Incorporating MAXQ Hierarchical Reinforcement Learning in Robotic Urban Search and Rescue

Author: Alexander Hong

Supervisor: Goldie Nejat

April 2014

University of Toronto
Division of Engineering Science

Incorporating MAXQ Hierarchical Reinforcement Learning in Robotic Urban Search and Rescue

ESC499Y Thesis
Final Report

Author:
Alexander Hong

Supervisor:
Prof. Goldie Nejat

April 2014

Abstract

Teams of autonomous robots can provide valuable assistance in Urban Search and Rescue (USAR) environments by gathering useful information about the complexity of the environment and accessing confined spaces. However, USAR scenes are complex and unpredictable. Many autonomous approaches are not effective due to their static nature in a dynamic environment. Learning-based semi-autonomous controllers allow robots to learn from their surroundings and previous experiences in order to optimize their own behaviours. The goal of this thesis is to incorporate MAXQ Hierarchical Reinforcement Learning (HRL) based semi-autonomous controller for a rescue robot team in a 3D simulation software. The simulation serves to evaluate HRL algorithm's performance in map exploration and victim identification in unstructured USAR environments. This thesis sets up the framework required to evaluate the MAXQ HRL architecture's performance in USAR environments. Using the 3D simulation software, Unified System for Automation and Robot Simulation (USARSim), accurate robotic movements and depth profile generation were successfully simulated. Terrain categorization and obstacle avoidance algorithms were implemented and tested to work in 3D space. In addition, teleoperated performance of USAR exploration was recorded for benchmark testing. It is hopeful that rescue robots will be able to effectively explore complex USAR environments intelligently.

Acknowledgement

I would like to express my deepest gratitude to Prof. Goldie Nejat for providing me with the opportunity to work in her research group. This thesis would not have been possible without her on-going support, guidance, and vital feedback. Special thanks to Yugang Liu for committing his time in helping me understand the complex concepts that is inherent in MAXQ Hierarchical Reinforcement Learning and providing me with constant technical feedback. I would also like to thank Julio Vilela for guiding me through the tedious problems I have encountered while using the USARSim simulator. Last but not least, thank you to all my Engineering Science friends for their continuing support throughout the year.

Contents

List of Figures

List of Tables

Acronyms

API	Application Programming Interface
GUI	Graphical User Interface
HRL	Hierarchical Reinforcement Learning
INS	Inertial Navigation Sensor
MDP	Markov Decision Problem
P3AT	Pioneer 3-AT (Research Robot)
RL	Reinforcement Learning
ROS	Robot Operating System
SHMR	Single Human Multi-Robot
SLAM	Simultaneous Localization And Mapping
TCP	Transmission Control Protocol
UI	User Interface
UDK	Unreal Development Kit
UT	Unreal Tournament
USAR	Urban Search and Rescue
USARSim	Unified System for Automation and Robot Simulation

1 Introduction

1.1 Robots in Urban Search and Rescue

Multi-robot rescue teams have promising applications in the field of robotic Urban Search and Rescue (USAR). The great losses in human lives caused by many tragedies have demonstrated the need for the development of multi-robot rescue teams to assist rescue workers in time critical tasks [4]. Rescue robots provide a compelling solution to assist rescue workers reduce human risk by entering unstable structures and increasing the speed of rescue response by extending reach into inaccessible and hazardous regions [4]. To date, the majority of rescue robots used in real-world USAR situations are teleoperated single robots [4]. However, single robots experience task-handling limitations [4]. Multi-robot rescue teams increase efficiency and system robustness compared to single rescue robots [4]. Furthermore, teleoperated robots depend on human operators, who may experience stress [4], and have perceptual difficulties in trying to understand cluttered 3D environments via remote visual feedback [3]. As a result, there is a need to develop autonomy for multi-robot rescue teams in order to reduce the stress and workload of rescue robot operators in time critical disaster scenes.

1.2 Current Research in Autonomy of Rescue Robots

Recent research has focussed on several areas for improving autonomy of rescue robots. One focus has been to improve low-level control of rescue robots by equipping them with the ability to autonomously climb stairs and navigate over uneven terrain [4]. Other research areas have focussed on improving high-level semi-autonomous control strategies that enable sharing of scene exploration and victim identification tasks between operator and rescue robots [4]. Moreover, the incorporation of Simultaneous Localization And Mapping (SLAM) algorithms allow single robots [5, 6] and multi-robot teams [7] to develop 3D maps of USAR scenes in order to locate victims within cluttered environments [4]. However, only primitive research has been done to incorporate robotic learning into a control scheme to deal with unknown and unpredictable USAR environments [3]. Learning-based semi-autonomous control for USAR robots will improve teamwork in Single Human Multi-Robot (SHMR) systems, and help mitigate stress that the operator experiences by having robots learn their own optimal behaviours [3]. In particular, Hierarchical Reinforcement Learning (HRL) allows rescue robots to learn from their previous experiences, as well as the experience of other team members [3]. HRL has been used in structured environments to address mobile navigation and obstacle avoidance problems [4]. However, robotic HRL is

a fairly new development and has only been prematurely applied to robotic USAR applications [3].

1.3 Objectives

The goal of this thesis is to incorporate robotic HRL techniques in a 3D simulation software package known as the USARSim [8] in order to provide rescue robots with varying levels of autonomous capabilities in disastrous environments. The goal is divided into two main objectives:

1. Implement the HRL algorithm [3] into USARSim environment
2. Evaluate robotic learning performance and compare results with teleoperated control performance in order to obtain knowledge on the feasibility of HRL algorithms in USAR scenes.

1.4 Simulation Approach

Given the value of simulation techniques and the need for reliable robotic assistance, this thesis incorporates the HRL architecture using USARSim platform with a team of P3AT robots. The implementation of the architecture will generate 3D maps of the virtual environment during scene exploration and identify potential victims.

USARSim is a high-fidelity simulation of USAR robots and environments that is used as a research tool in many multi-robot investigations [8]. This provides a virtual testing environment for robots and allows rapid prototyping of robotic learning algorithms. The idea of using USARSim to test HRL-based robotic learning techniques in multi-robot USAR is both feasible and economical. This is because all testing is done virtually. When the algorithm is fully developed, code can be transferred onto a physical robot with minimal code changes and expect similar results to the simulation. The experimental method to implement the HRL technique in USARSim is the following:

1. Generate unstructured USAR environments in USARSim using Unreal Development Kit (UDK) Editor
2. Create interface for communication between robot (client) and USARSim (server) via Transmission Control Protocol (TCP) connection
3. Implement HRL algorithm into client to provide semi-autonomous and autonomous capabilities
4. Conduct experimental tests to evaluate semi-autonomous algorithmic performance for a robot team in the unstructured USAR environments

5. Compare performance results between teleoperated and HRL semi-autonomous control trials [3].

It is hopeful that rescue robots will be able to efficiently explore cluttered USAR environments and find more victims compared to teleoperated control.

2 Literature Review

The main goal of this thesis is to place teams of robots in a virtual USAR map and perform semi-autonomous exploration of the environment while searching for potential victims. There are various exploration techniques to accomplish this. This section will introduce the idea of exploration in the scope of USAR missions, summarize current exploration techniques, and detail the application of the HRL architecture in robotic USAR.

2.1 Robotic Exploration

Robotic exploration and victim identification in USAR is a challenging task due to the sensing challenges in USAR environments. Vision-based techniques are difficult to implement due to the inherent challenges of computer vision under unstructured lighting conditions [4]. Low-cost acoustics and CO_2 sensors may experience unexpected behaviour due to potential existence of noise and other gases in USAR environments [4]. Thermal sensors prove to be unreliable for victim identification as collapsed structures can give similar heat readings to humans [4]. Furthermore, robots flipping over are one of the biggest challenges for rescue robots.

In USAR, rescue robots need to explore USAR environments while detecting hazards and searching for victims. There is minimum presumed knowledge about the environment. This includes having no knowledge of the victims' locations, existence of hazardous areas (i.e. leaked gas, fire, holes), and scene information (i.e. cluttered, climbable surfaces, open space). Furthermore, there is no indication of how long exploration will take. USAR exploration techniques must be robust and applicable to any unstructured environment. The following section will present current exploration algorithms used in USAR applications.

2.1.1 Simultaneous Localization And Mapping

Simultaneous Localization And Mapping (SLAM) addresses the problem of a robot navigating an unknown environment. In SLAM, the robot seeks to acquire a map while navigating the environment, and at the same time, it wishes to localize itself using its map [5]. SLAM provides detailed environment models and accurate sense of a mobile robot's location. However, SLAM techniques mostly deal with static environments, yet nearly every actual robot environment is dynamic. More work is needed to understand the interaction of moving and non-moving objects in SLAM [5].

Various SLAM approaches have been developed for USAR applications [4]. SLAM provides a way for robots to autonomously generate 3D maps of their environments and localize themselves as well as victims and objects of interest within these environments [4]. However, SLAM relies on iterative feedback from processes of locating and mapping under conditions of errors and noise. Thus, errors build cumulatively due to the inherent uncertainties in the robot's relative movement from its various sensors, resulting in poor accuracy [5].

2.1.2 Frontier-based Exploration

Frontier-based exploration is a direction-based exploration technique based on the concepts of frontiers, regions on the boundary between open and unexplored space [6]. By moving to new frontiers, a mobile robot can extend its map into new territories until the entire environment has been explored. This exploration technique requires no previous knowledge of the map's topology.

The central idea behind frontier-based exploration is to gain the most new information about the world by moving to boundaries between open space and uncharted territory. The environment is mapped by employing occupancy grids to associate areas in the real world with grid cells in the map. Each grid cell is marked as either open, unknown, or occupied based on occupancy probability, the probability of the robot accessing that cell. An advantage to this approach is its ability to explore both large open spaces and narrow cluttered spaces, with walls and obstacles in arbitrary orientations [6]. This type of exploration is completely scalable to multiple robots [7]. Robots can share individual occupancy grids and perceptual information with each other. By doing this, a global grid can be created to be shared amongst group of robots. This approach enables robots to make use of information from other robots to explore more effectively, but it also allows exploration to be more robust to loss of individual robots [7].

This technique has been implemented on real robots, and demonstrated that they can explore and map office environments as a team [7]. Moreover, it has been implemented into USARSim for multiple robots to search an unknown cluttered environment [9]. This technique considers both the terrain information of an environment by classifying obstacle cells as climbable or non-climbable cells, and the direction of a robot to determine its ability to traverse a cell of interest. The performance of the semi-autonomous exploration approach has proved to have a significant increase in exploration coverage compared to autonomous exploration approach for this algorithm [9]. However, there is a limitation in the frontier-based

exploration approach. The current implementation does not consider the optimal path for exploration. For instance, it may be more optimal for a robot to move backwards in some situations compared to rotating 180 degrees and proceeding forward [10]. The robot is programmed to behave in an isolated way and a more optimal path may exist. This is important in USAR applications because time is of the essence.

