A HYBRID METHOD FOR DISTRIBUTED MULTI-AGENT MISSION PLANNING SYSTEM

A Thesis

Submitted to the Faculty

of

Purdue University

by

Nicholas S. Schultz

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Aeronautics and Astronautics

May 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Shaoshuai Mou, Chair

School of Aeronautics and Astronautics

Dr. Dengfeng Sun

School of Aeronautics and Astronautics

Dr. Inseok Hwang

School of Aeronautics and Astronautics

Approved by:

Dr. Gregory Blaisdell

Head of the Graduate School Program

I dedicate my thesis work to my family and friends. A special feeling of gratitude to my loving parents whose never-ending support helped me throughout each semester. To my brother who is always there for me when I need him. I also dedicate this thesis to all the friends I've had the privilege of making at Purdue. I have great memories from this place and you all have helped make this an invaluable experience.

ACKNOWLEDGMENTS

First, I would like to acknowledge my advisor, Dr. Shaoshuai Mou. A special thanks to him for his advice and continued support of my project.

Second, I would like to acknowledge my lab mates, Mark Duntz, Arthur de Waleffe, Kevin Shi, Wanxin Jin, Xuan Wang, Paolo Heredia, Jeff Hall, and Daniel Liang. I am very grateful to you all for your help and support for my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

SYMBOLS

A	connectivity matrix
a	MDP action
c	task profit
Δ_G	communication graph diameter
$\mathbb{E}$	expectation
ϵ	number specifying criteria
f	objective function
G	environment graph
γ	discount factor
$L(\theta)$	loss function
λ	discounted embark time
m	agent
n	task
P	task bundle
π	MDP policy
$Q(s, a)$	Q-value
R	MDP reward
r	reward
s	MDP state
t	time-step
τ	traversability index
θ_i	neural network parameters
$V(s)$	value function
W	waypoints
X	state space

x	x coordinate position
y	y coordinate position
ζ	situation assessment error

ABBREVIATIONS

AC	Actor-Critic
A2C	Advantage Actor-Critic
AI	Artificial Intelligence
CBBA	Consensus-Based Bundle Algorithm
CBCA	Consensus-Based Collision Avoidance
CBS	Conflict-Based Search
DEM	Digital Elevation Model
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
GA	Genetic Algorithm
GUI	Graphical User Interface
IACA	Iterated Auction-Consensus Algorithm
LiDAR	Light Detection and Ranging
MAPP	Multi-Agent Path Planning
MDP	Markov Decision Process
MILP	Mixed-Integer Linear Programming
MPS	Mission Planning System
MRTA	Multi-Robot Task Allocation
PRM	Probabilistic Roadmaps
RRT	Rapidly-Exploring Random Trees
SA	Situation Assessment
SAR	Search and Rescue
SLAM	Simultaneous Localization and Mapping
TD	Temporal Difference
TI	Traversability Index

UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
WA*	Windowed A*

ABSTRACT

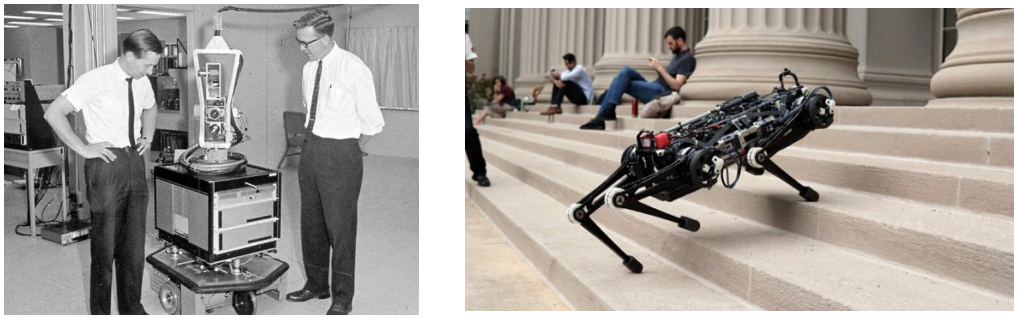
Schultz, Nicholas S. M.S., Purdue University, May 2020. A Hybrid Method for Distributed Multi-Agent Mission Planning System. Major Professor: Shaoshuai Mou.

This thesis presents work concerning a distributed heterogeneous multi-agent robotic team for outdoor applications such as search and rescue or surveillance. The goal of this research is to develop a method of control for a team of unmanned aerial and ground robots that is resilient, robust, and scalable given both complete and incomplete information about the environment. The method developed and presented in this paper integrates approximate and optimal methods of path planning integrated with a market-based task allocation strategy.

This thesis also presents a solution to unmanned ground vehicle path planning within the developed mission planning system framework under incomplete information. Methods such as genetic algorithm and deep reinforcement learning are proposed to solve movement through unknown terrain environment. The final demonstration for Advantage-Actor Critic deep reinforcement learning model elicits successful implementation of the proposed model.

1. INTRODUCTION

Humanity has come a long way since the early 1970's when Joseph Engleberger, dubbed the "father of robotics," developed the first successful autonomous industrial robot. The robot named "Shakey" was an advanced and specialized industrial robot that could perceive its environment and make appropriate decisions based off of sensor inputs [1]. After nearly 50 years, it is easy for one to consider widespread autonomy to be a fait accompli, an achievement on the cusp of attainment. Modern autonomous robots can be seen nearly everywhere in society from simple machines such as the Roomba to complex vehicles such as Tesla's self-driving car.



(a) "Shakey" Robot (b) Modern Robot

Figure 1.1.: Evolution of Robotics

At the base of all autonomous systems is their ability to perceive the environment and process data from the outside world while simultaneously making complex decisions. In terms of perception, it is common nowadays to see robots equipped with stereo vision or LiDAR (Light Detection and Ranging) which allow robots to develop an accurate layout of the surrounding environment.

With knowledge of the environment, the robot must then make appropriate decisions towards accomplishing a specific task. This type of control is described as motion planning, or path planning. According to Steven LaValle, an expert in the field of robotics and planning algorithms, "a fundamental need in robotics is to have algorithms that convert high-level specifications of tasks from humans into low-level descriptions of how to move" [2].

A classical version of motion planning is sometimes referred to as the *Piano Mover's Problem* [3]. This problem outlines one's ability to take a piano and move it room to room within a house without hitting any walls. In artificial intelligence (AI), the terms motion planning, or path planning, describe an agent's ability to move from point A to point B while simultaneously updating knowledge of its environment, making decisions, and avoiding collision with other agents or members of the environment.

1.1 Motivation

Within the past 20-30 years, robotics and autonomy have become a large focus of industry, academia, and the military. Researchers are finding new and innovative ways to utilize robots for problems that may be too dangerous or impossible for humans to solve on their own. More specifically, emphasis has been placed on controlling and coordinating teams of robotic agents in order to solve large-scale tasks more efficiently.

Autonomous and intelligent multi-agent systems have found wide-spread application in both civilian and military sectors [4] [5]. Unmanned aerial vehicle (UAV)/unmanned ground vehicle (UGV) systems have become a focal point for such applications as surveillance, navigation, crowd control, sensing, and emergency rescue operations in regions where it is difficult or dangerous for human beings to access. UAVs and UGVs have different characteristics that, when employed cooperatively, better address problems associated with preceding applications [6].

For example, in a disaster relief scenario multiple UAVs and UGVs can be deployed to increase system efficiency, provide real-time assistance to those in need while decreasing the overall time required to cover the target region [7]. UAVs have superior mobility and agility and typically conduct exploration, simultaneous localization and mapping (SLAM), and surveillance tasks. UGVs, on the other hand, are more payload-capable and conduct such tasks as package delivery, extensive computation, and the supply of medical assistance [8]. An example of a UAV/UGV team for search and rescue (SAR) can be seen in Figure 1.2.

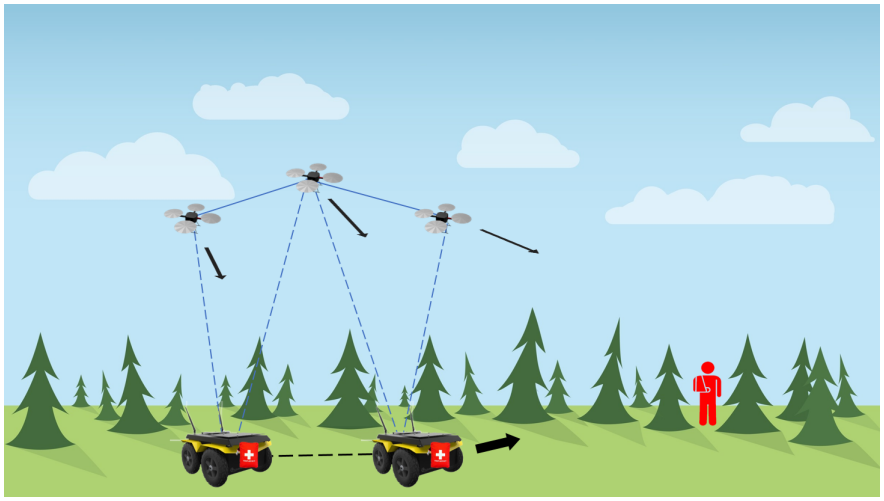


Figure 1.2.: Robotic Search and Rescue Team Example

There are many challenges associated with assembling a successful autonomous team of agents for situations such as disaster relief or SAR operations. The environment, for one, offers a multitude of obstacles. Some examples include foreign debris blocking the robots' path, hazardous operating conditions that affect stability or communication, etc. It is important that robust and efficient algorithms be created to account for all possible points of failure [6] [9].

In addressing the subsequent problem, this paper proposes a hybrid solution in which a team of heterogeneous agents must conduct distributed task allocation and path planning with both complete and incomplete information. First, this paper be-

gins by examining a solution for task allocation and path planning of a heterogeneous team of UAV/UGV agents with a complete map and knowledge of surroundings. Moving from this point, it is assumed that during the process of exploration, perceived information is obtained concurrently with subsequent navigation [10] and the UGV will need to navigate towards its goal with dynamic environmental updates. An extension to the hybrid algorithm is described for conducting task allocation and path planning of a heterogeneous team of UAV/UGV agents with incomplete information of the environment.

1.2 Literature Review

Task assignment and path planning are two fundamental problems in decision and control of multiple robots. The combination of both in the presence of multiple agents and multiple tasks presents problems ranging from scalability, robustness, complexity, and computational feasibility [11]. When scaling the problem to a large environment, multiple agents, and/or multiple tasks, there becomes a tradeoff between optimality and computational expense [12]. Optimal solutions typically require an expensive solution or a centralized planner, whereas successful suboptimal solutions are distributed, but require techniques to reduce waste or unnecessary expense [13]. Mission planning methods need to efficiently coordinate a multitude of tasks to multiple heterogeneous agents, plan paths accordingly, avoid obstacle and agent collisions, and maximize optimality while performing in real-time. This becomes a complex and challenging issue.

The term mission planning system (MPS) refers to the combined task allocation and path planning problem. To date, there have been many methods developed for the multi-robot task allocation (MRTA) problem as well as the multi-agent path planning (MAPP) problem. However, little research has addressed the combination or coupling of both [14] which is the core aspect addressed in this paper. This section will outline popular methods developed for task allocation, path planning, and the

combination of both for multi-agent systems. Additionally, this section will outline common methods for agents dealing with incomplete or partial information of the environment.

1.2.1 Multi-Robot Task Allocation

There are many benefits when considering multi-robot systems and some of these include resolving task complexity while increasing team performance, reliability, and simplicity in design [13]. Many systems today are increasingly complex and require multiple agents to assess multiple tasks. Growing numbers of agents and tasks require methods that effectively handle the system as a whole. [13] outlines current state-of-the-art approaches for solving the MRTA problem. Initially, approaches to this problem can be categorized as centralized and decentralized.

Some popular centralized algorithms proposed are GRAMMPS [15], Mtap-masim [16], event assignment [17], and fair subdivision [18]. Centralized approaches to this problem are among the most widely reported. The solution assumes that there is full communication and complete information across a system and one central agent is able to allocate tasks among the network simultaneously. These methods reduce duplication of effort, resources, and increase saving of cost and time. However, one major downfall of this approach is the lack of robustness. A failure among one agent can cause the entire system to fail, which is especially undesirable.

In contrast, distributed, or decentralized, approaches disperse administrative tasks and authorities between all agents within the system. Here, the system is assumed to be strongly connected and information is shared among agents through a multi-hop relay network. Visualization of how a decentralized method is realized can be seen in Figure 1.3.

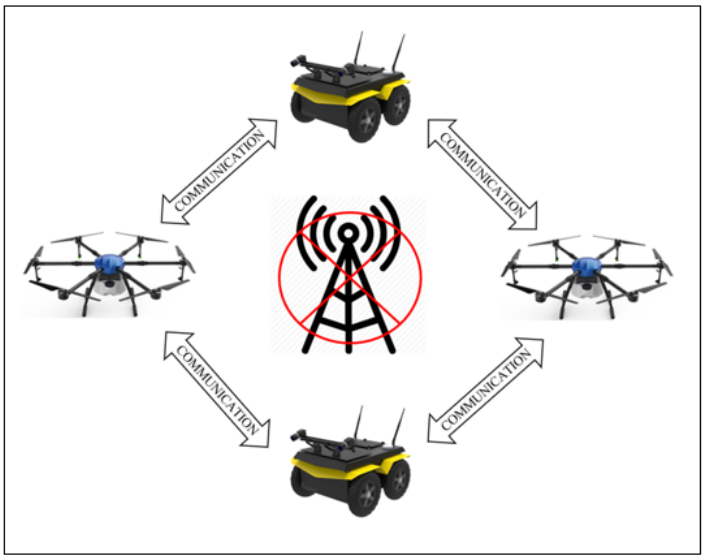


Figure 1.3.: Decentralized System

Within this domain, many existing solutions have been classified between optimization and market-based. Optimization approaches, such as Genetic Algorithm (GA) [19] and mixed-integer linear programming (MILP) [20], operate on a set of constraints and the optimum solution is chosen according to a set of certain criteria. While this branch of distributed solutions appropriately solves many variations of the MRTA problem, it is generally expensive in terms of computation and complex in nature. A new distributed approach called market-based task allocation appeared in 2004 with TraderBots [21] where agents follow a scheme of bid, auction, and assign. More reputable papers utilizing this approach include consensus-based bundle algorithm (CBBA) [22] and iterated auction-consensus algorithm (IACA) [23].

In general, distributed, market-based approaches to the MRTA problem have found widespread application due to its practicality, optimality, and overall simplicity in design. Strongly-connected agents can handle bids, auctions, and assignments individually in a computationally efficient manner.

1.2.2 Multi-Agent Path Planning

MAPP is a well studied research area and includes many sub-fields with narrowed variants. At a high level, MAPP can be dissected into two broad categories: planning with complete information and planning with incomplete information. The former is a simpler look at the problem where each agent operates within a simple environment assumed to know start, goal, agent, and obstacle locations. Incomplete information is a way of varying the problem by solving MAPP when there is unknown or uncertain information presented within the system.

Solving the compete information MAPP problem, while simplistic in nature, gives proper insight into how multiple agents can cooperate within the same domain and still applies to many real-life situations. Common graph-based solution methods include variants of the A* graph search algorithm [24], rapidly-exploring random trees (RRT) [25], and probabilistic roadmaps (PRM) [26]. The A* graph search algorithm is optimal, but more computationally expensive and less scalable to its approximate sample-based counterparts, PRM and RRT. Common optimization-based solution methods include conflict-based search (CBS) [27] and MAPP algorithm [28]. While most optimization methods offer efficient and complete results, they are considered centralized planners and therefore lack robustness.

With incomplete information, agents must typically take into account dynamic updates to the environment and respond appropriately. There are numerous approaches and variants within this domain, but the main split in current research is between graph-based/optimization-based approaches and learning approaches. Popular graph and optimization approaches include the D* algorithm [29], evolutionary algorithms, and diffusion maps [30]. Learning, more specifically deep reinforcement learning (DRL), has become a popular field because of an agent's ability to perceive large data sets and execute an effective control policy almost instantaneously [31].

1.2.3 Combined MRTA and MAPP

As previously stated and identified in research, less attention has been given to systems that effectively couple task allocation and path planning. These mission planning systems are applicable to many real-life multi-robot/swarm technologies.

The combined problem was solved recently with an optimal algorithm [14], however it is not scalable in the presence of large numbers of tasks. One method [32] looks to use path planning as an effective bid valuation within a market-based task allocation system, but does not take into account collision between agents and conflicting paths.

