem. The hierarchical structure of the Options method, as well as the use of a feature based relative state is expected to translate to accelerated learning for larger problems, as indeed the strategy used to navigate a two room gridworld should be no different than that used to navigate a four room gridworld. Figure 15 shows the learning curve for the four room problem using the two room hierarchical policy. It is apparent that convergence is essentially immediate, with the exception of a few instances, which can be attributed to the learning required for the unvisited states. Therefore, it can be conclusively stated that for the presented problem setup, learning is accelerated for a four room problem when making use of prior knowledge from a similar problem.

Figure 15. Reward received per episode for completing the four room task as seen in Fig. 12 using Options using a high level policy learnt for a two room problem

In looking at the four room problem presented in Fig. 12 one might hypothesise that the four room problem contains within it all the states that could possible be visited in terms of navigating a potentially infinite number of rooms. Therefore, it is feasible to postulate that should the four room policy be applied to a larger six room problem (as depicted in Fig. 12), convergence should be immediate. Figure 16 confirms this expectation as immediate convergence is apparent. It should be noted that depending on the layout of the rooms, convergence may not always be this immediate as the agent could find itself in the wrong room more frequently, however in any event it is incontrovertible that learning will always be accelerated. In terms of states, the feature based relative state means that the problem space size can theoretically be infinitely increased without there being any negative impact on the agent’s ability to use prior knowledge to accelerate learning for a similar yet different problem.

Figure 16. Reward received per episode for completing the six room task as seen in Fig. 12 using Options using a high level policy learnt for a four room problem

Table 2 provides an overview of the performance for the two room, four room and six room problems in terms of number of iterations until convergence, the converged value and the the computation time. It should be noted that comparing two problems with different number of rooms is somewhat like comparing apples and oranges as the total available reward is different for the different scenarios, but also the size of the problem to be explored are different. Therefore, differences in values should be seen in perspective and not be used to conclude that performance for the four room scenario is for instance better than that of the two room case. What can be conclusively stated is that the converged value for the four room problem when
learning from scratch is the same as that obtained when using prior knowledge, and therefore there is no intrinsic sub optimality that is incurred when using prior knowledge. Further, in looking at the convergence rate for the six room problem, it is clear that beyond a certain extent, as the problem size increases, learning from scratch becomes exponentially more difficult. Therefore, as the size of the problem gets bigger, the advantages of using prior knowledge becomes all the significant. This is apparent in the fact that training for a six problem from scratch requires over 1000 episodes, whereas using prior knowledge requires approximately 350 episodes trained on a smaller map therefore also requiring less computation time.

<table>
<thead>
<tr>
<th></th>
<th>Iterations until convergence</th>
<th>Converged value [reward per episode]</th>
<th>Computation time [seconds per 100 iterations]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Rooms</td>
<td>350</td>
<td>18.0</td>
<td>3.3</td>
</tr>
<tr>
<td>4 Rooms no prior knowledge</td>
<td>400</td>
<td>50.7</td>
<td>7.4</td>
</tr>
<tr>
<td>4 rooms with prior knowledge</td>
<td>50</td>
<td>49.2</td>
<td>6.9</td>
</tr>
<tr>
<td>6 rooms no prior knowledge</td>
<td>1100</td>
<td>54.8</td>
<td>18.6</td>
</tr>
<tr>
<td>6 rooms with prior knowledge</td>
<td>0</td>
<td>41.1</td>
<td>10.5</td>
</tr>
</tbody>
</table>

2. Learning as problem complexity scales up

This section aims to evaluate the agent’s ability to learn based on prior knowledge as the complexity of the problem increases. Therefore, rather than evaluating how the size of the problem affects learning, this section will determine the impact of adding additional elements to the agent’s primary task. The problem being tackled is similar to the aforementioned four room gridworld problem, with the addition of a battery element, wherein the agent must visit a charging station location periodically to ensure that it has enough battery to complete the mission of visiting all four goals. The goals are still randomly located within each room, however the locations of the charging stations are fixed. Figure 17 depicts the problem to be solved. The map is the same as seen in the previous section, however the charging stations, denoted by P1 to P4 are added to the map. These charging stations are always located at the same place and the agent is given a maximum of 500 actions to complete an episode.

Figure 17. Map used for the four room problem with battery element. The agent always begins at S and must visit states G1 to G4 in any order to complete the episode. The goal states are randomly generated within a room every episode while the door states are always located at the same locations. If the agent’s battery level is low, visiting states P1 to P4 replenishes the agent’s battery level.
In order to evaluate how learning is affected by an increase in complexity, the problem presented in Fig. 17 will be solved for two initial settings. The first will be a case where the agent is provided enough power such that it is able to complete an episode without needing to visit a charging station. The policy learnt here will then be applied to a scenario where the agent has a lower battery level and must therefore learn to visit a charging station when in a low battery state.

a. States, actions, options and rewards

The agent’s state is defined in the same way as described in the previous subsection. In addition, the agent is able to perceive the charging stations $P_1$ to $P_4$ as unique states. These charging stations are always located at the same point so as to distinguish looking for the charging station from performing a general navigation task of looking for a goal state. Furthermore, in order to ensure that decision making is Markovian, the agent is also provided with knowledge of its battery level. The agent starts each episode with its maximum power level of 150, and every step results in the agent losing 1 point of its battery level. If the battery level is greater than 2/5 of its total battery level, a 2 is perceived and if it is lower then a 1 is perceived. If the agent visits a charging station, its power level is fully replenished however it should be noted that the agent can only recharge if it is in a lower power level. This is to avoid the agent randomly recharging while looking for a goal, and to rather make the charging more similar to a conscious decision. Two states for the battery level were deemed to be sufficient to provide enough information to the agent for the decision making process while containing the total number of states for this scenario to 2000.

In this scenario, the agent is free to choose from 5 different options. The general navigation option, room exiting option and staying in the room option are the same as described in Subsection 1 of Section IV. A fourth option for recharging and finally a fifth option wherein the agent chooses the greedy choice for the general navigation option are also added. The fifth option is added to optimise behaviour for cases such as an agent being adjacent to a goal state or in a door state. The idea is that at such instances, the agent will choose the greedy action with a 100% probability. While this does add a burden to the learning task, the expectation is that the learnt policy will be more optimal. Therefore, if selected, both the newly added options are executed in a greedy manner. In terms of abstraction of the available choices, the agent is always able to choose the general navigation option, the greedy navigation option and the room exiting option. The staying in room option is only selected when faced with a door state and the battery option can only be selected when the agent is in its lower battery state. While perhaps limiting the agent’s learning ability, the assumption that the agent only needs to recharge in its low battery state is a valid one and allows for accelerated learning without a compromise on broader applicability of the learnt policies.

The rewarding strategy for this problem is as described in Subsection 1 of Section IV except for the fact that running out of battery results in the episode being terminated and the agent incurring a reward of -500. This high negative reward is used to ensure that the agent learns that it should never run out of battery, and should prioritise moving towards a charging station when in a low battery state.

b. Results

The problem presented in Fig. 17 is initially solved for a case where the agent has for all intents and purposes an “infinite” battery level. Therefore, the agent must only navigate a room and find a goal, and then proceed to the next room, without having to pay any attention to the battery level. Therefore, the problem here is comparable to the four room problem depicted in Fig. 12 with the exception that this problem has twice as many active states. Figure 18 shows the learning curve for this problem, and it can be concluded the convergence occurs after approximately 600 episodes. Since the agent never runs out of battery, the maximum negative reward is never lower than -500. In looking at this figures, as well as all other figures in this section, it should be noted that x axis scales are not always the same, and therefore even if two figures seem similar at first glance, they might not necessarily be that similar.

Figure 19 shows learning where a finite battery level is provided to the agent. As previously mentioned, the agent has a maximum battery level of 150, and every action results in a decrease by 1 point of the total power level. If the agent is in the lower 2/5 of the total power, it is then able to recharge back to its maximum power level by visiting the charging station. Convergence in this case occurs at approximately 170 episodes. The initial runs have a total reward lower than -500 indicating that the agent runs out of battery. Within 25 runs the agent appears to learn a near optimal policy in terms of recharging. Although surprising that the agent subsequently requires over 100 runs to “re-learn” the basic goal finding and room exiting aspects
of the mission, this can be explained by the fact that the previously learnt policy deteriorates due to the huge negative reward from the agent running out of battery.

In order to make conclusive remarks about an agent’s ability to use prior knowledge as complexity scales up, it is integral to evaluate how an agent might solve the problem described in Fig. 17 without any prior high level knowledge. Figure 20 shows learning for the four room problem with a maximum battery level of 150. Convergence in this case takes place after approximately 1000 episodes, which is significantly higher than the 170 episodes seen in the case where prior knowledge is used. Even considering the combined number of episodes for the case where prior knowledge is required, the making use of past knowledge still leads to accelerated learning in terms of episodes until convergence. Training for sub problems is a significant strength of hierarchical learning, and as is demonstrated here, is not limited to just the sub policies but can also have a positive impact on learning of the global hierarchical policy.

Figure 18. Reward received per episode for completing the four room task as seen in Fig. 17 with an “infinite” battery level using Options with no prior high level knowledge

Figure 19. Reward received per episode for completing the four room task as seen in Fig. 17 with a finite battery level using Options using prior high level knowledge from a case where the battery level is “infinite”.

Figure 20. Reward received per episode for completing the four room task as seen in Fig. 17 with a finite battery level using Options with no prior high level knowledge
In order to lend further credibility to the above findings, Table 3 provides other performance parameters. The data reiterates the fact that in terms of episodes required for convergence, the use of prior knowledge helps accelerate learning. In terms of computation time, it should be noted that this is the average time for 100 episodes, and not the computational time for convergence. Therefore, even though the computational time for 100 episodes is greater for the combined scenario where prior knowledge is required, this is not an indication of the fact that convergence takes longer for this approach. In terms of converged value, it is clear that using past knowledge leads to a significantly higher reward than when learning from scratch. Although this could be considered an anomaly, considering that this value is an average of 250 data points leads one to conclude that this is the true converged value. The assumption is that if the method without prior knowledge is allowed to train for more episodes, it too will converge to the same high value as the case where prior knowledge is used. This however has not been verified and therefore cannot be stated with certainty.

Table 3. Performance in terms of convergence rate, converged value and computation time for the four rooms with battery problem, evaluating the usage of past knowledge to accelerate learning

<table>
<thead>
<tr>
<th></th>
<th>4 Rooms with high battery level</th>
<th>4 Rooms with low battery using prior knowledge</th>
<th>4 Rooms with low battery and no prior knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations until convergence</td>
<td>600</td>
<td>170</td>
<td>1000</td>
</tr>
<tr>
<td>Converged value [reward per episode]</td>
<td>41.1</td>
<td>71.4</td>
<td>48.0</td>
</tr>
<tr>
<td>Computation time [seconds per 100 iterations]</td>
<td>10.9</td>
<td>6.5</td>
<td>14.2</td>
</tr>
</tbody>
</table>

B. Flat RL Simulation Results

Having demonstrated the effectiveness of the Options HRL method at solving complex navigation problems, as depicted in Fig. 17, it is important to assess how conventional flat Q-learning performs for the same task. To begin with, a two room version of Fig. 17 is considered, wherein the agent must go through one door and find two goals. The agent is initially provided with a battery level of 150, which means that it has more than enough power to complete an episode without needing to visit a charging station. As can be seen in the upper figure of Fig. 21, at approximately 250 episodes some form of convergence is reached, however it must be noted that there are still far too many outliers to categorise this as true convergence. In any event, one could conclude that a two room problem where the battery level is not a critical element can be solved to some extent by traditional Q-learning. For comparative purposes, the lower figure in Fig. 21 shows the same problem solved using options, except for the fact that the battery element is not ignored. As can be seen, learning takes place significantly faster than in the flat case, and the converged result is far more robust with fewer outliers.

In order to evaluate how the agent behaves and learns when the battery plays an active role in the mission, and the agent cannot complete a majority of episodes without recharging, the maximum battery level is halved to 75. Figure 22 demonstrates that when the battery plays an active role, the agent is unable to effectively learn and no convergence is seen. A part of the problem could be related to agent’s ability to find the charging station even if it realises that it should visit the charging station. When Options with sub-policies are used, entering the charging sub policy results in a very high probability of reaching the charging station, however since all decision making in the flat method is done on the same level and there is no prior knowledge afforded to the agent, wanting to recharge does not translate to being able to recharge. This sometimes results in the agent hovering around the charging point and not exploring for fear of running out of battery. The scale difference between the two figures in Fig. 22 should be noted, as one runs for 500 episodes and the other for 2000.
Although the agent’s inability to solve a two room problem using flat Q-learning provides a clear indication of the abilities of flat RL methods for such a problem, in order to compare like for like, the problem as described in Fig. 17 is solved in its entirety using Q-learning. The agent is given a maximum of 1000 steps to complete an episode, as opposed to 500 for all other scenarios, so that it has a higher chance of finishing an episode and learning faster. The $\epsilon$ value is initially set to 0.1 and is decreased by a factor of 0.999 until a value of 0.05 is reached, at which point the $\epsilon$ value is kept constant. As can be seen in Fig. 23 even after 10000 episodes, although behaviour does seem to improve, there is absolutely no indication of convergence. Therefore, it can be concluded the flat RL is not slow at solving the complex navigation problem being considered here, but is in fact incapable of learning to solve it.
C. Sensitivity Analysis

The designer plays an important role in terms of whether learning takes place or not, and further how fast learning takes place. In order to understand the effect of design decisions on learning, a sensitivity analysis is carried out on the problem depicted in Fig. 17 wherein certain parameters are varied and learning performance is evaluated.

