Coordinated Searching and Target Identification Using Teams of Autonomous Agents

Christopher Lum

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

2009

Program Authorized to Offer Degree: Aeronautics & Astronautics

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Christopher Lum

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of the Supervisory Committee:

_____

Juris Vagners

Reading Committee:

_____

Juris Vagners

_____

Rolf Rysdyk

_____

Dieter Fox

Date: _____

Signature_____

Date_____

University of Washington

Abstract

Coordinated Searching and Target Identification Using Teams of Autonomous Agents

Christopher Lum

Chair of the Supervisory Committee:
Professor Emeritus Juris Vagners
Aeronautics & Astronautics

Many modern autonomous systems actually require significant human involvement. Often, the amount of human support and infrastructure required for these autonomous systems exceeds that of their manned counterparts. This work involves increasing both the tactical and strategic decision making capabilities of various autonomous systems. The application considered is the problem of searching for targets using a team of heterogeneous agents. The system maintains a grid-based world model which contains information about the probability of a target being located in any given cell of the map. Agents formulate control decisions for a fixed number of time steps using a modular algorithm that allows for capabilities and characteristics of individual agents to be encoded in several parameters. The resulting search patterns executed by the agents guarantee an exhaustive search of the map in the sense that all cells will be searched sufficiently to ensure that the probability of a target being located in any given cell is driven to zero. This system was simulated using high fidelity simulations with heterogeneous agents in complex and dynamic environments. After performing successfully in simulation, these algorithms were then verified and validated on a distributed human-in-the-loop simulator. This system allows a human operator to handle low level tasks such as state stabilization and signal tracking while preserving the contributions of the autonomous algorithm. Finally, flight test results are presented showing the benefits of augmenting a human system with these types of autonomous algorithms.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

A	Original set of arcs (edges) in network
$\tilde{A}$	Arcs to add in network
A'	Total arcs in network
B	Set of $\overline{z}$ values defining spatial domain of occupancy based map
B_R	Locations reachable by agent in d steps
$\tilde{B}$	Set of $\overline{z}$ values defining center of occupancy based map cells
$\tilde{B}_R$	Cell centers reachable by agent in d steps
$\overline{b}$	Total magnetic field
C_s	Node subset at step s
C_t	Sum of particle filter weights at time t
$D(j)$	Span interval of arc j
d	Prediction horizon, number of waypoints in path
d_i	Minimum distance from generator i to set $\tilde{B}_{max}$
$d^-(j)$	Lower span interval of arc j
$d^+(j)$	Upper span interval of arc j
E	Incidence matrix of network
f	Feature vector
$f_0()$	Instantaneous cost function for path following problem
$f_C()$	Returns cumulative score of occupancy map cells covered by arc j
$f_\chi()$	Reward function for course deviation in ($\wp_2$)
$f_d()$	Reward function for distance in ($\wp_2$)
$f_G()$	Returns node coordinate ($f_G : i \in I \rightarrow \Re^2$)
$f_h()$	Reward function for high score cell in ($\wp_2$)
$f_M()$	Returns minimum occupancy map score covered by arc j

$g()$	Sampling function
$H(Q)$	Entropy of partition Q
$H(Q^s, Q^o)$	Joint entropy between partition Q^s and Q^o
h	Sensor reliability factor in range $[0, 1]$
$h()$	Target magnetic signature function
$\tilde{h}_i()$	False anomaly i signature function
I	Set of nodes (vertices) in network
I	Index of agent closer to $\tilde{B}_{max}$ than any other agent
$I(Q^s, Q^o)$	Perspectives mutual information between partitions Q^s and Q^o
I_n	Set of integers from 1 to n
$I_{\hat{n}(m)}$	Indices corresponding to runs where at least m agents find the target
$j \sim (i, i')$	Arc starting at node i and ending at node i'
$J()$	Total cost for path following performance
$J_0()$	Total reward function for $(\wp_2)$
K	Number of data points between waypoints
L_x, L_y	Width and height of map cell in x-direction and y-direction, respectively
l	Lower bound for box set in $(\wp_2)$
M	Total number of particles
M	Number of function range partitions
M	Number operations for $(\wp_{3,b})$ w/ Progressive Frontier Algorithm
M'	Number operations for $(\wp_{3,b})$ w/ combinatorial approach
N	Number of times spatial network algorithm is run
N	Number of function domain partitions
$N(\mu, \sigma^2)$	Gaussian distribution with mean μ and variance σ^2
N^+, N^-	Starting/ending node sets in min path algorithm
N_x, N_y	Number of columns and rows, respectively, of occupancy based map
$\tilde{N}(i)$	Number of agents who find target in run i
$\tilde{N}_{ave}$	Average number of agents who find target

n	Dimension of particle state in $(\wp_2)$
$n^{[m]}(t)$	Noise added to particle m at time t in $(\wp_2)$
$\hat{n}(m)$	Number of runs where at least m agents find the target
P_E, P_N	East and north position of agent, respectively
P	Path of arcs from N^+ to N^-
P	Set of Voronoi generator points
P_i	Set in $(\wp_{3,a})$
$(\wp_1)$	Subproblem of creating future world state estimates
$(\wp_2)$	Subproblem of finding desirable cells for agent to search
$(\wp_3)$	Subproblem of finding trajectories for agent's flight path
$(\wp_{3,a})$	Problem of finding feasible waypoints
$(\wp_{3,b})$	Problem of adding arcs to network
$(\wp_{3,c})$	Problem of finding optimal path in network
$p(A\|B)$	Conditional probability of A given B
p_A	Starting waypoint of current track segment
p_B	Ending or active waypoint of current track segment
p_i	Waypoint i of path
p_i	Voronoi generator point i
Q	Domain partition of function
Q_i	Set in $(\wp_{3,a})$
Q_j	Element j in partition Q
$q(a, b)$	Calculates absolute angular difference between angles a and b
R_i	Set in $(\wp_{3,a})$
R_{max}	Maximum distance agent can travel in a d steps
r	Radius of radial occupancy map update
$\hat{r}$	Distance agent can estimate target state
r_{max}	Distance agent can travel in a single step
$rand(l, u)$	Function which produces a uniformly distributed random number in the range $[l, u]$

$S(i,k)$	Cumulative map score at step k for run i
$S_{ave}(k)$	Average cumulative map score at step k
s_j	Limit of range partitioning defining δ_j
s_k	Score of given occupancy map cell at step k ($p(X_k = x_B)$ at step k)
T	Number of particle filter updates for a trajectory
$T_m(i)$	Time when the m^{th} agent finds the target for run i
$T_{m,ave}$	Average time when the m^{th} agent finds the target
t_i	Desired arrival time at waypoint p_i
u	Upper bound for box set in ($\wp_2$)
$\overline{u}$	Control vector
u_0	Initial potential on nodes
$\overline{V}$	Voronoi diagram
V_{tgt}	Actual velocity of target
V_a	Airspeed
V_{max}	Max velocity of agent
$V_{max,tgt}$	Max velocity of target
V_t	Variance of weights at time t
$V(i,k)$	Variance of map scores at step k for run i
$V_{ave}(k)$	Average variance of map scores at step k
W_t	Weight set at time t
$\overline{w}$	Decision vector in $\Re^{2 \cdot d}$
$\overline{w}^{\star}$	Optimal solution to ($\wp_3$) (path from current location to $\overline{z}^{\star}$)
$w_t^{[m]}$	Weight on particle m at time t
X	A box set
$\overline{x}$	State of agent for target identification purposes
x	Spatial coordinate
x_0	Agent's starting coordinate in ($\wp_3$)
x_d	Agent's desired ending coordinate in ($\wp_3$)