2.2 MAXQ Hierarchical Robotic Learning

The HRL technique addresses both robot exploration and the victim identification problem in USAR environments. This approach allows rescue robots to learn from their previous experiences, their current surroundings, and experiences of other team members [3]. Since the robot learns from experiences, it does not experience the same limitation as the frontier-based exploration approach. That is, robots can learn to optimize their paths as opposed to having an isolated behaviour.

Hierarchical approaches to Reinforcement Learning (RL) problems promise many benefits, including improved exploration and faster learning for new problems [2]. The MAXQ method for HRL is an extension of the Q-learning method, which is non-hierarchical. Experiments show that the MAXQ learning algorithm learn much faster than ordinary flat Q-learning [2]. This is because the Q-learning method suffers from unaccounted scenarios that are difficult to solve. To be more specific, the process of hierarchical learning is broken up into subtask, and each subtask corresponds to a Markov Decision Problem (MDP) [2]. The difference between MAXQ learning and Q-learning lies in the abstraction techniques, such as temporal abstraction, state abstraction, and subtask abstraction of the problem. Task decomposition is only present in MAXQ HRL. With task decomposition, the dimension of state or action spaces can be reduced. Thus, the results for lower level subtask can be reused by other high level subtasks. For the Q-learning method, the learning task is not broken down. In contrast, the MAXQ method is general purpose as it can be applied to difficult problems by breaking it into simpler subtasks. As a result, a hierarchy is form with many simple subtasks. At each level of the hierarchy, the task is Markovian and can be solved by standard RL methods [2]. The MAXQ HRL technique is an approach to HRL based on the MAXQ decomposition of the value function. The following sections will dive further into the technical details of the development of MAXQ HRL and its application in USAR.

2.2.1 Q-Learning

Q-learning is an unsupervised learning method [1]. A robot with Q-learning will explore from state to state until it reaches a goal. The terminology in Q-learning includes the terms “state” and “action”. The state space is the abstraction of a real problem, and an “action” allows one to move between different states. For example, the states in Figure 1 are the rooms, and the movement from one room to another is the ”action”. The initial state is Room 2 and the goal state is Room 5.

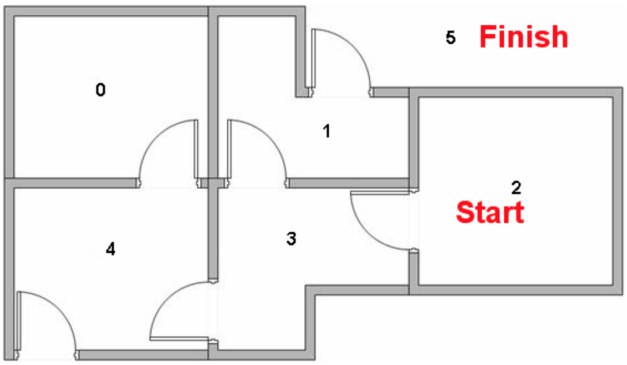


Figure 1: Standard Maze Problem [1]

For learning to happen, the doors in the building must be associated with a reward value. For the case of simplicity, doors that immediately lead to the goal state will have an instant reward of 100. Other doors not directly connected to the goal state will have a reward of zero. Figure 2 illustrates the graph representation of Figure 1. In Q-learning, the goal is to reach the state with the highest reward [1].

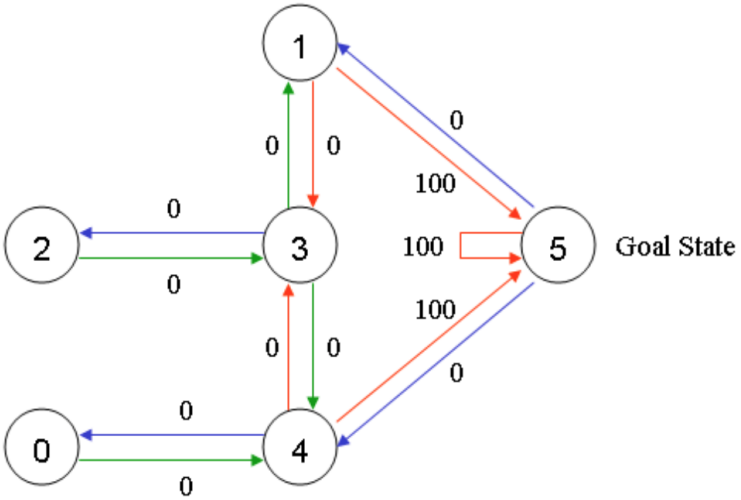


Figure 2: Graph Representation of Figure 1 [1]

The rewards and graph of the environment can be represented by the matrix R [1]:

$$R = \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix}$$

where each row corresponds to the initial state and each column corresponds to the final state after an action has taken place (-1 values in the table represent null values). The brain of the robot is represented by a Q-matrix, where the rows of matrix Q represent the current state of the robot, and the columns represent the possible actions leading to the next state [1]. The Q-matrix is initialized to be a zero matrix as the robot starts out by knowing nothing. In the above example, the number of states is presumed to be 6. For exploration applications, there are an unknown number of states. In this case, the Q-matrix starts out with one element and columns and rows are added if a new state is found.

The transition rule [1] of Q-learning is

$$Q(s, a) = R(s, a) + \gamma \times \texttt{max}[Q(s', a)] \tag{1}$$

where s is the current state, s' is the next state, a is the current action being evaluated, γ is the learning parameter ($0 \leq \gamma \leq 1$)[1], and R is the rewards matrix. The robot learns through experience. Each exploration is called an episode and each episode consists of the robot moving from the initial state to the goal state [1]. Each episode can be considered as a training session to condition the robot's brain represented by the Q-matrix. More training results will result in a more optimized Q-matrix and the optimal Q-matrix will find the fastest route to the goal state. The psuedo-code for Q-learning is presented in Appendiex A.1.

[1]If γ is closer to 0, then the robot favours immediate reward and if γ is closer to 1, then the robot will consider future rewards with greater weight, willing to delay the immediate reward

2.2.2 Taxi Problem

The Q-learning method can be applied to the popular Taxi Problem. The Taxi Problem is an extension of the example shown in Section 2.2.1. A simple Taxi Problem is illustrated in Figure 3. In each episode, the taxi starts in a randomly chosen state and with a randomly chosen amount of fuel [2]. There are four designated locations in the world marked as R, B, G, and Y[2]. There is a passenger at one of the four locations chosen randomly and that passenger wishes to be transported to one of the four locations also chosen randomly. In order to solve this problem, the taxi must go to the passenger's location, pick up the passenger, go to the destination location and drop off the passenger. When the passenger is dropped off at the destination location, the episode ends.

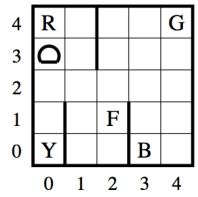


Figure 3: The Taxi Domain [2]

As one can see, this is clearly an extension of the Q-learning example previously explained before. There are seven actions: move (i) up, (ii) left, (iii) right, (iv) down, (v) pick up passenger, (vi) drop off passenger, (vii) fill up fuel tank at location F [2]. Rewards are assigned as follows:

Action	**Reward**
Each action	-1
Deliver Passenger	+20
Illegal (v) or (vi) action	-10
Fuel < 0	-20
Hit wall	-1

Table 1: Rewards for Taxi Problem [2]

The goal of this problem is to find a policy, a mapping from perceived states of the environment to actions to be taken when in those states [1], that maximizes the average reward per step [2]. Equivalently, this is the same as maximizing the total reward per episode. For randomly chosen amount of fuel ranging from 5 to 12 units, it has been shown that with optimal policy, an average reward per step of 0.92 can be attained [2].

[2]F is designated for filling up fuel tank.

2.2.3 The Hierarchical Approach

The MAXQ HRL method improves the Q-learning method as it has the ability to interpret MAXQ graphs as representations of the value function for hierarchical policies. A MAXQ graph representation for the Taxi Problem is shown in Figure 4. The Taxi Problem has a simple hierarchical structure in which there are three subtasks: (i) pick up passenger, (ii) refuel taxi, and (iii) drop off passenger [2]. Each subtask involves navigating to the desired location, and then picking up passenger, dropping off passenger, or filling up fuel. Two types of nodes are used in Figure 4: Max nodes (triangles), and Q-nodes (ovals). Max nodes with no children denote actions, and Max nodes with children represent subtasks [2]. The immediate children of each Max node are Q-nodes and each Q-node represents an action that can be performed to achieve its parent's subtask [2]. Each Q-node undergoes Q-learning and will learn the optimal path to achieve a high cumulative reward of performing its subtasks.

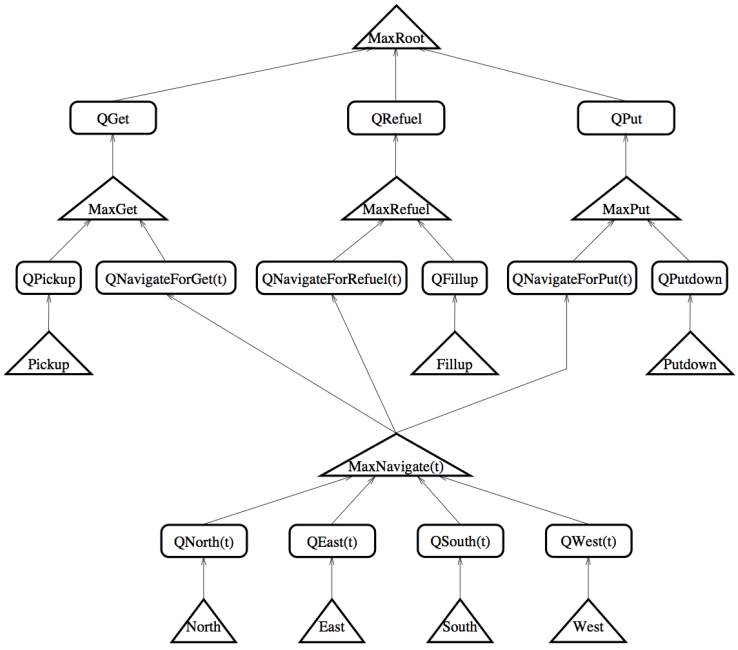


Figure 4: MAXQ Graph for Taxi Problem [2]

A hierarchical policy for a MAXQ graph is a set of policies $\pi = \{\pi_0, ..., \pi_n\}$, one for each Max node, that determines how each Max node should choose its actions. Essentially, the hierarchical policy is executed the same way that subroutines are executed in ordinary programming languages [2]. The MAXQ graph can be viewed as a subroutine call graph. Like subroutines, Max nodes can have parameters. One major difference between subroutines and the MAXQ graph is that the children of each Max node are unordered [2]. This means that they can be called in any order, and a Max node can execute each of its children multiple times before it completes

its subtask. The expected cumulative reward for following the hierarchical policy π starting in state s for Max node i can be written as:

$$V_i^\pi(s) = V_a^\pi(s) + \sum_{s'} P_i(s'|s,a) V_i^\pi(s') \tag{2}$$

where $P_i(s'|s,a)$ is the probability that the environment will move from state s to state s' when action a is executed, $V_a^\pi(s)$ is the immediate reward for node i of executing action a, and $V_i^\pi(s')$ is the expected culmulative reward in state s' [2]. The equation gives a recursive decomposition of the value function. That is, the value function of the root node is the value function of the entire MDP[3] and each subtask is a separate MDP. Equation (2) can be rewritten in action-value representation as:

$$Q_i^\pi(s,a) = V_a^\pi(s) + C_i^\pi(s,a) \tag{3}$$

$$V_i^\pi(s) = \begin{cases} Q_i^\pi(s,\pi_i(s)), & \text{if } i \text{ is composite} \\ \sum_{s'} P(s'|s,i) R(s'|s,i), & \text{if } i \text{ is primitive} \end{cases} \tag{4}$$

$$C_i^\pi(s,a) = \sum_{s'} P(s'|s,a) V_i^\pi(s') \tag{5}$$

where $Q_i^\pi(s,a)$ is the expected cumulative reward for a MDP performing action a in state s and then proceeding the hierarchical policy π thereafter, $C_i^\pi(s,a)$ is the completion function and is the expected cumulative reward of completing the MDP[4], and $R(s'|s,i)$ is the reward function, which is the reward received upon entering state s' as a result of performing action i in state s [2]. In essence, each Q-node with parent i and child a stores the information $C_i^\pi(s,a)$ for each state s, and each Max node i returns the Q-value of the child chosen by π_i [2]. The psuedo-code for MAXQ HRL is presented in Appendiex A.2.