Thus, the goal of this research is to investigate a proper mission planning control framework for integrating multiple heterogeneous agents in the presence of complete and incomplete information.

1.3 Contributions

The main contributions of this work are

- A hybrid distributed multi-agent mission planning system that is scalable and robust given complete information of the environment
- Extension to the hybrid mission planning system to assess incomplete knowledge of terrain structure using various methods
- Software implementation and simulations demonstrating successful implementation of the designed algorithms

2. A HYBRID METHOD FOR MISSION PLANNING

2.1 Introduction

The first section of this paper addresses control of a heterogeneous, communicating team of agents within a complete (fully known) environment. More specifically, this section analyzes the application and coupling of optimal and approximate planning methods within a distributed framework. An algorithm is presented utilizing proposed methods for task assignment, path planning, and collision avoidance. Task assignment is implemented as an extension to techniques proposed in the recently developed framework of CBBA and collision avoidance is presented in a consensus-based framework coupled with a windowed replanning algorithm.

2.2 Problem Formulation

The foundation for this problem is the consideration of an outdoor environment. [33] provides quality insight into global path planning for an outdoor environment. The first assumption made is that the ground robot will have knowledge of its initial position and orientation as well as global position and orientation in order to properly localize itself within the environment. For outdoor applications, vehicle constraints are less important due to the relative size of the obstacles and planned path, thus it is logical to assume that the vehicle is capable of holonomic motion. The on-board local navigator will plan around small obstacles and take care of vehicle dynamics within the immediate area.

Since not all information may be known prior to planning and resolution images from aerial surveillance may not be complete, it becomes difficult to represent the environment topologically. Thus, it is best to consider metric navigation where

information is discretized and algorithms can better handle updates to the environment. For developing a planning space, configuration space is commonly used which represents each possible configuration for the vehicle. With holonomic motion and consideration of a 2D environment, the configuration space can be reduced to just x and y coordinates, which is helpful for reducing the overall computational cost. Additionally, since the environment can be relatively large compared to the ground robot, discrete time is used in order to manage the processing power required to plan each path.

Current methods of comparison, A*, PRM and RRT*, all use cell decomposition within discrete metric navigation and are well-suited for planning in an outdoor environment. For the scope of this particular section, consider a complete map of the environment. An undirected graph, $G(V, E)$, is used to represent this environment where V is a list of the possible locations for each agent given in (x, y) coordinates and E is a list of the edges, or the possible transitions between states. An 8-grid approach is used with the consideration of diagonal movements. The system is computed on a time interval $T \subset \mathbb{R}$ that is unbounded to yield $T = [0, \infty]$.

For a heterogeneous team consider m agents and n tasks with the following notation:

$$m_a \subseteq m \triangleq \text{UAV agent},$$

$$m_b \subseteq m \triangleq \text{UGV agent},$$

$$n_u \subseteq n \triangleq \text{UAV task},$$

$$n_v \subseteq n \triangleq \text{UGV task},$$

$$n_w \subseteq n \triangleq \text{heterogeneous task}.$$

Furthermore, each agent is assigned a task bundle limit, l_m, which defines the maximum number of tasks allowed for an agent to complete. Additional basic assumptions for this problem assume that all obstacles are point obstacles and there are no obstacles present for UAVs.

Connectivity within the network is modeled using an unweighted adjacency matrix, A. It is assumed that agents are strongly connected meaning that there is a path

from any node to any other node. An example environment with "star" connectivity among agents can be seen in Figure 2.1 below.

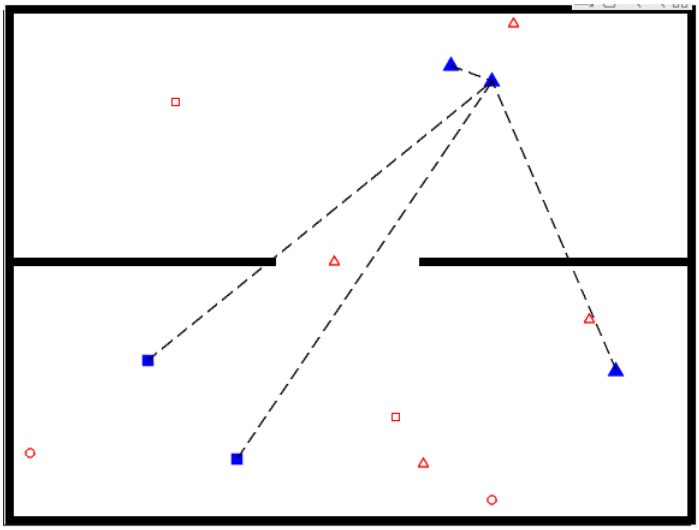


Figure 2.1.: Star Connection Map

To further explain this figure as well as additional similar figures that follow within this paper, a legend is provided below in Figure 2.2.

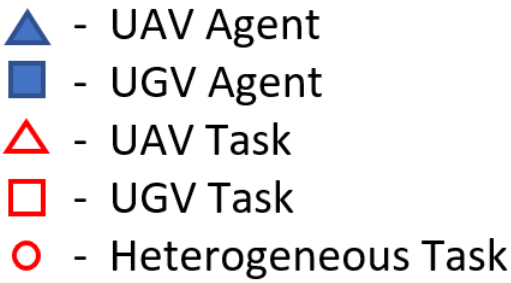


Figure 2.2.: Figure Legend

Time is discretized within the system where agent m is located at point $[x_m(t), y_m(t)]$ at time step t. For each agent, current start location, $[x_m(0), y_m(0)] \in V$, and all goal locations, $n \in V$, are known previously. Since time is discretized, the path from start

location to goal location is defined in terms of (x, y) coordinates as well as the time step, t, at which the robot will be at the specific vertex. Time step is included to incorporate a consensus-based collision avoidance technique. This approach will be discussed further on in the report.

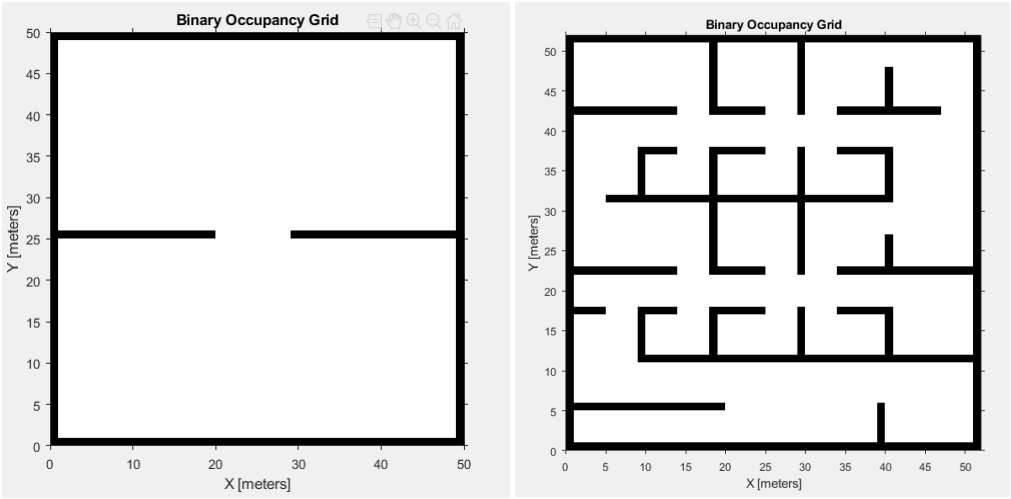
[2] helps to express problem formulation for multi-agent path planning. The configuration space is modeled as $C_m = [x_m, y_m]^T$ for each agent with configuration $q = [x, y]^T \in \mathbb{R}$ and the state space X is the Cartesian product of each agent's configuration space as seen below:

$$X = C_1 \times C_2 \times \cdots \times C_m. \tag{2.1}$$

A state $x_m \in X$ is denoted as $x_m = (q_m, t)$ with configuration q and time t components. The obstacle region is defined as X_{obs} within the state space as see below:

$$X_{obs} = \{q \in X \mid q \cap O \neq \emptyset\} \qquad \forall\{m_a, m_b\} \subseteq m. \tag{2.2}$$

With a complete map, the assumption is made that each agent has knowledge of all obstacle locations within the environment. Obstacles for this research are represented in a binary occupancy grid with 0 signifying the free space and 1 signifying the obstacle space. Two different grids are used to represent the environment, a simple map with few obstacles and a complex map with many obstacles. Examples are shown in Figure 2.3.



(a) Simple Map (b) Complex Map

Figure 2.3.: Obstacle Map Examples

The free region is the region defined outside the obstacle space. This area is represented as $X_{free} = X \notin X_{obs}$ and a goal region is $X_G \subset X_{free}$. The goal region is represented below:

$$X_G = \{(q_{G_n}, t) \in X_{free} \mid t \in T\} \qquad \forall\{n_u, n_v, n_w\} \subseteq n. \tag{2.3}$$

Constraints for this problem define that two agents cannot occupy the same state at the same time step, agents cannot occupy spaces with obstacles, and agents may not exceed their task bundle. A solution is a set of non-conflicting paths, where a path for agent m is a sequence of (x, y) waypoints.

The solution layout for this problem is given in two different sections. The first section discusses integration of distributed task assignment within a path planning infrastructure and the second discusses distributed path planning in combination with a collision avoidance technique based on consensus and waypoint sharing between agents.

2.3 Method

2.3.1 System Overview

The task assignment and path planning problem of multiple robots requires coupling of information. [34] outlines why distributed, market-based strategies coupled with path planning techniques is the best approach to this problem. More specifically, an iterative approach to this problem allows for dynamic updates within the uncertain environment. Moving on from this strategy, an iterative MPS hierarchical framework is proposed consisting of three stages between UAVs and UGVs of pre-processing, task assignment, and post-processing. A basic overview can be seen below in Figure 2.4 with subsequent discussion of each stage.

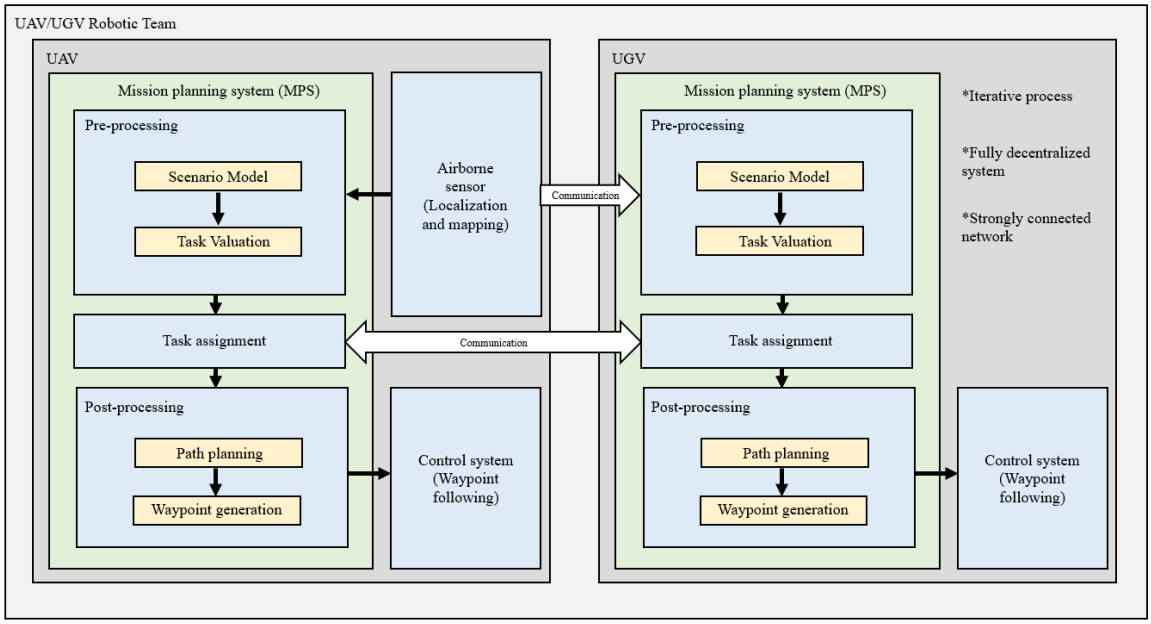


Figure 2.4.: MPS Hierarchical Framework

A detailed description of methods will be provided in following sections, but a general overview is provided here. An assumption made in this work is that a scenario model is pre-made from UAV surveillance and airborne sensors. UAVs and UGVs coordinate control with 2D information outlining free space, obstacle space, and goal

locations. From here, agents value tasks individually. Bids are time-value based which converts well to embark time and distance traveled. UAVs assume a straight line distance with no obstacles while UGVs must account for the obstacle space. To reduce heavy computational requirements in bidding for multiple tasks, UGVs utilize an approximate sample-based planning technique with scalability guarantees.

Post task valuation, agents coordinate among each other through an extension of IACA to acquire their task bundle. This is a market-based, iterative strategy that is resilient to time-bounded attacks on the system. The final stage of path planning and collision avoidance is required for UGVs. With a distributed system, each agent can use an optimal technique to plan for single tasks without consideration for computational expense. Conflicts between UGV agents are resolved concurrent with waypoint following. Nearby agents share a short series of future waypoints and reach a consensus on which agent will perform a quick replan. Problem models and algorithms are presented in this section.

2.3.2 Distributed Task Allocation

The problem of task allocation is first considered for properly assigning each agent a task within the system. This problem has been properly formulated and addressed by both articles presenting CBBA [22] and IACA [23]. The basis of IACA is assigning a time-discounted reward $r_{ij}(t) \in \mathbb{R}$, depending on the time that agent i is able to finish task j. For task allocation consider UAV agents: $I_a = \{1, 2, \ldots, a\}$, UGV agents: $I_b = \{1, 2, \ldots, b\}$, UAV specific tasks: $I_v = \{1, 2, \ldots, u\}$, UGV specific tasks: $I_v = \{1, 2, \ldots, v\}$, and heterogeneous tasks: $I_w = \{1, 2, \ldots, w\}$. The goal of distributed task allocation is to find a group of values for agent $x_{ij} \in 0, 1$ which:

$$\text{maximize:} \quad \sum_{i=1}^{m} \left(\sum_{j=1}^{n} r_{ij}(t) x_{ij} \right) \tag{2.4}$$

$$\text{subject to:} \quad \sum_{i=1}^{m} x_{ij} \leq 1, \quad \forall j \in \{I_u, I_v, I_w\}, \tag{2.5}$$

$$\sum_{j=1}^{n} x_{ij} \leq l_i, \quad \forall i \in \{I_a, I_b\}, \tag{2.6}$$

$$\sum_{i=1}^{a} x_{ij} = 0, \quad \forall j \in \{I_v\}, \tag{2.7}$$

$$\sum_{i=1}^{b} x_{ij} = 0, \quad \forall j \in \{I_u\}, \tag{2.8}$$

Equation 2.4 defines the objective of this distributed optimization problem. The goal is to maximize the sum of discounted rewards across all tasks assigned within the network. $r_{ij}(t)$ signifies the reward that agent i will receive for being assigned task j at time t. x_{ij} is an indicator function that defines whether task j is assigned to agent i. 2.5 is a constraint that limits a task being assigned to multiple different agents, 2.6 constrains the amount of tasks assigned to an agent to stay within their bundle limit, 2.7 states that UAV agents cannot be assigned UGV-specific tasks, and 2.8 states that UGV agents cannot be assigned UAV-specific tasks.

This algorithm is split into two main parts: auction and consensus/validation. For the purpose of this paper, consensus and validation will be grouped into the same category. The CBBA algorithm performs auction and consensus all in one step. IACA is different in that agents run auction and consensus in an iterative fashion. They build their bundle by at most one every iteration throughout the algorithm. An advantage of this method is that iterative auction-consensus is resilient to time-bounded attacks on the system. An overview of this process can be seen in Figure 2.5 below.

Figure 2.5.: IACA Overview

First, a look at the mechanism of auction for both UAV and UGV agents. Once all agents are set within the environment, they are introduced to a task list, $\{I_u, I_w, I_v\} \in I_T$, that is pre-determined from the scenario model. All agents will bid for appropriate tasks if their bundle is not already full. An agent with a full bundle will be removed from the task allocation process. Task bids differ between agents. Generally, the task bid calculation is given in the following form,

$$[y_i]_j(t) \triangleq r_{ij}(t) = c_j \lambda_j^{\delta_{i,0j}(t)}, \tag{2.9}$$

where c_j is the initial profit for task j that is pre-defined according to the relative importance of task j and $\lambda_j^{\delta_{i,0j}(t)}$ is the embark time, or distance, for agent i. This is the cumulative distance traveled by agent i given their task bundle. Embark time for UAV agents is determined by Euclidean distance from the agent position and the task position. Embark time for UGV agents is different because they have to account for obstacles within the environment. Thus, this is determined by the path output from PRM or RRT* approximate path planning algorithms. The mechanisms for these path planning methods will be discussed in later sections. An overview of the auction process can be seen in the flow diagram in Figure 2.6.