$x_{agt}(t)$	Agent's location at time t ($\in \Re^2$ or $\Re^3$)
x_A, x_B	Event of target not in cell and target in cell states, respectively
x_{min}, x_{max}	Minimum and maximum x value of occupancy based map
$x_p(t)$	Point on path segment closest to $x_{agt}(t)$
$x_w(k, \overline{z})$	Actual state of the world at time step k and location $\overline{z}$
$\hat{x}_w(k, \overline{z})$	Estimated world state at time step k and location $\overline{z}$
$\overline{x}_{agt}(k)$	Agent state $(P_E \ P_N \ \chi)^T$ at step k
$\overline{x}_{tgt}(k)$	State of the target at time step k
$\hat{x}_{tgt}(k)$	Estimated target state at time step k
$x^{[m]}(t)$	Particle m at time t in $(\wp_2)$
$\tilde{x}^{[m]}(t)$	Resampled particle m at time t (no noise added yet) in $(\wp_2)$
$x_{w,nom}$	Nominal score of occupancy based map
$\overline{x}^\star$	Average of all particles after particle filter terminates
y_{min}, y_{max}	Minimum and maximum y value of the occupancy based map
$\overline{z}$	(x, y) coordinate $(P_E \ P_N)^T$
$\overline{z}^\star$	Most desirable location in $(\wp_2)$
$\overline{z}_{agt}$	Agent's current (x, y) position
z_t	Sensor measurement made by agent at time t
$z_t^{[m]}$	Sensor measurement expected by particle m
z_A, z_B	Observation of target not in cell and in cell, resectively
$\overline{z}_0$	Agent's current coordinate
$\overline{z}_H$	Location of cell center with highest score in map and closest to agent
$\overline{z}_h$	Location of cell center in agent's reachable set closest to $\overline{z}_H$
$\overline{z}_s$	Location of cell center of same cell that agent is currently in
α	Scalar tradeoff parameter for $\hat{x}_w(k + d, \overline{z})$
β	Scalar tradeoff parameter for $f_\chi(\overline{z})$
χ_t	Particle filter set at time t
χ_{tgt}	Course angle of target

Δ	Range partitioning of function
$\Delta\psi$	Change in heading
Δt	Time step used to record data
ΔT	Time between steps/waypoints
δ_j	Element j of range partition Δ
ϵ	Small number
η	Maximum score of cell within agent's reachable set
γ	Scalar tradeoff parameter for $f_d(\overline{z})$
$\Gamma()$	Predicted future state function
$\kappa^\star$	Scalar parameter used when computing pilot path following performance
λ	Scalar in $[0, 1]$ denoting agent's capability to estimate target state and control
$\mu(Q)$	Lebesgue measure of set Q
$\overline{\mu}$	center of radial occupancy map update
ν	max $f_C()$ over all j
ν	Domain partition of function using connected components
ν_i	Element i in partition ν
ψ	Heading
σ_{sensor}	Standard deviation of sensor
τ	Time constant for occupancy based map scores
φ	Distance from agent to target
ξ	Confidence factor in radial occupancy map update
ζ_i	Flag used in definition of $\overline{z}_{h_i}$

ACKNOWLEDGMENTS

This dissertation is the product of many years of research at the University of Washington in the Department of Aeronautics and Astronautics. This department has been instrumental in fostering and furthering my academic career and I am very grateful for my positive experiences and supportive peers and advisors. First and foremost, I would like to thank my advisors, Professor Juris Vagners and Dr. Rolf Rysdyk. Both individuals provided invaluable guidance and motivation throughout my career in graduate school. They created an environment conducive to learning and focused on both theory and application. Without their influences, I would not be where I am today. This sentiment extends to the other members of my committee, Professor Dieter Fox, Professor Mehran Mesbahi, and Professor Nathan Kutz. These individuals were sources of inspiration and exposed me to many new ideas in their perspective fields, much of which I have incorporated into this dissertation. Additionally, I would like to thank other excellent faculty and staff in the Department of Aeronautics and Astronautics, including Professor Uy-Loi Ly, Professor Kristi Morgansen, Professor Adam Bruckner, Wanda Frederick, and Robert Gordon. All of these individuals played a large positive role in my academic journey.

I would also like to extend my gratitude to my labmates in the Autonomous Flight Systems Laboratory. Dr. Anawat Pongpunwattana, Dr. Richard Wise, and Matthew Rowland were extremely helpful during many projects. But of all my peers, my greatest thanks goes to Dr. Daniel Klein. He was one of the most helpful, knowledgable, and intelligent individuals that I had the pleasure of meeting during my tenure at the UW. He served as a great research accomplice and companion.

Financially, I would like to thank the George E. Solomon Academic Award, University of Washington Aeronautics and Astronautics Alumni Scholarship, the Andris Vagners Memorial Fellowship, the Washington Technology Center (grants F04-MC2 and F04-MC3), the

Osberg Family Trust fellowship, the NASA Space Grant Consortium Graduate Fellowship, and the Air Force Office of Scientific Research (grant FA-9550-07-1-0528) for their generous support. Additionally, I would like to thank the Boeing Company, the Insitu Group, and Fugro Airborne Surveys for their collaborative efforts and data sharing.

I would like to thank my family and friends, including my mother, Judy, and my brothers, Jonathan and Timothy. They always supported my decisions and provided me with many needed opportunities to relax and focus on other things besides research. Research projects may come and go, but I know they will always be there for me.

Finally, I would like to thank my father, Matthew. He is the one who originally motivated me to become an engineer and he continues to inspire me and my family with his endeavors and leadership. He serves as my role model of what an engineer and father should be. I am extremely proud to be your son.

DEDICATION

To my wonderful wife, Alison. You are all I have ever wanted and all I will ever need.

Chapter 1

INTRODUCTION

This chapter serves as an introduction to the main dissertation. A brief background section reports on some current autonomous systems and the state of the art in various fields. Some of the major contributions of this work are also presented along with the general layout and structure of this dissertation.

1.1 Background and Motivation

The field of modern unmanned systems is a very well studied and engaging subject. It is also an extensive field that spans many subcategories. It encompasses topics such as machine learning, artificial intelligence, path planning, task allocation, human factors, systems integration, and many more. The study and application of these disciplines is required in order to develop a viable autonomous system with a large set of capabilities. For an autonomous system to be a useful system, it must be capable of deployment in a wide variety of missions and able to perform required tasks competently. The situations that these systems encounter in their missions often require that the non-human components must execute tasks or make strategic decisions that are typically handled by humans. For example, search-and-surveillance type missions typically involve many tasks which are currently handled by human operators. Tasks of assigning regions to search and coordinating sensor measurements are usually left to human decision making and analysis. In a noisy environment, it becomes difficult for a human operator to classify sensor readings and assign confidence in these readings. Determining regions of high target-location probability and coordinating nearby agents to converge on a particular spot while allowing other vehicles to continue searching is also difficult. These missions are often complex and difficult operations due to dynamics in the environment. Targets may not be stationary and observations become less reliable as time progresses. In addition, many of these search missions are initiated

with a poor estimate of the target's actual position. To aggravate matters, often the target is moving or evading. These problems illustrate some of the challenges encountered by an autonomous system during a mission. The current state of the art is such that in many cases, human operators still outperform autonomous systems. Maximizing performance of these systems often leads to many situations where humans are required to interact with these systems and the size of the crew required to operate these "unmanned" systems exceeds that of traditional manned systems. Therefore, the primary limitation to concurrent operation of multiple vehicles remains the lack of autonomy of these vehicles. This lack of autonomy when performing tasks such as path planning and target identification in the face of rapidly changing conditions severely increases operator workload. In order to minimize human interaction, an efficient method for adding autonomy to these systems is required.