2.2.3.1 MAXQ Hierarchical Reinforcement Learning in USAR

The MAXQ HRL method provides a unique approach to USAR exploration. An HRL-based controller enables a team of rescue robots to learn and make decisions regarding on which rescue task need to be executed at a given time [3]. Semi-autonomous controller has proved to be more effective than just autonomous controllers [4]. With semi-autonomous MAXQ HRL, rescue robots can benefit from human operators' experience and knowledge, while reducing the stress and mental workload of operators [3]. USAR environments are unknown and highly cluttered,

[3]In MAXQ, the overall task is to solve a MDP defined over a set of states and actions with rewards. Each Max node corresponds to a separate MDP.

[4]A node is only primitive if it has no children. Otherwise, the node is composite.

which can increase the complexity of the learning problem. However, the MAXQ approach can decompose the search and rescue task into smaller manageable subtasks that can be learned concurrently [3]. One main advantage of using MAXQ compared to other HRL approaches is that MAXQ requires less prior knowledge about the environment [2].

The MAXQ task hierarchy developed by University of Toronto Autonomous System and Biomechatronics Lab is presented in Figure 5[5]. Herein, the *Root* task defines the overall goal of the rescue robot team in a USAR mission. The *Root* task is further decomposed into the following subtasks: *Search Sub-scene, Navigate to Unvisited Regions, Victim Identification, Navigate*, and *Human Control* [3].

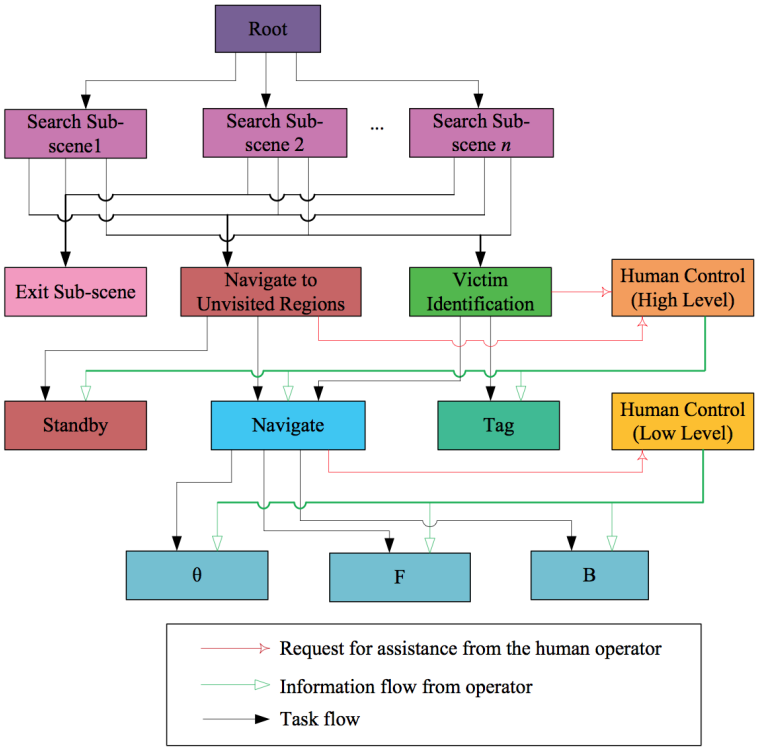


Figure 5: MAXQ Task Hierarchy for a Team of Rescue Robot [3]

The following lists the functions of each subtask [3]:

1. *Search Sub-scene* subtask breaks up the entire USAR map into smaller USAR scenes. This is done so that exploration with multiple robots can be done in parallel.
2. *Navigate to Unvisited Regions* subtask explores unvisited regions within the sub-scenes of the USAR environment via frontier-based exploration. The current

[5]The primitive actions θ, F, and B corresponds to rotate a robot by an angle θ, move a robot forward, and move a robot backward respectively.

3D map of the sub-scene can be decomposed into a 2D occupancy grid map for frontier-based exploration.

3. *Victim Identification* subtask identifies victims in sub-scenes.
4. *Navigate* subtask performs local navigation and obstacle avoidance. Rubble piles in USAR environment can be categorized into open cells, climbable obstacles, and non-climbable obstacles based on the slope and smoothness information obtained from the scene's depth profile [3].
5. *Human Control* subtask passes over control of the robots to the human operator when robots are unable to perform any of its tasks autonomously.

The MAXQ task hierarchy updates the value functions of the *Root* task and non-primitive subtasks recursively. This is accomplished in an online manner during a search and rescue task [3]. The following table summarizes the rewards of a robot state transition for Q-learning:

Subtask	**Robot state transition**	**Reward**
Root	The mission is completed successfully	+100
Search Sub-scene	Exit a sub-scene after it has been successfully explored	+50
Search Sub-scene	Exit a sub-scene when there are still accessible unknown cells	-10
Navigate to Unvisited Regions	Exit into Standby after exploring all unvisited regions in the sub-scene	+10
Navigate to Unvisited Regions	Exit into Standby when there are still accessible unvisited regions	-10
Victim Identification	Tag a victim correctly	+10
Victim Identification	False identification by tagging an object that is not a victim	-10
Navigate	Move into an unvisited region in the desired global exploration direction	+15
Navigate	Avoid an obstacle	+10
Navigate	Collide with an obstacle, a victim or another robot in the team	-20
Navigate	Repeatedly revisit an explored region	-1
Human Control	Human Control is requested when necessary	+10
Human Control	Human Control is unnecessarily requested	-10

Table 2: MAXQ Transition Rewards for Multi-Robot USAR

Simulation of this architecture has been previously conducted in a 2D environment. With a total of 500 training trials, the average percentage of scene explored is 93.9%, and the average percentage of victims identified is 99 % over a 100 randomly generated USAR map [11]. There has yet to be any performance tests done in a high-fidelity 3D simulation environment.

2.3 Terrain Categorization

Terrain Categorization is used in the *Navigate* subtask in order to classify areas in USAR environments into open cells, climbable obstacles, and non-climbable obstacles. There are many terrain categorization techniques reported in literature. Each technique varies with the type of sensor being used and its application.

With 3D laser range finders, the slope, obstacles, and roughness has been detected by calculating the terrain's surface normal. This has been tested on off-road terrain environments [12]. With image cameras, slope, roughness, and hardness of unstructured environments has been detected by using fuzzy logic and neural network systems [13]. With stereo vision cameras, plane-fitting techniques has been used to detect inclines, drops, and obstacles in urban areas [14]. With Kinect cameras, Fast Normal Calculation and static state binary Bayes filter has been used to detect slopes and height of platforms [15]. Kinect has also been used to detect inclination and roughness using statistical filters in indoor and outdoor environments [16].

2.4 Simulation Environment

In order to simulate USAR missions and test the performance of exploration algorithms, various computer platforms have been used. Some of these platforms are based on 2D simulations, while others are more realistic and use a 3D game engine.

MobileSim is a 2D simulation platform and has been used for testing autonomous navigation techniques, obstacle avoidance, and artificial intelligence with robot teams [10]. However, these simulation are two-dimensional and do not give a true representation of a real environment. In contrast, USARSim is a high-fidelity simulator that runs in a 3D game engine. Game engines are modular simulation code that can be used in creating a family of similar games [8]. For USAR missions to be as realistic as possible, USARSim uses an advanced 3D game engine known as Unreal Engine. This engine is the same engine used to build popular realistic games such as Unreal Tournament and Gears of War [8].

In the context of USAR applications, USARSim provides robot packages based on real-life models, and its virtual maps incorporate advance geometry and textures that faithfully simulate USAR environments [8]. In the past, USARSim has been used as the main platform for conducting performance evaluation in USAR robot competitions [4, 17]. Various algorithms to find victims in USAR environments have been implemented. Some of these algorithms include frontier-based exploration, and

SLAM [17].

USARSim is both feasible and cost-effective. It simulates a realistic environment for robots to behave in and interact with the environment in real time. In addition, it has the ability to simulate real USAR missions in a 3D environment. This provides the advantage of transferring a simulation robot's main program to a physical robot with minimum code changes and performance can be expected to be similar [8]. The simulator is cost-effective as it uses modular code and does not require research groups to develop their own high-fidelity testing environments.

For this thesis, USARSim has been chosen to be the simulation environment due to its advantages. Very few research use 3D environments to evaluate performance of exploration missions [4]. The potential for simulating robotic teams exploring USAR environments could open up many possibilities, such as finding an optimal robot team size for a particular exploration [10], and optimizing collaborative multi-robot exploration techniques. By using USARSim, no investment in physical hardware is required to investigate the performance of the MAXQ HRL algorithm in semi-autonomous USAR exploration.

3 Simulation of USAR Missions

This section will detail the simulation method used in order to test teams of robots in USAR environments.

3.1 Unified System for Automation and Robot Simulation

As stated in Section 2.4, the 3D high-fidelity simulation environment has been chosen to be USARSim[6]. The more realistic the simulation is, the more likely we are to obtain simulation results which are consistent with that of real-life tests. Unfortunately, the USARSim project is not funded anymore and lacks any up-to-date manual on working with the simulator. This section will be dedicated to the configurations chosen in USARSim for setting up the initial working environment where the MAXQ HRL architecture can be implemented.