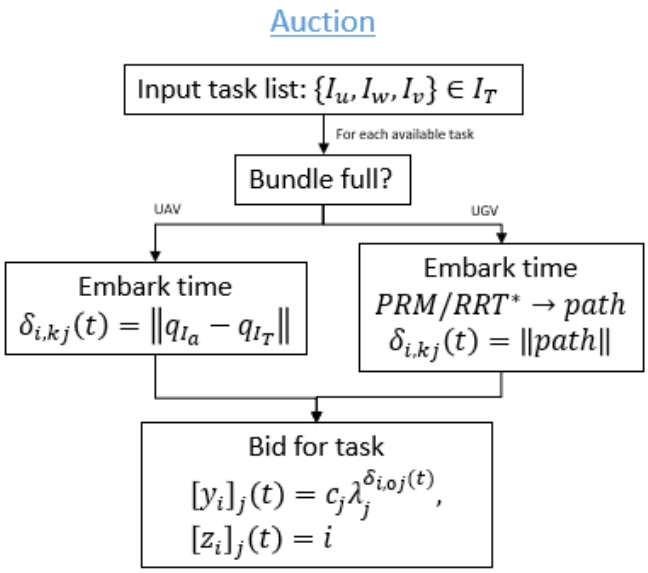


Figure 2.6.: Auction Flow Diagram

Code implementation of the auction mechanism can be seen in the pseudocode in Figure 2.7.

Algorithm 1: Auction for agent i at time t

Result: Bid matrix for each agent $\{m_{a_i}, m_{b_i}\} \in m$
Input: $P_i(t), \{I_u, I_v, I_w\} \in I_T$
for $j \in I_T \quad and \quad |P_i(t)| < l_i$ **do**
  if $i \in I_a$ **then**
    if $j \in P_i(t)$ **then**
      $[y_i]_j(t) = 0, \ [z_i]_j(t) = 0.$
    else
      $\delta_{i,kj}(t) = ||q_{I_a} - q_{I_T}||,$
      $\delta_{i,0j}(t) = \sum_0^{k-1} \delta_{i,(k)(k+1)},$
      $[y_i]_j(t) = c_j \lambda_j^{\delta_{i,0j}(t)}, \ [z_i]_j(t) = i.$
    end
  else
    if $j \in P_i(t)$ **then**
      $[y_i]_j(t) = 0, \ [z_i]_j(t) = 0.$
    else
      $RRT^* \rightarrow path,$
      $\delta_{i,kj}(t) = ||path||,$
      $\delta_{i,0j}(t) = \sum_0^{k-1} \delta_{i,(k)(k+1)},$
      $[y_i]_j(t) = c_j \lambda_j^{\delta_{i,0j}(t)}, \ [z_i]_j(t) = i.$
    end
  end
end
Output: $y_i(t), z_i(t)$

Figure 2.7.: Auction Pseudocode

Within the same iteration, once agents have their bids placed for specific tasks, all agents within the system will reach a consensus on task allocation as well as validate the results. Given a strongly-connected communication topology, agents need at most $\Delta_{G(t)}$ (graph diameter) time steps in order to reach a consensus. The goal here is for an agent to increase their bundle by at most one task in one iteration. For consensus, agents input their bids, $y_i(t)$ and utilize the method of MAX-consensus [35] where agents agree on the highest reward for the overall system. From this process, a winner is assigned and agents look to validate results.

Validation is performed by individual agents with task bundle $P_i(t)$ and winning bid $y^*(t)$ as inputs. Agent i will gather their valid consensus bundle $H_i(t)$ and the

winning task $h_i(t)$. If the winning task matches their valid consensus bundle, then the agent is able to add the task to their final task bundle. An overview of the consensus process can be seen in the flow diagram in Figure 2.8.

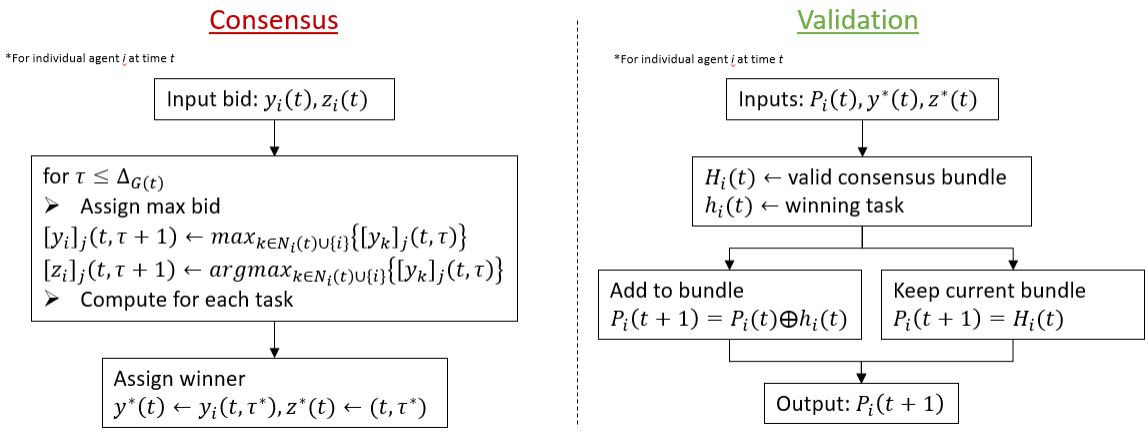


Figure 2.8.: Consensus/Validation Flow Diagram

Code implementation of the consensus mechanism can be seen in the pseudocode in Figure 2.9.

Algorithm 2: Consensus for agent i at time t

Result: Consensus on highest bid $y^*(t)$ for each task
Input: $y_i(t), z_i(t)$
Initialization
$y_i(t,0) \leftarrow y_i(t), \quad z_i(t,0) \leftarrow z_i(t), \quad \tau \leftarrow 0$
for $\tau \leq \Delta_{G(t)}$ **do**
  for $j \in I_T$ **do**
    $[y_i]_j(t, \tau+1) \leftarrow max_{k \in N_i(t) \cup \{i\}}\{[y_k]_j(t, \tau)\}$;
    $[z_i]_j(t, \tau+1) \leftarrow argmax_{k \in N_i(t) \cup \{i\}}\{[y_k]_j(t, \tau)\}$.
  end
end
$y^*(t) \leftarrow y_i(t, \tau^*), \quad z^*(t) \leftarrow z_i(t, \tau^*)$.
Output: $y^*(t), z^*(t)$

(a) Consensus

Algorithm 3: Bundle update for agent i at time t

Result: Validation for highest bid among agents, bundle update
Input: $P_i(t), y^*(t), z^*(t)$
$H_i(t) \leftarrow$ valid consensus bundle,
$h_i(t) \leftarrow$ winning task,
if $H_i(t) = P_i(t)$ *and* $|P_i(t)| < l_i$ **then**
  $P_i(t+1) = P_i(t) \oplus h_i(t)$;
else
  $P_i(t+1) = H_i(t)$;
end
Output: $P_i(t+1)$

(b) Validation

Figure 2.9.: Consensus/Validation Pseudocode

In summary, UAV and UGV agents bid between UAV-specific, UGV-specific, and heterogeneous tasks in a distributive and iterative method. This distributed market-

based strategy is desirable because approximate and optimal path planning information can be coupled. Additionally, this method scales well for large environments and a large number of agents. The next section will discuss novel path planning methods utilized and their place within the mission planning system.

2.3.3 Distributed Path Planning

Distributed, or decentralized, path planning is completed in the same manner as distributed task allocation. Each agent computes paths and trajectories on their own accord and communicates essential information across the network. Path planning is performed in both parts of the mission planning system and their implementation will be discussed further in this section.

First, as previously discussed, novel approximate path planning methods of PRM and RRT* are used during auction within distributed task allocation. These two sample-based methods work well with the assumption of holonomic motion and assume a static environment during the planning process. With stationary obstacles, this is acceptable so long as it is possible to update paths for agent to agent collision avoidance. The first method of comparison for fulfilling embark time in UGV-agent bids is PRM.

[26] defines the mechanism behind PRM, so a short overview is given here. There are two phases for this method: a learning phase and a query phase. In the learning phase, a pre-determined number of nodes are computed within the free space of the environment. The number of nodes chosen for a given environment depends on its complexity. More nodes are required for larger and obstacle-rich environments. If a path is not found for a given set of nodes, these are increased incrementally until the number is sufficient. Generally, obstacles are represented within a binary occupancy map where the free space is identified with a 0 and the obstacle space is represented with a 1. Nodes are then connected in collision-free configurations.

Only pairs of configurations whose relative distance is smaller than some constant threshold, $maxdist$ are connected. This defines,

$$N_c = \{\tilde{c} \in N \mid D(c, \tilde{c}) \leq maxdist\}, \tag{2.10}$$

where it connects c to all nodes in N_c in order of increasing distance from c. A layout of the probabilistic roadmap with node generation and subsequent connections can be seen in Figure 2.10.

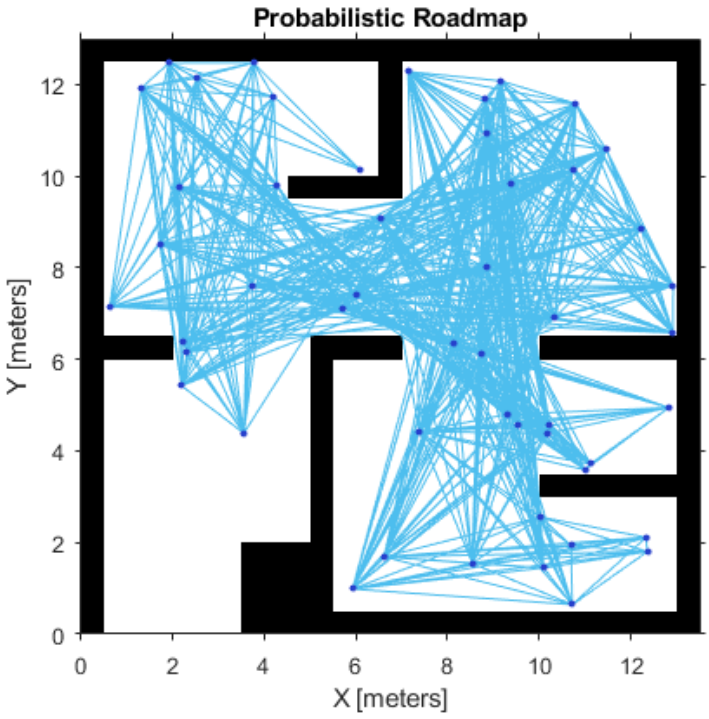


Figure 2.10.: Probabilistic Roadmap Example

In the query phase, a start node and goal node are determined. A search is computed for all possible configurations between the two paths and the minimum distance path is chosen. This is much more computationally efficient than Dijkstra's algorithm [36] where each individual node within the search space is computed for an optimal path.

[25] defines the mechanism behind RRT, so a short overview is given here. For holonomic planning, one can define $f(x, u) = u$ and $||u|| \leq 1$. A max number of

iterations and a max step-size is pre-determined, Δq. Below is a simple walkthrough of the algorithm.

1. Root node q_{init} is chosen within configurations space
2. Randomly sampled node q_{rand} is determined
3. Nearest node q_{near} within the tree is calculated
4. Step size from nearest node to random node is taken q_{new}
5. Repeat until path is complete

q_{new} is only accepted if it does not conflict with obstacles within the environment. A visual of the mechanism for this algorithm can be seen in Figure 2.11.

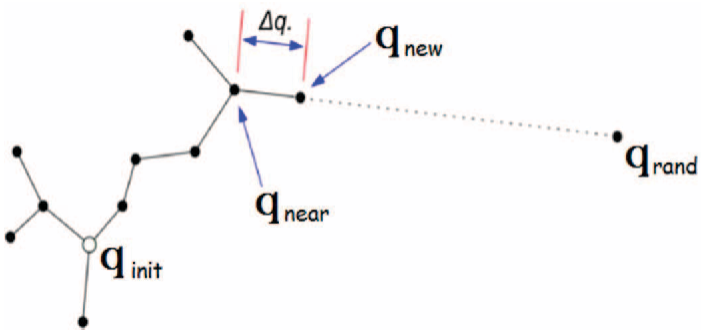


Figure 2.11.: Rapidly-Exploring Random Trees Example

The method used in the MPS is a variant of RRT called RRT*. This algorithm is an optimized version of RRT. The difference is shown when a new node is generated, q_{new}, and the nearest node in the tree is less than the pre-defined step size Δq. When this happens, a circle neighborhood of radius r_q is generated and all nodes within this area attached to the tree are connected to the randomly generated node q_{rand}. The least cost (or least distance) path is calculated and the random node q_{rand} is connected with the nearest node that gives this least cost path.

Empirical data was generated in comparison of these two methods. The IACA extension was ran in a 100x100 resolution grid environment with an empty map,

simple map, and complex map. The overall computation time was recorded for 50 separate runs with both methods. Overall, as seen in Table 2.1, RRT* outperformed PRM in terms of computational expense and overall speed. The optimality of the outputted trajectory was less important in this comparison since both methods are approximate and an optimal path is planned later in the system.

Table 2.1.: PRM/RRT* Comparison

Avg computation time (sec) per agent (100x100)			
	Empty Map	Simple Map	Complex Map
PRM	0.1751	0.1895	0.1995
RRT*	0.0099	0.0241	0.0544

Different from the approximate path planning methods used in task allocation, an optimal solution is looked towards when performing the act of trajectory generation for the purpose of reaching pre-determined goals. As previously discussed, a simple, well-known, effective method for optimal graph-based path planning is the A* algorithm developed by Peter Hart [24]. His paper outlines the mechanism behind this graph search method, so a short overview is given here.

The A* graph search algorithm is a heuristic search method meaning that it uses special knowledge about the domain of the problem to improve the computational efficiency of solutions. The basis for this algorithm is the computation of the following node function:

$$f(n) = g(n) + h(n). \tag{2.11}$$

The function evaluation here at each node within the graph is an estimate to the total cost of finding a path within the graph. $g(n)$ is the running total cost of reaching a specific node n and $h(n)$ is a heuristic function that estimates the remaining cost left to the goal. A successful heuristic used with this method is the Euclidean distance heuristic. This is simply the straight line distance computation between the current node and goal node. A proof that this function is admissible is discussed in Hart's

paper as well as why the Euclidean distance heuristic generates an optimal solution. By admissible it is meant that the graph search algorithm is guaranteed to find an optimal path from start node to goal node. A simple example of an agent utilizing this algorithm can be seen in Figure 2.12.

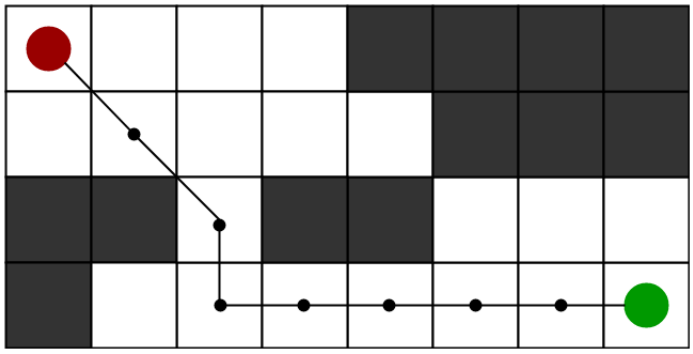


Figure 2.12.: A* Algorithm Example

A drawback of this method is that it expands a large amount of nodes compared with approximate path planning methods of PRM and RRT*. This means the algorithm is more computationally expensive and not scalable to large networks of agents. This is why the A* algorithm is used when task bundles are finalized. UGV agent i will only need to calculate one trajectory to the next task j within its bundle P_i.

Each agent running A* individually means real-time performance, but there still is no guarantee of non-conflicting paths. Thus, methods of avoidance are left up to either local navigation utilizing onboard sensors [37] or computed through consensus-based collision avoidance methods simulating an intersection between agents [38]. To handle inter-agent conflicts during execution, a consensus-based collision avoidance method is developed along with an extension to the A* algorithm to resolve locally.