The potential application and utility of these types of autonomous systems is almost limitless. These agents can be used in situations which may be too dull, dirty, or dangerous for a manned system. For example, the functionality of such systems will be extendible to a wide variety of system configurations such as forestry patrol, border patrol, maritime search, and ISR (intelligence, surveillance, and reconnaissance) missions. Homeland Security Presidential Directive (HSPD)-5 directed the development of a national plan for response to manage radiological, hazardous chemical, or biological release incidents. The algorithms developed in this dissertation have direct application to hazardous substance detection and tracking tasks. The Environmental Protection Agency and Department of Energy have indicated interest in evaluation of unmanned aerial vehicles (UAVs) for these applications. The development of autonomous algorithms will increase the performance of these agents, thus providing a direct outlet for military and commercial market demands requiring multiple vehicles. Currently, these types of operations are restricted to the experimental robotics research community.

In 2004, there were over 250 models of UAVs in production. In the United States alone, there were over 160 models manufactured by over 40 different companies ranging in size from Boeing to small businesses sustained by small business innovation research (SBIR) grants. The annual world wide expenditures on UAV systems exceeded $2 billion [84]. These numbers have since increased dramatically and the demand for capable autonomous

systems continues to grow.

However, at the present time, the autonomous decision making capabilities of these systems are insufficient to meet the demands of many complex missions. Many of the subsystems and algorithms have limitations which hinder the performance of the overall system. Removing or ameliorating these limitations is the goal of this research.

1.2 Problem Statement

The overwhelming majority of current missions tasked to autonomous systems revolve around intelligence, reconnaissance, and surveillance (ISR). These are tasks such as tracking or loitering to observe or searching a domain for a target. This work seeks to increase the autonomy and capabilities of a heterogeneous team of agents involved in a search and surveillance type mission. The system maintains a world model which includes an estimate of possible target states and the state of the environment. The issue of compelling agents to converge on possibly moving targets and continuing to search new regions is formulated as a model predictive control problem. The world model is propagated forward in time and autonomous strategic decisions are made based on the predicted future state of the world. Agents formulate control decisions for a fixed number of time steps by optimizing a team based objective function which allows for control and timing constraints.

The overall objective of this work is to develop robust and scalable control laws to apply to a heterogeneous group of agents so that they operate in a cooperative fashion to achieve a common goal. To sharpen the research focus, we consider the specific mission of coordinating a group of agents to search a two dimensional space for a target based on its magnetic signature. Although this goal may seem narrow in scope, many of the technologies developed for this application are easily adaptable to more general tasks encountered in autonomous systems. Developing a strategic level searching algorithm often requires developing and implementing lower level algorithms to support it. Several of these algorithms do not exist and must be developed for this application. For example, methods to perform target identification and path planning must be developed and implemented to support the searching algorithm. However, other lower level algorithms such as a state stabilization system are not the main focus of the research, so methods to test the strategic algorithm

while circumventing these low level algorithms is also needed.

1.3 Contributions of this Research

A multitude of algorithms currently exists that allow many autonomous systems to perform a wide variety of tasks. The state of the art and works of others are presented in Chapter 2. This section serves to outline the unique contributions of this research to the field of autonomous systems.

This dissertation describes the development of several autonomous algorithms which are used to solve the problem of searching for a target using a team of heterogeneous agents. The main contribution of this work is a modular, autonomous algorithm which manages a group of heterogenous agents involved in a search and surveillance mission. This algorithm is scalable to a large number of agents and maintains a single belief of the state of the world at any given time. The algorithm has several unique characteristics, such as guaranteed performance and coverage, which make it suitable for these types of missions. The algorithm captures a search strategy that can be used to coordinate a searching mission for a static or dynamic target. The resulting search strategy ensures that the team of agents will exhaustively search the map for the target in the sense that the target will be found or the probability that the target is located in any given cell of the map approaches zero.

In addition to the main searching algorithm, this dissertation discusses several algorithms that are used in the overall searching strategy but can be applied to different situations with equal success. These include an algorithm for autonomous path planning in an complex environment. This is a graph based approach that provides paths which meet timing constraints and are guaranteed to be feasible for a given agent. Although the results presented here do not guarantee accuracy within a user defined limit, the advantages over previous work by others include the ability to satisfy agent performance and timing constraints and extensions to path planning in three dimensional space.

Another major contribution is the development of an autonomous target identification algorithm. This system uses low dimensional, simple sensor measurements and adds system consistency and temporal smoothness by incorporating information about the agent's motion and sensor model to develop a highly accurate classifier which allows the agent to

determine if it has encountered the target or not. This is a system that processes the output from a low dimensional sensor and extracts relevant features from which to train a classifier. We investigates the use of particle filters instead of the traditional hidden Markov models to incorporate temporal smoothness and regularity into the target identification system.

These algorithms were verified and validated using a high fidelity and human-in-the-loop simulator developed in the course of the research. This simulator isolates the contributions and functionality of a strategic algorithm by allowing a human operator to handle lower level tasks. This system can be used to expedite the testing of strategic algorithms without the burden of developing low level control laws. The simulator can be used to train operators in preparation of flight tests and field deployment of strategic algorithms. Results from simulation, human-in-the-loop simulation, and actual flight tests are presented as well.

1.4 Document Layout

This dissertation is partitioned into several chapters. Chapter 2 reviews previous work in the field of autonomous systems and algorithms in general. It also defines the distinctions between different types and classes of autonomous algorithms. In this application, there are two main tasks that the team of agents must perform. One is the tactical task of performing autonomous target identification based on sensor readings from the agents. Chapter 3 details the autonomous target identification algorithm and includes results and simulations.

Once the agents are able to perform tactical target identification, the system requires a high level strategy for searching the domain for the targets. The framework for representing environmental states is presented in Chapter 4. This lays the foundation for the overall strategic searching algorithm which is discussed in detail in Chapter 5. This is a large chapter which is broken down into several smaller sections which mirror the algorithm's modularity.

Chapter 6 provides simulation results and performance analysis of the search strategy including comparisons with alternative searching methods. Once the searching algorithm was proven in simulation, it was used in conjunction with the human-in-the-loop simulator to prepare for deployment. This process is discussed in Chapter 7. After passing verification and validation on the simulator, final flight test results are presented in Chapter 8. Finally,

conclusions and future research directions are presented in Chapter 9.

Chapter 2

LITERATURE REVIEW

The work presented in this dissertation covers many fields of research. Many of these subjects are well studied and extensive literature exists regarding pertinent methods and techniques. This chapter serves to highlight some of the related work done by others in specific areas related to this research. The limitations of these approaches are not detailed here. Instead, as the corresponding topic is discussed in the dissertation, the deficiencies of these methods are outlined which illustrate the advantages and need for the methods developed in this dissertation.