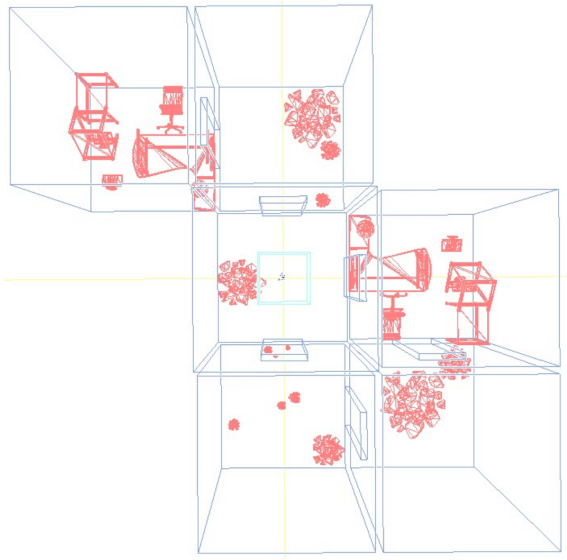
The USARSim environment allows teams of robots to be placed in predefined maps that accurately simulate real USAR environments. These robots are computer models that are currently divided into four different types: (i) legged robots, (ii) wheeled robots, (iii) submarine robots, and (iv) aerial robots. In this thesis, wheeled robots will only be considered as the agents in USAR missions. USARSim provides some useful predefined models of wheeled robots, such as the P2AT and the P3AT. This thesis will primarily use the P3AT as it is an improved model of P2AT and has been used in RoboCup Rescue competitions [17]. It also closely resembles the physical robots found in the Autonomous Systems and Bio-mechatronics Laboratory at University of Toronto.

3.1.1 Creating Urban Search & Rescue Maps

The actual graphical generation of USARSim is obtained using the Unreal game engine. In order to create the desired USAR maps in USARSim, maps are built from the Unreal game engine libraries. Fortunately, the Unreal Development Kit (UDK) provides support for rendering USAR maps into USARSim. In this thesis, USAR maps are created by obstructing an office environment with rubbles and collapsed structures. This method of creating test maps is appropriate, as real-world USAR maps have unleveled terrains, walls, and obstacles. Figure 6a illustrates an example of an USAR map generated in UDK. The collapsed office desk and rubbles in Figure 6a are typical scenarios resulting from an earthquake. In addition, the rubbles and furniture introduce both climbable and non-climbable obstacles to the default flat-surfaced office

[6]This thesis uses USARSim v.1.4 downloaded from http://sourceforge.net/projects/usarsim/files/usarsim-UDK/

environment. These unleveled terrains will eventually be analyzed by a team of robots to determine whether they are traversable. Figure 6b illustrates a rendered USAR office environment.



(a) Floor Plan of an USAR environment in UDK

(b) Rendered USAR Environment

Several maps have been created in this manner in order to test the scalability of the MAXQ HRL architecture. Table 3 shows the maps created for this thesis. The maps generated ranges from simple environment to gradually more complicated USAR-like environments. For instance, World 1 consist of an empty 4 by 4 units[7] hallway[8], which is surrounded by 4 walls placed on the boundary of the world. This structured environment serves to test the basic exploration functions of rescue robots, as this map is as simple as it can get. World 2 is an increment of complexity to World 1 as there are added rooms. Rooms present additional obstacles as each room is encapsulated by non-climbable walls and a small door opening. World 3 and World 4 are gradual increments of complexity to World 2 as they contain rubbles and collapse structures. These worlds are unstructured and model realistic USAR environments. The four worlds serve to evaluate the MAXQ HRL architecture in term of scalability. They also serve to test and debug the functions of the architecture during the implementation phase.

[7]Each unit is a room size; each room size is about 3 × 3 P3AT robot units
[8]Hallways in USARSim are open spaces (or rooms without walls as boundary)

World	Grid Size	Obstacles
World 1	4×4	Boundary Walls
World 2	4×4	Rooms
World 3	3×3	Rooms, Furnitures & Rubbles
World 4	3×4	Rooms, Furnitures & Rubbles

Table 3: USAR Environment Testing Maps

3.1.2 Interface with USARSim

In order to perform semi-autonomous missions in USARSim, an interface is required to communicate with USARSim, in accordance with the simulator's Application Programming Interface (API). The USARSim package contains a GUI for communicating with USARSim called Iridium. Iridium, however, lacks some key features that make incorporating the multi-robot MAXQ HRL architecture feasible.

1. Iridium does not allow one to teleoperate more than one robot at a time. This defeats the purpose of testing a multi-robot implementation of the MAXQ HRL algorithm.
2. Autonomous behaviour for robots is impractical to implement in Iridium. Iridium has the ability to run pre-defined scripts to control a robot. However, it lacks the ability to send real time feedback from sensors back to the user.
3. The GUI is difficult to use for teleoperated control. The GUI requires user to input low-level parameters such as controlling the speed of each individual wheel in order to move forwards, backwards, and turn.

For these reasons, Iridium does not fit the requirements in order to incorporate the multi-robot MAXQ HRL architecture. It is hard to find an interface with all the necessary features needed for this thesis. Hence, designing an interface with all the necessary features for this problem is imperative.

Interface Design Requirements

The interface...

1. shall be able to control any number of robots. This allows the MAXQ HRL algorithm to be implemented in more than one robot.
2. shall be capable of controlling robots through autonomous functions.
3. shall provide real-time feedback from robots' sensors to the user. This provides features such as camera viewing for teleoperated control.

4. shall be able to perform teleoperated control with any robot. Combined with Design Requirement 2, this provides semi-autonomous capabilities for robots to operate in.
5. shall provide an expandable exploration map applicable for any USAR scene. This serves to track the exploration performance of robots in various sized maps.

With these design requirements, a GUI was created to address the issues that were present in Iridium and to tackle the problem of incorporating MAXQ HRL architecture in USARSim. Figure 7 shows the designed GUI. This interface allows the user to select the robot he/she chooses to control. Teleoperational control is done through the arrow keys on the keyboard, where ← maps to turn left, → maps to turn right, ↑ maps to drive forward, and ↓ maps to drive backwards. The grid provided in the interface is the exploration grid, where it tracks the performance of frontier-based exploration. The grid is also able to expand when the robot exceeds the pre-defined grid size. Each square in the grid represents a robot unit[9]. The dimensions of the simulated robot is not important as grid size can be scaled up or down to give an accurate representation of the real-world.

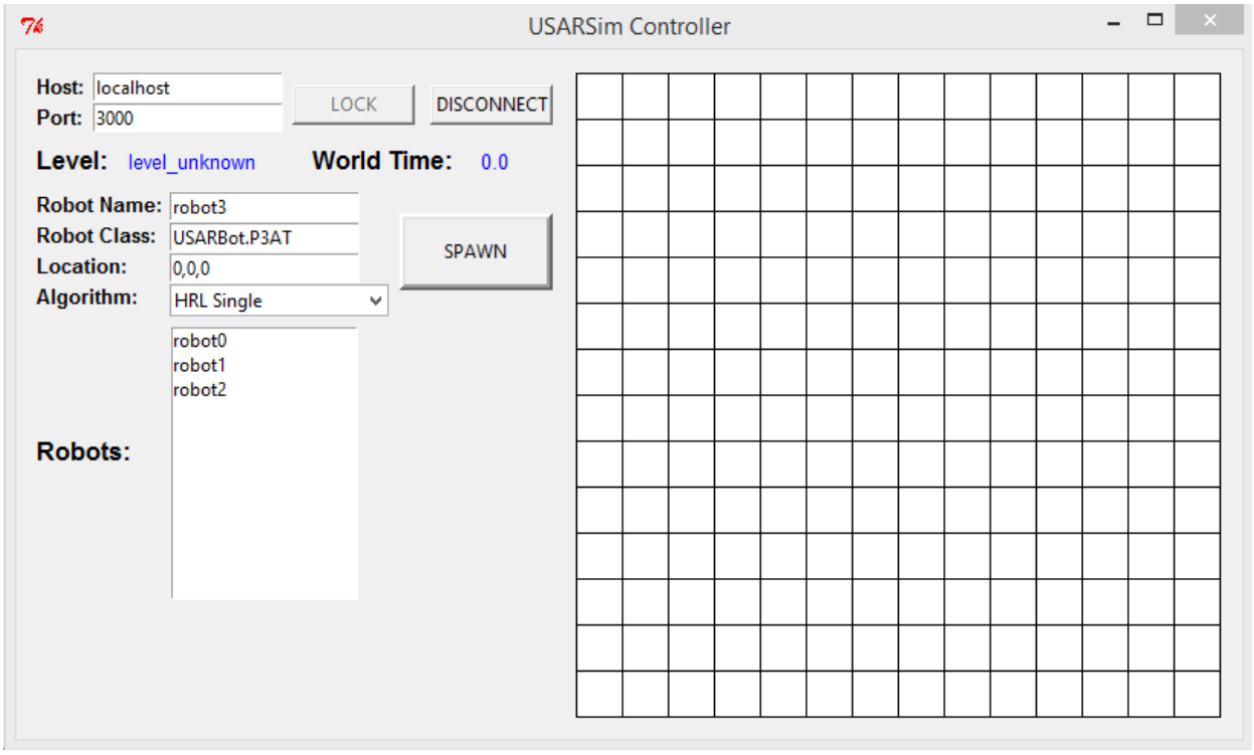


Figure 7: GUI for USARSim

Figure 8 shows the back-end communication between USARSim and the GUI, which is called *Controller Application* in the diagram. The *Image Server* provides the

[9]One robot unit is the size of the robot.

camera feedback to the application. *Gamebots* allows robots to be controlled through TCP/IP socket. Seemingly, USARSim is a low-level application that relies on socket data communication between client (controller) and host (game engine).