As previously mentioned, the learning method implemented in this paper makes use of abstraction in terms the availability of options. Therefore, not all options are available to agent at all states, thereby accelerating the learning process. Figure 24 shows the learning curve for a scenario where the agent is always allowed to choose from all five options. As can be seen, convergence seems to occur after approximately 1500 episodes. When comparing this to the learning curve of the same problem with abstraction as seen in Fig. 20, it is clear that the design decisions help significantly increase the learning rate. In effect, when the agent has fewer options to evaluate for a given state, the number of episodes until convergence are less by about 33%.

Figure 23. Reward received per episode for completing the four room task as seen in Fig. 17 using flat Q-learning with a decreasing \( \epsilon \) value starting and 0.1 and decreasing until 0.05

Figure 24. Reward received per episode for completing the four room task as seen in Fig. 17 with a finite battery level with all options available in all states

The options for staying in a room and looking for the charging station are defined as multistep actions, and therefore once the option is invoked, a series of actions are carried out before option selection is re-evaluated. Needless to say, this has an effect on learning. In order to evaluate the effect of this design parameter, all multistep options are reduced to two steps, so as to make them highly representative of primitive actions. This case still makes use of abstraction in that not all options are available at all times. Figure 25 shows the learning curve for this case. Convergence occurs after approximately 1700 episodes, however it must be noted that even after convergence outliers and poor episodes are still extremely frequent.
Figure 25. Reward received per episode for completing the four room task as seen in Fig. 17 with a finite battery level with all multistep options consisting of two primitive actions

Another important element that is known to influence the learning rate is the initial epsilon value as well as the rate at which epsilon is decreased. Table 4 shows the number of iterations until convergence, the converged value and the computation time for six combinations of epsilon values and decay rates. The $\epsilon$ value is updated from one episode to another and is computed as $\epsilon_{\text{current}} \cdot$ decay rate, where $\epsilon_{\text{current}}$ for the first episode is the initial $\epsilon$ value. An initial $\epsilon$ value of 0.1 with a decay rate of 0.999 is what is used in the previously presented results, with the benchmark for comparison being shown in Fig. 20 and in the first column of Table 4. As can be seen, the previously used setting is not optimal, and therefore the above presented learning curves do not represent the fastest possible learning. However, it should be noted that the presented data in Table 4 only tells a part of the story, as even though the numbers look very promising, the learning curve as seen in Fig. 26 demonstrates that even after convergence there are several outliers. That being said, it is certainly true that there are still design possibilities for accelerating learning with respect to what has been previously demonstrated.

Table 4. Performance comparison when solving the four room with battery problem for varying initial epsilon values as well as varying decay rates

<table>
<thead>
<tr>
<th></th>
<th>Initial:0.1</th>
<th>Initial:0.1</th>
<th>Initial:0.1</th>
<th>Initial:0.05</th>
<th>Initial:0.05</th>
<th>Initial:0.05</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Decay:0.999</td>
<td>Decay:0.9975</td>
<td>Decay:0.995</td>
<td>Decay:0.999</td>
<td>Decay:0.9975</td>
<td>Decay:0.995</td>
</tr>
<tr>
<td>Iterations until convergence</td>
<td>1000</td>
<td>650</td>
<td>450</td>
<td>900</td>
<td>450</td>
<td>550</td>
</tr>
<tr>
<td>Converged value [reward per episode]</td>
<td>48.0</td>
<td>42.5</td>
<td>10.6</td>
<td>56.3</td>
<td>73.6</td>
<td>34.7</td>
</tr>
<tr>
<td>Computation time [seconds per 100 iterations]</td>
<td>14.2</td>
<td>14.2</td>
<td>15.2</td>
<td>14.6</td>
<td>12.8</td>
<td>14.1</td>
</tr>
</tbody>
</table>

Figure 26. Reward received per episode for completing the four room task as seen in Fig. 17 with an initial Epsilon value of 0.1 and a decay rate of 0.995
These findings relating to the effect of the $\epsilon$ value naturally raise questions about the validity of the other conclusions drawn concerning the use of prior knowledge. In order to address this issue, learning is demonstrated on the two room with battery case using an initial $\epsilon$ value of 0.1 with a decay rate of 0.9975. Figure 27 as well as Table 5 demonstrate that using the new $\epsilon$ setting results in faster learning of the two room case as well, thereby allowing one to conclude that all learning is similarly accelerated and thus validating the previously presented results. The learning in Fig. 27 can be compared to the learning in the lower figure of Fig. 21 as both relate the same problem but with different $\epsilon$ settings.

![Figure 27. Reward received per episode for completing the two room variation of task as seen in Fig. 17 with an initial Epsilon value of 0.1 and a decay rate of 0.9975](image)

| Table 5. Performance comparison when solving the two room with battery problem for varying epsilon decay rates |
|--------------------------------------------------|--------------------------------------------------|
| Initial:0.1 & Decay:0.999 | Initial:0.1 & Decay:0.9975 |
| Iterations until convergence | 240 | 90 |
| Converged value [reward per episode] | 18.8 | 30.2 |
| Computation time [seconds per 100 iterations] | 7.5 | 6.7 |

D. Verification of Results

Given that the methods presented in this paper as well as the specific problems being considered are unique and novel, there is no model problem that can be solved to verify and validate the obtained results. That being said, certain measures were taken to ensure that obtained results are both accurate and reliable.

Firstly, significant visual verification was carried out to ensure that the behaviour of the agent was as expected. On a low level, RL is based on states, actions and rewards. Therefore, it was verified whether for a given location, the state that the agent observed was indeed accurate, whether a chosen action resulted in an appropriate next state and whether the reward received for performing an action and entering a new state was as expected. In terms of action selection, once a policy was derived, a greedy selection was adopted, and it was verified whether for a given state, the best action was selected. All these elements were found to be as expected and therefore in terms of flat RL and sub policy learning the derived results can be trusted.

Secondly, in terms of option selection, the fundamentals of the algorithm and process are the same as selecting primitive actions for flat RL. The only exception is for cases where abstraction is used and not all options are available to the agent in all states. For these states, it was verified that the Q value corresponding to an option that cannot be selected was 0 (the initial value). With this element also accounted for, from a theoretical perspective it can be concluded that the implementation is robust for both flat and hierarchical RL.
Figure 28. Reward received per episode for completing the four room task as seen in Fig. 17 using a hand crafted deterministic high level option policy

Having concluded that action/option selection, rewarding and state updating take place as desired, the last remaining element to verify is the learning process. Although already demonstrated by the fact that initial rewards per episode are consistently worse than those obtained after episodes of training, a more explicit verification method is demonstrated here. In effect, the agent is deterministically told to enter a navigation policy when looking for a target, to use the room exiting option once a target has been found, to invoke the staying in room option when faced with a door without having found the target in said room, to enter the charging mode when the battery level is low and to use the greedy navigation policy when in a door state or when near a goal. Figure 28 shows the reward per episode when using a hand crafted high level policy, with the average reward per episode being approximately 38. For comparison purposes, the converged reward when solving this problem without any prior high level knowledge is 48. Therefore, it can be concluded that the obtained results are in line with what one might expect, and further that the agent is able to learn a more optimal policy through HRL than can be easily deterministically coded by a designer.

V. Conclusion

The primary aim of this paper was to demonstrate a method to efficiently solve complex navigational problems, with a focus on indoor navigation and locating points of interest. To that end, this paper effectively evaluated the Reinforcement Learning (RL) methods of Options, MAXQ and flat Q-learning on a simple navigation problem so as to understand in more detail the strengths and weaknesses of said methods. Subsequently, the Options and flat Q-learning methods were implemented on larger and more complex problems. This paper effectively demonstrated the strengths of Hierarchical Reinforcement Learning (HRL) methods over flat RL and further successfully solved a problem using HRL that could not be solved using traditional flat methods.

In evaluating the MAXQ and Options methods on a simple goal finding and room exiting problem, it was determined that while both approaches showed promise, the Options method would be used in tackling the more complex problems. This was based on the quantitative performance of the two methods in terms of number of iterations until convergence, value of the converged policy and computation time. In addition, the implementation characteristics were also considered in deciding which method to pursue in more detail. At this stage already, the gross limitations of flat RL became apparent.

In solving the more complex navigational problems a focus was placed on the ability to reuse prior knowledge as a) the size of the problem increases and b) the complexity of the problem increases. This paper effectively demonstrated that training on a two room problem led to significant gains in learning time when the learnt policy was applied to a larger four or six room problem, as opposed to training for a four room problem from scratch. Furthermore, the ability to use prior high level knowledge about a task and apply it to a similar problem with an additional element in terms of complexity was also demonstrated. However, beyond just showing the ability to transfer low and high level policies, it was also demonstrated that this transfer of learning leads to accelerated learning. In specific terms, this was demonstrated by comparing learning for a problem with four rooms and a battery element with learning when a policy for a navigational problem where the battery element was irrelevant was used on a problem where the recharging played an active role in the mission. The flat RL approach was entirely ineffective at solving the most complex version of this problem, and no learning was apparent.
A sensitivity analysis was also carried out in order to examine the influence the designer exerts on learning. Three aspects, namely the availability of options, the number of steps within an option and the epsilon value were evaluated. In all three studies, convergence took place regardless of the parameters, however there was undeniably a significant impact on the rate of convergence as well as the value of the converged function. This forces one to be critical of the results, as indeed attempts should be made to ensure that learning takes place as fast as possible, yet designer imposed restrictions should not in any way limit the functionality of the learning approach should it be applied to a different problem.

RL in and of itself is arguably the most fundamental form of learning, and is predicated on a very simple relationship between actions and rewards. Therefore, it is understood that the potential of such a learning method is essentially limitless and can be applied to several different problems. However, the curse of dimensionality as well as the agent’s ability to effectively learn as the problems become increasingly complex has often raised doubts about the practical applications of RL in real world problems. The findings of this paper, demonstrating the use of HRL to solve a complex navigation problem, not only provide an alternative approach to performing autonomous navigation, but further display the strength of RL in solving complex problems. These findings, along with the extensive research done by others in the field, lend great confidence to the effectiveness and applicability of RL, even when dealing with complex problems, and provide sufficient indications to suggest that further research in this field will be highly rewarding.

VI. Recommendations for Future Work

Despite presenting several promising results, there are still limitations to the findings and therefore a need for further research. A primary limitation of the derived approach is in the problem setup, in that every room has a goal within it, and therefore an agent does not need to explore between rooms when looking for a target. In real applications, there is no guarantee of finding a target in every room, and therefore the agent does not have an explicit trigger for when it should leave a room. Hence, there is a need to explore in more detail how an agent might for instance solve a navigation problem wherein a goal state randomly appears in only one or two of four rooms.

In keeping with the point of practical applications, the ability to avoid collisions with walls and obstacles is an important one. There is extensive evidence in literature to suggest that implementing obstacles in the current problem structure and adding an option to go around obstacles should not prevent convergence (Parr and Russell [1998] and Feng et al. [2015]), but only delay it. When facing walls or obstacles however, the presence of randomness in actions could lead to the agent colliding with an obstacle even when it knows it should not. Therefore, some sub policies should make use of softmax action selection methods so as to ensure that the worst action is never (or very rarely) chosen. This differs from an $\epsilon$-greedy action selection where when the non greedy choice is made, all actions have the same probability of being chosen, regardless of how good or bad they might be.

When executing the general navigation option, the agent can choose between a greedy option or an option with some randomness to encourage exploration. The greedy option is for the case where the agent is in a doorway or adjacent to a goal state. The effect of this additional option on learning time is not entirely clear, however it would be worth exploring the use of stochastic action selection for the navigation sub policy. This means that based on its state, the agent should learn for itself a specific epsilon value so as to determine how greedy its actions should be. While this might not necessarily lead to faster learning, it will demonstrate another element of learning and should lead to greater optimality.

All learning in the presented problem is done in a gridworld setup using discrete states. One of the strengths of HRL lies in its ability to execute sub policies in the continuous domain, thereby decreasing computational efforts. Future work should aim to incorporate a combination of discrete and continuous options such that performance, learning rate and computational time can all be improved.

Finally, future work should involve deploying the derived approach on an actual platform so that the findings can be validated in more than a theoretical framework. That would lend further credibility to the results in this paper and would further efforts for HRL to be more prevalently used in complex navigation problems.
Bibliography


Part II
Chapter 1

State of the art / Literature review

The earliest references of Reinforcement Learning (RL) can most likely be traced back to Minsky, who described a reinforcement process as "one in which some aspects of the behavior of the system are caused to become more (or less) prominent in the future as a consequence of a reinforcement operator" (Minsky [1961]). The reinforcement operator in this case fulfils the role of a reward, thereby encouraging the learner to take some actions over others. However, Minsky also noted that the real challenge arose in complex systems where credit had to be distributed between the various actions when several decisions are involved (Minsky [1961]). The system being considered for this thesis can be categorised as complex, and is therefore a challenging case where RL cannot be applied very easily.