2.1 Autonomous Systems

The majority of this work falls under the umbrella classification of autonomous systems. This is a fairly broad classification in the sense that an autonomous system may refer to any system which contains decision making capabilities on some level. These systems typically start with a vehicle such as an aircraft or boat. These vehicles are augmented with autonomous algorithms to modify their behavior. A vehicle augmented in this manner with basic decision making capabilities is referred to as an agent. An autonomous agent is a complex system comprised of many separate algorithms. It is useful to define a metric for determining the class of a given autonomous algorithm. One common metric is to classify an autonomous algorithm by the level of autonomy it achieves and types of tasks it handles. Three common classifications are shown in Figure 2.1.

Figure 2.1 shows the first level of autonomy as the strategic level. These algorithms often operate at a low bandwidth and may be in charge of tasks such as mission planning. They are sometimes referred to as "high level control schemes." Next, the tactical level of control operates underneath the strategic level. This level of control is concerned with more specialized tasks such as path following or orbit coordination. Lastly, the dynamics

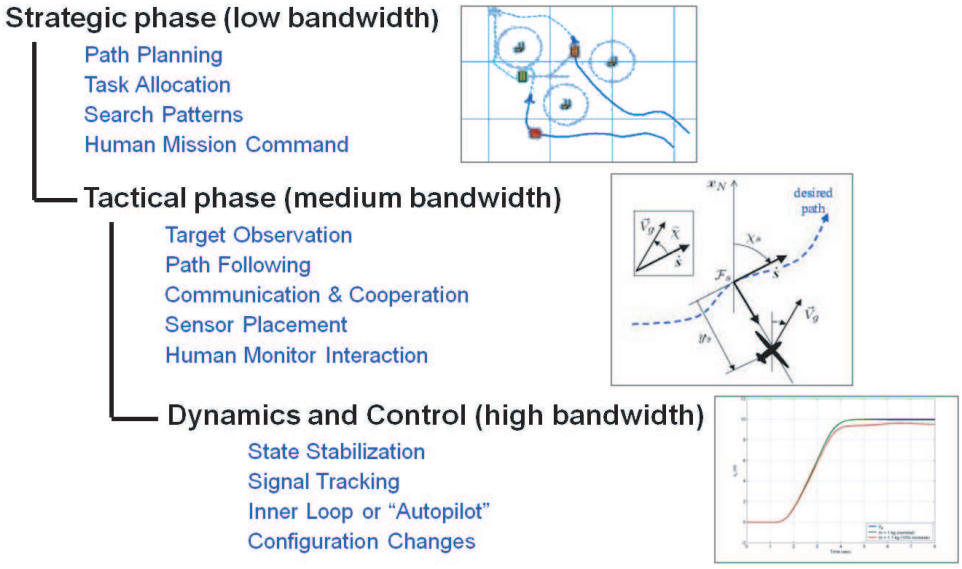


Figure 2.1: Different levels of autonomy.

and control level consists of the classical control problems such as state stabilization or signal tracking. These are the most specialized and "narrow-minded" and operate at a high bandwidth. A successful autonomous system uses algorithms from all different levels to work together to achieve a common goal. This hierarchical structure of automation and the challenges it entails has been studied and analyzed by groups such as Passino et al [12].

Others such as Clough [27], Huang et al [49], and Sholes [107] have proposed more detailed metrics for measuring a system's level of automation that operate in a similar fashion but provide a more detailed classification system.

Often, the terms "autonomous systems" and "autonomous algorithms" are used interchangeably. The distinction made in this work is that autonomous algorithms are the routines which manage specific tasks without human interactions and autonomous systems are comprised of many sub-systems including hardware and autonomous algorithms. The autonomous system may refer to a single actual agent or the entire team and infrastructure to manage a group of agents.

An example of a modern autonomous system is the Boeing/Insitu ScanEagle UAV. The ScanEagle is a medium sized unmanned aerial vehicle (UAV) that evolved from the

Aerosonde, the first aircraft to autonomously cross the Atlantic [79]. The original mission for the Aerosonde was the collection of weather data for meteorological purposes [47], [80]. The ScanEagle continues this tradition of data gathering and is used in a variety of missions. Variations of this system include models with a gimballed camera mounted in its nose which is used extensively for ISR missions [5], [1]. An alternative configuration swaps the nose mounted camera for a cesium vapor magnetometer. This variation is called the Georanger and is shown in Figure 2.2(a). This system is used for magnetic anomaly surveys [11], [88], [108]. The performance specifications of a typical ScanEagle are shown in Table 2.1.

Table 2.1: ScanEagle performance specifications.

Specification	**Value**
Max Takeoff Weight	41.9 lb /19 kg
Payload	15.4 lb / 7 kg
Endurance	15 hours
Service Ceiling	16400 ft / 5000 m
Max Level Speed	70 knots / 36 m/s
Cruise Speed	49 knots / 25 m/s
Wing Span	10.2 ft / 3.1 m
Fuselage Diameter	7.0 in / 0.2 m
Length	4.9 ft / 1.5 m

Currently, these systems have basic autonomous flight capabilities such as waypoint following, altitude and airspeed hold, and automated launch and retrieval. In some situations, the camera is able to lock onto pixels and track them autonomously.

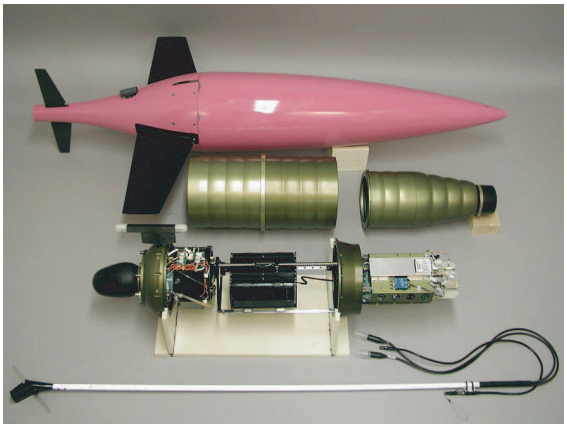
Agents in a heterogeneous team are not only restricted to aircraft; unmanned systems such as the SeaFox unmanned surface vehicle, the Seaglider unmanned underwater vehicle, and the Mule unmanned ground vehicle could potentially make up a heterogeneous team. These vehicles are shown in Figure 2.2.

More well known and larger scale autonomous systems includes the Predator which is manufactured by General Atomics Aeronautical Systems and the Northrop Grumman Global Hawk. These systems are also used extensively in ISR type missions and can be equipped with a variety of sensors such as infrared cameras and synthetic aperture radar.

(a) The Georanger autonomous air vehicle

(b) The Mule autonomous ground vehicle

(c) The SeaFox autonomous surface vehicle



(d) The Seaglider autonomous underwater vehicle

Figure 2.2: Possible agents of a heterogeneous team involved in searching mission.

Although the capabilities of these systems are formidable, the key distinction is that these are integrated systems rather than merely aircraft. They require a support infrastructure which contains ground stations, communication links, and human operators. In the case of the ScanEagle, the system requires at least two human operators for flight operations: one to manage flight paths and the avionics systems and another to operate the camera system. Personnel required for launch, retrieval, and ground tasks may further increase this number. A portion of the supporting infrastructure for the ScanEagle system is shown in Figure 2.3.