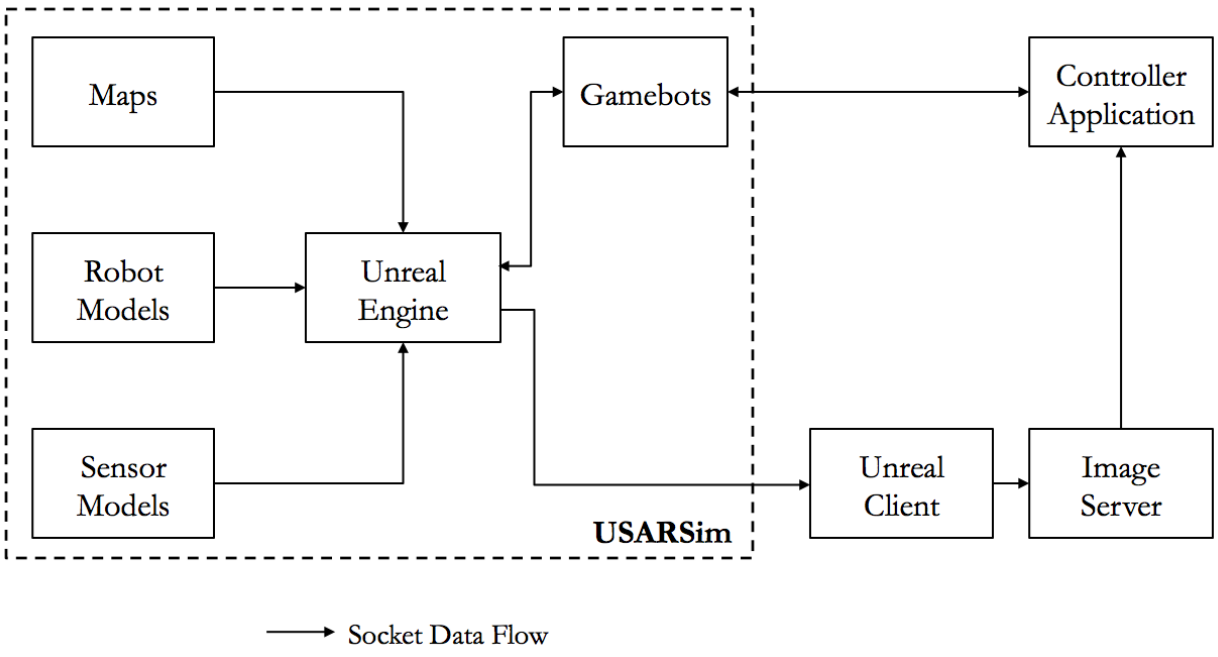


Figure 8: Back-end Communication in USARSim

The following socket commands were used to communicate with USARSim:

- `INIT {ClassName <classname>} {Location <location>} {Rotation <rotation>}`
  Spawns a robot in the desired position and orientation. This command is used to spawn a team of robots into the simulated USAR environment.
- `DRIVE {Left <left_value>} {Right <right_value>}`
  This command is used to initiate an action for the robot, such as driving the robot forwards, backwards, stopping the robot, and turning the robot.

3.1.3 Sensors

The P3AT may be equipped with a number of sensors available in libraries of USARSim. For this thesis, the P3AT robot has been chosen to be equipped with the following sensors:

1. **Range Scanner**: The Range Scanner uses a range sensor and emits one detection line to scan the environment. This means that the scanner is only limited to provide distance from nearby obstacles across a single plane.
2. **Inertial Navigation Sensor**: The Inertial Navigation Sensor (INS) estimates the robot's current location and orientation. Noise is added to the ground truth

in angular velocities and distance travelled per time step [8]. A Global Positioning System (GPS) was not chosen due to its reliance on location with signal coverage in the real world. USAR missions may happen anywhere in the world.

3. **Victim Sensor**: The Victim Sensor simulates a victim location sensor. The sensor sends out a number of traces and looks to see if they hit a victim.
4. **Camera**: The Camera provides a point of view in perspective of a robot. The Unreal Client is used for video feedback in USARSim. Camera is necessary for human control in the semi-autonomous exploration approach.
5. **Tachometer**: Tachometer measures the revolution per minute of the robot's wheel. This can provide information about how fast the robot is travelling at given time.
6. **Encoder**: Encoders measure the spin angle of the robot's wheels. This can provide information about how far a robot has travelled in a given time.
7. **Battery**: Battery gives a realistic time limit for the robot to be operational.
8. **Ground Truth**: This is a virtual sensor that exists in the simulator to provide true values about the robot's position and orientation. The values read off here are used to compare with the INS, Encoder, and Tachometer results to calculate the errors associated in the real world.

Ideally, the robot would be equipped with Kinect sensors, rather than range scanners. This is due to the fact that Kinect sensors can return a depth profile of a terrain. The Kinect returns a point cloud consisting of (x, y, z) coordinates belonging to a spatial area. The point cloud provides the capability to display maps in the form of 3D surfaces. With the Kinect, robots can acquire a depth map of the USAR environment, which enables them to categorize areas as climbable, non-climbable and open cells in the occupancy grid. This is due to the sensor's capabilities to provide the inclination and roughness of the surrounding environment. However, multiple trials have been attempted on implementing the Kinect sensor onto the robot in simulation with little success[10]. Due to the discontinued support from research groups and the community for USARSim, implementing the Kinect sensor was abandoned.

[10]Null values are read from the Kinect sensor output. Using `SET {Type RangeImager} {OPCODE SCAN}` in USARSim activates the scan function for the Kinect sensor, but the values outputted could not be decoded. No documentation was found on how to use the Kinect sensor in USARSim

3.1.4 Creation of Depth Profiles

Alternatively, a depth profile can still be created in simulation by using range scanners. In order to create a full depth profile of the robot's surroundings, 4 range scanners were used. Each range scanner was placed 30 cm above the floor in the middle of the robot oriented at facing $\theta = \{0, \frac{\pi}{2}, \pi, -\frac{\pi}{2}\}$ relative to the robot. Figure 9 illustrates the robot's orientation.

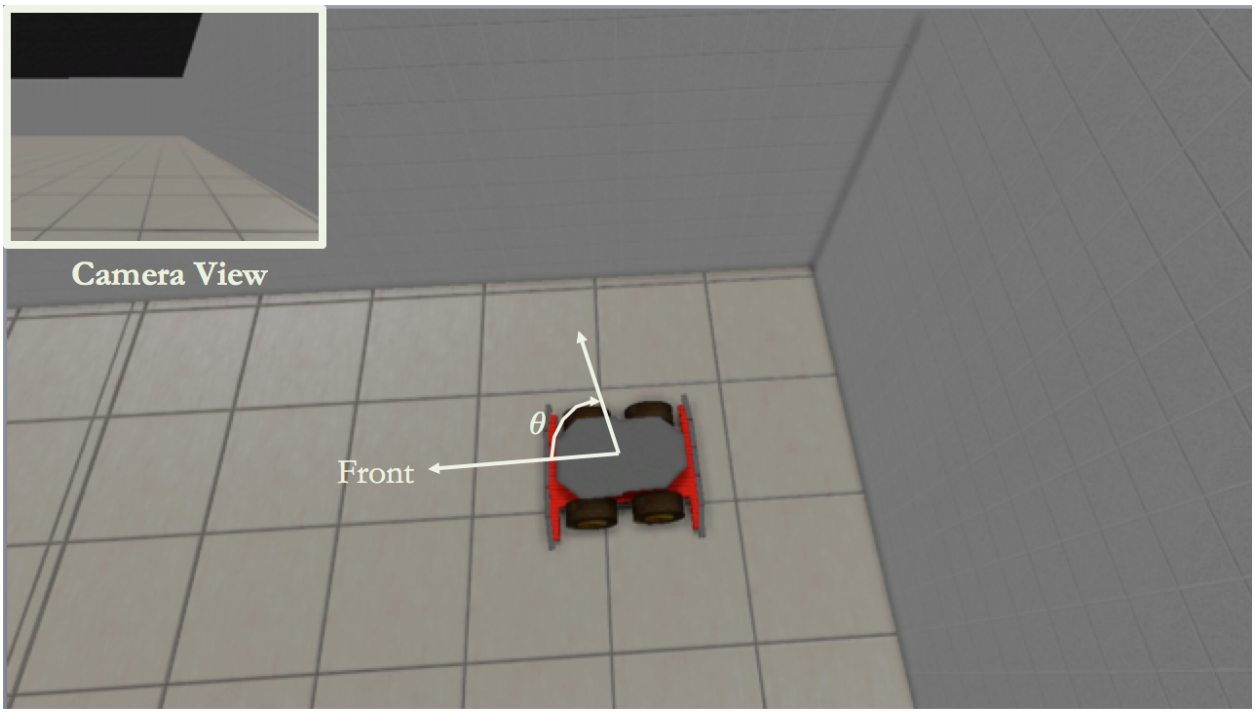


Figure 9: Robot Orientation

In order to capture the depth of the surrounding areas, the range scanners scanned vertically, as oppose to horizontally, with a field of view of $\frac{2\pi}{3}$ and a resolution of 100 points. This configuration records points from 17 cm ahead of the robot on the ground (relative to the middle of the robot). Figure 10 illustrates this configuration. The depth of a plane in each direction can be capture this way. For capturing multiple planes of depth, the range scanner must rotate. USARSim has pan-tilt functionality for some robots, such as a submarine, QRIO (humanoid), and Soryu (robot dog), but it lacks this feature for the P3AT. To compensate, the robot rotates to scan ± 10 degrees (a field of view of 20 degrees) scanning at each degree. As a result, the depth image is 20×100 points. The field of view, 20 degrees, was chosen because it covers the adjacent neighbour cells surrounding the robot.

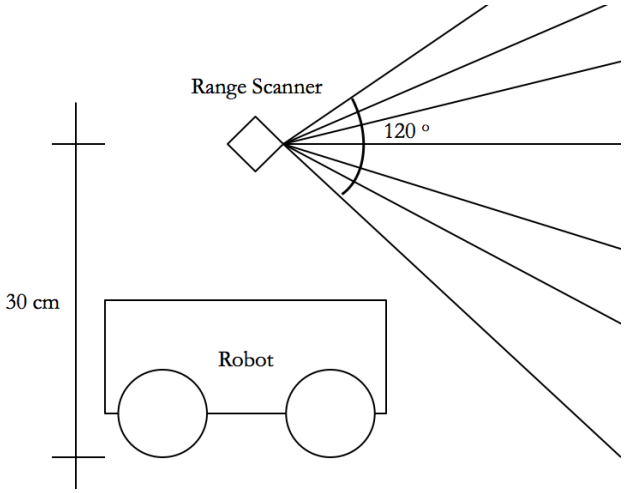
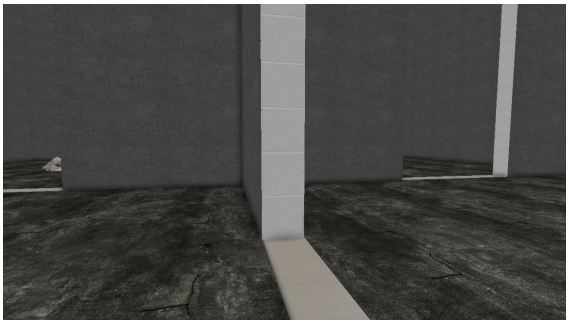
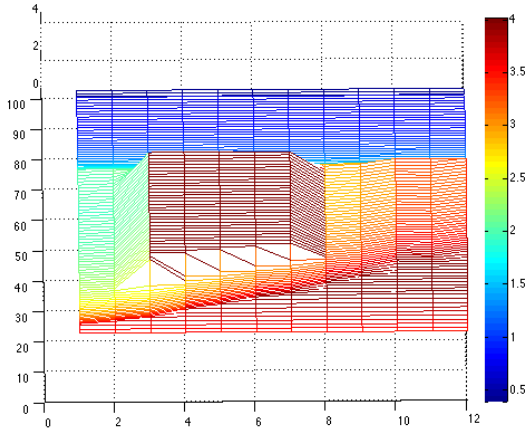


Figure 10: Range Scanner Configuration



(a) Surrounding Environment



(b) Depth Image reproduced by Range Scanner

In order to verify and validate this technique, depth points that were recorded during scan were then plotted to recreate the depth image. Figure 11b shows the recreated image of Figure 11a. The depth graph is not to scale because it assumes each point recorded by the range scanner is uniformly spaced. The red-orange regions indicate objects that are close. In Figure 11b, the pillar-like structure is apparant. Red regions are also at the bottom of the graph because the range scanner also scans the ground. This is used for detecting drops and obstacles. There is a sudden blue region due to the sensor's range limit. This most likely indicates areas close to the room's ceiling. A limitation to this method is that there is a trade-off between scan speed and resolution. The depth graph may not look like the real environment due to the scan resolution. In order to improve on this, scans should be recorded more frequently than every degree of the robot's rotation, However, this affects the scan speed. For USAR

missions, the current settings take approximately 10 seconds to scan surrounding cells. Since every cell needs to be scanned for exploration to happen, anymore time allocated for scanning would be time-consuming. The depth image generated by using the range scanner's data is then processed by a terrain categorization method (see Section 4.1).