Conflict between agents is dealt strictly between UGV agents as it is assumed that UAVs can avoid collision by flying at differing altitudes. An example of conflict between UGV agents during path planning can be seen in Figure 2.13.

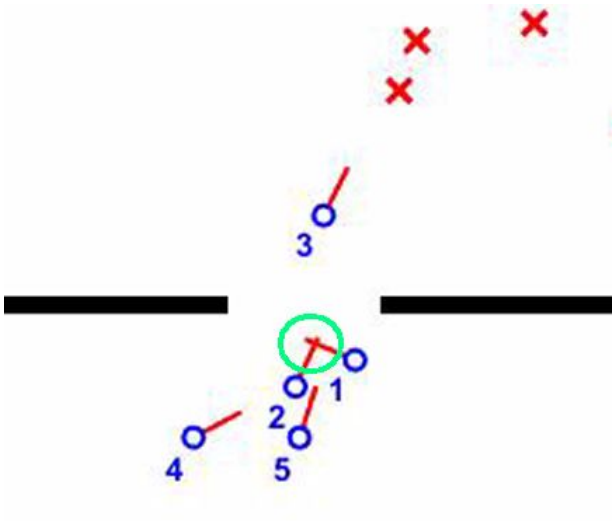


Figure 2.13.: Collision Between UGV Agents

To clarify, a "danger region" is a region in which agents determine when to run consensus-based collision avoidance (CBCA). A criterion ϵ is set that initiates CBCA if two agents are operating within this distance. Agent i and j share waypoints W_i and W_j. One agent will initiate a new set of waypoints W_μ upon consensus of a certain condition. Agents share $\epsilon + 1$ waypoints among the small network and identify conflicts between paths. Should a conflict exist, agents use the method of MAX-consensus [35]. Certain conditions that differentiate agents are given below:

- Path length
- Task priority
- Agent priority
- Battery status

For the purpose of this system, MAX-consensus is utilized on path length meaning that in the event of a collision, the agent with the shortest embark time will replan. Lastly, replanning is approached using an extension of the A* algorithm called windowed-A* (WA*). The A* algorithm is constrained to a window of size Δ

and the goal is chosen as an outlying point in the window that is within the agents waypoint list. A visualization of WA* can be seen in Figure 2.14.

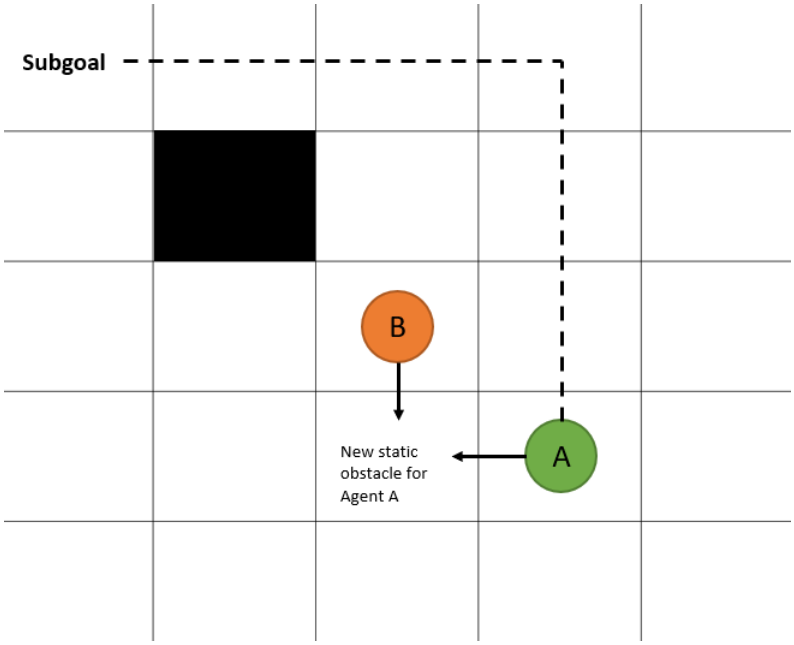


Figure 2.14.: Windowed A* Example

The final output from CBCA will be a modified waypoint list W_{μ}. A limitation of this method occurs if three or more agents are clustered within the defined "danger region". However, research here considers an outdoor environment where this is a highly unlikely event. Implementation of the CBCA algorithm can be seen in the pseudocode in Figure 2.15.

Algorithm 4: Consensus-based collision avoidance for agent i at time t

Result: Collision free paths between agent i and agent j
Input: $W_i(t), W_j(t)$
for $i \in I_b$ **do**
  if $||q_i - q_j|| \geq \epsilon$ **then**
    return
  else
    $w_i \leftarrow W_i(t) \quad \forall [t, t+\epsilon-1]$,
    $w_j \leftarrow W_j(t) \quad \forall [t, t+\epsilon-1]$.
    if $w_i \cap w_j \neq \emptyset$ **then**
      $\mathcal{O} = w_i \cap w_j$,
      $\mu = max\{W_i(t), W_j(t)\}$,
      $W_k([t, t+\epsilon]) \leftarrow$ A*,
      $W_\mu = W_k \oplus W_\mu$.
    end
  end
end
Output: $W_\mu(t)$

Figure 2.15.: Consensus-Based Collision Avoidance Pseudocode

2.4 Main Result

2.4.1 Mission Planning System

The final result utilizing the hybrid method described is a distributed multi-agent mission planning system that is able to effectively distribute tasks among the network and accomplish heterogeneous tasks in real-time for a large, simple outdoor environment. Thus, the hybrid method is defined as the coupling of novel path planning algorithms PRM/RRT* and WA* with an iterative market-based task allocation algorithm, IACA. Implementation of the final model can be seen in the pseudocode in Figure 2.16.

Algorithm 5: Hybrid Method

Result: High level hybrid multi-agent task assignment and path planning

Initialization

$\{I_a, I_b\} \in I_A, \quad \{I_u, I_v, I_w\} \in I_T, \quad t \leftarrow 0$

while *tasks not assigned* **do**

  Auction,

  Consensus,

  Validation.

end

$W \leftarrow$ A*,

if $||W_{optimal} - W_{approx}|| \geq \zeta$ **then**

  Rerun IACA.

end

while $I_T \notin \emptyset$ **do**

  $t = t + 1$,

  $q(t) = W(t)$.

  CBCA

end

Figure 2.16.: Hybrid Method High-Level Pseudocode

One last item of discussion is the modeling of situation assessment (SA) errors [34]. SA errors within this system describe the comparison between outputs of approximate and optimal path planning algorithms. Large errors between these two trajectories may deplete the optimality of task allocation among the system. As seen in the pseudocode overview, it is important to identify the approximate path W_{approx} and optimal path $W_{optimal}$. Should the difference exceed some criterion ζ, task allocation is reran. In iteration, the number of nodes generated in the novel sample-based planner is increased to generate more optimal assignment. Additionally, there is an aspect of stochasticity to these methods, therefore there is a higher chance for an increase in optimality among task allocation.

A graphical user interface (GUI) was created in Matlab to demonstrate effective execution and simulation of this method. A brief visualization of this simulation can be seen in Figure 2.17. Please refer to Figure 2.2 for the GUI legend.

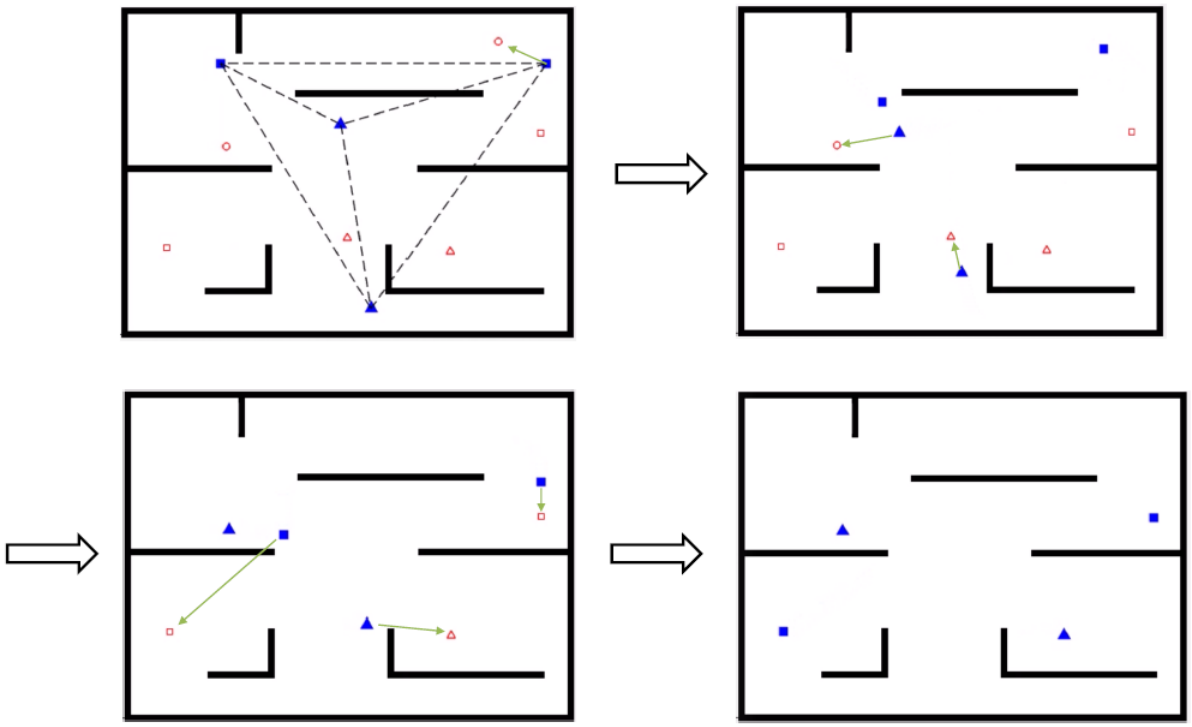


Figure 2.17.: Graphical User Interface Simulation

2.4.2 Advantages and Disadvantages

Maintaining a distributed market-based allocation system and managing sub-optimality in this way promote a system that is robust and efficient in real-time performance. Distributed sample-based computation paired with local consensus-based repair to an optimal planner allow for a system that is scalable to many heterogeneous agents and tasks. Utilization of novel path planners such as the RRT* and A* algorithms maintain a level of simplicity within the model and the windowed replan relieves the burden of path deconfliction prior to execution. The SA error check manages sub-optimality while improving results through node increases within the PRM and RRT* approximate solution.

The method developed here is effective for a simple environment with complete information. While it is a simplified model of the combined mission planning problem,

the system is scalable, robust, and effective in its solution. With coordination and control of multiple agents, the mission is completed faster and with the iterative design, the SA error within the system is reduced. Some other assumptions that limit the method is that the agents are holonomic and have perfect connection.

In order to further extend research in this field, this paper look towards a more complicated environment where incomplete information is introduced. In this variation it is assumed that the ground robots have no prior knowledge of the environment terrain. That is, they do not know the elevation along their path of travel. With dynamic updates from the terrain, they must make appropriate changes to their path in real time so as to minimize their risk of failure. Thus, the next section discusses extending the hybrid method for heterogeneous multi-agent mission planning system with incomplete information.

3. A HYBRID METHOD FOR MISSION PLANNING IN PARTIALLY KNOWN ENVIRONMENT

3.1 Introduction

Recent research has focused on increasing complexity of the multi-agent mission planning problem to include a dynamic environment or a partially/unknown environment. Naturally, it is logical to assume that these issues are more complex and complicated than the original mission planning problem. Solvers must manage the tradeoff between optimality and computational expense while also dealing with new information posed by the environment [39]. Managing unnecessary waste or expense among the system becomes increasingly more complex the more that is unknown to the agent. Additionally, there is an abundance of data available from onboard sensors and managing this data can require large computational resources and increase complexity of the system [12].

This section addresses control of a UGV agent within an unstructured environment with dynamic updates. The model here is kept the same, only a solution for dealing with unknown aspects of the environment is proposed. This solution will be integrated within the current hybrid MPS that is developed with a simulation for demonstration of the final result. The focus is on a new path planning strategy for the UGV agent among an unstructured environment. By unstructured environment, it is meant that the terrain, or elevation, is unknown before planning.

3.2 Problem Formulation

Within the current MPS structure, agents conduct their optimal path planning through A* assuming complete knowledge of the environment. With unknown struc-

ture added to the problem, there is a need for the agent to efficiently make decisions based on a constant input of information from the environment. Thus, the goal is to effectively substitute a method for A* that can handle incomplete information of terrain structure within the environment.

This problem is formulated as controlling one UGV agent to one task in order to simplify problem formulation while maintaining performance and success within the greater multi-agent system. The initial simple environment is maintained and each agent has complete knowledge of start, goal, and obstacle positions. This is information taken from the scenario model described in the MPS framework. The difference here is that terrain structure (or elevation) is gathered dynamically as agents explore the environment through LiDAR [40]. It is assumed that LiDAR data attained from the UGV agent is perfect and the vehicle is capable of holonomic motion.

LiDAR data has been obtained from OpenTopography [41]. The data is taken from various regions within Yellowstone National Park to include Norris Geyser Basin and Stonetop Mountain. Yellowstone National Park is used because it contains steep areas which pose high risk considerations for a UGV agent. The resolution for this data is $0.5m$ and organized into a Digital Elevation Model (DEM). The DEM contains elevation information in the form of a matrix representing (x, y) locations with a z coordinate identifying the elevation. A mesh map representation of Norris Geyser Basin and Stonetop Mountain can be seen in Figure 3.1 and 3.2, respectively.

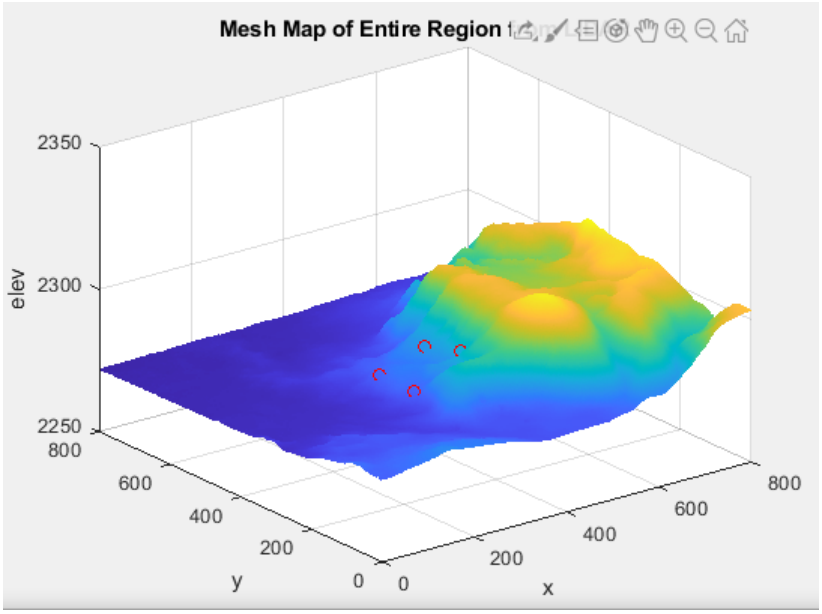


Figure 3.1.: Norris Geyser Basin Mesh Map

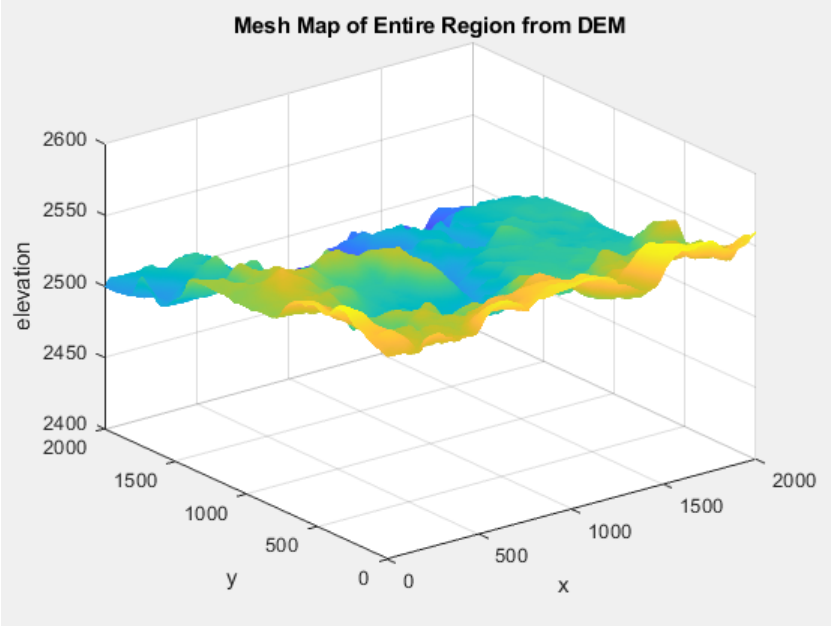


Figure 3.2.: Stonetop Mountain Mesh Map

With consideration of another dimension within the problem, agent representation is the following:

$$q = [x, y, z]^T \in \mathbb{R}. \tag{3.1}$$

Additionally, the agent will be operating within an unbounded time interval $T \subset [0, \infty]$, and time is discretized agent i is located at point $(x_i(t), y_i(t), z_i(t))$ at time step t. Constraints are placed so the agent cannot exceed terrain slope/roughness criteria and the agent cannot occupy spaces with obstacles. The solution is a set of (x, y) non-conflicting waypoints. With this information, the UGV must find a path from start to goal avoiding obstacles and minimizing risk of failure from traversing along steep terrain.