The Predator and Global Hawk each require a flight crew of three operators [6], [7]. For the Global Hawk, there is a pilot for the mission control element, a pilot for the launch and recovery element, and a sensor operator. Although these systems succeed in shielding the human operator from the danger present in these missions, the amount of human interaction

required for these autonomous systems exceeds that of their manned counterparts.

(a) ScanEagle pilot and sensor operator interface (b) Ground control station

Figure 2.3: Supporing infrastructure for ScanEagle and Georanger operations.

Cooperation between agents in a heterogeneous team is under development for systems such as the U.S. Military Cooperative Operations in Urban Terrain (COUNTER) program [53]. This system uses agents with varying sensors and capabilities to provide cooperative surveillance for military bases. Although this integrated system is in the later stages of development, it still relies on human operators to process the vast amounts of data gathered by the agent's sensors [37].

2.2 Autonomous Algorithms

The intelligence of autonomous systems stems from the various autonomous algorithms employed by the agents. Autonomous algorithms can be thought of as procedures or strategies which govern the behavior and mission related judgements made by the agent. A complex autonomous system is typically comprised of many autonomous algorithms which handle specific tasks required for a mission. The capabilities of autonomous algorithms run the spectrum from simple algorithms that stabilize a system about an operating point to algorithms that perform task allocation between agents in the team. Some major aspects of this work are in the field of autonomous searching, path planning, and target identification algorithms.

2.2.1 Search Algorithms

Searching for a target in a two dimensional domain is a common problem for modern autonomous systems. Determining regions of high target-location probability and coordinating nearby agents to converge on a particular spot while allowing other vehicles to continue searching is also difficult. Open problems such as these limit the number of agents that can participate in any given mission due to the high level of human interaction required to manage these systems. Many of these types of schemes are often referred to as "lawnmower" algorithms due to the fact that the search patterns they tend to generate resemble how one might choose to mow a rectangular lawn. For simple environmental geometries, heuristic strategies such as these work well and often can be trusted to search the entire map for the target. However if the environment is complex, such as containing areas which may be more or less desirable to search or require avoidance of areas posing threats to the search vehicles, the performance and guarantees of these heuristic patterns deteriorate and eventually disappear. The main research in this dissertation investigates a modular algorithm which can efficiently coordinate a team of possibly heterogeneous agents and ensure that the team performs an exhaustive search of the map.

This type of search mission has been studied before by several other groups. Classically, this has been addressed in exploration [82], [30] and search and rescue [61], [58], [51] scenarios. A survey of classical searching techniques by Benkoski [16] provides additional background and references.

One modern approach that has become popular is to consider possible target locations as a continuous or discrete probability density function. Groups such as Durrant-Whyte et al. [21], [20], [122] have studied the problem of searching for a target using a Bayesian probabilistic approach and have investigated some of the communication issues involved in such a search. Polycarpou et al. [89], [39], [52] have applied optimization techniques to generate search patterns over a finite amount of steps. Many of these methods are successful and effective for single target locations but have difficulty providing guarantees on target detection and map coverage. To address this, Erignac [36] has developed exhaustive searching strategies which also provide guarantees about map coverage with ideas based on

pheromone maps. Coverage of maps and domains have also been studied in the context of minimum service time to spontaneously occurring targets. Many of these techniques use Voronoi partitioning of the domain to maintain agent separation and coverage and have been studied by Du [33], Cortes [28], [68], and Frazzoli [42], [13].

Searching and map building has been studied extensively by the robotics community as well. Groups such as Fox et al. [41] have looked at generating searching algorithms with the ideas of exploration and map building in mind. Others such as Baillieul et al. [14] and Hoffman [46] have posed the searching mission in an information theoretic framework which provide useful metrics for search effectiveness. Previous work at the University of Washington by Rubio et al. [100] investigated searching algorithms in the context of adaptive algorithms. Previous work by the author's group [75], [70] explored the use of novel optimization techniques to address the search problem.

2.2.2 Path Planning Algorithms

The autonomous path planning problem is another subject that has been well studied in the past. Finding a feasible path for a vehicle subject to dynamic or kinematic constraints in a complex environment is often a difficult problem. Assuming that feasible paths exist, optimality of the path is a subjective matter and varies with environmental conditions and constraints. Efficient computation of optimal paths under such conditions remains an open problem.

The path planning problem is often addressed as a nonholonomic planning problem. Although this is accurate, Donald et al. [31] have shown that finding exact, time-optimal trajectories for a system with point-mass dynamics and bounded velocity and acceleration in an environment filled with polyhedral obstacles is NP-hard. In order to find solutions in a reasonable amount of time, several approximations to this problem have been made. LaValle and Kuffner [67] addressed the nonholonomic constraints and solved the kinodynamic planning problem using rapidly exploring random trees. Groups such as Capozzi et al. [25] have also looked at semi-randomized methods which provide quasi-optimal solutions to the path planning problem using evolutionary programming. Later, Pongpunwattana et al. [91]

incorporated these ideas into overall mission planning and task management schemes which address an agent's state and timing constraints. Other related work is found in [87]. Previous work by the author's group [75] investigated classical convex optimization techniques to generate simple paths from a starting point to a goal location for agents with constrained velocity limits. These algorithms are most similar to work done by Sun and Reif [110] where they introduce the concept of optimal path planning through a non-uniformly discretized two dimensional space (referred to as a weighted region) and efficiently compute a path which is an ϵ-accurate approximation of the true optimal path though this space.

In general, many groups have extensively studied the path planning problem and there is a significant amount of literature detailing the problem and proposed solutions. For additional material see [81], [9], [100], [92], and [66]. Difficulties with these strategies include: extensive computational power requirements (i.e. evolutionary algorithms [91], [97]), they are limited to generating simple paths (i.e. convex optimization techniques [75]), and many other open problems.

2.3 Target Identification

Another major type of autonomous algorithm deals with target identification and classification. A primary goal in a searching mission is typically to find a target in some region. However, there may be objects in the environment other than the desired target. If the agent encounters one of these objects, it needs to be able to discern between a false and true target. This problem can be addressed using machine learned classifiers.

The topic of machine learned classifiers is a large and well studied field. One of the most well known family of classification algorithms are the boosting algorithms generated by Schapire [106] and Viola et al. [117]. Groups such as Lester et al. [69] have applied this idea and studied the problem of fusing data from a wearable unit with multiple sensors and extracting features from which to train boosting algorithms for automatic classification of human activities. Raj [96] and Fox [40], [62] addressed a similar problem using standard particle filters, Rao-Blackwellized particle filters, and hidden Markov models (HMMs) to incorporate temporal smoothness into the task of classifying human activities from the same wearable multi-sensor unit. Particle filters are a popular method for estimating a system's

state based on non-linear motion and sensor models [112], [56] and have been successfully applied to applications such as robust control [18] and fault detection [119]. Modern tools such as the Weka tool set [121] have allowed for rapid training of classifiers using training and validation data. Previous work by the author's group [74] investigated the potential for using particle filters for target identification on a basic level.

2.4 Belief Maps

Traditionally, possible target locations in a two dimensional region have been represented by two dimensional probability density functions [21]. In this paper, an alternative method referred to as an occupancy based map is investigated. This method represents the belief of target locations by discretizing the search domain into a finite number of cells similar to work by Elfs [35], [34]. This map represents the state of the world in the sense that cells with high scores correspond to areas where the target is more likely to be located. This occupancy based map is shared and updated by all agents in the team. This method of representing the state of the environment is popular in computer science and artificial intelligence applications [101], [112].