3.1.5 Robotic Movement

From literature review, the results from using MAXQ HRL in a 2D grid assumes perfect movement from cell to cell with no errors [11]. In 3D exploration, errors can accumulate as the robot can be off from the center of cells. In order to prevent accumulation of errors, the robot's position and orientation were recorded using the INS. The following list the procedures used for linear motion and rotational motion of the robot.

Linear Motion (Forwards or Backwards)

1. Drive both wheels at the same speeds.
2. Check for deviation:
   - If robot deviates to the right, rotate to the left and continue linear motion.
   - If robot deviates to the left, rotate to the right and continue linear motion.
3. Stop all wheels when target position (next cell) is reached.

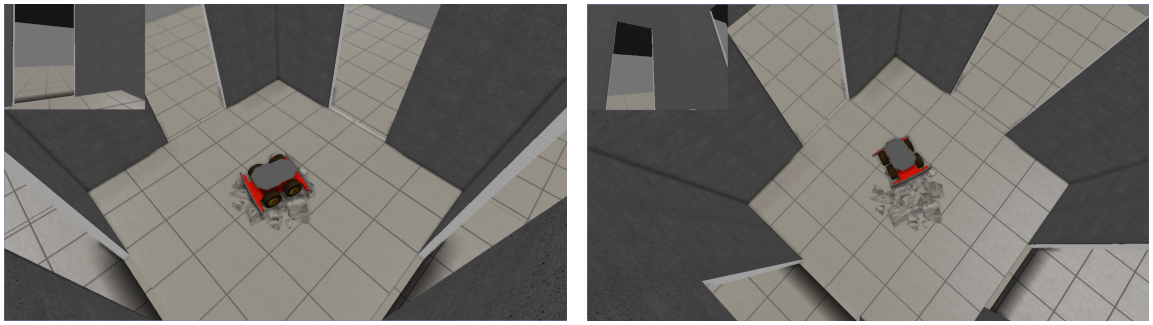
Rotational Motion (Clockwise or Counterclockwise)

1. Determine desired orientation[11]:
   - FRONT: 0 rad
   - RIGHT: $\frac{\pi}{2}$ rad
   - BACK: π rad
   - LEFT: $-\frac{\pi}{2}$ rad
2. Drive opposite wheels with opposite speeds
   (i.e. `DRIVE {Left 0.3} {Right -0.3}`).
3. Stop all wheels when target orientation is reached.

The INS is fairly accurate and the errors are insignificant when compared to the ground truth. The method used above is independent on the robot's current position and orientation and depends entirely on the feedback of the INS. This prevents error propagation as any new movement will be adjusted according to the INS reference frame, and not based on the previous robot pose. Figure 12 shows an example of a robot rotating in place on top of a rubble. There is error in robotic movement as the

[11]The choices of angles follows the convention of USARSim.

robot also moves out of its initial position due to the uneven surfaces of the rubble. This is a potential problem for creating depth images as rotational motion is used when the robot wants to record depth images of the surrounding cells. However, since the rotation is only ±10 degrees, the errors in the depth image are not significant and this method suffices. Depth images do not have to have perfect accuracy as they are only used to categorize climbable and non-climbable cells during terrain categorization.



(a) Start Rotation (b) End Rotation

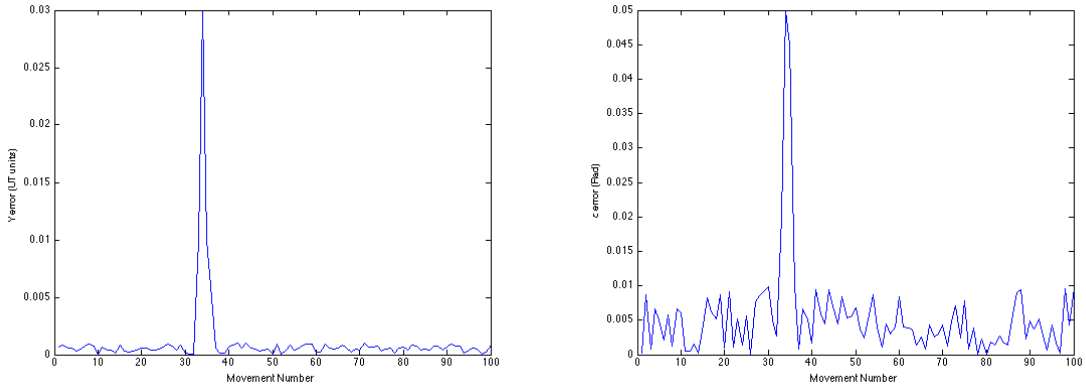
Figure 12: Example of Errors in Robotic Movement

Errors are higher after each movement with higher wheel speeds. Different wheel speeds have been tested to minimize error in robotic movement. The 2D errors in position and orientation are recorded in Figure 13[12] with the best-found wheel speed configurations (Table 4). The 2D errors were recorded across 100 movements, where one movement is equivalent to a robot's action (i.e. move forward, rotate right). In the 100 movements, the robot travelled in a straight line on a flat surface until it hit a climbable rubble at movement number 33. After going over the obstacle, the robot turned right at movement 58 on a flat surface, and continued moving forwards. As the error plots show, the robot deviates when it climbs over an obstacle and attempts to adjust itself after (relative to the INS frame of reference). It takes the robot a few movements to adjust itself back on track and the errors became relatively low again. Thus, this method attempts to minimizes error propagation.

Movement	**Left speed [rad/s]**	**Right speed [rad/s]**
Forward	0.3	0.3
Backward	-0.3	-0.3
Rotate Right	-0.3	0.3
Rotate Left	0.3	-0.3

Table 4: Best-found Robot Wheel Speed Configurations

[12] X position and β angle plots have the same behaviour. Also, 1 cm = 0.525 UT units.



(a) Y position (b) α angle

Figure 13: Self-correction of Errors in Position and Orientation after Climbing a Rubble

4 MAXQ Task Hierarchy Implementation

The MAXQ task hierarchy consists of the following tasks: *Root, Search Sub-scene, Navigate to Unvisited Regions, Victim Identification, Navigate* and *Human Control*, as described in Section 2.2.3.1. For each of these subtasks, a state function is defined [2]. These state functions are implemented into the robot controller for USARSim.

1. **Root**: The MAXQ state function is defined as $S(V, S_s, M_{xyz})$ where V represents the presence of potential victims in sub-scene, S_s denotes the status of the sub-scenes, and M_{xyz} represents the 3D maps of USAR.
2. **Search Sub-scene**: The MAXQ state function is defined as $S(V_i, L_R, M_{xyz,i})$, where V_i represents presence of victims in sub-scene, L_R is the robot's location with respect to the local coordinate frame[13], and $M_{xyz,i}$ represents the 3D map of the USAR sub-scene the robot is exploring.
3. **Navigate to Unvisited Regions**: The state definition is $S(L_R, M_{xyz,i})$, where L_R and $M_{xyz,i}$ have the same definition as above. This is where the current 3D map of the sub-scene is decomposed into a 2D grip map for frontier-based exploration.
4. **Victim Identification**: The state is defined as $S(L_{V/R}, M_{xyz,i})$, where $L_{V/R}$ represents the locations of potential victims in the scene with respect to the robot's location and $M_{xyz,i}$ is the same as above. This is where tagging the location of the victim within the 3D map of the sub-scene happens.
5. **Navigate**: The state definition is $S(C_j, D_E, D_{xy}, L_{V/R})$, where C_j represents grid map information for the 8 surrounding cells of the robot, D_E corresponds to the desired exploration direction, D_{xy} contains the depth profile information of rubble pile in the surrounding environment, and $L_{V/R}$ is the same above. This is where rubble piles are categorized into open cells, climbable obstacles, and non-climbable obstacles. The categorization is based on the slope and smoothness information obtained from their depth profile, which is captured by the Kinect sensor.
6. **Human Control**: No state definition is required as the human is in control of the robots.

Luckily, most of the implementation already exist for a 2D map application[14]. The next step is to implement the architecture and the required 3D features into the robot's controller in USARSim. The following subsections details the implementations of key algorithms used in the robot's controller.

[13]Local coordinate frame of a robot is defined to be at the location at which the robot enters the sub-scene

[14]MAXQ HRL has been previously implemented by the University of Toronto Autonomous Systems and Biomechatronics research group [11].

4.1 Terrain Categorization

Terrain Categorization is used in the *Navigate* subtask in order to classify areas in USAR environments into open cells, climable obstacles, and non-climbable obstacles. A simple plane fitting method was implemented into the robot's controller. The heights of the points in the voxel grid were extracted from depth images and then fitted into a plane using the least squares method.

$$\begin{bmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i y_i & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i y_i & \sum_{i=1}^{n} y_i^2 & \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} y_i & \sum_{i=1}^{n} 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} x_i z_i \\ \sum_{i=1}^{n} y_i z_i \\ \sum_{i=1}^{n} z_i \end{bmatrix} \tag{6}$$

$$R = \sum_{i=1}^{n} |z_i - (Ax_i + By_i + C)| \tag{7}$$

Values of A, B, C were solved for in Equation (6) and the residual, or accumulated error of each point from the plane, was used as the roughness parameter, R, in Equation (7). The value, B, represents the slope. This method was tested on various terrains ranging from climbable obstacles, such as small rubbles, to non-climbable structures, such as walls and large furnitures. The results of the experiment are shown in Table 5.

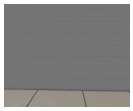
Terrain	Cell	Slope	Roughness
Flat Plane		0.03	0.54
Wall		4.54	0.62
Small Rubbles		0.21	2.72
Large Rubbles		1.37	2.42
Office Chair		3.75	1.65

Table 5: Terrain Categorization Test Results

The results in Table 5 are reasonable as the slope and residuals from plane fitting corresponds to the obstacle in the cell of interest. For instance, the flat plane has a relatively small slope and low roughness value. These parameters can serve as indicators

that the scanned cell is an open cell. In comparison, the wall has a relatively high slope and low roughness. The high slope parameter indicates that there is a large obstacle in the cell and is most likely non-climbable. The small rubbles and large rubbles have higher roughness values than other terrains as their surfaces are not uniform within the cell. Lastly, the office chair has both moderately high slope and roughness values. The roughness value and the slope parameter, in combination, can indicate whether a cell is open, climbable (obstacle), non-climbable (obstacle). The threshold for these values are determined by the MAXQ HRL algorithm and are adjusted as the robot learns to optimize its own behaviour. In other words, the *Navigate* subtask calibrates the threshold values of both R and B, in Equation (6)-(7), in order to categorize areas in the terrain as open space, climbable obstacle, and non-climbable obstacle.