This section aims to demonstrate the current understanding of flat RL methods, evaluate why there is a need for development in HRL and further present the necessary information to implement an HRL method to solve the problem being considered. To that end, this section will begin with a description of the fundamentals of RL in general and describe the governing principles. Subsequently, insight into flat RL methods, namely Dynamic Programming, Monte Carlo and Temporal Difference, will be provided so as to understand how these methods work. This leads to highlighting the limitations of flat RL methods, with a focus on the curse of dimensionality. Subsequently, HRL methods will be presented, with a focus on Options (Mcgovern et al. [1998]), HAMs (Parr and Russell [1998]) and MAXQ (Dietterich [2000]). Combinations and variations of these fundamental methods will also be reviewed. Finally, the section is concluded with an analysis, explicitly linking the literature review to the goals of this thesis.

1.1 Fundamentals of Reinforcement Learning

Before understanding the differences between flat RL and HRL and evaluating them, the natural starting point is to understand the fundamental principles and concepts that govern RL as a whole. This subsection aims to describe the fundamentals and relate them to the task at hand.

1.1.1 Key elements- states, actions and rewards

RL as it is now known is composed of three elements, namely states, actions and rewards. In the context of a quadcopter, the state could refer to its attitude, the rotational velocity of its different rotors, its velocity and possibly its location. The actions would be to increase or decrease the rotational speed of each of the rotors such that a new (next) state is achieved. The concept of rewarding is something that needs to be evaluated in more detail, however on a simple level could for instance consist of assigning a reward every time the quadcopter locates an obstacle on the ground. These three elements come together to form the concept of a policy, which is defined as "a mapping from states to actions" (Watkins [1989]). As per Watkins [1989] "the aim of the learner will be to construct a policy that is optimal in the sense that, starting from any state,
following the policy yields the maximum possible expected return that can be achieved starting from that state”. Ultimately, the goal of all RL is to determine a policy that results in the optimal value function.

1.1.2 Markov Decision Process and Semi Markov Decision Process

The framework that governs most reinforcement learning problems, including the one being considered here, is known as the Markov Decision Process (MDP). MDPs provide a framework where the outcome of a decision is partly random and partly under the control of the decision maker (Puterman [1998]). As per this framework, a learning agent interacts with its environment at discrete time steps and at every step "perceives the state of its environment, \( s_t \in S \) and on that basis chooses a primitive action \( a_t \in A \)" (Mcgovern et al. [1998]). Once an action is taken, at the next time step the system produces a numerical reward \( r_{t+1} \) and a next state \( s_{t+1} \). The one step dynamics of the environment is what enables the prediction of the next state of the system \( s' \) and its expected reward \( r \). The transition probability \( P \) as well as the one-step expected rewards \( r \) can be computed as seen in Eqs 1.1 and 1.2 (Mcgovern et al. [1998]) respectively. Working within the MDP framework is extremely useful as it implies that all information that is needed to decided what action to take is "stored" within the current state of the system and therefore knowing about the system’s past is not required.

\[
P^a_{ss'} = P\{s_{t+1} = s' \mid s_t = s, a_t = a\} \quad (1.1)
\]

\[
r^a_s = E\{r_{t+1} \mid s_t = s, a_t = a\} \quad (1.2)
\]

Therefore, effectively, an MDP is a tuple composed of a set of states, \( S \), an action set \( A \), transition probabilities \( P^a_{ss'} \), and an immediate reward \( R^a_s \) which is received upon executing an action \( a \) in state \( s \). The goal of RL, as enabled by the MDP framework, is to determine a policy \( \pi \) which determines the action to be taken for any given state. An optimal policy, \( \pi^* \) is one where actions are taken such that the expected reward is maximised, and depending on parameters, can either maximise short term or long term rewards.

While flat RL problems fit within the MDP framework, HRL makes use of a generalisation of the MDP known as Semi-Markov Decision Processes (SMDPs). The primary difference between the two is that an action in SMDPs can last multiple time steps (Ponsen et al. [2006]). In other words, while a flat RL approach consists only of primitive actions (executed over one time step), an SMDP consists of actions that are either primitive or temporally extended. This concept will become clearer when HRL approaches are discussed in Section 2.4. SMDPs are also constituted of a tuple, however the transition probabilities and reward function are dependent on a random variable \( \tau \), which denotes the number of time steps required to carry out an action \( a \) (Gao [2015]).

1.1.3 Balancing Exploration-Exploitation

A critical trade-off encountered by the agent in terms of learning is finding an optimality between exploration and exploitation. In effect, based on previous experience (trials), the agent may already have an estimate of the state-action function, \( Q^\pi_{s,a} \) and therefore must decide whether to build on the existing knowledge, or follow a previously untested action that could lead to greater rewards. If the agent decides to exploit the knowledge it already has so as to improve the state-value action, it is referred to as the greedy choice. On the other hand, if the agent decides to explore so as to find a better optimal function, it is referred to as a nongreedy action. In the short run, exploitation leads to better rewards, however in the long run making use of some exploration eventually leads to better rewards.

A greedy action is one where 100% exploitation takes place, and therefore no exploration occurs. This could result in a sub optimal result being found. Non greedy policies include \( \epsilon \)-greedy and Softmax action selection methods (Sutton and Barto [1998]). An \( \epsilon \) - greedy policy simply means that the non greedy action is chosen with a probability of \( \epsilon \) and the greedy action is chosen with a \( 1 - \epsilon \) probability. Figure 1.1 shows the effect of using a greedy policy (\( \epsilon = 0 \)) as well as an \( \epsilon \)-greedy policy with \( \epsilon = 0.1 \) and \( \epsilon = 0.01 \), as applied to the 10-armed bandit problem. What is noteworthy is that the greedy policy has the worst performance and
reaches a sub-optimal solution rapidly. Generally speaking, starting off with more exploration and decreasing it as more runs are performed should lead to the best results. Not only will an optimal value function be reached, but it will be reached faster than if \( \epsilon \) is constantly very small. This is also supported by Fig. 1.1.

A shortcoming of the above described \( \epsilon \)-greedy action selection method is that "it is as likely to choose the worst-appearing action as it is the next-to-best action" [Sutton and Barto 1998]. To overcome this problem, the action probabilities should be varied as a graded function of their estimated value. Therefore, when making a non-greedy choice, the agent is informed to some extent of the effect of its decision. Ranking and weighting actions according to their value estimate in such a way is referred to as softmax action selection and is advantageous for tasks where the worst actions could have very bad consequences. In the case of navigation for a quadcopter, in general it is better to avoid the worst options since there is little to be gained from them.

In terms of implementation, \( \epsilon \)-greedy is often considered simpler as the temperature, \( \tau \), parameter in softmax action selection requires information about the likely action values. As per [Sutton and Barto 1998], there are no "careful comparative studies of these two action-selection rules". However, research conducted by [Tokic and Palm 2011] comparing \( \epsilon \)-greedy and softmax action selection methods concluded that decisions should be made by an agent by balancing exploration and exploitation based on changes in the value functions—thereby endorsing softmax action selection. In any event, it is unclear what form of exploration-exploitation should be used for the task at hand, thereby creating an opening for further research during the thesis.

### 1.1.4 Delayed Rewards

Another conflict within reinforcement learning is finding a balance between immediate rewards and rewards received in the future—delayed rewards. In many ways this relates closely to the aforementioned issue of exploration versus exploitation. Formally, this leads to a distinction between what is called rewards and returns. Rewards are indeed the value gain or loss associated with achieving or failing to achieve a goal for each episode, time-step or however rewards may be allocated. However, what is more interesting for the agent is to be able to maximise the expected return over a period of time, rather than for instance at every time step or after each episode. On the most basic level this is described by Eq. 1.3 [Sutton and Barto 1998], where \( r_t \) is the expected return after time step \( t \), \( R \) is the reward per game and \( T \) is a final time step. For an episodic task like Blackjack, where each game ends and a reward is received, this approach makes sense.

\[
r_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + r_T
\] (1.3)

Another important concept in terms of returns is that of discounting. In effect, using a discount rate determines the present value of future rewards and is characterised by Eq. 1.4 [Sutton and Barto 1998]. Here, \( r_t \) is the expected discounted return over time step \( t \), \( r \) is the reward per game, \( \gamma \) is the discount rate. This equation translates to the fact that a reward received \( k \) time steps in the future is worth only \( \gamma^{k-1} \) of what it would be worth if received immediately. The discount rate is a value between 0 and 1 and
when set it 0, implies that the learner only cares about immediate rewards. In the case of the navigational application, this means that the quadcopter will focus more on the completion of some final mission and consistently delivering results in the long run, rather than only focusing on short term rewards that may not necessarily lead to optimal completion of the entire mission.

\[ R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} = \gamma^t r_{t+k+1} \] (1.4)

1.2 Predominant Flat Reinforcement Learning Approaches

Having understood the key elements involved in RL as well as a key conflict to address, it is now possible to look in some detail at the different types of flat RL methods that have already been developed and that could potentially be applied to navigational tasks.

1.2.1 Dynamic Programming

Dynamic programming based methods can be characterised as model based methods as they "require a model of the transition probabilities" as seen in Eqs. 1.1 and 1.2 so as to determine the value function (Kober et al. 2013). The model does not have to be entirely preprogrammed, but can be learned through data, through methods such as policy iteration and value iteration (Kober and Peters 2014). The primary difference between the two is that the former only updates the policy once the policy evaluation has converged. The latter on the other hand combines evaluation and improvement of the policy by updating the value function once a state is updated. The primary drawback of dynamic programming with respect to the task at hand is that it requires a perfect model of the environment as an MDP so as to compute the optimal policy (Sutton and Barto 1998). As was already established, the system being considered here is complex, and therefore producing an accurate MDP model will prove to be extremely challenging.

1.2.2 Monte Carlo Methods

Monte Carlo methods are based on the idea of repeatedly sampling the states and corresponding actions so as to obtain an estimate of the value function. Since an explicit transition function as seen in Eqs. 1.1 and 1.2 is not required, Monte Carlo methods can be described as model-free. Monte Carlo methods learn only through experience, and therefore by simulating enough runs, a good estimate of the action-value function and state-value function can be obtained. In effect, Monte Carlo methods work by averaging the reward obtained for an action, every time a particular state is visited. The update is done entirely on past experience and not through any computation. In evaluating the obtained state-action values, there are several algorithms that can be considered, two of which can be seen in Eqs. 1.5 and 1.6 (Sutton and Barto 1998). Equation 1.5 shows \( Q_t(s, a) \), the Q value of action \( a \) given state \( s \) at time \( t \). The direct averaging method shown here is perhaps the most simple and intuitive way to determine the value of an action. In effect, the mean reward of an action is estimated simply by averaging the rewards actually received when the action was selected for a particular state, \( r_1 + r_2 + \ldots + r_{k_a} \), divided by the number of times the state was visited (\( k_a \)).

\[ Q_t(s, a) = \frac{r_1 + r_2 + r_3 \ldots + r_{k_a}}{k_a} \] (1.5)

A downside of the method shown in Eq. 1.5 is that "the memory and computational requirements grow over time and without bound" (Sutton and Barto 1998). This is due to the fact that every additional reward after a run needs to be stored and must be considered during the following computation of the state-action function. Needless to say, this adds an unnecessary burden to the processor. In order to circumvent this issue, an incremental update formula is used, such that averages are small and constant computation is required to process each new reward. This is represented by Eq. 1.6 where \( Q_{k+1}(s, a) \) is the most recently updated \( Q(s, a) \) matrix and \( Q_k(s, a) \) denotes the average of the first \( k \) rewards. Using this average along
with the $k + 1$ reward (reward of the just completed game), $r_{k+1}$, the average of all $k + 1$ rewards can be computed easily.

$$Q_{k+1}(s,a) = \frac{1}{k+1} \sum_{i=1}^{k+1} r_i = Q_k(s,a) + \frac{1}{k+1} [r_{k+1} - Q_k(s,a)] \quad (1.6)$$

1.2.3 Temporal Difference Methods

Temporal difference methods can in some ways be considered a combination of Monte Carlo and Dynamic Programming. While they are based on sampling, unlike Monte Carlo methods, the value function is updated at the end of every time step using temporal errors and not at the end of each episode (Kober et al. [2013]). The temporal difference (or error), is the difference between the old estimate of the value function and new estimate, after accounting for the reward received during the current sample. Contrary to Dynamic Programming, the updates of the value functions are done only considering the sampled successor states and not the entire probabilistic distribution for the successor states. Therefore, like with Monte Carlo methods, Eqs. 1.1 and 1.2 are not required, thereby making Temporal Difference methods, model free methods. Two common Temporal Difference algorithms are Q learning and SARSA.

Q learning, as seen in Eq. 1.7 (Sutton and Barto [1998]) is a Temporal Difference method that approximates the optimal state-action function ($Q^*$) on each time step. It is an off policy in that learning takes place using an arbitrary policy. It is a good method for approximating a navigation strategy because it allows learning to take place during the run. A basic requirement here is that all state-action pairs are sampled often enough and therefore exploration is key. The amount of exploration is critical and often plays a very important role in convergence rates. Given that all state action pairs are continually updated, the state value function is guaranteed to converge at the optimal state-value function with a probability of 1 (Sutton and Barto [1998]). The parameter $\alpha$ in Eq. 1.7 is referred to as the step size parameter and determines the step size on each time step, thereby influencing the learning rate. This value is between 0 and 1, with a typical value being around 0.1. $\gamma$ is the discount rate and as previously explained relates to optimising for expected returns in the long run rather than short term rewards. As can be seen, the state-action estimate $Q(s_t, a_t)$ is updated using the current state-action function, the reward $r$ at time step $t + 1$ (the observed reward), the $\text{max}Q$ value of the next state $s_{t+1}$ given a certain action $a$ has been executed and the previously described parameters.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \text{max}Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1.7)$$

The on-policy variant is referred to as SARSA and ensures that both the current action $a$ and the following action $a'$ are chosen based on the current policy $\pi$. The SARSA equation can be seen in Eq. 1.8 (Sutton and Barto [1998]).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (1.8)$$

Since this approach combines both Dynamic Programming and Monte Carlo, it is considered to be the method that usually leads to best results and is best suited to tackle complex problems. Therefore, in terms of traditional flat RL methods, this approach would be optimal to solve the problem at hand.