Alternatives to the fully populated occupancy based map involve quad-tree representations such as those studied by Kang [56] and Amin [10].

In the research reported here, the occupancy based map provides the framework for formulating the search strategy.

2.5 Human Interface and Simulators

Another major contribution of this research is in the area of verification and validation of strategic autonomous algorithms [71]. Once the autonomous algorithms are developed, they need to be tested in both software simulation and hardware-in-the-loop simulation before deployment to flight test hardware. Many other research laboratories use ground based simulators to verify and validate control algorithms. The UAV Lab at Georgia Tech [54], [55] has developed significant software and hardware for testing algorithms and implementing flight tests [94]. Many of the interfaces and architecture presented in this dissertation are based on similar ideas. Previous work at the Autonomous Flight Systems Laboratory

at the University of Washington by Ponpunwattana et al. [93] regarding flight testing of autonomous algorithms has led to the development of the current ground based testing facility.

This simulator aims to replace the inner loop control algorithms with a human operator in order to expedite the process of verification and validation of the strategic level algorithms. This naturally introduces the aspect of human factors into the system. Human factors with respect to unmanned and remotely piloted vehicles have been studied by groups such as McCarley et al. [77], [78] and Tvaryanas et al. [113], [114]. A more in depth look at integrating human decision making capabilities into autonomous systems and strategies has been investigated by groups such as Savla et al. [104], [105], Cao et al. [24] and Morgansen et al. [118].

Chapter 3

TARGET IDENTIFICATION

A primary goal in a searching mission is typically to find a target in some region [75]. As an agent performs its search, it may encounter the target it is searching for or it may encounter an object which produces a sensor reading but is not the desired target (a false anomaly). The main challenge in this situation is to correctly identify or classify these encountered anomalies. Typically, the target identification process is handled by a human operator. It is the operator's responsibility to monitor sensor outputs and determine if readings correspond to the desired target, false anomalies, or extraneous noise. Many unmanned systems carry optical sensors such as cameras or infra-red imaging devices. Autonomously processing the output of these sensors and classifying readings requires significant computational resources and effort and in many cases, a human operator will outperform a computer generated algorithm in terms of classification accuracy and reliability. This can be mainly attributed to the fact that the output of the sensor is a complex multi-dimensional set of data with many features being explicit and more easily extracted and recognized by a trained human operator. As the number of agents involved in the search mission increases, so does the required number of human operators. A large team of expensive agents with sophisticated sensors may not be the most effective system.

A more scalable and implementable team would be comprised of several smaller agents with less complex sensors. These more primitive sensors may output a smaller dimension signal which a human operator may have difficulty interpreting. Automating the target identification process may have more promise in this context. In a noisy environment, it becomes difficult for a human operator to classify sensor readings and assign confidence in these readings because the useful features are hidden in the signal. In this situation, the useful features in the signal can be extracted and then used to train a machine learning algorithm to perform the classification which may have better accuracy and reliability than

the human operator.

This work contributes to the large area of research in autonomous target identification. The main research focus is to develop a system which processes the output from a low dimensional sensor and extracts relevant features from which to train a classifier. Some topics covered in this chapter include sensor models of environments and targets, machine learning algorithms for autonomous target identification, and investigation of the use of particle filters instead of the traditional HMMs to incorporate temporal smoothness and regularity into the target identification system.

The example used in this chapter involves an agent searching for a target based on its magnetic signature. The agent is equipped with a simple scalar magnetometer which can measure the magnetic field magnitude at its current location. The unique contribution of this research is a highly accurate target identification system which uses low dimensional sensor returns to classify anomalies. Section 3.1 covers the concepts of the general sensor used in this application as well as the model of the magnetic field and various factors which affect it. The models of the true target and false anomalies are detailed in Section 3.2. Section 3.3 describes the method used to generate data and training examples for this application. The various data features that are extracted from this simulation are described in Section 3.4. Section 3.5 explains how the autonomous target identification algorithm is trained and implemented. Finally, Section 3.6 presents results and remarks about the overall target identification system.

3.1 Sensor Capabilities and Models

This work focuses on identifying sensor readings made by an autonomous agent and classifying these readings as either the desired target or a false anomaly. The sensor of interest is a scalar magnetometer which measures the magnitude of the magnetic field at a given location. This section documents the sensor platform of the autonomous agent as well as the models of the sensor and environment.

3.1.1 The Georanger Autonomous Aeromagnetic Survey Vehicle

For aeromagnetic surveys, the agent (UAV) is essentially a mobile sensor. The vehicle autonomy serves the engineering user, who simply specifies an area of interest and after some processing, receives a corresponding set of data. In this idealized perspective, the data analyst is unconcerned with the method in which the data was obtained. To achieve such an objective, autonomy is required at several hierarchical levels. The current work focuses on target detection and classification tactics. The vehicle serving as the sensor platform in this work is the Fugro Georanger, provided by Boeing/Insitu, shown previously in Figure 2.2(a).

Each agent is equipped with a magnetometer to measure the total magnetic intensity at its current location. This data is relayed to a ground station. The crucial piece of information required by the ground station is a local magnetic map of the region where the search is taking place. This map of the total magnetic intensity (TMI) of the region may be acquired using analytical models such as the WMM-2000 or WGS-84 model [85]. However, since these models are coefficient-based analytical models, they do not capture temporal or small local variations in magnetic field strength. Therefore, a more accurate map is obtained by performing an actual survey over the area of interest to collect the necessary data. It is worthwhile to note that a large database of these surveys are readily available from either proprietary companies [88] or government resources such as the United States Geological Survey [4] so it may not be necessary to perform extra missions whose sole purpose is to obtain these maps.

3.1.2 Magnetometer Capabilities and the Magnetic Field

One of the most important pieces of equipment for this application is the sensor used to measure magnetic field strength. In this work, each agent is assumed to be equipped with a cesium-vapor scalar magnetometer and a fluxgate vectored magnetometer. These sensors measure the absolute field strength and the three component magnetic field vector, respectively. A typical fluxgate vectored magnetometer can measure fields in the range of 0 to 2,000,000nT with a resolution of better than 1nT. Furthermore, the measurement signal

is expected to have a noise profile of 0.1nT.

A reasonable model of the magnetic field measured by an agent is given by the linear superposition of various sources [32].

$$\overline{b} = \overline{b}_{tgt} + \overline{b}_{m} + \overline{b}_{agt} + \overline{b}_{var} \tag{3.1}$$

The magnetic field at the sensor, $\overline{b}$, is a linear combination of several different magnetic fields. The magnetic field predicted by an analytical model such as the International Geomagnetic Reference Field (IGRF) [3] or the 2000 World Magnetic Model (WMM-2000) [85] is given by $\overline{b}_m$; the magnetic field created by the target is given by $\overline{b}_{tgt}$; the magnetic field induced by the agent itself is given by $\overline{b}_{agt}$; and unaccountable magnetic field variations are given by $\overline{b}_{var}$. One of the main goals of this work is to be able to discern the magnetic signature of the target, $\overline{b}_{tgt}$, from the other sources. In order to do this, each term must be analyzed and compared to the capabilities of the sensors.

Mean Magnetic Intensity

The magnetic field at all points on the earth is modeled fairly accurately using analytical models such as the IGRF or WMM-2000. These models provide the three component estimated magnetic vector at any coordinate on the planet. The mean magnetic intensity of the magnetic field varies from roughly 30,000nT to 65,000nT. An example of the output of one of these models at the Boardman, OR closed flight test range (coordinates 47.780°N, 119.708°W) is shown below in Figure 3.1.