Related works suggest that UGVs are able to sense the surrounding slope of the environment and classify the information in many ways. The problem of terrain classification is formulated through [40] using a Mamdani Fuzzy Logic model. With the LiDAR data, terrain will be classified for an 8-grid surrounding region. The process of terrain classification consists of calculating the slope and roughness of the sector, determining a value within a membership function, using a fuzzy rule to determine the degree of membership, and defuzzifying the information into a traversability index, τ. A visualization of the UGV reading terrain can be seen in Figure 3.3.

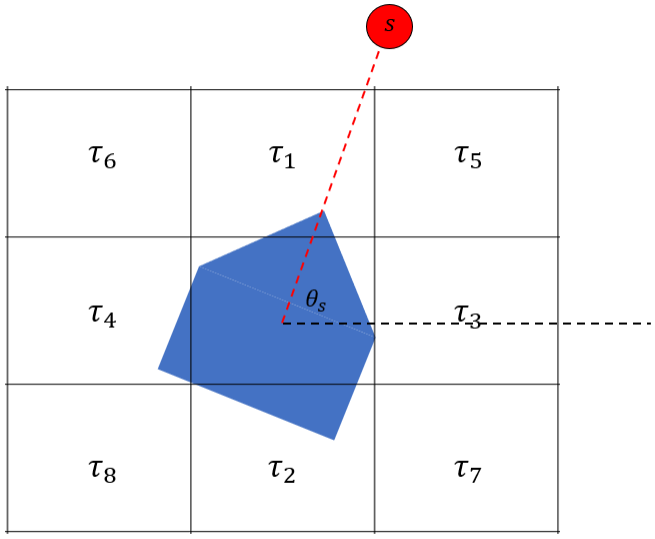


Figure 3.3.: Visualization of LiDAR Readings

For each region, the slope and roughness are calculated as inputs into the membership function. These calculations can be seen below:

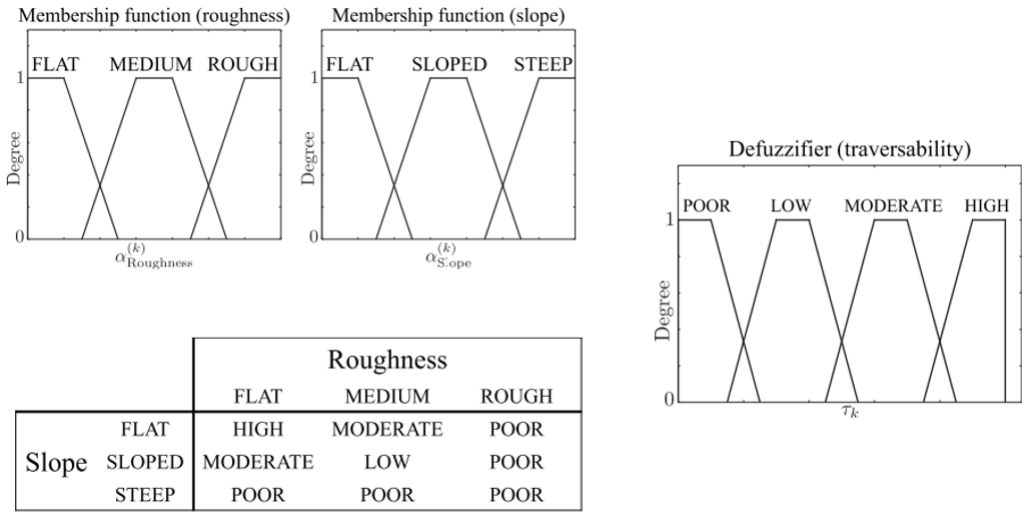
$$\alpha^S_{Slope} = \tan^{-1} \frac{|z_s - z_c|}{0.5}, \tag{3.2}$$

$$\alpha^S_{Roughness} = \sqrt{\frac{1}{N} \sum_{x_{s-1}}^{x_{s+1}} \sum_{y_{s-1}}^{y_{s+1}} (z_{x,y} - \overline{z_s})^2}, \tag{3.3}$$

where,

$$\overline{z_s} = \frac{1}{N} \sum_{x_{s-1}}^{x_{s+1}} \sum_{y_{s-1}}^{y_{s+1}} (z_{x,y}). \tag{3.4}$$

N is the number of surrounding cells and $z_{x,y}$ is the elevation information at the point (x, y). The roughness is calculated as a standard deviation over a selected area around the measured cell. These values are then used as inputs into a membership function for roughness and slope. This outputs values of "flat", "sloped", or "steep" for slope and "flat", "medium", or "rough" for roughness. A membership value is outputted for each measurement and a defuzzifier function converts to a traversability index, τ. An overview of this model can be seen in the figure below.



		Roughness		
		FLAT	MEDIUM	ROUGH
Slope	FLAT	HIGH	MODERATE	POOR
	SLOPED	MODERATE	LOW	POOR
	STEEP	POOR	POOR	POOR

Figure 3.4.: Mamdani Fuzzy Inference Model

The Mamdani fuzzy inference model was coded within Matlab and a visualization of the input-output system along with an example of how the model works can be seen in the figure below.

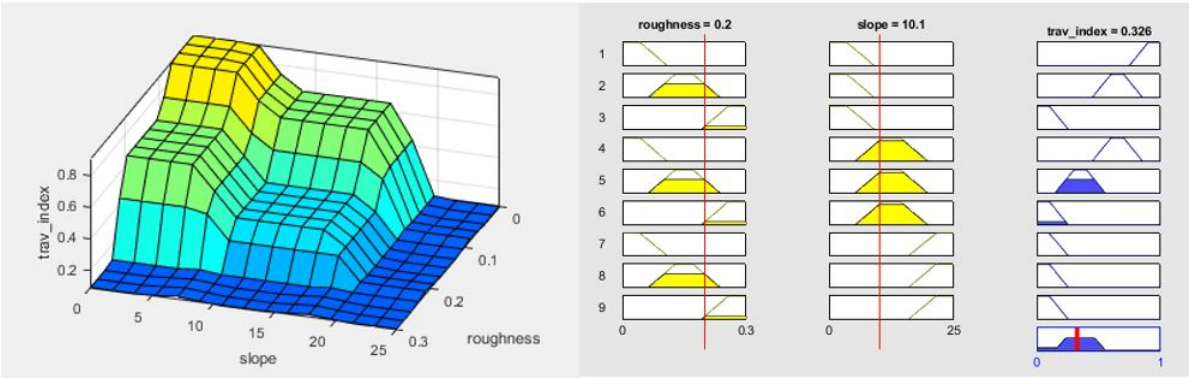


Figure 3.5.: Visual Representation of Mamdani Model

For subsequent solutions to the incomplete information problem, this fuzzy model will be used to classify terrain as the agent moves throughout the environment. The first approach uses a Genetic Algorithm and is discussed in the next section.

3.3 Genetic Algorithm

3.3.1 Problem Description and Assumptions

The first solution approach to this problem uses a Genetic Algorithm (GA) to minimize the traversability risk of a UGV moving within an environment with incomplete information. For this problem, the input variables that are being evaluated are the range and direction of travel for the UGV. This is expressed below:

$$x = \begin{bmatrix} r \\ \theta \end{bmatrix} \quad \begin{bmatrix} m \\ rad \end{bmatrix} \tag{3.5}$$

Since a GA is used, the initial value is generated within the population. Space around the UGV is separated into n sectors, each of which are assigned a traversability index from zero to one through the Mamdani fuzzy inference model. Since a high traversability index is good, the goal is to minimize the negative of this value. Additionally, RRT* is chosen to plan an approximate path from the starting location to some given global goal location. This will ensure that the UGV has some knowledge of direction in a complex environment. A visualization of how this is implemented can be seen in Figure 3.6.

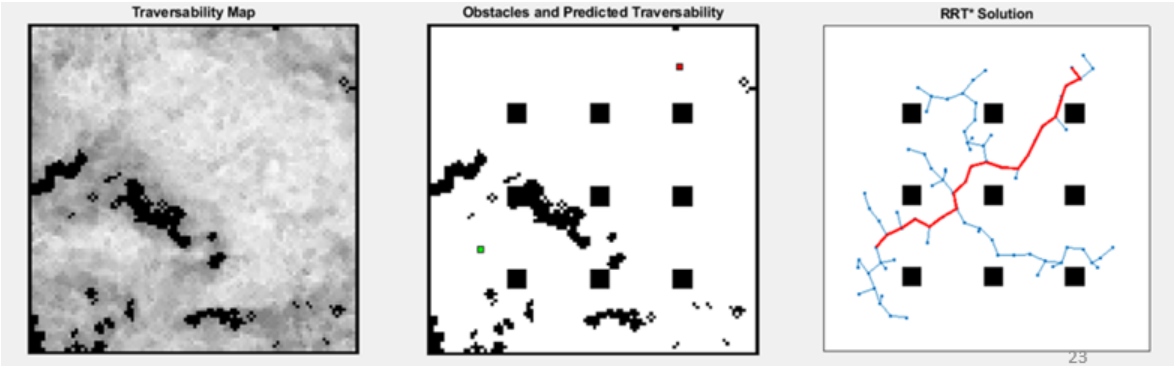


Figure 3.6.: RRT* Path with Incomplete Information

Waypoints are generated by this approximate method and the goal is to minimize the distance to this approximate path so that the robot stays on course. This is a

multi-objective problem and weighted sum method is used to solve. The objective function can be seen below:

$$f(x) = a_1\phi_1(x) + a_2\phi_2(x), \tag{3.6}$$

where,

$$\phi_1(x) = \tau_s, \tag{3.7}$$

$$\phi_2(x) = ||P_i - P_p||^2. \tag{3.8}$$

Here, τ_s is the function based on the Mamdani fuzzy inference model, P_i is the location of the search point, and P_p is the closest point location on the approximated path. τ_s is a reasonable formulation for the traversability index because an explicit function is not available.

For this problem, the only constraint is placed on maintaining a minimum specified distance away from obstacles. Exact locations of nearby obstacles are sensed and a buffer ϵ is chosen to avoid possible collision. This constraint is kept on an order of one and represented below:

$$g(x) = -\sqrt{(o_x - s_x)^2 + (o_y - s_y)^2} + \epsilon \leq 0, \tag{3.9}$$

where o is the coordinate position of the obstacle and s is the coordinate position of the search point. The following bounds were placed on the range of travel and direction:

$$0.5 \leq r \leq 1 \quad [m] \tag{3.10}$$

$$0 \leq \theta \leq 2\pi \quad [rad] \tag{3.11}$$

Constraints are handled within this problem using a quadratic exterior penalty function. This function can be seen below:

$$\Phi(x) = f(x) + r_p P(x), \tag{3.12}$$

where,

$$P(x) = max\big[0, g(x)\big]^2. \tag{3.13}$$

This penalty function derives a high function value when the constraint is not satisfied. The value for r_p here was chosen to be 100.

3.3.2 Optimization Algorithm Description

For this problem, a pre-coded GA within Matlab is used obtained from Purdue AAE 550 course [42]. In this problem, there is an implicit and discontinuous function of τ_s. In order to find a global solution with discontinuous function, an approximate global solver must be used. The Genetic Algorithm is able to handle this function, as well as apply any bounds or constraints in the problem.

As a brief overview of the algorithm, GA can be decomposed into five main parts. Variables are discretized and coded into chromosomes, then a population is initialized. Next, agents in the population are tested against a cost function to determine a fitness score. Selection, crossover, and mutation are performed to push along "the fittest" members of the population as well as randomize over the data to explore for a global solution. A visualization of this method can be seen in Figure 3.7.

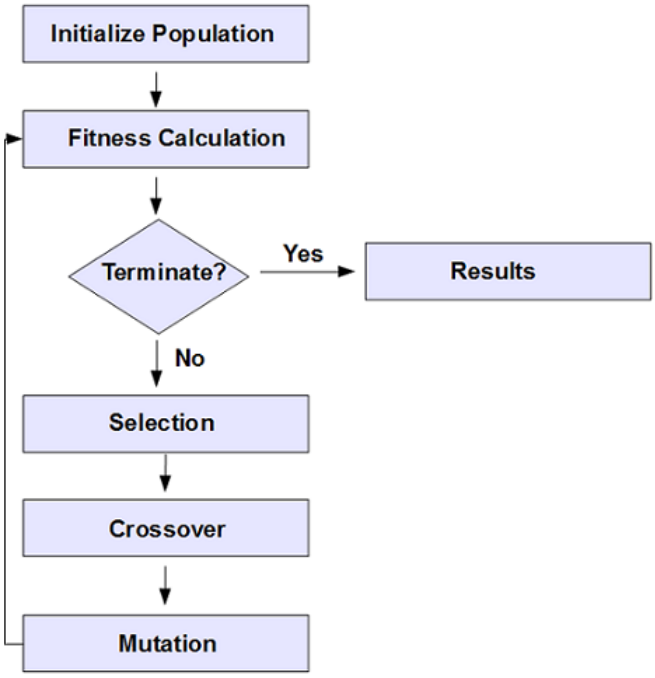


Figure 3.7.: Flowchart of Genetic Algorithm

Additionally, there are explicit bounds on the direction and range of travel. Bounds are discretized into different chromosomes that are inputted into the algorithm. 8 bits was chosen to code the vehicle travel range and 10 bits to code the direction of travel. Resolution is calculated below:

$$r_r = \frac{1 - 0.5}{2^8 - 1} = 0.00196 \quad [m], \tag{3.14}$$

$$r_\theta = \frac{2\pi - 0}{2^{10} - 1} = 0.00614 \quad [rad]. \tag{3.15}$$

The population size and mutation rate were calculated as the following:

$$N_{Pop} = 2l = 36, \tag{3.16}$$

$$P_m = \frac{l + 1}{2N_{Pop}l} = 0.01466. \tag{3.17}$$

Operators for the Matlab solver are tournament selection and uniform crossover and the default coding is Gray. This means that the crossover probability, $P_c = 0.5$. Stopping criteria is determined using a bit string affinity value of 0.9. When using this algorithm, it is important to note that the results are not truly optimal, but this approach typically provides a good enough solution.

3.3.3 Results

First, this problem was solved on a small scale in Matlab where the agent performs one step. As a brief summary, the space around the agent is split into 16 separate, equal regions and a traversability score is randomly picked between 0 and 1. Initial parameters in the environment are chosen:

$$p = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$g = \begin{bmatrix} 5 \\ 5 \end{bmatrix},$$

$$o = \begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix},$$

where p is the current coordinate position of the UGV, g is the coordinate position of the next waypoint goal, and o is the coordinate position of the obstacle. The UGV must use the waypoint goal and terrain information to make a decision about the optimal next step length and direction of travel. The algorithm was ran with varying weights and a Pareto Front was obtained by comparing the values for each objective function. Below is a visual of the curve.