1.3 Limitations of Flat Reinforcement Learning

Cuayahutl et al. [2014] highlight two primary limitations of flat RL methods with respect to HRL methods, both of which have been previously echoed by several other researchers. Firstly, the issue of scaling up the architecture to deal with a complex problem, such as a quadcopter having to search an area, focus on recharging its battery, having to periodically return to base to upload on board data, etc can prove to be difficult or even impossible when dealing with a flat approach. Although the agent would eventually learn what to do, the lack of any understanding of the importance of certain tasks as compared to others, implies
that executing real world tasks would prove to be near impossible. In explicit terms, this relates to the designer’s ability to add additional knowledge to the agent, such that it has a better understanding of how a task may be executed optimally. This ties in in some ways to the second limitation of flat RL, which is that learning is much slower as compared to an HRL approach. This is due to the fact that RL methods scale very poorly, both in terms of the aforementioned complexity of the problem but also in terms of the problem space that needs to be dealt with. In effect, an increase in the number of states and possible actions for each state means that the rate of learning will decrease exponentially (and realistically speaking become infeasible) and is what is referred to by many scholars in the field as the "curse of dimensionality" [Dietterich 2000]. This problem is further extended by the fact that flat RL is bound by the MDP which is limited to discrete-time models, thereby adding extra computational burdens on the agent. HRL methods on the other hand are free to exploit the SMDP wherein computations can be performed within the continuous time domain. The nature of many real world tasks in terms of complexity and problem space results in reinforcement learning not being a suitable fit to solve many such problems. It is this problem that has forced researchers to think beyond flat RL methods, and is what has given rise to the branch of HRL.

In looking into hierarchical methods, another limitation of flat RL methods has become apparent to researchers, namely the inability to transfer learnt policies and knowledge from one task to another. In effect, the use of subtasks and subgoals in HRL implies that learnt policies for subtasks can indeed be transferred to different problems with the same or similar subtasks, and by doing so lead to accelerated learning. In the case of flat RL, the agent has no sense of task division and therefore even if it fully masters a certain subtask, it has no notion that the same policies can be used elsewhere, thereby limiting the ability to speed up learning.

1.4 Hierarchical Reinforcement Learning Methods

In the years since understanding limitations of flat RL when applied to many real world problems, academics have been studying ways in which the curse of dimensionality can be overcome. The foundation of these studies have been in trying to enable agents to learn in the same way that a human being might. In effect, while flat RL is based on the agent making decisions on a micro level, it is clear that human beings generally deal in macro actions. For instance, when asked to walk over to a table and open a bottle of water, the human does not need to think about every little decision, but might loosely divide the task into subtasks such as walking to the table, picking the bottle up and then opening it. This division means that the human can act much more easily and quickly than if he had to consider for instance how to walk and think about every step he might take towards the table.

Therefore, in order to combat the curse of dimensionality, the concept of using a hierarchy for decision making has become prevalent, thereby giving rise to HRL. The three main approaches in HRL are Options [Mcgovern et al. 1998], Hierarchies of Machine Abstraction (HAMs) [Parr and Russell 1998] and MAXQ [Dietterich 2000], with research focussed on either combining approaches or improving the existing ones.

1.4.1 Options

Mcgovern et al. [1998] use the term options to define a "generalization of primitive actions to include temporally extended courses of action". An option is composed of three elements, namely an input set, a policy and a termination condition. An option can be chosen if and only if the the state it is in at a certain point in time is a subset of the input set that constitutes the option. Once an option is chosen, the policy of that option is followed until the termination condition is met. Once an option has terminated, a higher level controller determines the next course of action, which could include selecting another option.

Sutton et al. [1999] highlight that the input set and termination condition of an option allow for the range over which the option must be defined to be limited in a useful manner. In effect, since policies in the options method need to be learned independently of the macro task being performed (either pre-programmed or learnt through some other training method), it is advantageous if the policy only needs to be defined for an input set and not all the states of the entire macro problem. The termination condition allows one to
define the input set for all the relevant states that are within the termination state. Therefore, in terms of policy generation, the input set and termination condition apply a very useful constraint to the subtask.

Reinforcement learning generally takes place within the MDP framework, which implies that all information that is needed to decide what action to take is "stored" within the current state of the system and therefore knowing about the system’s past is not required (Puterman [1998]). However, in the options method, the decision making process depends not just on the current state but also the option being followed. This gives rise to an extension of the MDP process known as the semi-Markov decision process. Figure 1.2 (Sutton et al. [1999]) illustrates how the state trajectory for an MDP and SMDP vary and how they behave with respect to options. Within the SMDP framework, the action probabilities for a certain policy are based on the entire history of states, actions and rewards for the option that is currently being executed. Within the options method, state trajectories can be analysed with either the detail of MDPs or the macro decision making aspect of SMDPs.

Sutton et al. [1999] saw great value in treating options as much as primitive options as possible. To that end, they presented the concept of a multi-time model of an option, which aims to extend the single step model seen in flat RL, such that it can be used for the SMDP case considered here. Having successfully achieved that, a Q-learning algorithm and a Dynamic programming algorithm were derived such that the Bellman optimality equation (Barto and Mahadevan [2003]) can be solved. Equation 1.9 (Barto and Mahadevan [2003]) shows the Q learning algorithm to be followed for updates of the state-action matrix, $Q$.

$$Q_{k+1}(s,o) = (1 - \alpha_k)Q_k(s,o) + \alpha_k [r + \gamma \max Q_k(s',o')]$$

$$Q_{k+1}(s_t,o) = (1 - \alpha_k)Q_k(s_t,o) + \alpha_k [r_{t+1} + \gamma U_k(s_t,o')]$$

where

$$U_k(s,o) = (1 - \beta(s))Q_k(s,o) + \beta(s)\max Q_k(s,o')$$

In terms of implementation of the options method, Wang et al. [2014] presents a three step method that can easily be applied to the navigation task at hand. The first step is to determine a set of local subgoals from the primary target task. This means breaking down the primary target task into subtasks. Once this is
done, options are determined for each of the subtasks as far as possible. This essentially means determining a
policy for a certain subtask given some input set and termination condition. Finally, the options are framed
such that they can be reused for subtasks that resemble the subtask they are devised for, and in doing so
can accelerate learning for the primary target task.

In looking at the presented approach, two issues come to mind; firstly, how does one determine a set
of subgoals, and secondly how does one ensure that options can be reused in similar tasks. In terms of
determining subgoals, McGovern [2002] propose a method wherein subgoals are automatically identified by
the agent by determining which regions are visiting frequently on successful trials but not on unsuccessful
trials. In effect, by using the example of room to room navigation to reach a goal state McGovern [2002]
demonstrates that "If the agent can recognize that a doorway is a kind of bottleneck by detecting that the
sensation of being in the doorway always occurs somewhere along successful trajectories but not always on
unsuccessful ones, then it can create an option to reach the doorway as a subgoal". Similarly, Xu et al.
[2009] also exploits bottlenecks in a task to determine appropriate subgoals, and presents an approach where
the agent can simultaneously obtain an initial policy for the various options. This approach is successfully
implemented on the gridworld problem. A lot of recent research in terms of subgoal discovery has been
focused on graphical approaches, as demonstrated by Taghizadeh and Beigy [2013], Moradi et al. [2012]
and Rad et al. [2010]. Much like the previously proposed methods, these approaches also rely on exploration
of the environment, however in this case the agent "saves the history of interactions as a directed graph"
Taghizadeh and Beigy [2013]. Once again, subgoals in these methods are also determined in a sense by
locating bottlenecks, as subgoals are defined as the regions of the state space that lie between two densely
connected regions. This form of graphical approach gives rise to several algorithms for subgoal discovery
which have proven to be effective.

The second aspect, namely the transferability of options is both a major weakness and potential strength
of options and is related to the issue of transfer learning. This is an issue that has been addressed by
several researchers in the field, including McGovern [2002], Konidaris and Barto [2007, Wang et al. [2014],
Masuyama et al. [2013] and Feng et al. [2015]. Transfer learning refers to the ability to use past knowledge
so that future learning can take place at an accelerated rate. Given the specific way in which options are
defined for a given input set and termination condition, it is often challenging to use what is learned in a
modified version of the same problem. Konidaris and Barto [2007] propose a solution wherein options are
learnt in an agent space and not a state space, as is traditionally done. The agent space is defined as "the
space generated by a feature set that is present and retains the same semantics across successive problem
instances". Therefore, the idea is that options defined in the agent space can be easily reused in different
problem spaces. Masuyama et al. [2013] propose an intrinsically motivated reinforcement learning method
which is hinged on using options and knowledge from successful past experiences. In effect, the success of a
past experience is measured in all environments with respect to an invariant feature and in doing so, skills can
be easily transferable. That being said, the authors admit that their formalisation has limited applicability
in cases where the problem space becomes very large, and therefore this might not be an interesting option
for the thesis topic being considered. Wang et al. [2014] propose a novel method wherein before learning
takes place, the agent performs extensive exploration runs, and then frames the the task to be performed in
terms of previously learnt options. The proposed algorithm is based on the premise that similar options have
similar state-action probabilities, thereby allowing the agent to identify similar options for varying tasks.
While in theory this sounds legitimate, it is not hard to imagine scenarios where this form of thinking could
lead to slower learning. That being said, the authors claim to have performed extensive testing of their
algorithm and have achieved promising results. What is advantageous here as well is that there is no need
for a complex hierarchical structure for transfer learning to take place.

1.4.2 HAMs

HAMs are an approach to RL wherein the number of actions that an agent is allowed to take within a given
state is limited. This is formally expressed by Parr and Russell [1998] as constraining the learning process
by hierarchies of partially specified machines. Therefore, while there might be several possible actions,
HAMs constrain the set of possible actions to a subset of said actions. While in essence a simple principle,
HAMs in the context of RL attempt to build on this concept and create ways to express these constraints at varying levels of specificity depending on where in hierarchy they are defined. HAMs are defined by "a set of states, a transition function and a start function that determines the initial state of the machine" [Parr and Russell 1998]. The set of states here refers to the various states encountered by the agent within the defined problem space. The transition function allows for the next machine state to be determined once an action has been carried out and is usually made using only a partial description of the environment. The start function is used to determine the initial state of the machine in which execution begins. Machine states can be categorised into four types, namely an action, a call, a choice and a stop. The action state executes an action, the call state executes another machine, the choice state selects a next machine and a stop state halts the execution of a machine and reverts to the previous state [Parr and Russell 1998].

Like in the case of the options method, HAMs also rely on the SMDP framework, however, Du et al. [2009] and Du et al. [2008] both highlight the fact that this does not naturally lend itself to decomposing the overall task at hand. In other words, within the HAMs method, understanding how to form sub-tasks and decompose a primary target task into a collection of subtasks can be challenging. Du et al. [2008] introduces a unifying framework for HRL with the inclusion of concepts such as HAM-decomposable and HAM homomorphism. This framework is demonstrated to be able to easily perform state abstraction for each individual subtask and allows for policy optimality to be freely selected. Du et al. [2009] also makes use of HAM-decomposable, however their primary contribution is the fact that they provide a method for hierarchical decomposition of "policy-coupled" SMDPs and present an approach that can be used to determine whether or not a certain task can be hierarchically decomposed. This is founded largely on work by Hengst [2003] who demonstrated that decomposition is possible when some variables change faster than others and when variables that change frequently keep their transition properties.

In the context of navigation, Parr and Russell [1998] demonstrated the power of HAMs when applied to a problem now referred to as Parr’s maze. In this problem, an agent must navigate through a maze filled with obstacles until it reaches its goal which is located at some random point in the maze. Fundamentally, this is a navigation problem similar to what will be implemented in this thesis as it relates to an agent exploring an unknown area both quickly and intelligently. As seen in Fig. 1.3, using HAMs allows for Parr’s maze to be solved much quicker in terms of computational time than using standard flat reinforcement learning methods. This is testament to the fact that using an abstract machine for quadcopter navigational tasks could have a significant impact on processing times.

Andre and Russell [2002] introduced the concept of Programmable HAMs (PHAMs), which increased the expressive power of HAMs, which allows for state abstraction to be formulated in a concise manner but also greatly increases the ability to transfer learned skills from one problem to another, a feature that would be highly useful for solving the problem being considered.

Figure 1.3: Required computational time comparison of HAM and flat RL method for Parr’s maze (Parr and Russell 1998)
1.4.3 MAXQ

The MAXQ approach, as developed by Dietterich [2000] works by decomposing the value function of the primary target task into a group of value functions for subtasks. Therefore, integral to MAXQ is the ability to decompose a value function into a set of value functions. As with options and HAMs, MAXQ methods also rely on the SMDP framework, and solves problems by creating a hierarchy of SMDPs that are processed simultaneously to solve the primary target task. However, unlike the aforementioned methods, the MAXQ method does not aim to directly reduce the target task to a single SMDP, but rather creates a hierarchy of SMDPs that are solved simultaneously.