Figure 14 shows a logic flow diagram of the terrain categorization implementation, where $D_{xy} \in \{$`obstacle, drop, downhill, climbable, open`$\}$ is the depth map of the entire USAR scene.

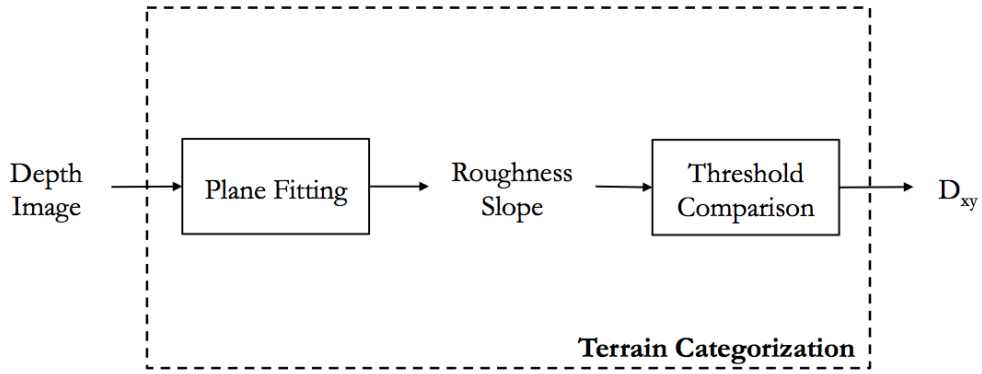


Figure 14: Terrain Categorization Implementation

4.2 Rewards

A more detailed version of the rewards system stated in Table 2 is implemented. In order for the robot to determine its next action, the MAXQ value function, Equation (3)-(5), is evaluated and is entirely dependent on the rewards system. Table 6 shows the general nature of the implementation of the rewards system, where smart decisions yield positive rewards and unfavourable decisions yield negative rewards. The implementation takes into account a combination of cases, such as moving into an unvisited area but hitting an obstacle along the way, and returns a net reward value to be used for evaluating the MAXQ value function.

Task	Action	Reward
Navigation	Hit obstacle	-
	Move to open space	+
	Hit victim	-
	Drop	-
	Move to unvisited area	+
	Move to visited area	-
	Climb obstacle	+
	Avoid obstacle	+
	Avoid open space	-
	Avoid climbable area	-
Sub-scene Searching	Exit	+
Victim Identification	Tag correctly	+
	False identification	-
	Tag with low certainty	-
	Tag with high certainty	+

Table 6: Rewards Implementation

4.3 Obstacle Avoidance

Obstacle avoidance can optimize a robot's behaviour and can lead to faster exploration. For this thesis, a simple obstacle avoidance algorithm is implemented. The robot controller first detects if there exist any concave boundary obstacle in the direction of travel. It does this by checking for continuous row of obstacles from its mapping. If there is a concave boundary obstacle in the direction of travel and no unexplored cells inside, then that region can be avoided. If there are still unexplored cells inside, then the robot needs to explore them. Figure 15 shows the logic flow diagram for the obstacle avoidance algorithm. The algorithm checks for continuity from the left most obstacle to the right most obstacle. If there are open spaces in between, then the result is not a concave obstacle boundary.

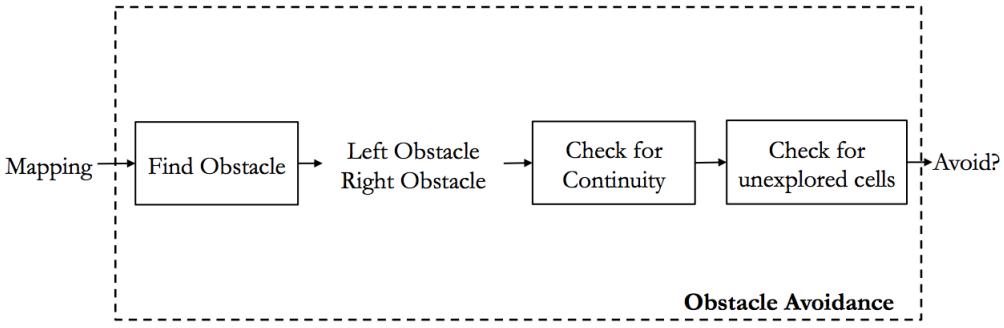


Figure 15: Obstacle Avoidance Implementation

Obstacle avoidance is used in exploration. In exploration, the robot travels in the direction of unexplored cells via frontier-based exploration, while avoiding explored concave obstacle boundaries.

5 Results

The following section details the results of the year-long thesis. Unfortunately, the MAXQ HRL performance could not be evaluated due to time constraints. However, the thesis did accomplished setting up the framework for the evaluation.

5.1 The Framework

The framework for evaluating MAXQ HRL has been developed. First, USAR maps have been created, ready to be tested in, in USARSim. Second, the GUI for controlling and evaluating the robot has been developed. Lastly, the MAXQ HRL architecture for 3D environments has been implemented in the robot's controller. Figure 16 shows the overall communication framework between all the modules developed for this thesis. Variables, F, B, θ are robot movements outputted by the MAXQ HRL controller and variables, `Depth Image F, Depth Image R, Depth Image L, Depth Image B, INS`, are sensing information outputted by the robot in the USARSim environment.

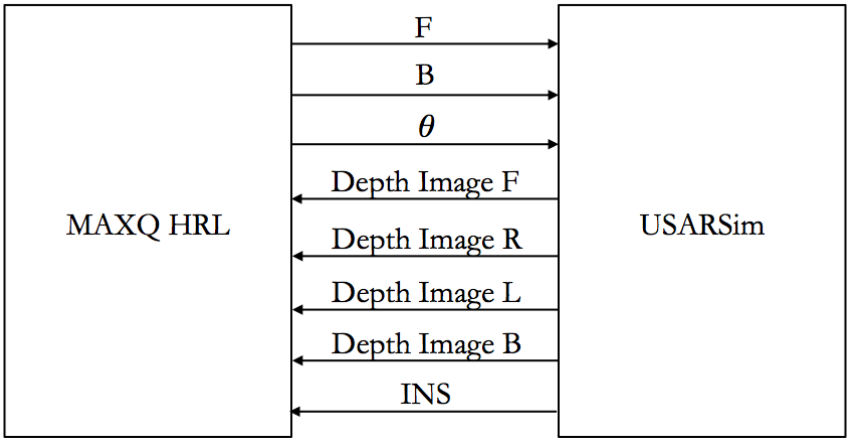


Figure 16: Communication Framework

5.2 Teleoperated Performance

Teleoperated performance of the exploration in USAR environments have been recorded using the robot's camera and the developed GUI. The teleoperated method is used as a bench-mark for comparison with MAXQ HRL because it is assumed that a human controller has better spatial awareness and will be more efficient in exploration while avoiding complex obstacles. In addition, current USAR missions use mostly teleoperated robots. The USAR environments that have been used for testing are from Table 3.

World	Grid Size	Obstacles	Time (s)	Explored Area (%)
World 1	4×4	Boundary Walls	432	100%
World 2	4×4	Rooms	526	100%
World 3	3×3	Rooms, Furnitures & Rubbles	920	93.8%
World 4	3×4	Rooms, Furnitures & Rubbles	1226	92.5%

Table 7: Teleoperated Performance

World 1 was a large empty map with 4 boundary walls surrounding the world. It was a relatively easy exploration for the teleoperator as most of the surrounding environment can be seen via visual feedback. Adding rooms introduced additional walls as unclimbable structures, as in World 2. Even though the grid size of World 2 was the same as World 1, extra time was required to maneuver through obstacles. World 3 had higher complexity than the first two worlds. The complex obstacles were difficult to maneuver through via camera. Moreover, there was a potential to get stuck if the teleoperator was not careful. Figure 17 illustrates the floor plan for World 3. A significant amount of time was spent on exploration maneuvering through doors and avoiding obstacles. Only 76/81 cells were explored as the robot got stuck on one of the furnitures. World 4 was similar to World 3, but larger. It significantly took more time than any other world due to the larger map and unstructured environment. The teleoperator repeatedly moved the robot to already visited cells to check for missing paths in order to finish exploration. This was an inefficient approach, as shown in Figure 18, which illustrates World 4's teleoperated performance over time. The exploration finished in World 4 with 100/108 cells explored. The teleoperator could not find anymore unexplored cells during exploration.

From Figure 18, it could be seen that the majority of the exploration occured at the beginning. After about 50% of the map had been explored, the teleoperator struggled to find unexplored areas and searched through visited areas to find other possible paths that may lead to more unexplored paths. Evidently, teleoperated approached become less effective over time. Most of the time was spent searching through already visited areas. After about 1000 seconds, the teleoperator believed the exploration was complete and gave up searching for anymore unexplored cells.