Self Induced Magnetic Field

The agent creates its own magnetic field which is measured by the sensor. The static effect of the agent can easily be calibrated and subtracted from the sensor readings. It is more difficult to characterize the dynamic effects of actions such as changing engine RPM or servo actuation on the magnetic field. These effects are shown below in Figure 3.2, courtesy of Boeing/Insitu. In this figure, the effects of different actions on the magnetic field are shown for different sensor locations. The red line denotes data collected when

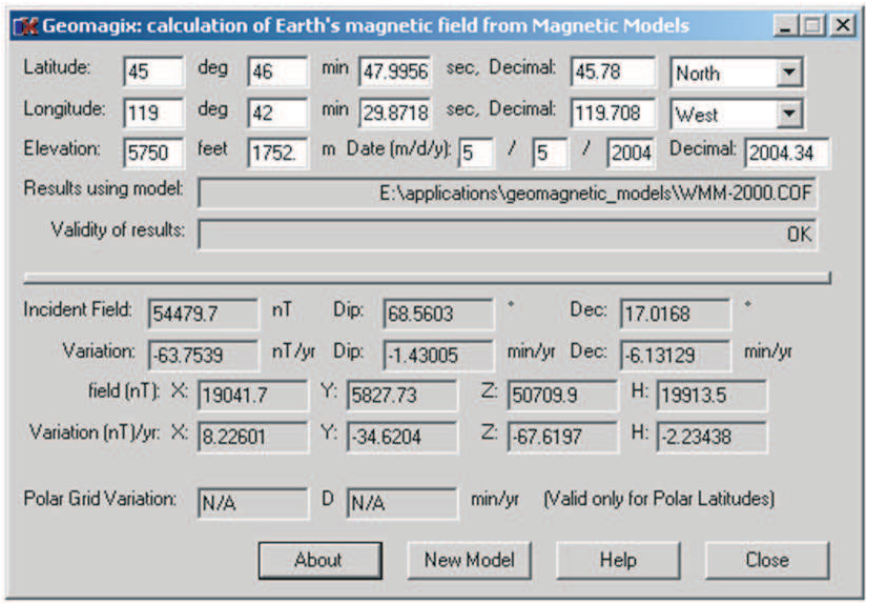


Figure 3.1: Predicted magnetic field at the Boardman, OR test range.

the magnetometer was closest to the agent's core, whereas the purple line corresponds to moving the magnetometer as far away as possible from the agent's core. Due to proprietary concerns, the scale of these readings is not shown; but it suffices to show that this self-induced magnetic field is present and may have an effect on the measured magnetic field. If an agent is optimized to make magnetic readings, these dynamic effects can be minimized.

Total Magnetic Intensity Maps

The magnetic field also has variations due to other unaccountable factors. These can include effects such as short period magnetic fluctuations (≈1-2nT), long term drift (≈60nT/year), and unaccounted magnetic signatures of structures or land formations. The effect of the short period magnetic fluctuations can be satisfactorily modeled as white noise with amplitude 1-2nT. The unaccounted magnetic signatures of structures and land formations are characterized by a total magnetic intensity (TMI) map. This map is a function that maps a $\overline{z} = (x, y)$ coordinate to an scalar representing the deviation from the predicted magnetic field strength ($T : \Re^2 \rightarrow \Re$). The graph of this function is essentially the resultant field

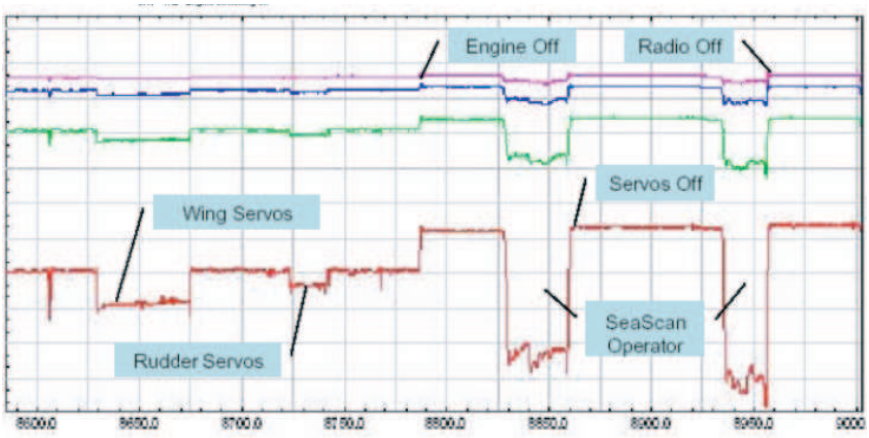


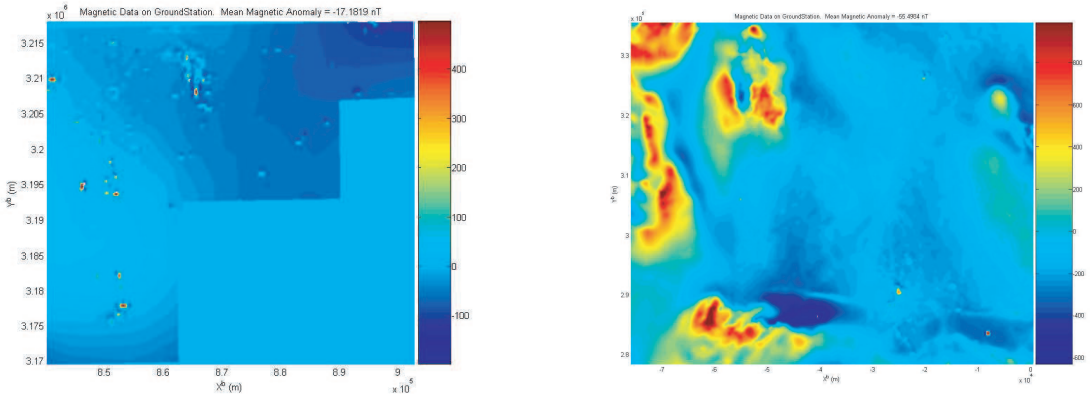
Figure 3.2: Self induced magnetic field effects. Courtesy of Boeing/Insitu.

after correcting the observed field for a regional gradient field (using IGRF or WMM-2000).

$$T(\overline{z}) = ||\overline{b}_{observed}(\overline{z}) - \overline{b}_m(\overline{z})|| \tag{3.2}$$

Two examples of actual TMI maps are shown below in Figure 3.3. Since these surveys are obtained using specialized vehicles, the self-induced magnetic field is negligible ($\overline{b}_{agt} \approx 0$) and no targets are assumed to be present during the survey ($\overline{b}_{tgt} = 0$) so the TMI map gives a measure of accuracy of the predictive models. The areas of light blue are where the magnetic anomaly is near zero. This implies that the measured magnetic field is very close to the field predicted by a regional model. However there are many regions where the measured field is significantly different than the predicted field. In Figure 3.3(b), the magnetic anomaly ranges from roughly -600nT to 900nT. Since the peak anomaly of the target is expected to be around 115nT, the predictive regional models cannot be relied upon and an actual survey of an area must be obtained in order to discern the $\overline{b}_{tgt}$ from $\overline{b}_{var}$.