The MAXQ approach is best described by Fig. 1.4 wherein the task diagram for a common reinforcement learning problem, known as the taxi problem, is shown. The taxi problem is concerned with how an agent may optimise picking up and dropping people, much like a regular taxi. Once again, the relevance to navigation is explicit, as the agent must navigate intelligently so as to maximise rewards. In this case, the problem is decomposed into four subtasks, namely Root, Get, Put and Navigate. For each of these subtasks, a corresponding primitive action or set of primitive actions is indicated, which are executed to complete a subtask. In the case of navigation for instance, the primitive actions are to go North, South, East or West. Depending on the complexity of the problem, a subtask can lead to either a primitive action or a further division into subtasks. It should be noted that the order in which primitive actions or subtasks on a certain level are shown is arbitrary, and that the action that is actually executed depends on the policy of a higher level controller. The graph only depicts the available action choices at a certain level.

![Figure 1.4: Task graph for the Taxi problem](image)

Each subtask within the MAXQ framework is composed of three elements (Barto and Mahadevan [2003]), namely a subtask policy, a termination condition and a pseudo reward function per subtask. A subtask is similar to a hierarchi- cal option, and therefore a subtask policy executes actions or selects other subtasks that fall under it in the task graph.

The issue with decomposing a task in such way is that complexities can arise when attempting to transition from one subtask to another. Harris et al. [2015] propose a viable method to stitch together subtasks with wide boundaries in a "near-seamless manner" by exploiting termination approximation. For cases where the transition from one sub-task to another cannot always be clearly defined, this is highly beneficial. In the case of the navigational task considered in this thesis, it is likely that the primary task will be broken into macro subtasks, with significant overlap between tasks, thereby making the findings of Harris et al. [2015] significant.

The MAXQ method is generally considered a very complex framework and amongst the more computationally intensive approaches in terms of HRL. In effect, extensive research has been carried out so as to improve the efficiency of learning. To that end, Gräve and Behnke [2014], Dai et al. [2010], Mehta et al. [2011], Cao and Ray [2012] and others have attempted to integrate Bayesian learning - a model based...
approach— with MAXQ learning. The global idea is that although developing models can be very complex, providing the agent with even a very simplified model can lead to huge gains in terms of learning rates. The model approaches extend beyond Bayesian learning, but include methods such as Dyna and Priority Sweeping, amongst other. Dai et al. [2010] demonstrate through their Bayesian-MAXQ learning algorithm that using statistical technologies to model the core MDP on any level can enable the agent to avoid making computationally costly repetitions. This further allows for HRL methods to overcome the curse of dimensionality more effectively and deal with high dimensional state spaces in a more feasible manner. Results demonstrating the performance of the Bayesian-MAXQ method for the taxi problem can be seen in Fig. 1.5, where it can be seen that the Bayesian-MAXQ method converges much faster than traditional MAXQ method while converging to almost the same value. Therefore, for a marginal loss in optimality, there is a significant increase in convergence speed.

In addition to further combating the curse of dimensionality, Gräve and Behnke [2014] makes use of a Bayesian and MAXQ combination to protect the agent from actions with an unpredictable outcome. In effect, having a model gives the agent a better sense of which actions might lead to severely negative consequences, thereby allowing them to be avoided. In the context of the navigational task at hand, it is clear that as long as the quadcopter is not colliding into objects, there is no negative outcome in terms of having complete randomness during exploration, and therefore this aspect is not particularly interesting. While there are several different algorithms and strategies that have been developed and show good results, they all attempt to increase efficiency of learning by combining HRL and learning through demonstrations. Whether this can still be considered reinforcement learning is a highly discussed topic in the field, however the added advantages of using such methods, especially for continuous state spaces is incontrovertible.

1.4.4 Combinations of Options, HAMs and MAXQ

Although options, HAMs and MAXQ are the most widely used methods in HRL, researchers have considered the implications of using a combination of the above methods. Shen et al. [2006], Harris et al. [2015] and Cai et al. [2013] all propose methods of combining options and MAXQ to achieve improvements in performance. The MAXQ framework is often considered to be one that uses hierarchical options (Dietterich 2000), and therefore in many ways, a merging of these methods seems natural. Shen et al. [2006] were able to successfully demonstrate the potential of their method, called OMQ (combination of Options and MaxQ), by applying it to the taxi problem as described previously. The OMQ method is also governed by the SMDP framework and its architecture is best described by Fig. 1.7, which shows a task graph used in the MAXQ method, as modified for the OMQ approach. As can be seen, the task graph does not vary much from that seen in Fig. 1.4 except for the fact that a certain level of subtasks consists of learned policies in the form of options. The results of using such an approach can be seen in Fig. 1.6 where it is demonstrated that OMQ requires fewer iterations for convergence than both MAXQ and Options applied independently. In effect, for partially known environments, such as the navigation problem that needs to be solved in this thesis, a combination of methods, if applied intelligently, could lead to better results (Shen et al. 2006). Similarly, Cai et al. 2013 also demonstrated the potential of combining MAXQ and options, within the context of a searching task involving multi-robot cooperation. Ignoring the fact that their work deals with multiple agents, the task
carried out in many ways is similar to what this thesis aims to develop, as it involves an agent searching an unknown area for certain points of interest. Incorporating multiple agents in an area could be an interesting extension for this thesis, however keeping time restrictions in mind, might prove to be beyond the scope of this thesis.

1.4.5 Other Methods

Other algorithms that have been developed include HEXQ \cite{Hengst2003}, Hierarchical Average Reward RL \cite{Ghavamzadeh2007}, Economic Hierarchical Q-Learning \cite{Schi10} and the HASSLE algorithm \cite{Bakker2004}. Each of these approaches brings its own set of strengths and weaknesses when assessed with respect to the navigational task at hand.

\cite{Hengst2003} presents the HEXQ Automatic Decomposition approach wherein the MDPs are decomposed based on a "variable-wise search for Markov sub-space regions". The governing principle is based on the fact that variables associated with lower levels in the decision hierarchy will change more frequently than those associated with higher level control. This allows for the agent to automatically determine a hierarchy and construct multiple SMDP regions, each with its own subgoal. These subgoals are based on the agents desire to leave one region and enter another. Globally speaking, HEXQ has the ability to work on a broad range of unknown problems, however it is limited by the fact that variables must have different time scales (change at varying frequencies) and must be consistently labelled.

\cite{Bakker2004} present the HASSLE algorithm, which aims to achieve HRL with sub-policies specialising for learned subgoals. In effect, \cite{Bakker2004} highlight that most HRL methods rely on a predetermined hierarchical structure, something that can place a significant burden on the designer. To overcome this, it would be ideal if "high level policies automatically discovered subgoals and low level policies learnt to specialise for different subgoals". The HASSLE algorithm aims to do exactly that, however has certain drawbacks in terms of dealing with large number of parameters, lack of convergence guarantees and a heavy dependence on being able to identify appropriate high level goals.

\cite{Ghavamzadeh2007} present a method, that as opposed to the previously proposed methods, is well suited for continuous tasks and is not limited to the discrete time SMDP framework. The average reward RL being considered here is nothing new, however placing it in the context of HRL is far less common. The problem in the thesis will be considered to be discrete and therefore further investigation into this method is redundant. That being said, this is definitely an avenue for future work as average reward HRL is appropriate for a wide range of continuous tasks, enables for the optimal policy to be determined such that rewards are maximised on a per step basis and allows for more efficient state abstraction in HRL than within the SMDP framework \cite{Ghavamzadeh2007}.

39
1.5 Discussion and Conclusions

Based on the current state-of-the-art and the navigational task that RL needs to be used for, it is clear that conventional flat RL methods will prove to be insufficient. The large number of states involved with navigation in a realistic area results in the curse of dimensionality, implying that traditional methods will be too slow at learning and therefore infeasible. This gives rise to the concept of HRL, wherein decisions are made on different levels, and not every action needs to be thoroughly evaluated. In effect, a target problem is broken into subtasks which can solved independently and simultaneously with less computational effort as fewer states need to be evaluated. The three main approaches are options, HAMs and MAXQ, all of which have their own respective strengths and weaknesses. Although HRL has already been applied to several navigational tasks, such as the racetrack problem (Harris et al. [2015]), the taxi problem (Shen et al. [2006], Dietterich [2000], Jong and Stone [2008], Dai et al. [2010]), Parr’s maze (Parr and Russell [1998]) and many others, the author of this paper knows of no instance where HRL has been applied to a system such as a quadcopter exploring a relatively large state space.

1.5.1 Specifying/Discovering Subtasks and Hierarchies

The three most prominent methods in the field of HRL, namely options, HAMs and MAXQ were all studied during this review. All these methods, and indeed all HRL methods one might argue, rely on dividing a primary task into several subtasks. These subtasks can be defined, loosely speaking, in three broad ways. The first, as seen in the Options method by Sutton et al. [1999] aims to define each subtask in terms of a fixed policy. This policy is independent of the learning that takes place in order to solve the primary task. Therefore, the first challenge that arises with options is that the policy for each subtask must either be programmed by a human or learnt using some other program. For complex problems this can prove to be time consuming for the designer. To that end, McGovern [2002] presents a viable method to automatically determine subgoals and consequently hierarchies as well. This is elaborated on by work done more recently by Taghizadeh and Beigy [2013], Moradi et al. [2012] and others, who use similar concepts to automatically determine subgoals and task hierarchies. Research done here is extremely interesting as it minimises the workload on the designer, and allows for developed algorithms to be applicable to several different forms of problems.

The HAMs approach attempts to "define each sub subtask in terms of a non-deterministic finite-state controller" (Dietterich [2000]). In this method, the programmer is able to apply certain constraints to the action set, thereby defining a partial policy that the agent should follow. The advantage of HAMs lies in the fact that depending on the subtask and its place in the hierarchy, a range of constraints from no constraint to a fully specified solution can be implemented. HAM techniques developed by Parr and Russell [1998] help ensure that exploration takes place in a focussed manner, and exploration of a new environment is not done blindly. There is minimal literature on self discovery of subtasks and task hierarchies, and therefore while the superior performance of HAMs is clear (see Fig. 1.3), it does seem like an approach that places increased burden on the designer. This becomes more apparent when looked at in terms of optimality and reusing learnt policies, as will be discussed later in this chapter.

The third well known method of dividing a task into subtasks is the one employed by the MAXQ method and aims to define each subtask in terms of its own termination condition and local reward function (Dietterich [2000], Mehta et al. [2011]). This method creates subtasks by hierarchically decomposing the value function of the target MDP, and allows for a unique approach to be applicable in many domains. That being said, the proposed method still has difficulties in fully discovering a hierarchy for domains where only some good policies exhibit a hierarchical structure.

The HEXQ approach (Hengst [2003]) is also very interesting in terms of automatically learning subtasks and determining hierarchies. Much like MAXQ, this method creates subtasks by hierarchically decomposing the value function of the target MDP, and allows for policy and state abstraction to be employed at various
levels. As Hengst [2003] notes, this method allows the agent to "decompose its environment based on its own experience", thereby minimising the designer’s workload.

As Dietterich [2000] notes, the ability to exploit state abstraction is something that is highly beneficial, especially when dealing with large domain spaces like that of most real world problems. Making use of a hierarchical structure allows for state abstraction to be implemented as it creates room to evaluate decisions with different levels of detail. That being said, making use of state abstraction in a safe way can be challenging, as ignoring certain elements of the state space could lead to sub optimality, with potentially serious consequences for real world applications. Dietterich [2000] also states that successful use of state abstraction requires a termination predicate, as seen in MAXQ and HEXQ. Therefore making use of options or partial policy methods like HAMs can be significant drawback. That being said, using a termination predicate introduces a lot of complexity and therefore from an implementation perspective is much less appealing.

1.5.2 Optimality

The primary drawback with dividing a task into subtasks is that the agent will focus on maximising each local reward rather than the global reward, thereby perhaps leading to an overall suboptimal result. Therefore, the programmer must carefully evaluate how the sub tasks and rewards are defined as they can have a significant impact on learning and performance. Needless to say, this could be a drawback for essentially all HRL methods, wherein the distinction between the hierarchical optimal and the recursive optimal should be made Cao and Ray [2012]. A hierarchically optimal policy is a hierarchical policy that has the maximum expected reward whereas a recursively optimal policy is defined as one where the local policy for each subtask, and not the global policy for the primary target task, is optimal. Making use of subtasks invariable means that there is a chance of finding the recursive and not hierarchical optimal. An example of recursive optimality can be seen in Fig. 1.8 wherein the global task of reaching the goal "G" has a subtasks of exiting the first room and then moving towards the goal. In an attempt to exit the first room optimally, actions are taken that in fact lead to the goal being reached sub-optimally. As per Barto and Mahadevan [2003], "SMDP dynamic programming or RL methods applied to HAMs or to MDPs with options whose policies are fixed a priori, yield hierarchically optimal policies" and it is also relatively easy to achieve hierarchical optimality for the MAXQ method. That being said, there is interest in the recursive optimality as well as it is a context free optimality and therefore greatly enables transferability of learned knowledge. Therefore, while not ideal for a specific task, enabling an agent to determine the recursive optimal can be beneficial in terms of the bigger picture. Despite what Barto and Mahadevan [2003] state, Hengst [2003] highlights that options, HAMs and MAXQ provide no guarantee of how close the hierarchical solution will be to the globally optimal solution of the original MDP. The global optimal here refers to the optimal of the original problem, without any constraints that might be introduced by the designer when creating a hierarchy. It states that the constraints on subtask policies imposed by the designer result in the obtained solution being optimal in terms of the limitations imposed on the hierarchy. This of course is further worsened for cases where the agent has no contextual understanding of subtasks and determines the recursive optimal. Interestingly, the fact that certain complex problems can only be realistically solved using hierarchical methods, it is potentially impossible to know if the hierarchical optimal is indeed the global optimal. In general terms, achieving the global optimal requires a huge set of available policies and furthermore limits the ability to implement state abstraction. State abstraction however, as demonstrated in Dietterich [2000], has significant impacts in terms of reducing computational time, and is therefore very useful to have.
1.5.3 Reusability of subtasks

Traditionally speaking, option learning methods are limited to work within a specified state space, and are not easily transferable from one environment to another, thereby greatly reducing the usability of learnt options. However, Konidaris and Barto [2007], Wang et al. [2014], Masuyama et al. [2013] and Feng et al. [2015] discuss methods that allow for “skill transfer in reinforcement learning” within the options framework, thereby greatly improving the return on effort in terms of having to program a certain policy for a subtask. This is built on the work presented in McGovern [2002] who created a framework for automatic discovery of subgoals, which further facilitated the easy transfer of learnt policies. A drawback of this approach is the fact that for complex systems it cannot easily understand the fact that there can be good actions that in the big picture lead to negative outcomes. As the complexity of the system rises, the ability to determine subtasks decreases quickly.