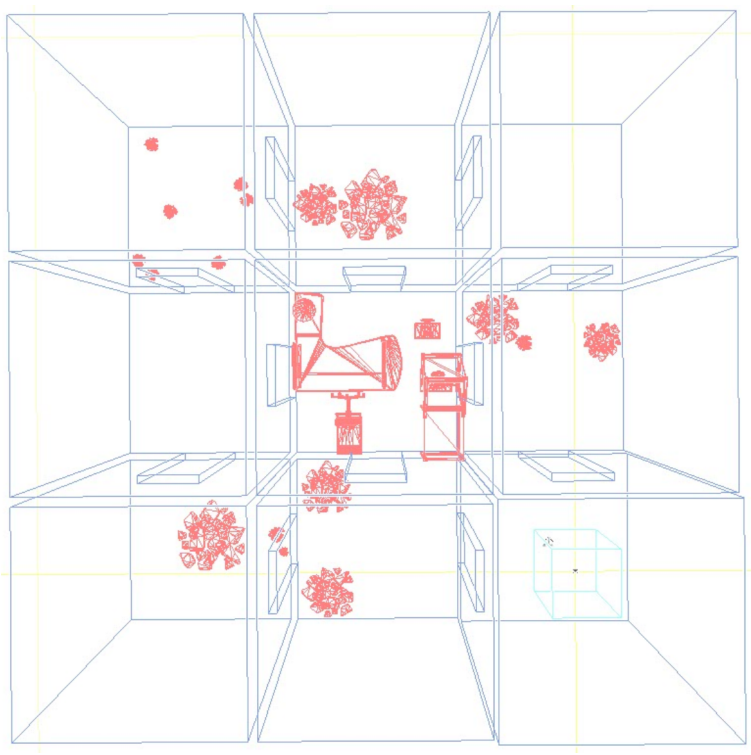


Figure 17: World 3 Floor Plan

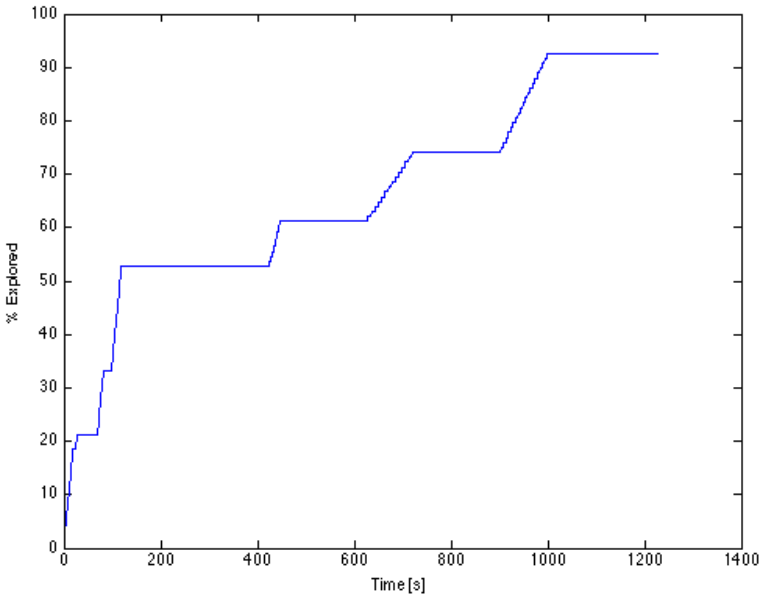


Figure 18: Teleoperated Performance in World 4

6 Future Work

This thesis lays the foundations for evaluating MAXQ HRL in USARSim. The system architecture allows multiple agents to be inserted onto a virtual map. This section indicates possible future work in the area.

6.1 MAXQ Hierarchical Learning in USARSim

Several approaches can be done to evaluate the performance of MAXQ HRL. The following are a list of tests to evaluate MAXQ HRL with different configurations.

1. Single robot autonomous exploration in structured environments[15].
2. Single robot semi-autonomous exploration in non-structured environments with human assistance for learning.
3. Multi-robot autonomous cooperative exploration in structured environments.
4. Multi-robot semi-autonomous cooperative exploration in non-structured environments with human assistance for learning.

The 3D implementation of the MAXQ HRL will use the same evaluation metrics as the 2D implementation [3]. These metrics are the control mode (teleoperation vs. semi-autonomous), average trial time, average percentage of scene explored, average percentage of victims identified, and number of collisions[16]. This will give us a clear indication of the architecture's performance, and conclusions can be drawn about the feasibility of MAXQ HRL in USAR applications.

6.2 Robotic Movement Improvement

The current implementation of robotic movement is simple and takes a few movements to converge to proper position and orientation after climbing rubbles (see Section 3.1.5). In reality, USAR scenes have uneven surfaces and does not provide flat surfaces for convergence of proper pose. To try and improve this limitation, other control architectures can be implemented to predict the deviation of the robot when it travels over rubble. This would lower the errors and the chances of error propagation in robotic movement.

[15] Structured environments have little to no obstacles

[16] A collision is when a robot is stuck and unable to perform exploration (i.e. flipping over, stuck in rubble)

6.3 Teleoperated Performance Evaluation

The author of this paper was chosen as both the map designer and teleoperator. This generated biased results, as the teleoperator had prior knowledge of the environment. This problem could be solved by recruiting more teleoperators for testing, as done in previous research studies [3].

7 Conclusion

The MAXQ HRL implementation for USARSim has been developed and is ready to be used to evaluate its performance in scene exploration and victim identification. Conclusions can be drawn when comparing its performance with teleoperated controls. Using this method, one can verify that using learning-based semi-autonomous controllers are feasible architectures for USAR missions. It should take less time to explore complicated environments with higher exploration coverage. In this thesis, the framework for interacting with USARSim is developed and USAR missions are simulated using various USAR designed maps. A depth profile creation algorithm is implemented and is later used for terrain categorization. Furthermore, a robotic movement controller is tested for error convergence. Moreover, teleoperated performance are recorded through various USAR scenarios. Many future work lies ahead in this field and MAXQ HRL seems promising. It is hopeful that rescue robot will be able to intelligently explore unstructured USAR environments in the future and assist rescue workers for faster response time.

References

[1] MnemStudio. (2012) A painless q-learning tutorial. [Online]. Available: http://mnemstudio.org/path-finding-q-learning-tutorial.htm

[2] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of Artifical Intelligence Research*, pp. 227–303, 2000.

[3] Y. Liu, G. Nejat, and B. Doroodgar, "Learning based semi-autonomous control for robots in urban search and rescue," *IEEE, International Symosium*, 2012.

[4] Y. Liu and G. Nejat, "Robotic urban search and rescue: A survey from the control perspective," *Journal of Intelligent and Robotic Systems*, March 2013.

[5] S. Thrun, "Simultaneous localization and mapping," *Springer-Verlag*, 2008.

[6] B. Yamauchi, "A frontier-based approach for autonomous exploration," *IEEE*, pp. 146–151, July 1997.

[7] B. Yamauchi "Frontier-based exploration using multiple robots," *Autonomous Agents*, pp. 47–53, 1998.

[8] J. Wang, M. Lewis, and J. Gennari, "A game engine based simulation of the nist urban search and rescue arenas," in *Winter Simulation Conference*, 2003, pp. 1039–1045.

[9] J. Vilela, Y. Liu, and G. Nejat, "Semi-autonomous exploration with robot teams in urban search and rescue," *IEEE, International Symposium*, 2013.

[10] J. Vilela, "Exploration in urban search and rescue environments with robot teams," *ESC499Y, University of Toronto Engineering Science Thesis*, 2013.

[11] Y. Liu, G. Nejat, and J. Vilela, "Learning to cooperate together: A semi-autonmous control architecture for multi-robot teams in urban search and rescue," *IEEE, International Symposium*, 2013.

[12] J. Larson, M. Trivedi, and M. Bruch, "Off-road terrain traversability analysis and hazard avoidance for ugvs," *IEEE Intelligent Vehicles Symposium*, 2013.

[13] A. Howard and H. Seraji, "Vision-based terrain characterization and traversability assessment," *J. Robotic System*, 2001.

[14] A. Murarka and B. Kuipers, "A stereo vision based mapping algorithm for detecting inclines, drop-offs, and obstacles for safe local navigation," *IEEE/RSJ International Conference on Robotics and Intelligent Systems*, pp. 1646–1653, 2009.

[15] I. Bogoslavskyi, O. Vysotska, J. Serafin, G. Grisetti, and C. Stachniss, "Efficient traversability analysis for mobile robots using the kinect sensor," *Winter Simulation Conference*, 2013.

[16] M. Bellone, G. Reina, N. I. Giannoccaro, and L. Spedicato, "Unevenness point descriptor for terrain analysis in mobile robot applications," *International Journal of Advanced Robotic Systems*, vol. 10, 2013.

[17] G. Lakemeyer, E. Sklar, D. G. Sorrenti, and T. Takahashi, "Development of an autonomous rescue robot within the usarsim 3d virtual environment," *RoboCup 2006: Robot Soccer World Cup X*, pp. 491–498, 2007.

A Pseudo-Code of Algorithms

This section details psuedo code of some algorithms found in literature.

A.1 Q-Learning

```
function Qlearn

Set gamma;
Set environment rewards for matrix R;
Initialize matrix Q to zero;
for each episode:
    Select random initial state;
    do while goal state hasn't been reached:
        Select one among all possible actions for current state;
        Using this possible action, consider going to next state;
        Get maximum Q value for this next state based on all possible actions;
        Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)];
        Set the next state as the current state.
    end do
end for

end Qlearn
```

A.2 MAXQ Hierarchical Reinforcement Learning

A.2.1 Execution of hierarchical policy

```
function ExecuteHierarchicalPolicy(Policy pi)

Let time t = 0;
Let K_t (state of the execution stack at time t) = empty;
s_t (state of the world at time t) is observed;
push(0,NULL) onto stack K_t; // invoke the root task with no parameters
do while K_t+1 != empty
    while top(K_t) != primitive action
        Let (i,f_i):=top(K_t);
        // i is the name of the current subroutine
        // f_i gives the parameter bindings for i
        Let (a,f_a):=pi_i(s,f_i);
```

```
        // a is the action
        // f_a gives the parameter bindings chosen by policy pi_i
        push(a,f_a) onto stack K_t;
    end while
    Let (a,NULL):=pop(K_t) // primitive action on the top of the stack
    Execute primitive action a;
    Observe s_t+1;
    Receive reward R(s_t+1 | s_t,a)
    if any subtask on K_t is terminated in s_t+1 then
        Let M' be the terminated subtask that is highest on stack
        // highest is closest to the root
        while top(K_t)!=M'
            pop(K_t);
        end while
        pop(K_t);
    end if
    K_t+1:=K_t is the resulting execution stack
end do

end ExecuteHieraarchicalPolicy
```

A.2.2 MAXQ learning algorithm

```
function MAXQ(MaxNode i, State s)

let seq = {} be the sequence of states visited while executing i
if i == primitive MaxNode
    execute i;
    receive r; // rewards
    observe result state s'
    V_t+1(i,s):=(1-alpha_t(i)) * V_t(i,s) + alpha_t(i) * r_t;
    push s onto beginning of seq;
else
    let count = 0;
    do while T_i(s) == false
        choose action a according to current exploration policy pi_x(i,s);
        let childSeq = MAXQ(a,s);
        //childSeq is sequence of states visited while executing action a
        //in reverse order
```

```
        observe result state s';
        let a* = argmax_a' (C_t(i,s',a') + V_t(a',s'));
        let N = 1;
        for each s in childSeq do
             C_t+1(i,s,a):=(1-alpha_t(i))*C_t(i,s,a)+
                         alpha_t(i)*gammaˆN*(R_i(s')+C_t(i,s',a*)+V_t(a*,s));
             C_t+1(i,s,a):=(1-alpha_t(i))*C_t(i,s,a)+
                                      alpha_t(i)*gammaˆN*(C_t(i,s',a*)+V_t(a*,s'));
             N:=N+1;
        end for
        append childSeq onto the front of seq;
        s:=s';
    end while
end if
return seq;

end MAXQ
```

A.2.3 Greedy execution of MAXQ graph

```
function EvaluateMaxNode(i,s)

if i == primitive Max node
    return <V_t(i,s),i>;
else
    for each j in A_i
        let <V_t(j,s),a_j>=EvaluateMaxNode(j,s);
        let jˆhg = argmax_j(V_t(j,s)+C_t(i,s,j));
        return <V_t(jˆhg,s),a_jˆhg>;
    end for
end if

end EvaluateMaxNode
```

B Thesis Project Plan

The following illustrates the work done for this thesis throughout the year.