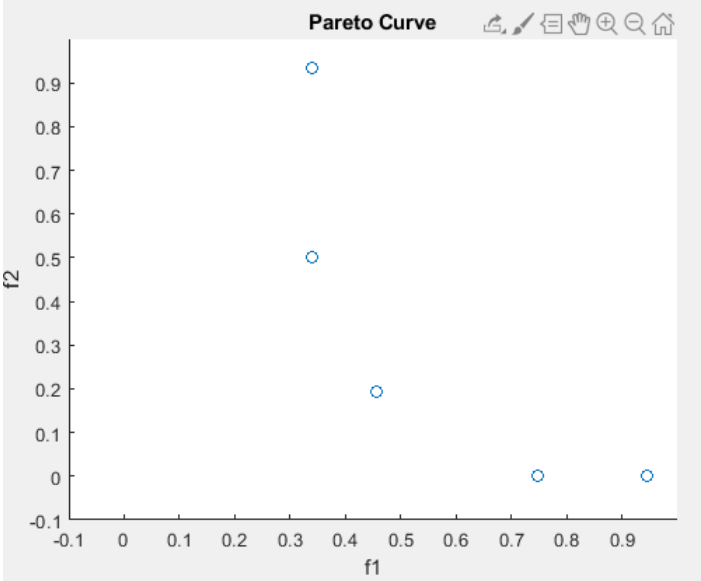


Figure 3.8.: Pareto Front

This Pareto Front is convex and the weighted sum approach will be able to handle all solutions based on various weight inputs. In solving this problem, weights of $a = [0.5; 0.5]$ were chosen in order to balance the decision between heading towards the goal and avoiding more risky traversable areas. The solution to this particular run can be seen below:

$$x^* = \begin{bmatrix} 0.5020 \\ 1.1792 \end{bmatrix} \quad \begin{bmatrix} m \\ rad \end{bmatrix}. \tag{3.18}$$

What this solution means is that the UGV will take a step length of 0.5020 $[m]$ in the direction of 1.1792 $[rad]$ measured from the x axis. These polar coordinates are converted to Cartesian coordinate locations and the position of the UGV is then updated within the environment.

This algorithm was tested within a larger, more realistic environment and its effectiveness was analyzed. The environment used for this example was Norris Geyser Basin. A smaller 100x100 map within the environment was chosen to simulation the UGV agent.

Within this section, an initial and goal location are selected and obstacles are predefined throughout. With this information, the RRT* generates a path to the goal and waypoints from this path are used as sub-goal points in the algorithm. The agent uses 'sensed' elevation data to calculate slope and roughness for eight surrounding grid locations. Thus, here $n = 8$ as apposed to 16 sections in the previous example. A Mamdani fuzzy inference model is coded to output traversability indices. For simulation, values outputted by the Genetic Algorithm were used to move the UGV accordingly. New sensed values were recorded and the agent continued to move until it reached its goal.

In the first number of tests, the ground agent would often get stuck within local minima, traveling back and forth between the same two points. This would typically occur when the agent left its path to travel to regions of lower traversability. In order to help this, the RRT* would replan if the agent reached some specified maximum distance from the approximated path. Previously determined untraversable areas were noted as new obstacles and the agent could then work its way around the map towards its goal and avoid local minima. With this same example, a solution can be seen below.

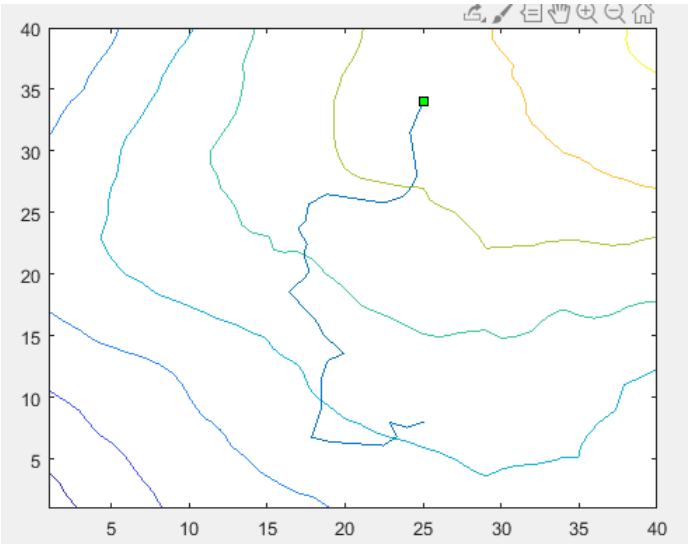


Figure 3.9.: Genetic Algorithm Simulation

3.3.4 Advantages and Disadvantages

This method was able to effectively search for a global solution around obstacles and implement local collision avoidance for a UGV. Terrain and traversability were handled in a manner that provided a global solution. It completed the goal of path planning with incomplete information about the terrain structure, nevertheless there were still some complications that arose during simulation.

There were times when the agent would still become stuck in local minima within the environment. Particularly, this would occur when the approximate path was obstructed by a large untraversable area. This is because there is a tradeoff between terrain considerations and the global solution as seen by a single pareto curve example. Additionally, the genetic algorithm is typically a higher computation cost with many function evaluations which may lead to the "freezing robot" effect in implementation. Lastly, for this approach the agent needs full knowledge of obstacles within the environment for a global solution with RRT*.

Future promising work to address the local minima is, as mentioned before, to update untraversable obstacles within the region as they are discovered and replan

with RRT*. This would ensure the agent is able to eventually find a path around these large areas. For now, deep reinforcement learning is evaluated as a promising solution because of the algorithm's ability to handle a large search space and output an action real-time based on past experiences.

3.4 Reinforcement Learning

3.4.1 Problem Description and Assumptions

The second solution approach is more promising in regards to planning with incomplete information and processing stochastic, real-time environmental updates. Here, the method proposed utilizes reinforcement learning to make real-time decisions as the agent moves through the unknown terrain environment. Formulation of this problem is similar to [31] where the goal is to minimize the expected time to goal (or distance).

For reinforcement learning methods, Stonetop Mountain was used for training since it is a larger environment. Multiple smaller regions within the DEM can be used to give more variety to the input data set.

Before defining the objective function here, the problem is formulated as a Markov decision process (MDP) with key notation defined. An MDP is a discrete time stochastic control process where at each time step t, an agent is in some state s, and may choose some action a to a new state s'. From a single step in the process, an agent will acquire a reward from the state transition, $R_a(s, s')$. Next, state transition probability is defined as the probability of reaching the next state, $P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, \ldots, S_t]$ which is a memory-less random process. Lastly, γ defines the discount applied to all future rewards. The tuple, $\langle S, A, P, R, \gamma \rangle$ defines an MDP with a policy, π that maps actions to specific states. A visualization of an MDP example can be seen in Figure 3.10.

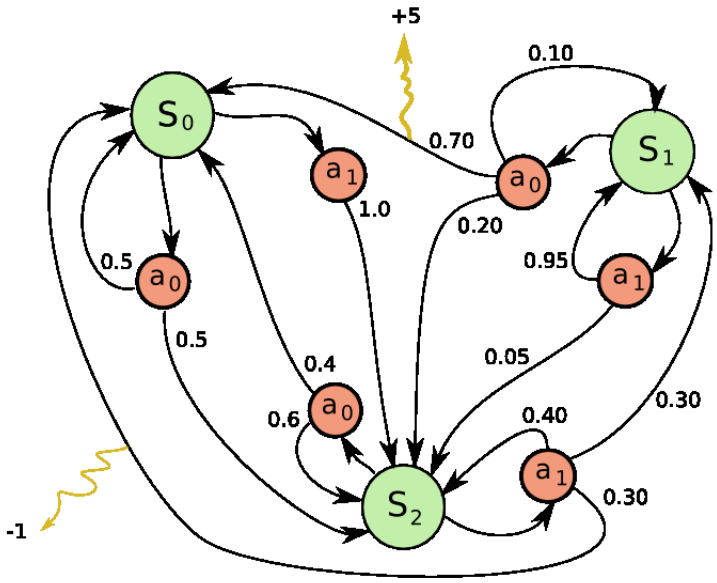


Figure 3.10.: Markov Decision Process Example

A value function, $V(s)$ is a value defining how good it is to be in any given state. The Q-Value, $Q(s, a)$ is a value defining how good an action is to take given any state. In reinforcement learning, the value function and the Q-Value function are used to improve or evaluate a policy, $\pi(s) = a$ which is a mapping of actions to states.

To narrow down how reinforcement learning is utilized in this problem, the goal, state space, and value function must be defined. The goal is to learn a policy, $\pi(s)$, that maximizes a sum of discounted rewards, $R_a(s, s')$.

$$\max \sum_{t=0}^{\infty} \gamma^t R(x(t), u(t)), \tag{3.19}$$

where γ discounts all future rewards to account for uncertainty in future states. The purpose of reinforcement learning here is to solve the optimal control problem without knowledge of system dynamics. This is desirable because the dynamics of terrain within the environment are unknown.

In discussing the state space of the problem, consider first what is known to the agent. A global path from RRT* is provided to the agent in a set of waypoints and

the agent knows immediate terrain classification of surrounding 8-grid square. Thus, the agent's state is determined to be,

$$s = [\theta_p, \tau_1, \ldots, \tau_8], \tag{3.20}$$

where θ_p is the direction to the subgoal within the approximate path. It is important to note that the action space will be limited to a 4-grid approach in order to manage computational complexity and can be seen below,

$$a = [N, S, E, W]. \tag{3.21}$$

As defined by [43], reinforcement learning methods can typically be split between policy evaluation algorithms such as TD(0), TD(λ), and Gradient TD(0) and policy improvement algorithms such as Q-Learning, SARSA, and Actor-Critic (AC). Policy evaluation algorithms generate a random policy and continuously evaluate policies until an optimum is reached. Policy improvement algorithms on the other hand continuously work to update parameters until a given policy converges to the optimal policy.

Specifically, the methods chosen to solve this problem as means of comparison were Tabular Q-Learning, Deep Q-Network (DQN), and Advantage Actor-Critic (A2C). Subsequent sections in this paper will discuss each of the three methods used along results, advantages, and disadvantages. Overall, it will be noted that A2C provides the best results at a fraction of the computational expense.

3.4.2 Tabular Q-Learning

Q-Learning was one of the early breakthroughs in reinforcement learning [44] because of its desirable off-policy behavior. Off-policy in this sense means that the agent will follow a policy that is different than the policy that is learned. The agent can continuously update the policy in both training and implementation. This off-policy TD control algorithm is defined by,

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\big[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\big]. \tag{3.22}$$

The key idea here is utilizing the Bellman Equation [45] to update Q-Values as the agent gains experience. This will in turn give an optimal policy for the problem. The optimal update equation can be seen below,

$$Q_{\pi^*}(x,u) = R(x,u) + \gamma Q_{\pi^*}(x', \pi^*(x')), \tag{3.23}$$

$$\pi(x) = \arg\max_u Q(x,u). \tag{3.24}$$

The tabular method stores Q-Values within a lookup table. Upon convergence of a policy, the greedy policy is chosen utilizing max Q-Values in the table for any given state. A simple layout of this method can be seen in Figure 3.11.

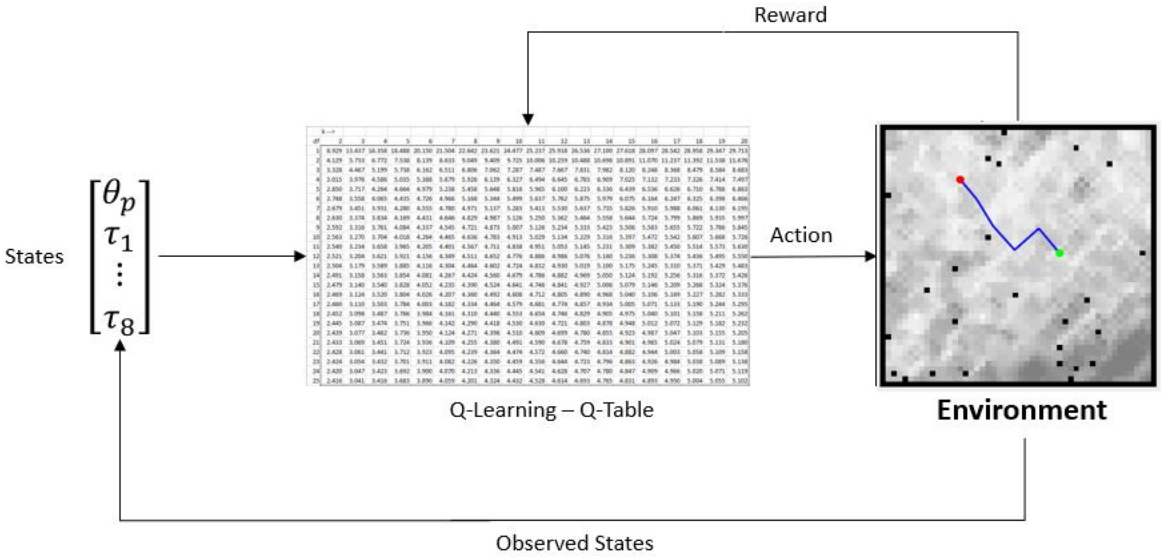


Figure 3.11.: Tabular Q-Learning Model

As an agent moves and makes decisions within the environment, i.e. as the UGV moves and learns about new terrain, the environment will output a new state for the UGV and will classify the action based on a reward function. For the reward function, there are three main considerations in motivating the agent to act appropriately:

- Terrain consideration
- Global solution

- Minimum distance

With this in mind, the reward function for the reinforcement learning model was crafted empirically to be the following:

$$R_a(s, s') = c_1\tau_{s'} + c_2(D_s - D_{s'}) - \mathbb{I}_{\mathscr{O}} + \mathbb{I}_{\mathscr{G}} - 1, \tag{3.25}$$

where c is a constant adjusting the affect that terrain consideration and minimum distance has on the final policy, $\tau_{s'}$ is the traversability index for the search point s', D defines the distance from the goal, $\mathbb{I}_{\mathscr{O}}$ is an indicator function for hitting an obstacle, and $\mathbb{I}_{\mathscr{G}}$ is and indicator function for reaching the goal. This reward function is similar to the function used for GA to incorporate all important aspects of the system. It is important to note that this reward function was obtained empirically for all three reinforcement learning methods and will be used consistently throughout.

In order to solve the exploration-exploitation dilemma of reinforcement learning, epsilon-greedy exploration is used to converge upon a policy as the agent acquires a larger sample distribution. This tabular Q-Learning method was utilized in Matlab on a smaller environment. Terrain for the smaller example can be seen in Figure 3.12.

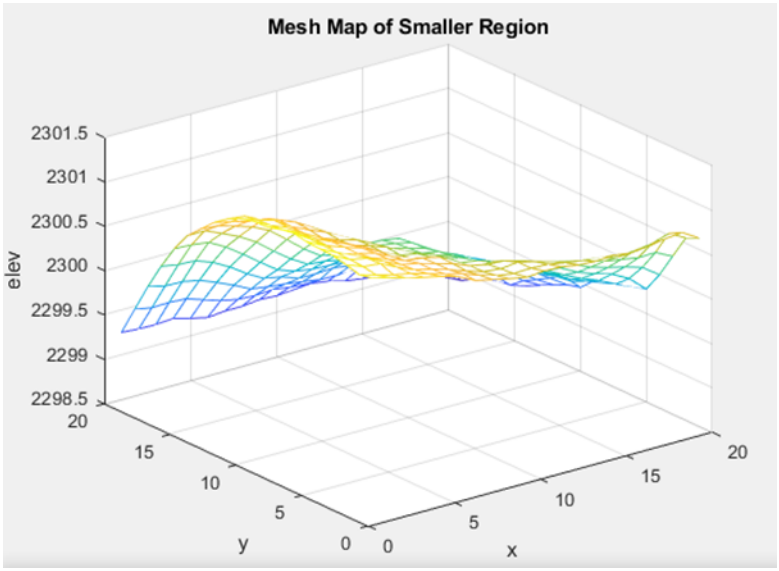


Figure 3.12.: Mesh Map of Smaller Region

The idea of utilizing a small section within the larger mesh terrain DEM is used through each method. Methods such as DQN and A2C switch between terrains within the larger dataset in order to develop a policy that is generalizable.

Training data for a UGV moving through this environment shows convergence upon an optimal policy showing success and promise for the reward function. For most trials, the policy converged within 50 episodes. Convergence of this data can be seen below.