The literature on transfer of subgoals within the HAMs framework is rather limited and no explicit cases where methods were developed and or applied were found during the literature study phase.

In terms of the MAXQ framework, Dietterich [2000] made the difference between being able to find the recursive and hierarchical optimal explicit. While the recursive optimal seems trivial, computing it suggests complete learning of a subtask, in a context free environment. Therefore, fundamental to the MAXQ framework is the ability to find recursive optima and thereby transfer learnt skills to different contexts. Needless to say, integrating a learnt skill in a different context is challenging, however the decomposition approach of MAXQ lends itself nicely to facilitating transfer learning. In addition, the HI-MAT approach (Mehta et al. [2011]), while good at enabling transfer learning in the MAXQ framework as well, has limitations which include the fact that there has to be some form of underlying causal relationship between the source and target problems. A system where the two are decoupled leads to a breakdown of this approach.

1.5.4 Synthesis

The primary purpose of this literature review is to gain insight into the current practices and state of the art literature in the field of HRL, and relate that to navigational tasks for a quadcopter. To that end, this review successfully outlines the various methods that can be applied in order to solve the problem and provides sufficient information to implement options, HAMs, MAXQ or a combination of these methods. In addition, potential points of difficulties as well as methods to tackle them have been highlighted. Unfortunately, this survey is inconclusive in terms of which method would be best suited to solving the problem at hand as there is no explicit best choice for the task being considered. To that end a more in depth review during the thesis itself is required. That being said, based on the survey, it can be concluded that the lack of literature of cases where HAMs have been applied, allows for it to be safely ruled out and not considered any more during the thesis. Furthermore this review provides a strong foundation based on which specific steps can be taken so as to evaluate which method would lead to optimal performance for the considered problem.

In terms of determining which method to further develop and implement for a quadcopter navigational task, several different criteria need to be evaluated. These include, but are not limited to, the ability to automatically discover subgoals and hierarchies, the ability to transfer learnt knowledge between varying scenarios, the ability to achieve the global optimal for any given task and the ease with which a designer can implement the desired approach. Based on results obtained by Shen et al. [2006] it does appear as though a combination of methods, such as OMQ, may yield better results than using just one method, as the domain of the taxi problem is similar to the domain of the navigation problem to be tackled in this thesis. This is further supported by Harris et al. [2015] who also makes use of a combination of options and MAXQ on the racetrack problem to achieve promising results. In addition, the HEXQ method also presents several advantages over the traditional MAXQ as developed by Dietterich [2000]. Therefore, it is clear that looking beyond the conventional options and MAXQ framework could lead to added advantages in terms of performance, but will likely bring about significant challenges in terms of implementation. Understanding the trade-off between the two will be key in terms of determining an approach to implement.
Chapter 2

Research Plan

This chapter will aim to elaborate on the theoretical framework within which the research questions will be solved. Furthermore, the setup, procedure and tools to be used will also be discussed, and supported by the presentation of two RL problems that have been solved in preparation of the primary navigational problem to be solved during the thesis. That will lead to a discussion on the expected results from the thesis, as well as a Gantt chart outlining the planning for the thesis, including this preliminary report.

2.1 Theoretical Content/Methodology

Although a few other approaches do exist, most development in the field of HRL is governed by the Options, HAMs and MAXQ methods. These three methods operate within their own respective frameworks, and have certain advantages and disadvantages associated with them. A large part of the literature study, and subsequently the thesis, is focussed on understanding which of these three methods is best suited for the navigational task being considered, and whether indeed a combination or variation may yield better results. Therefore, it is at this stage premature to express in more detail the exact methodology that will be used in solving the problem, except for the fact that HAMs in their traditional form will not be implemented due to a limited amount of literature found on the topic as well as limited cases where the approach has been applied being found. In addition, what can be stated is that the solution strategy will be bound by the aforementioned MDP or SMDP, which govern almost all (if not all) discrete time RL problems. Furthermore, the strategy will rely on dividing tasks into subtasks, with their own respective subgoals, as implementing a hierarchy without a division of tasks would be infeasible. To that end, a navigation task will have as the most primitive actions, the ability to go North, South, East or West. This would further lead to higher level tasks of searching for points of interest or moving towards the charging station. Therefore, understanding which HRL method is best suited for the task, and implementing it for the problem being considered is indeed the primary focus of this thesis, and therefore it is natural that further details cannot be provided at this point.

2.2 Experimental Set-up

This thesis is primarily concerned with HRL methods, and merely uses a navigational task as a context in which to apply HRL methods. To that end, this thesis will be entirely theoretical and will only produce MATLAB simulations as the result, evaluating the methods mentioned in Section 2.1. Therefore, all experimentation will take place in MATLAB and will be done through programming.

In terms of programming, the first aspect of the setup that needs to be considered is the motion of the quadcopter and the environment in which it is placed. Given that this research is about HRL and not quadcopter dynamics, the motion of the quadcopter will be modelled in a very simplistic manner, so that
time and resources can be focussed on other tasks. To that end, the quadcopter will have x and y velocities of -1, 0 and 1 and will be able to accelerate in both directions with an acceleration of -1, 0 or 1.

Subsequently, the size of the state space to be explored must be determined. This is an important parameter, as the bigger the state-space, the more important the HRL algorithm becomes, as explained by the curse of dimensionality. Therefore, the chosen state space size has to be large enough such that it allows for a hierarchical approach to demonstrate both its need and its effectiveness. This state-space size will be determined in an iterative fashion but does indeed play an important role in determining the strength of HRL algorithm that is implemented. Subsequently, certain points of interest will be placed on the map, which when visited will provide the agent with a reward (the value of which is still to be determined). Finally, the battery level of the quadcopter will be monitored and a charging station will be placed somewhere on the map, such that the quadcopter must periodically visit this location. If the quadcopter runs out of battery, a negative reward will also be received by the agent. The actual value of all these rewards needs to be determined somewhat iteratively as well.

Another factor that has an influence in terms of gauging the strength of an algorithm is the computational power of the computer that it is run on. If a simulation is run on two different computers with different processing abilities, the results will be influenced not just by the algorithm but also by said processing power. Therefore, in order to ensure that the results are consistent, it is important that all simulations are done on the same computer such that the computational power variable is controlled.

In order to make explicit the experimental setup and procedures to be used in the thesis, the sub research questions will be addressed individually to present the proposed approach.

1. "To what extent can navigational problems be solved by HRL?"
   This question will be answered using a purely theoretical approach. To that end, literature will be evaluated and cases where RL and HRL have been used to solve navigation problems will be evaluated and the results will be analysed.

2. "What benefits and limitations does HRL possess as opposed to flat RL methods?"
   This question will be answered through the use of literature and simulations. Firstly, literature will be used to see results from work done by other researchers in the field, and understand how flat and hierarchical RL compare. Subsequently, flat RL and HRL will be implemented on a simplified version of the racetrack problem to better understand the challenges and advantages with respect to using HRL. More details of the racetrack problem being considered here can be found in Section 3.2.

3. "What are the benefits and limitations of the Options HRL approach with respect to the MAXQ HRL framework?"
   Using literature, the number of methods to be implemented has already been narrowed down to two, namely Options and MAXQ. This is to limit the scope of this thesis and focus on the method(s) that seem most promising. The selected methods will be implemented in the environment as described at the beginning of the section, and will be compared in terms of performance. This is a very significant part of the research to be carried out and will allow for comparison between not only flat and hierarchical RL methods, but also highlight the differences between Options and MAXQ.

4. "What is the effect of scaling up the problem space on performance of the HRL method in question?"
   Determining the size of the problem space, as mentioned before, is an iterative process. Therefore, the problem space will be iteratively scaled up to test the performance of the algorithm. This could involve increasing the size of the area to be navigated, or possibly adding additional velocity states. The former seems more useful while keeping learned policies in mind. Furthermore, the complexity of the problem will also be scaled up by incorporating additional tasks that must be carried out, such as the quadcopter having to periodically recharge its battery. The scaling up of the problem should allow one to gauge how performance varies when complexity and size is increased, but also analyse the extent to which policies learned in a "simple" problem space can be transferred and accelerate learning when dealing with more complex problem spaces.
In terms of execution, this thesis will begin by building on the work presented in Junell et al. [2015], a paper co-authored by the primary supervisor of this thesis. In effect, said paper presents a RL approach for a quadcopter to explore a completely unknown environment and locate points of interest. It also incorporates an element where the quadcopter must return to a specific location once the on board storage capacity is full. In all respects the research carried out is the same as what the thesis will aim to achieve, with the exception that the applied method is a form of flat RL. Therefore, the initial aim is to apply a hierarchical method to solve the described problem and in doing so enable the agent to deal with a larger number of states in the same or less amount of time.

Barring these elements, the author does not face any experimental limitations in terms of implementing an HRL method. The challenges that arise are on the theoretical side and relate to deriving an optimal algorithm and tuning parameters to achieve optimal performance.

2.3 Expected Results, Outcome and Relevance

The goal of any reinforcement learning method, hierarchical or otherwise, is to determine an optimal policy wherein the reward achieved by an agent is maximised. In the context of a quadcopter performing navigational tasks, the idea is to create a map with certain points of interest with a reward associated with them and enable the quadcopter to navigate such that it maximises its reward by moving from one point of interest to another in an optimal manner, without of course having insight of the map itself. An additional feature involves the quadcopter having to periodically move to a charging station to recharge.

The primary outcome of this thesis is to compare performance between a hierarchical and flat RL method in the context of navigation. As previously mentioned, performance here is defined in terms of a) the size of the problem space that can tackled, b) the number of iterations until convergence, c) the final value of the converged value function d) the computational power required for learning. Therefore, results would include data for all the above mentioned criteria, sampled over varying number of runs. Therefore, these values after for instance 10, 100 and 1000 runs can be recorded to observe trends in performance. The expectation is that the HRL method will converge to an optimal much quicker than the flat method, however in the long run the flat method will lead to a better optimal than the hierarchical method. That being said, there is an expectation that if the problem space is increased to a large enough size, the flat RL method will not function in any feasible manner. Achieving a policy using HRL that allows for convergence in little time and fewer iterations implies that in real life situations where large areas need to be explored, the derived algorithm can be used to achieve good results.

The basic elements considered here are the actions that the quadcopter can take, the states it operates within and the rewards it receives. In a hierarchical framework, the rewarding strategy is not a trivial matter and needs to be evaluated carefully. In addition, the HRL method, or combination of methods to be used also needs to be evaluated. Furthermore, the exploration strategy is also something that needs to tuned, and has to be done through an iterative process.
Chapter 3

Solved Blackjack and Racetrack problem using RL

In an attempt to gain further insight into RL and understand which elements might prove to be challenging, two problems were implemented using RL. Firstly, the popular card game Blackjack was solved. Given the limited size of the state-space involved, only flat Monte Carlo was applied. In addition, the common racetrack problem was solved, using both a flat and hierarchical approach. This section will present the implementation details as well as the obtained results.

3.1 Blackjack

Blackjack is a casino card game wherein the aim is to ensure that the sum of the cards in the player’s hand is as close to 21 as possible, without exceeding that value. The player plays against the dealer and ultimately whoever has the number closer to 21 without exceeding it takes the pot. If the hand value of either the player or the dealer exceeds 21 then the game is automatically over and the other wins. If both the player and the dealer have the same hand value then the game is considered a tie. No other players are considered in this simulation, and for the purpose of determining the optimal policy and state-value function this should not make any difference. If anything, having additional players could mean that the learner learns faster as it observes other states, actions and rewards.

The game begins with the player getting two cards and the dealer getting two cards, with one of the dealer’s cards being open for the player to see. All number cards are the value of the number they have, whereas all face cards (Jack, Queen and King) are counted as 10’s. The ace can either be counted as a 1 or 11, depending on whichever is more useful. Therefore, based on the options available to him, the player can decide to either hit (take another card to add to his existing sum) or stay (not take any more cards). There could be other actions such as splitting, however those are not considered in this assignment.

As per the form of BlackJack considered here, all the possible states are described below:

- **Player hand sum:** If the player’s hand sum is less than 12, then it is clear that he must always hit as there is no way that he can lose. This can be set as a deterministic policy as there is no learning required for this. If the player’s hand sum is somewhere in the range of 12 to 21 (10 states), then by observing the other factors he can decide whether he wants to stay or hit.

- **Dealer’s open card:** The card that the dealer is showing is another piece of information based on which the player can determine what course of action to take. The showing card can be in the range of 1 to 10 (10 states).

- **Usable ace:** Whether the ace can be used as an 11 or a 1 determines if the ace is ”usable” or not (2 states). An ace is considered usable if it can be used as 11.
Based on the above factors, it can be seen that there are 200 states (10x10x2) for which the choice of hitting or staying must be made. With enough iterations, the player should learn what action to do (policy to use) such that the reward is maximised. The rewards for this game will be set at +1, -1 and 0, for a win, loss and draw respectively.

It should be noted that the version of Blackjack used here omits many rules such as doubling down, splitting and winning by a natural. Although this does make the system easier, it still allows for clear demonstration of reinforcement learning. Another assumption made here is that the deck of cards is an infinite deck so there is no use keeping track of cards that have passed. This is a fair assumption as many casinos make use of multiple decks to make card counting more difficult.

In order to understand the fundamentals of RL, the problem as described in Sutton and Barto [1998] was implemented, so that a problem could be solved and validated. Thus, the unlikely case of Monte carlo exploring starts was run. This method ensures that every episode starts with a randomly initialised policy, and therefore all states are visited multiple times in a short period, thereby allowing one to quickly and accurately average the optimal state-action function, state-value function and policy. The optimal policy is presented in Sutton and Barto [1998]. As can be seen, Fig. 3.1 exactly matches Sutton and Barto [1998] and therefore the fundamentals of the Monte Carlo simulation in Matlab can be considered both understood and validated.

Figure 3.1: Optimal state value function and optimal policy derived using Monte Carlo Exploring Starts
As an initial parameter study, the effect of varying $\epsilon$ in an $\epsilon$ greedy approach was looked at and compared with a greedy policy. This was done by observing the learning of the system over the short run (1000 episodes) and the long run (5e6 episodes). The method used here is Monte Carlo, with a direct averaging approach for the state-action function, as seen in Eq. 1.5. Although this method is thought to be computationally intensive, it is also the most straightforward. As can be seen, four $\epsilon$ values were tested, namely 0, 0.1, 0.01 and a varying function. The varying function is defined as follows:

- $\epsilon=0.01$ for the first 5e5 runs
- $\epsilon=0.001$ for the next 1.5e6 runs
- $\epsilon=0.0001$ for the last 3e6 runs

By looking at Table 3.1, it is clear to see that for the first 100 runs, all options have extremely similar rewards, with the exception that $\epsilon = 0.1$ starts having a lag compared to the other options. This is due to the fact that there is too much exploration and not enough exploitation, thereby reducing the returns of the player. As far as the greedy policy and other $\epsilon$ greedy policies are concerned, there are almost no differences for the first 1000 runs. This can be explained by the fact that all of the options are heading towards some common optimum. Whether this is the actual global optimum is yet to be seen. Globally speaking, the greedy policy performs the worst in the long run as the lack of exploration means it is unable to find optimal solutions other than the one it thinks it has found. $\epsilon = 0.1$ has overtaken the greedy policy and is still improving, however the relatively high amount of exploration means that it is very slow at finding the optimum compared to the other two remaining options. In terms of the varying $\epsilon$ and $\epsilon = 0.01$, it is observed that until 5e5 hands, both options have the same performance, which is expected since they have the same $\epsilon$ value. However, at that point, it is observed that the varying function has a much quicker rise in performance as the amount of exploration has decreased and the focus is on exploitation. With enough runs, all options except the greedy one should eventually converge, the only difference being that some will be much slower than the others.

<table>
<thead>
<tr>
<th>Episode Range</th>
<th>$\epsilon = 0$</th>
<th>$\epsilon = 0.01$</th>
<th>$\epsilon = 0.1$</th>
<th>Varying $\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:10</td>
<td>-0.6000</td>
<td>-0.6000</td>
<td>-0.6000</td>
<td>-0.6000</td>
</tr>
<tr>
<td>10:20</td>
<td>-0.6000</td>
<td>-0.6000</td>
<td>-0.6000</td>
<td>-0.6000</td>
</tr>
<tr>
<td>20:40</td>
<td>-0.2500</td>
<td>-0.2500</td>
<td>-0.4500</td>
<td>-0.2500</td>
</tr>
<tr>
<td>40:100</td>
<td>-0.1667</td>
<td>-0.1667</td>
<td>-0.2167</td>
<td>-0.1667</td>
</tr>
<tr>
<td>100:1000</td>
<td>-0.1222</td>
<td>-0.1267</td>
<td>-0.1578</td>
<td>-0.1267</td>
</tr>
<tr>
<td>1e3:1e4</td>
<td>-0.1178</td>
<td>-0.1157</td>
<td>-0.1418</td>
<td>-0.1157</td>
</tr>
<tr>
<td>1e4:1e5</td>
<td>-0.1212</td>
<td>-0.0979</td>
<td>-0.1224</td>
<td>-0.0979</td>
</tr>
<tr>
<td>1e5:1e6</td>
<td>-0.1161</td>
<td>-0.0631</td>
<td>-0.1098</td>
<td>-0.0606</td>
</tr>
<tr>
<td>1e6:5e6</td>
<td>-0.1185</td>
<td>-0.0558</td>
<td>-0.1109</td>
<td>-0.0544</td>
</tr>
</tbody>
</table>

Table 3.1: Change in average reward for different Epsilon values with $P_{\text{initial}} = \text{random}, Q_{\text{initial}} = \text{zeros}$

### 3.2 Racetrack Problem

The second, more complex system considered was the racetrack problem. This is another common RL problem wherein an agent must navigate through a racetrack taking as few actions as possible. Each action the agent takes earns it a reward of -1 and each time it leaves the track it receives a reward of -5. Therefore, the aim is that after sufficient iterations, it will be able to perfectly navigate the racetrack with as few steps as possible.
The problem space is considered to be a discrete race track consisting of 16X10 "blocks" and there being 7 possible velocity states ranging from -3 to +3 for both the x and y direction. This therefore gives rise to \(10 \times 16 \times 7 \times 7 = 7840\) states. The possible actions are accelerations of -1, 0 and 1 in both the x and y direction giving rise to 9 possible actions. Therefore, the difference in terms of problem space is already apparent as the blackjack problem presented earlier has an action value matrix of 200X2 whereas the racetrack problem here has an action value matrix of 7840X9. Furthermore, for the chosen racetrack, the starting point can be any x-location with a value less than or equal to 5 and the race is complete once the x-location is greater than 16 (the maximum x location for this track).

In solving the problem, both a flat and hierarchical approach were implemented and were solved using a simplistic Monte Carlo approach, as represented by Eq. [1.6]. The reason this approach was opted for was primarily due to its simplicity with respect to implementation. Therefore, there is indeed a limitation in how well a hierarchical approach can be implemented using Monte Carlo methods, it should be kept in mind that this problem was solved only to understand the kind of thinking and challenges that might be faced with implementing a hierarchy.

The flat method is extremely straightforward and requires little explanation. As mentioned, it makes use of Eq. [1.6] to update the Q matrix (state-action matrix). The state action matrix in this case is 7840X9. The Q matrix was initially set to zeros and an epsilon-greedy policy was implemented, with \(\epsilon = 0.1\). This may seem rather high, but given the number of states that need to be visited it is acceptable.

The HRL method made use of the options framework. The options consisted of enabling the agent to accelerate either North, South, East or West. Therefore, if the agent for instance decides to go North, actions are taken within the subpolicy to increase the vertical speed and bring the horizontal speed to zero. In this case, the sub policies of going in one of the cardinal directions was hard coded and did not have to be learnt automatically by the agent. What can be seen here is that the state action matrix Q for the HRL approach is in fact a 7840x4 matrix as the agent only needs to make one of 4 choices. What happens after that is the following of a deterministic policy. Therefore, in this simplistic approach itself, the effect of making use of a hierarchy is evident.

Results obtained from this implementation can be seen in Figs. 3.2, 3.3 and 3.4. Figs 3.3 and 3.4 show the optimal path taken by the agent to travel from the start of the racetrack until the finish after 1000 training episodes for a flat and hierarchical approach respectively. As can be seen, the flat approach takes several small steps, especially at the beginning. Somehow it has determined an almost perfect trajectory for the second half of the track, which could be just luck of having visited all the best states during some runs. The hierarchical approach is much more what one would expect. The agent keeps moving in straight lines as it understand making turns leads to sub-optimality. That being said, it is clear that it is stuck in a local optimal and needs more training to understand better how to make turns. There is full confidence that with more training and exploration, the agent will understand that turns should be sooner. Figure 3.2 demonstrates the average reward per episode for the flat and HRL approach. As can be seen, despite starting off in somewhat similar ways, the HRL approach is able to improve its learning much faster, and is receiving higher rewards after few runs. That being said, it also seems to be caught in a local optimum which it does not seem
be able to break out off. This suggests that maybe the exploration strategy needs to be reconsidered. The flat approach on the other and will always converge to the global optimal, but as can be seen, requires more time to do so. Furthermore, there is full confidence that as the size of the problem space is increased and more states are incorporated, this difference in learning rate will only become more prominent. Furthermore, it can also be stated that there are significant differences in computation time for the flat and hierarchical method, with the hierarchical method being a lot faster than the flat.

Figure 3.3: Route followed by the agent in an optimal run after 1000 training episodes for a flat RL implementation

Figure 3.4: Route followed by the agent in an optimal run after 1000 training episodes for a HRL implementation
Chapter 4

Preliminary Conclusions

The proposed preliminary report highlights both the relevance and feasibility of enabling a quadcopter to use HRL to navigate an area intelligently. To that end, literature in the field is surveyed to determine whether there is a need for this research and how it might contribute to the existing body of work. In addition, this review successfully outlines the various methods that can be applied in order to solve the problem and provides sufficient information to implement options, HAMs, MAXQ, a combination or a variation of these methods. Subsequently, a research question and sub research questions are designed. Furthermore, an experimental setup, or procedure for testing and simulation with solved examples was also described, highlighting the expected results and outcomes. Finally, the report also provides a Gantt chart that should be followed so that the thesis stays on track with respect to time.

In determining the need for HRL for navigational tasks, the performance and capabilities of flat RL methods had to be evaluated. They are much easier to implement than HRL methods, and therefore if they perform as well as HRL methods, there is no need for HRL methods. However, it was found that Dynamic Programming, Monte Carlo, and Temporal Difference applied in a flat manner, all suffer from the curse of dimensionality. As the problem space scales up, flat RL methods are impotent and can never function in a realistic way. Therefore, the three primary HRL methods were surveyed, and it was concluded that these methods could lead to significant improvements in performance. In addition, variations such as HEXQ and combinations such as OMQ were also investigated and the added advantages of those frameworks also became apparent. Although no method can be explicitly categorised as the best choice, a decision was made to discard HAMs from further consideration, primarily due to a lack of cases where HAMs have been applied by other researchers. In answering the primary research question "How can a quadrotor exploit hierarchical reinforcement learning methods to optimally navigate an area and locate points of interest?" MATLAB simulations will be used along with theoretical input. The final output is to develop/implement an algorithm that generates an optimal policy, which when followed leads to a maximised reward. The Gantt chart provided should serve as a strict guideline during the thesis, such that all deadlines are respected.

HRL is a powerful learning approach that can be applied to several problems. At its core it is perhaps the most intuitive form of learning, and therefore development in this field has several applications, even in areas that extend far beyond quadcopters or aerospace. To the best of the author’s knowledge, an evaluation of HRL for a quadcopter including sequential tasks as proposed here has not been previously performed. With this in mind, there is little doubt about the value that this thesis would add to the existing body of research. This preliminary review opens the discussion to determining the optimal method(s) that should be used to solve the specified navigational task. The addition of sequential elements implies that the method chosen must take the full functionality of the algorithm into account, as the best algorithm for a simple navigational task might not necessarily be the best for a more complex task. To that end, the problem itself needs to be considered in more detail, and the possible subtasks must be evaluated. Depending on how the problem is “broken up” and whether hierarchies and subtasks need to be learned automatically, the provided literature should be sufficient to determine which method(s) will lead to optimal results as well as how said method can be implemented.
Part III
Additional Results for Autonomous Navigation using Feature-based Hierarchical Reinforcement Learning

I. Sub policies

This section presents the learning curves for the learnt sub policies. No explicit attempt was made to ensure learning took place as quickly as possible, as it was only considered important to see convergence in the end. The four policies that are trained are a navigation policy used to travel in straight lines until a wall is met (Fig. 1), a policy to exit a room (Fig. 2), a policy to stay in a room (Fig. 3) and a policy to find the charging station (Fig. 4). Some policies are learnt more rapidly than others, which is to be expected. What is surprising however is how much longer it takes the charging option to converge than the room exiting option. This is perhaps best attributed to the fact that sub optimal epsilon values were used, thereby prolonging learning. This aspect was not further explored, as as previously mentioned, the only truly important element is convergence. What is worth noting however is that the charging policy for some reason does not perfectly converge after 5000 runs, a flaw that contributes to poor runs when solving the primary target problem using this sub policy.

![Figure 1](image1.png)

Figure 1. Learning curve demonstrating a near perfect policy for the navigation option, where the agent always chooses an action that maintains it’s current heading when not faced with a wall.