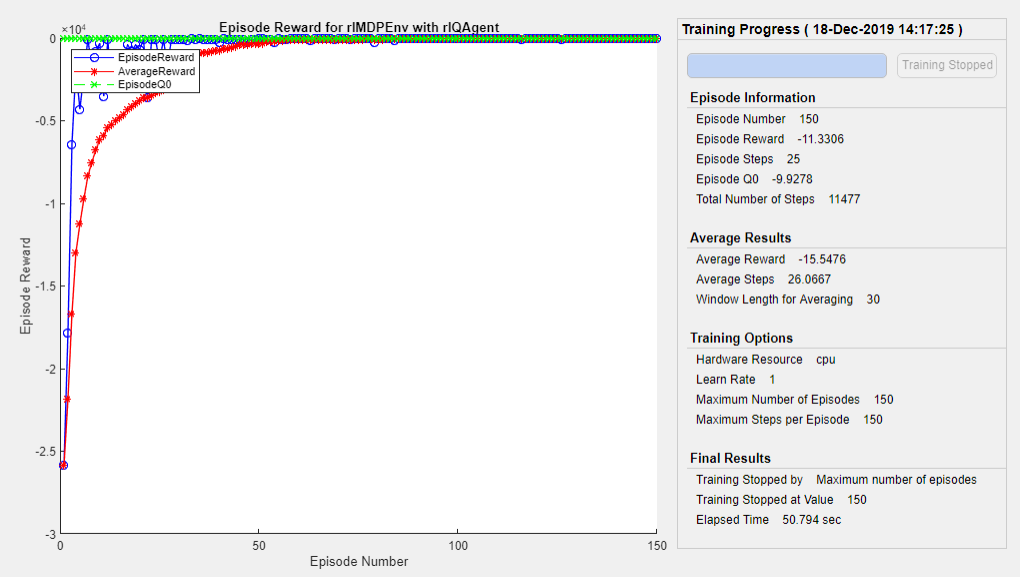


Figure 3.13.: Training Data: Tabular Q-Learning

For validation of the policy obtained from this training set on a smaller environment example, the trained Q-Agent obtained was ran one more time through the environment and the final path was outputted. A demonstration of the approximated path and the final path from the Q-Agent can be seen in Figure 3.14.

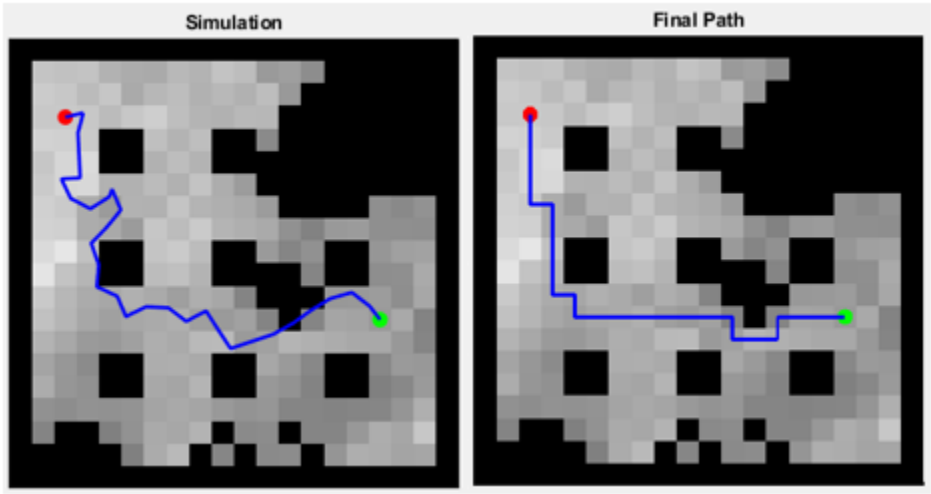


Figure 3.14.: Tabular Q-Learning Simulation

Though results seem promising for this simple Q-Learning method, it is important to note that convergence was only reached when the start, goal, obstacles, and terrain were kept the same throughout training. For any additional complications, the environment size would need to be drastically reduced. This is because this method stores optimal values within a table and slows significantly when having to update a large state space. Additionally, the policy is not generalizable to multiple different environments. DQN and A2C utilize deep reinforcement learning in order to deal with these issues.

3.4.3 Deep Q-Learning

Given a large state and action space, the reinforcement learning agent cannot possibly test all configurations. To handle this issue, an artificial neural network is used in place of the q-table as a universal function approximator. More specifically, a deep convolutional network is used to approximate the Bellman equation [46]. The approxmation to the optimal action-value function is below:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\big[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_t = s, a_t = a, \pi\big], \tag{3.26}$$

An issue with using this universal approximation is that reinforcement learning is known to be unstable or even diverge [47]. This is primarily due to the following causes:

- correlations present in sequence of observations
- small updates to Q may significantly change the policy
- data distribution and correlations between Q-values and target values change

Experience replay is introduced [48] to randomize over the data and remove correlations in the observation sequence. The agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ are stored at each time step in the data set $D_t = e_1, ..., e_t$. In combination with iterative update, Q-learning updates on minibatches of experiences that are drawn uniformly from the pool. At iteration i, the update uses the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')}\Big[\big(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\big)^2\Big], \tag{3.27}$$

where γ is the discount factor, θ_i defines the parameters of the Q-network and θ_i^- defines the parameters used to compute the target.

The DQN agent was tested on various regions within the same environment of Stonetop Mountain. The resolution of the map was 50x50 which was an increase from the alloted size with tabular Q-learning. It is important to note that the deep learning agents will incorporate an aspect of replanning within training. If an agent reaches a distance, ϵ, away from the approximate path, RRT* is reran and the agent utilizes the new set of waypoints. This is to help guide the agent when traversability becomes an issue.

The layout for this method is very similar to tabular Q-learning with the integration of a deep convolutional network. Please refer to Figure 3.15 below.

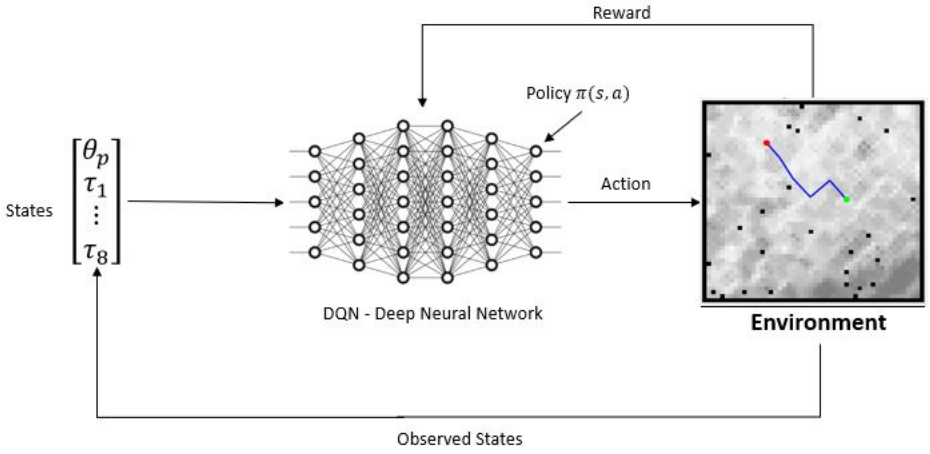


Figure 3.15.: Deep Q-Learning Model

During training, all aspects of the environment change per episode to include terrain, obstacles, start, and goal locations. Thus, the trained agent develops a policy that is generalizable across multiple domains. This aspect is desirable when planning in an unknown area in which an agent will need to respond to appropriate terrain inputs.

The DQN agent was trained at a maximum of 3500 episodes though the environment with a maximum of 500 steps per episode. Episodes would terminate upon reaching the maximum steps or the terminal location. From training data, it is clear that the DQN agent recognizes a pattern among state inputs to reach the terminal location and maximize total expected reward. This demonstrates convergence upon an optimal policy. Typically, the DQN agent reached convergence upon around 2000 episodes. Training data can be seen in Figure 3.16.

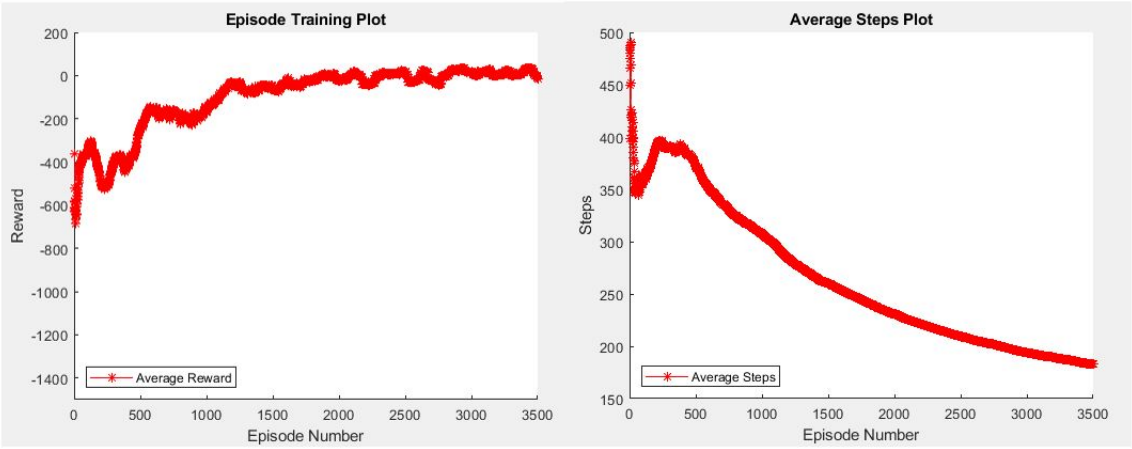


Figure 3.16.: Training Data: Deep Q-Learning

This method shows promise in the direction of finding a generalizable policy. However, one can determine from the training set above that the agent still managed to maintain approximately 200 steps within a 50x50 resolution environment. This method does not prove to be efficient in minimizing total distance traveled and would significantly decrease optimality within the hybrid MPS. To improve upon this method, A2C is utilized as it is more sample efficient and effective towards generating an optimal policy.

3.4.4 Advantage Actor-Critic

A downfall of implementing DQN is its actor-only structure. As described in [49], the gradient estimators for these methods have a large variance and new gradients are estimated independently of past gradients. Actor-critic methods were developed to enhance the policy gradient updates and increase sample efficiency by introducing a critic to the structure.

For A2C, first look at the baseline function approximator,

$$\Delta_\theta J(\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T-1} \Delta_\theta \log \pi_\theta(a_t|s_t) G_t], \tag{3.28}$$

in which then the expectation can be decomposed,

$$\Delta_\theta J(\theta) = \mathbb{E}_{s_0,a_0,\ldots,s_t,a_t}[\sum_{t=0}^{T-1} \Delta_\theta \log \pi_\theta(a_t|s_t)]\mathbb{E}_{r_{t+1},s_{t+1},\ldots r_T,s_T}[G_t]. \tag{3.29}$$

Notice that the second expectation is the Q-Value and can be written in final as:

$$\Delta_\theta J(\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T-1} \Delta_\theta \log \pi_\theta(a_t|s_t) Q_w(s_t, a_t)], \tag{3.30}$$

where w represents the Q-value that is determined by parameterizing the Q-function with a neural network.

With this method, the critic will estimate either the state-value (V) or the action-value (Q) and the actor will update the policy distribution in the direction suggested by the critic. It has been found that the state-value function makes an optimal baseline function. Thus, the advantage value can be calculated,

$$A(s_t, a_t) = Q_w(s_t, a_t) - V_v(s_t), \tag{3.31}$$

and the relationship between Q and V can be taken from the Bellman optimality equation,

$$Q(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1})], \tag{3.32}$$

thus the advantage can be finally written as:

$$A(s_t, a_t) = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t). \tag{3.33}$$

The A2C agent was tested within the same environment of Yellowstone National Park. The resolution of the map was again 50x50. To reiterate, this method utilizes the aspect of RRT* replanning where replanning occurs on the basis of a distance ϵ from the approximated path.

The layout for this method is slightly different that the DQN model. Two neural networks are used to generate an action and output an estimate to the state-value. The model for training the agent can be seen in Figure 3.17.

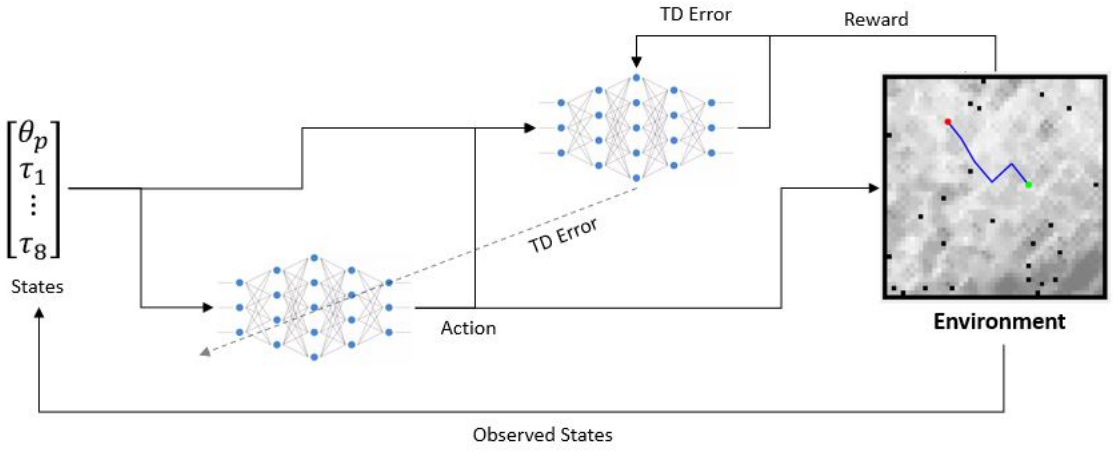


Figure 3.17.: Advantage Actor-Critic Model

During training all aspects of the environment change per episode to include terrain, obstacles, start, and goal. Thus, the trained agent develops a policy that is generalizable across multiple domains in a similar fashion to the DQN agent.

During training, the A2C agent went for 5000 episodes with a maximum of 400 steps per episodes. As seen in Figure 3.18, the agent reaches convergence around 2000 episodes. Most importantly, the agent averaged under 100 steps per episode as compared to 200 from the DQN model.

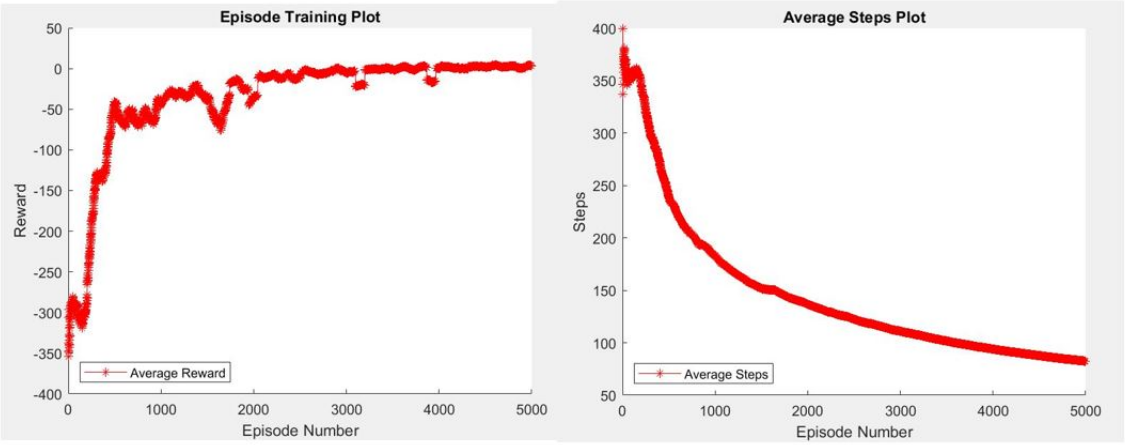


Figure 3.18.: Training Data: Advantage Actor-Critic

This main advantage of this method over Deep Q-learning is that it was much more efficient in finding a path to the goal. It will be easy to visualize in the final results section, but this agent was better suited towards managing global decisions and terrain considerations. The sampling efficiency and projection towards a target generated by the critic allowed for a better training set for the agent and development of a better overall policy. Integration of this method within the hybrid MPS is discussed in the next section.

3.4.5 Results

From the training data presented, it is clear that A2C provides an agent that can effectively adjust and locally repair the approximate global path with terrain consideration while choosing to minimize distance to the final goal. This method has demonstrated intelligent decision logic capability. To demonstrate this experimentally, key information has been recorded for varying map resolutions. The values are approximate averages across as many as 10 simulations per method and are clear enough to circumvent further testing. The results are seen in the tables below.

Table 3.1.: Reinforcement Learning Method Comparison: 20x20

Map Resolution: 20x20				
Method	Episodes	Convergence	Avg. Steps	Time to Train
Tabular Q	60	yes	25	50 sec
DQN	800	yes	74	14 min
A2C	750	yes	51	13 min

Table 3.2.: Reinforcement Learning Method Comparison: 50x50

Map Resolution: 50x50				
Method	Episodes	Convergence	Avg. Steps	Time to Train
Tabular Q	n/a	no	n/a	n/a
DQN	2000	yes	184	2 hr
A2C	2000	yes	86	1.5 hr

Table 3.3.: Reinforcement Learning Method Comparison: 80x80

Map Resolution: 80x80				
Method	Episodes	Convergence	Avg. Steps	Time to Train
Tabular Q	n/a	no	n/a	n/a
DQN	3900	yes	241	4.2 hr
A2C	3500	yes	119	3.6 hr

Thus, the goal is to integrate the agent within the overall hybrid MPS system developed. In theory, the UGV agents would not plan ahead with A*, but simply adjust their paths with the integrated policy. Local navigation sensors would have to take over in conjunction with the consensus-based collision avoidance since future direction

of each UGV is uncertain. For SA errors, each UGV agent measures distance from the approximate path as it stands within the environment and compares this with a predetermined value, ϵ. This maintains the SA consideration and ensures results that are near optimality. As a reference to how the system integrates, pseudocode is provided below.

Algorithm 1: Hybrid Method with Deep RL

Result: High level hybrid algorithm with deep RL for risk-aware planning
Initialization

$\{I_a, I_b\} \in I_A, \quad \{I_u, I_v, I_w\} \in I_T, \quad t \leftarrow 0$
while *tasks not assigned* **do**
    Auction,
    Consensus,
    Validation.
end
while $I_T \notin \emptyset$ **do**
    $t = t + 1$,
    $q_a(t) = W(t)$.
    $q_b(t) = \pi(s)$.
    CBCA
    if $|P_b - W_{approx}| > \epsilon$ **then**
        Rerun IACA
    end
end

Figure 3.19.: Hybrid MPS DRL Pseudocode

For final simulation, a random environment of 50x50 resolution was generated along with random obstacles, start, and goal location for 4 agents: 2 UAVs and 2 UGVs. There were a total of 6 tasks: 2 UAV, 2 UGV, and 2 heterogeneous. A visual of the simulation with subsequent discussion can be seen in Figure 3.20.

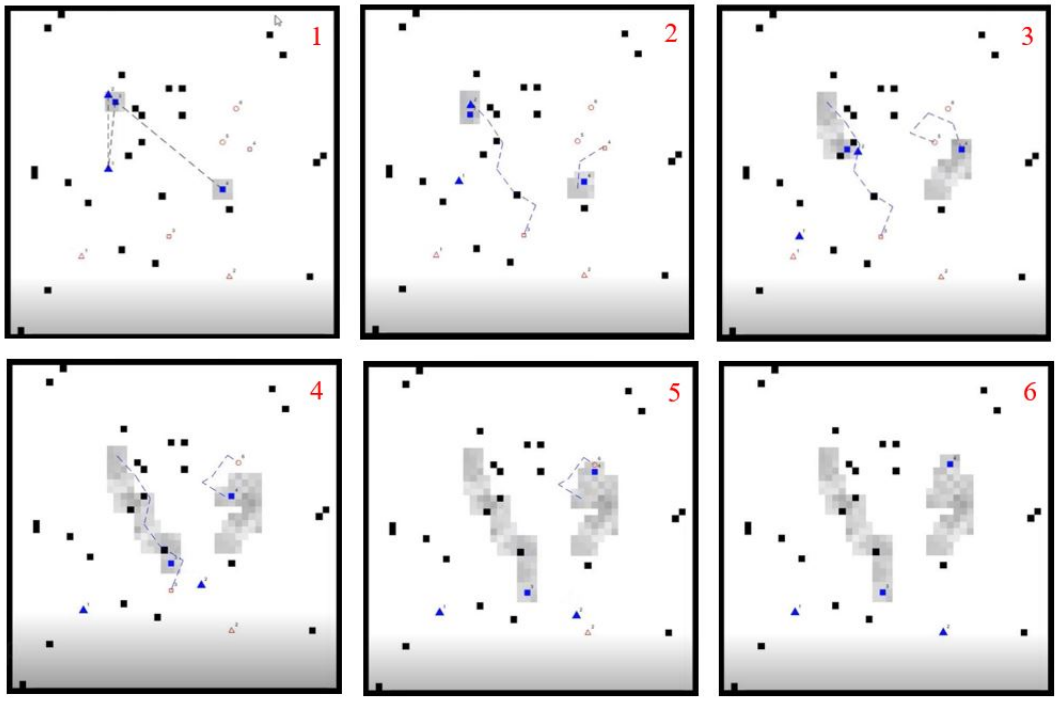


Figure 3.20.: Hybrid MPS DRL Simulation

Per the diagram shown, each agent begins with randomly designated strongly connected network. Consensus is reached within the system in a distributed fashion and all tasks are allocated through a market-based approach. UAV agents disperse along their straight line Euclidean distance paths and UGVs begin to move according to their policy with approximate paths and terrain information as inputs. UGVs continue towards their goal until they reach a specified distance from their approximate path. In this example that criteria is not met, but had it occurred agents would rerun task allocation for the appropriate tasks.

Overall, deep reinforcement learning can handle large data sets and generalize a policy for quality solution in an environment that the agent has never seen before. With an optimal policy, the agent will be able to escape local minima and provide a good solution as repair to the sub-optimal approximate RRT* solution. Utilization of deep reinforcement learning allows for real-time performance with constant updates from the environment where a novel replanning method based on the D* or

evolutionary algorithms lead to the freezing robot issue in hardware deployment. The integration of deep reinforcement learning within the overarching MPS framework allows for a system that is scalable to multiple agents and multiple tasks, resilient to time-bounded attacks, and computationally efficient given a complex and uncertain environment.

4. SUMMARY

4.1 Conclusion

The problem addressed within this study concerned itselft with efficient, resilient, and robust control of a multi-agent heterogeneous team given complete and incomplete information. The research was mainly directed towards a robotic team completing missions such as search and rescue and surveillance within a large, outdoor environment.

For a heterogeneous multi-agent team within a complete environment, the method proposed consists of three main parts to solve the mission planning problem: approximate planning, task allocation, optimal planning. An approximate RRT* planner is used to quickly place bids for UGV agents and UAV agents place bids based on Euclidean distance. Once all agents have a bid for each task, allocation is commenced through the IACA algorithm extension for heterogeneous tasks increasing robustness of the system. Planning is commenced with the A* algorithm for a single agent, single task scenario. Optimality of this method is desirable and computation is real-time for the isolated situation. Simplicity is maintained with the development of consensus-based local collision avoidance between agents where previous computational path check is not required. Approximate and optimal solutions are compared and an excess of sub-optimality initiates a new iteration of IACA extension. The nodes of RRT* are increased to provide a better approximation and manage sub-optimality throughout system.

To introduce a subset of the incomplete information problem, unknown terrain is introduced to the ground vehicle. The goal of the ground vehicle is to reach a task while minimizing risk of failure due to terrain considerations.

The first proposed method is a hybrid solution that combines aspects of approximate planning for a global solution and a genetic algorithm for local repair. The agent will utilize this approximate path generated with RRT* to aid in single agent, single task path planning. This will replace A* with windowed replan (CBCA) in the complete map scenario. The UGV agent generates traversability indices for surrounding locations through a Mamdani inference model. A genetic algorithm solves optimal next step for the agent while minimizing deviation from the approximated path and maximizing traversability. This solution is effective in simple environments but can become stuck in local minima for complex terrains and could lead to the freezing robot problem.

The second proposed method utilizes all known discrete information with continuous traversability index states as inputs to a reinforcement learning model. Known information of obstacles, agent location, and terminal location are paired with unknown information of surrounding traversability indices as state inputs. Three methods are addressed within the paper to include Tabular Q-learning, DQN, and A2C methods. Overall, A2C method demonstrated the most efficient results for UGV path planning. For the method itself, A2C utilizes all state inputs into fully connected network which acts as a universal nonlinear function approximator to the Bellman update equation. Thus, this concept of Advantage Actor-Critic is used to update state-action values (or Q-values), compare this through the actor network distribution, and develop an optimal policy for a UGV agent operating within unknown terrain environment. The goal is for a ground agent to learn, from experience, the optimal next best move as it discovers information from the environment about surrounding traversability. The policy developed is generalizable to many terrains and unknown environments and thus is the overarching hybrid MPS.

4.2 Future Work

Looking to the future, work will be needed to develop an appropriate collision avoidance system for the UGV agents. As of now it is an assumption that local navigation will direct agents through CBCA. This aspect of local navigation, however, is another widely studied area in robotics. Considering assumptions taken in the proposed solution, the two main factors are holonomic motion and no UAV collision. To create a more realistic representation of the environment for implementation, these two factors will need to be addressed.

Lastly, upon consideration of these assumptions, hardware implementation would be the next step. A multi-robot team consisting of drones such as Parrot and UGVs such as the Jackal would suit well as a testing platform for missions such as surveillance or search and rescue.

REFERENCES

[1] Standford. Robotics: A brief history. Website, 1998.

[2] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[3] Jacob T. Schwartz and Micha Sharir. On the "piano movers" problem. *Advances in Applied Mathematics*, pages 298–351, 1983.

[4] Julie Adams. *Robotic Technologies for First Response*, volume 27, chapter 1, pages 3–33. CRC Press, 1 edition, 2010.

[5] Mario Garzon, Joao Valente, David Zapata, and Antonio Barrientos. An aerial-ground robotic system for navigation and obstacle mapping in large outdoor areas. *Sensors*, pages 1247–1267, 2013.

[6] Jianqiang Li, Genqiang Deng, Chengwen Luo, Qiuzhen Lin, Qiao Yan, and Zhong Ming. A hybrid path planning method in unmanned air/ground vehicle (uav/ugv) cooperative systems. *IEEE Transactions on Vehicular Technology*, 65(12):9585–9596, 2016.

[7] Daniel Camara. Cavalry to the rescue: Drones fleet to help rescuers operations over disasters scenarios. *IEEE Conference on Antenna Measurements and Applications*, pages 1–4, 2014.

[8] Luqi Wang, Daqian Cheng, Fei Gao, Fengyu Cai, Jixin Guo, Mengxiang Lin, and Shaojie Shen. A collaborative aerial-ground robotic system for fast exploration. *Cornell University*, pages 1–6, 2018.

[9] Hadi Jahanshahi and Naeimeh Najafizadeh Sari. Path planning and robot's hardware. In *Robot Path Planning Algorithms: A Review of Theory and Experiment*, chapter 4, pages 94–132. University of Tehran, 2018.

[10] Brian Yamauchi. A frontier-based approach for autonomous exploration. *CIRA*, 1997.

[11] Weiran Yao, Nen Wang, and Naiming Qi. Hierarchical path generation for distributed mission planning of uavs. *IEEE 55th Conference on Decision and Control*, pages 1681–1686, 2016.

[12] D. C. Guastella, L. Cantelli, C.D. Melita, and G. Muscato. A global path planning strategy for a ugv from aerial elevation maps for disaster response. *Proceedings of the 9th International Conference on Agents and Artificial Intelligence*, pages 335–342, 2017.

[13] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. Multi-robot task allocation: A review of the state-of-the-art. *Cooperative Robots and Sensor Networks*, pages 31–51, 2015.

[14] Christian Henkel, Jannik Abbenseth, and Marc Toussaint. An optimal algorithm to solve the combined task allocation and path finding problem. Technical report, University of Stuttgart, 2019.

[15] B. L. Brumitt and A. Stentz. Grammps: A generalized mission planner for multiple robots in unstructured environments. *IEEE International Conference on Robotics and Automation*, pages 1564–1571, 1998.

[16] K. Al-Yafi and H. Lee. Mtap-masim: A multi-agent simulator for the mobil task allocation problem. *IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 25–27, 2009.

[17] B. Coltin and M. Veloso. Mobile robot task allocation in hybrid wireless sensor networks. *Intelligent Robots and Systems*, pages 2932–2937, 2010.

[18] J. Higuera and G. Dudek. Fair subdivision of multi-robot tasks. *IEEE International Conference on Robotics and Automation*, pages 3014–3019, 2013.

[19] G. Ping-an, C. Zi-xing, and Y. Ling-li. Evolutionary computation approach to decentralized multi-robot task allocation. *International Conference on Natural Computation*, pages 415–419, 2009.

[20] N. Atay and B. Bayazit. Mixed-integer linear programming solution to multi-robot task allocation problem. Technical report, Washington University, St. Louis, 2006.

[21] M. B. Dias. A comprehensive taxonomy for multi-robot task allocation. Technical report, Robotics Institute, Carnegie Mellon University, 2004.

[22] Han-Lim Choi, Luc Brunet, and Jonathan How. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4):912–926, 2009.

[23] Xuan Wang. Resilient multi-agent task allocation on constrained robotic platforms. Technical report, Purdue University, 2018.

[24] Peter E. Hart. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, pages 514–532, 1968.

[25] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Iowa State University, 1998.

[26] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. Technical report, Utrecht University, 1994.

[27] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, pages 40–66, 2015.

[28] Ko-Hsin Cindy Wang and Adi Botea. Mapp: A scalable multi-agent path planning algorithms with tractability and completeness guarantees. *Journal of Artificial Intelligence Research*, pages 55–90, 2011.

[29] Anthony Stentz. Optimal and efficient path planning for partially known environments. In *Intelligent Unmanned Ground Vehicles*, pages 203–220. Springer, 1997.

[30] Yu Fan Chen, Shih-Yuan Liu, Miao Liu, Justin Miller, and Jonathan P. How. Motion planning with diffusion maps. *International Conference on Intelligent Robots and Systems*, pages 1423–1430, 2016.

[31] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P. How. Socially aware motion planning with deep reinforcement learning. *International Conference on Intelligent Robots and Systems*, pages 1343–1350, 2017.

[32] Savas Ozturk and Ahmet Emin Kuzucuoglu. Optimal bid valuation using path finding for multi-robot task allocation. *Springer Science and Business Media*, pages 1049–1062, 2012.

[33] J Giesbrecht. Global path planning for unmanned ground vehicles. Technical report, Defense Research and Development Canada, 2004.

[34] Weiran Yao, Naiming Qi, Neng Wan, and Yongbei Liu. An iterative strategy for task assignment and path planning of distributed multiple unmanned aerial vehicles. *Aerospace Science and Technology*, pages 455–464, 2019.

[35] A. Tahbaz-Salehi and A. Jadbabaie. A one-parameter family of distributed consensus algorithms with boundary: From shortest paths to mean hitting times. *Decision and Control*, pages 4664–4669, 2006.

[36] E. W. Dijkstra. A note on two problems in connexion with graphs. Technical report, Mathematisch Centrum, Amsterdam, 1959.

[37] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. *IEEE International Conference on Robotics and Automation*, pages 1928–1935, 2008.

[38] Fabio Molinari and Jorg Raisch. Automation of road intersections using consensus-based auction algorithms. *Proceedings of the American Control Conference*, pages 1–22, 2018.

[39] J. Peterson, H. Chaudhry, K. Abdelatty, J. Bird, and K. Kochersberger. Online aerial terrain mapping for ground robot navigation. *Sensors*, pages 1–22, 2018.

[40] Yusuke Tanaka, Yonghoon Ji, Atsushi Yamashita, and Hajime Asama. Fuzzy based traversability analysis for a mobile robot on rough terrain. *IEEE International Conference on Robotics and Automation*, pages 3965–3970, 2015.

[41] Open Topography. Lidar data - yellowstone national park, 2009.

[42] Edwin A. Williams and William A. Crossley. Empirically-derived population size and mutation rate guidelines for a genetic algorithm with uniform crossover. *Soft Computing in Engineering Design and Manufacturing*, pages 163–172, 1998.

[43] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. The MIT Press, Cambridge, Massachusetts, 2018.

[44] Christopher J. C. H. Watkins. Q-learning. *Machine Learning*, pages 279–292, 1992.

[45] Richard Bellman. The theory of dynamic programming. Technical report, The RAND Corporation, 1954.

[46] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE 86*, pages 2278–2324, 1998.

[47] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, and Andrei A. Rusu. Human-level control through deep reinforcement learning. *Nature*, pages 529–533, 2015.

[48] J. ONeil, B. Pleydell-Bouverie, D. Dupret, and J. Csicsvari. Play it again: Reactivation of waking experience and memory. *Trends Neurosci*, pages 220–229, 2010.

[49] Vijay R. Konda and John N. Tsitsikilis. Actor-critic algorithms. Technical report, MIT, 2000.