![Figure 2](image2.png)

Figure 2. Learning curve demonstrating a near perfect policy for the exit room option.
Figure 3. Learning curve demonstrating a near perfect policy for the staying in room option

Figure 4. Learning curve demonstrating a near perfect policy for the charging option

The key performance parameters regarding training and convergence are presented in Table 1. The final converged value is not provided since the rewards are entirely arbitrary and no useful insights can be gained by knowing the converged reward received by the agent. What is worth noting is the fact that in terms of computation term, all sub policies are learnt within approximately 75s. While this may seem like quite a long time, understanding that these sub policies can then be used to solve a wide range of problems helps legitimise the investment in this extra learning time.

Table 1. Performance of learnt sub policies in terms of iterations for convergence and computation time

<table>
<thead>
<tr>
<th></th>
<th>Navigation</th>
<th>Exiting</th>
<th>Staying in room</th>
<th>Charging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations until convergence</td>
<td>300</td>
<td>200</td>
<td>1750</td>
<td>700</td>
</tr>
<tr>
<td>Converged</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Computation time [seconds per 100 iterations]</td>
<td>1.03</td>
<td>1.4</td>
<td>1.8</td>
<td>1.7</td>
</tr>
</tbody>
</table>

II. Training for 6 rooms without prior knowledge

Figure 17 of Part I shows the learning curve for the six room problem without the battery element using previously learnt high level knowledge. Observing the learning curve for the same problem but without prior high level knowledge as shown in Fig. 5 allows for a more complete understanding of the full extent to which the use of prior knowledge accelerates learning, especially as the size of the problem increases. Table 2 provides an overview of the key performance parameters. The two room performance is presented since the prior knowledge used consists primarily of learning done in this scenario and therefore provides a good comparison of the difference in learning ability. In terms of episodes for convergence as well as computation time, using prior knowledge provides significant advantages. However, the converged value for learning done
without past knowledge indicates that learning from scratch results in an episode being completed in 13 fewer steps on average. This difference is within the margin of error given the randomness of the goals, and therefore is not an explicit indication of better performance when learning from scratch.

Figure 5. Reward received per episode for completing the six room task without battery with one randomly generated goal per room using Options without any prior high level knowledge

Table 2. Comparison of performance of the six room problem using no prior knowledge with a case where prior knowledge is used

<table>
<thead>
<tr>
<th></th>
<th>2 Rooms</th>
<th>6 Rooms with prior knowledge</th>
<th>6 Rooms without prior knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations until convergence</td>
<td>350</td>
<td>0</td>
<td>1100</td>
</tr>
<tr>
<td>Converged value [reward per episode]</td>
<td>18.0</td>
<td>41.1</td>
<td>54.8</td>
</tr>
<tr>
<td>Computation time [seconds per 100 iterations]</td>
<td>3.3</td>
<td>10.5</td>
<td>18.6</td>
</tr>
</tbody>
</table>

III. Epsilon study for high level learning

This section presents the learning curves for the high level epsilon study that was carried out as part of the sensitivity analysis. The figures (Fig. 6 to Fig. 9) are self-explanatory and the captions provide all the relevant information so as to draw meaningful insights from the figures.

Figure 6. Reward received per episode for completing the task as seen in Fig. 18 of Part I with an initial Epsilon value of 0.1 and a decay rate of 0.9975
Figure 7. Reward received per episode for completing the task as seen in Fig. 18 of Part I with an initial Epsilon value of 0.05 and a decay rate of 0.995

Figure 8. Reward received per episode for completing the task as seen in Fig. 18 of Part I with an initial Epsilon value of 0.05 and a decay rate of 0.9975

Figure 9. Reward received per episode for completing the task as seen in Fig. 18 of Part I with an initial Epsilon value of 0.05 and a decay rate of 0.999

IV. Expanding applicability

The results derived in Part I are based on a specific setup, namely that every room consists of a goal state. This therefore results in learning of policies wherein the agent learns to remain within a room until a goal has been found. While this serves as a good platform to demonstrate the learning ability of an agent within a hierarchical framework, this problem setup poses certain limitations in terms of practical applications. Therefore, an attempt is made to expand the applicability of the derived learning strategies such that they can be applied to a more general case wherein goals might be more randomly dispersed. Therefore, a scenario with four rooms and the battery element (as seen in Fig. 18 of Part I), but only two goals, is explored. The placement of the goals is not entirely random, but rather there is always one goal in the two lower rooms.
and a second goal in one of the two upper rooms. Therefore, the agent must learn to sometimes travel from one room to another yet also learn when to stay in a room and look for a goal. Figure 10 shows the learning for the described problem, and as can be seen convergence occurs at approximately 1250 runs. The sizeable variation in reward even after convergence has occurred can be attributed to the fact that the likelihood of the agent missing the goal and moving to another room is relatively high, and therefore the agent must return to a previous room so as to find the goal.

**Figure 10. Reward received per episode for completing the four room task with two targets and battery element using Options**

Having achieved convergence for a four room with two target problem, an attempt is made to demonstrate learning on the even more general problem of four rooms with one target. Therefore, there is only one goal to be found by the agent, and it is randomly located in any one of the four rooms. Figure 11 shows the learning curve for this problem while a reward of 100 is provided for finding the goal. The learning here seems to be unexpected as the agent does not seem to fully learn the concept of visiting a charging station when the battery is low. This is unexpected as all prior learning experiments have seen the agent understand and learn this concept relatively easily. The best explanation for this behaviour perhaps lies in the rewarding strategy, wherein the agent receives a relatively low positive reward for finishing an episode, as opposed to the huge negative reward when it runs out of battery. Therefore, the reward for finishing an episode is increased to 1000 and the learning using this new rewarding strategy can be seen in Fig. 12. Although no convergence can be seen, it can be concluded that the agent no longer runs out of battery by the end of its training. The lack of convergence is best explained by the randomness of the goals, resulting in the agent leaving a room in which a goal is located (without finding said goal), and then having a low probability of returning to said room. Therefore, at this stage it is concluded that training for a four room problem with one random target has been unsuccessful and requires more research and the use of new strategies.

**Figure 11. Reward received per episode for completing the four room and battery task with one randomly generated goal using Options**
Figure 12. Reward received per episode for completing the four room and battery task with one randomly generated goal using Options with a high reward for finding the goal.

V. Addressing publication bias

The nature of reinforcement learning as a training method results in there being a lot of room for publication bias. The randomness of the episodes invariably means that learning will sometimes take place faster than other times, and some curves will experience more outliers than others. That being said, when a complex problem that requires several hundred episodes (or even thousands in some cases) to converge is dealt with, the variation from one trial to another is not particularly significant. In order to demonstrate this, the learning curve for the four room and four goal problem with the battery element (as depicted in Fig 18 of Part I) is presented in Figs. 13 and 14 with learning taking place under the same setting as seen in Fig. 21 of Part I. As seen in Table 3, although there is some variation present, globally speaking the learning in all three cases exhibit the same characteristics. Perhaps the biggest variation is seen in the converged value, with trial 3 converging to a significantly higher value. It is observed that this is not due to the optimal behaviour being better than in the other trials, but rather due to the fact that there are fewer relatively bad episodes. This is the kind of bias that is inherent to RL, especially when the goals are randomly generated, however it should be noted that these findings do not in any way negatively impact the findings of the presented research.

Figure 13. Repeated trial showing reward received per episode for completing the four room, four goal and battery task using Options.
Figure 14. Second repeated trial showing reward received per episode for completing the four room, four goal and battery task using Options

Table 3. Comparison of performance for the four room problem with four goals and battery element using the same setting so as to demonstrate variation between trials

<table>
<thead>
<tr>
<th></th>
<th>Trial 1</th>
<th>Trial 2</th>
<th>Trial 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations until convergence</td>
<td>1000</td>
<td>800</td>
<td>950</td>
</tr>
<tr>
<td>Converged value [reward per episode]</td>
<td>48.0</td>
<td>45.7</td>
<td>75.4</td>
</tr>
<tr>
<td>Computation time [seconds per 100 iterations]</td>
<td>14.2</td>
<td>16.7</td>
<td>13.2</td>
</tr>
</tbody>
</table>
Conclusion

The primary aim of this report was to investigate the potential of Hierarchical Reinforcement Learning (HRL) in terms of enabling agent’s to successfully perform autonomous indoor navigation. Further, an attempt was made to demonstrate the need for HRL and highlight the limitations of traditional flat methods. To that end, a thorough assessment of existing approaches and standard practices was carried out, as presented in Part II of this report. Although there were several methods that showed a lot of promise, a decision was made to pursue the MAXQ and Options methods in more detail, thereby resulting in only these two HRL approaches being analysed in depth.

In evaluating the MAXQ and Options methods in Part I of this report on a simple goal finding and room exiting problem, it was determined that while both approaches showed promise, the Options method would be used in tackling the more complex problems. This was based on the quantitative performance of the two methods in terms of number of iterations until convergence, value of the converged policy and computation time. In addition, the implementation characteristics were also considered in deciding which method to pursue in more detail. At this stage already, the gross limitations of flat Reinforcement Learning (RL) became apparent. In solving the more complex navigational problems a focus was placed on the ability to reuse prior knowledge as a) the size of the problem increases and b) the complexity of the problem increases. This paper effectively demonstrated that training on a two room problem led to significant gains in learning time when the learnt policy was applied to a larger four or six room problem, as opposed to training for a four room problem from scratch. Furthermore, the ability to use prior high level knowledge about a task and apply it to a similar problem with an additional element in terms of complexity was also demonstrated. However, beyond just showing the ability to transfer low and high level policies, it was also demonstrated that this transfer of learning leads to accelerated learning. In specific terms, this was demonstrated by comparing learning from scratch for a problem with four rooms and a battery element with learning when a policy for a navigational problem where the battery element was irrelevant was used on a problem where the recharging played an active role in the mission. The flat RL approach was entirely ineffective at solving the most complex version of this problem, and no learning was apparent.

A sensitivity analysis was also carried out in Part I in order to examine the influence the designer exerts on learning. Three aspects, namely the availability of options, the number of steps within an option and the epsilon value were evaluated. In all three studies, convergence took place regardless of the parameters, however there was undeniably a significant impact on the rate of convergence as well as the value of the converged function. This forces one to be critical of the results, as indeed attempts should be made to ensure that learning takes place as fast as possible, yet designer imposed restrictions should not in any way limit the functionality of the learning approach should it be applied to a different problem.

The additional results presented in Part III also allow for some conclusions to be drawn. Firstly, in terms of expanding applicability, it is effectively demonstrated that the presented approach works for a case where there is a target in one of two rooms and not necessarily one in every room. This is a significant find and is an indication that measures can be implemented to ensure learning for the case with one goal in four rooms. That being said, at this stage such a conclusion cannot be convincingly made as initial results show that there is no convergence for a case with four rooms and one goal. Further, the additional results also demonstrate the quality of the sub policies being used, and it can be concluded that they are all nearly optimal if not optimal. Finally, the issue of publication bias is addressed, in that not all trials exhibit the
same behaviour. The additional results demonstrate that even though there is variation between runs, the nature of the problem ensures that statistically speaking the findings in Part I are still entirely valid, as the variations between trials do not have an influence on the overall learning trends.

HRL is a powerful learning approach that can be applied to several problems. At its core it is perhaps the most intuitive form of learning, and therefore development in this field has several applications, even in areas that extend far beyond quadcopters or aerospace. The successful demonstration of the use of HRL to solve complex navigation problems reaffirms the fact that RL can be used to solve real world problems and has applications beyond just theoretical sample problems.
The limited scope of this thesis implies that there are several elements within this topic that would make for interesting future research. This chapter will present those elements.

Firstly, the autonomous discovery of hierarchies and subtasks is a feature that is extremely interesting and would definitely lend itself nicely to the navigational task being considered. It would enable the quadcopter to be able to navigate different types of areas without additional input from the designer. Therefore, whether it be having to travel indoors from one room to another or having to explore an external area with trees, the agent would be able to determine for itself how best to formulate subtasks and create a hierarchy.

Secondly, the ability to transfer learnt knowledge from one problem space to another is a feature that is known to accelerate learning. This would result in a quadcopter being able to make use of policies learnt in previous scenarios and easily adapt them to new environments. This is demonstrated in a very limited scope in this thesis. Developing experiments to demonstrate the extent to which learnt knowledge can be transferred can prove to be challenging but if indeed learnt knowledge can be transferred, it can lead to significant improvements in learning.

Thirdly, this thesis has not been able to devote sufficient attention to neither combinations of the primary methods (Options, HAMs, MAXQ) nor other methods such as HEXQ. In effect, to fully understand the strengths, limitations and performance of these methods, a more detailed analysis of each of them needs to be performed and further they may even need to be implemented. While many such methods show a lot of promise, this thesis will focus on Options and MAXQ, thereby leaving several stones somewhat unturned.

Furthermore, a lot of research in HRL is geared towards making using of model based approaches such Bayesian learning, combined with HRL methods such as MAXQ as this is known to accelerate learning, especially in complex systems. Incorporating a model based approach to this problem would be challenging, but there is an almost certainty that this will lead to accelerated learning rates and would therefore be extremely beneficial.

Another interesting application of HRL is in terms of controlling multiple agents to execute a mission. In the context of the considered problem, it could involve a swarm of drones exploring an area and locating points of interest. The intrinsic ability of HRL to make decisions on different levels naturally lends itself to having a high level controller that decides how the individual drones should behave for best impact as a group. There are several advantages to using multiple drones, including but not limited to being able to explore an area faster and not facing serious problems should a drone be damaged in some way.
Bibliography