Once again, this map is only a function of two variables (namely the (x, y) coordinates). This forces the agents to fly at the same altitude that the original survey vehicle flew at when the map was acquired. This map could be corrected for altitude variations or multiple maps at different altitudes could be obtained. This modification would make the map more complicated (a function of three or more variables) but would increase the functionality and



(a) 76.4km by 57km map near the Gulf of Mexico (b) 62.5km by 48.2km map over Puget Sound

Figure 3.3: Total magnetic intensity maps.

versatility of the system. Methods to do this are beyond the scope of this dissertation.

When an actual search is executed, differences between the ground station map of the magnetic field and the actual magnetic field will appear as magnetic anomalies. In this work, to minimize the number of false anomaly encounters and to increase the accuracy of the evaluation, actual magnetic survey data is used as a local TMI map. This data is provided by Fugro Airborne Surveys. The data was collected by a manned aircraft equipped with a magnetometer to measure the TMI. This information, coupled with a GPS position, provides the TMI in "line data" form. This data can then be interpolated into a 100x100 meter grid. TMI readings at locations other than survey points are linearly interpolated from this grid. A magnetic map of a region in the Gulf of Mexico and a simple grid search trajectory are shown below in Figure 3.7(a). Here, the data is acquired in an approximate 60x50 km grid. The regions of uniform color denote areas where survey data is not available, creating the staircase appearance. Assuming that there are only permanent fixtures in the region when the map is acquired, this map now makes up the reference set of data on the ground station.

3.2 Target Models

The last term in Eq. 3.1 is the magnetic field anomaly due to the target or false anomaly in the environment. The effect of the target on the magnetic field is difficult to obtain in practice. For proof of concept purposes, it is modeled as a continuous function of the position of the sensor over the target. This type of magnetic signature model is also applied to generate models of the false anomalies.

3.2.1 True Target Models

As stated in Section 3.1.2, accurate magnetic maps and target signatures are necessary for a successful target identification process. The target of interest in this application is a submerged submarine operating at a constant depth. Based on limited knowledge at the current time, the submarine signature is modeled as an oblong, two-dimensional Gaussian with peak of approximately 115nT. The function's level sets are roughly ellipsoids. The rate of decay from the peak can be tailored using the covariance matrix of the Gaussian. Undoubtedly, this profile differs from a true submarine signature. Furthermore, this profile is a function of depth and sensor altitude. An accurate magnetic signature of the target is required in order to predict the target signature in different conditions. Some modeling has been completed by engineers at Fugro Airborne Surveys. One of the models is shown in Figure 3.4.

If modeling the complex target signature is not accurate enough or impractical, another option is to obtain this data experimentally. This would involve first obtaining a TMI map of the test area without the target and then mapping the same area again with the target present in a known position and orientation. By Eq. 3.1, the difference of the two maps should be the magnetic signature of the target.

The magnetic field intensity created by the target is the most interesting and is also the signature that is the most difficult to model. Modeling the submarine as an ellipsoid 80m long, 10m in diameter, at depth of 25 meters, and the sensor at an altitude of 25m above the surface, the expected peak magnetic anomaly is approximately 115nT [45]. The profile falls off quickly and at 100m to either side of the submarine, the magnetic anomaly

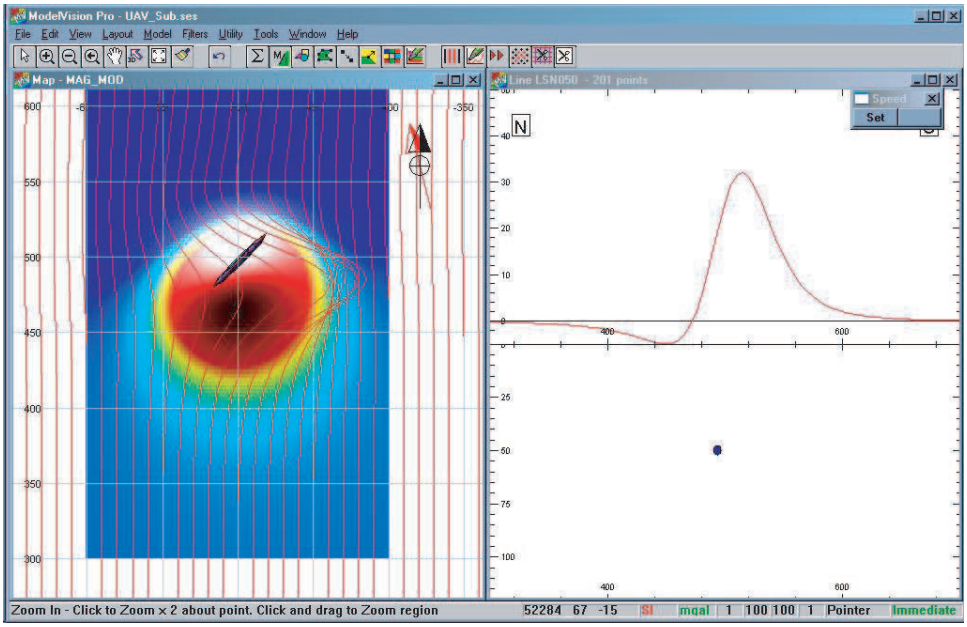
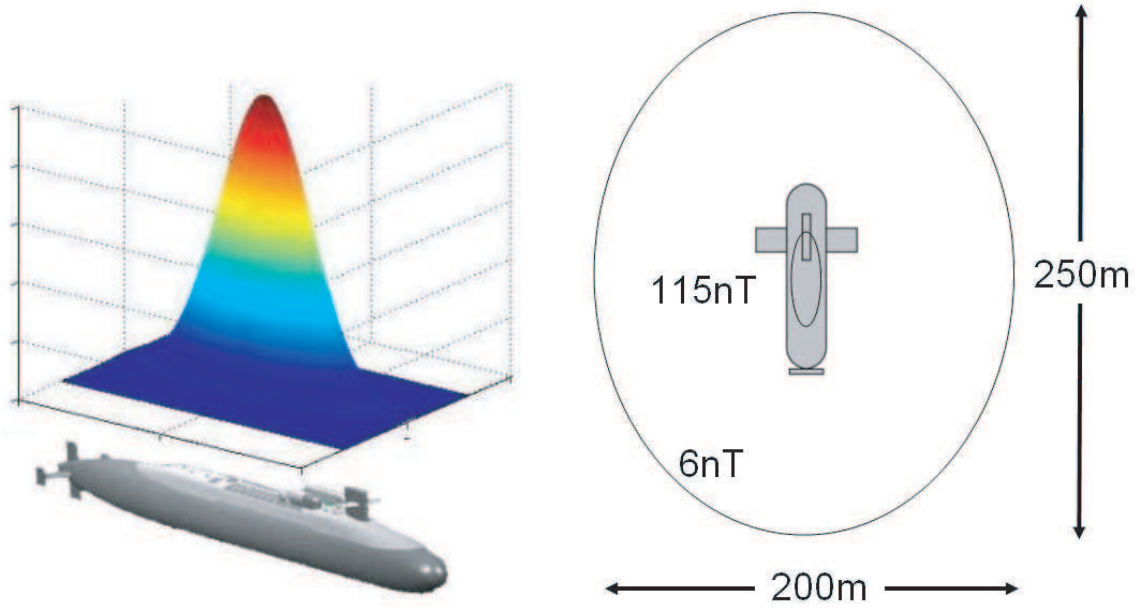


Figure 3.4: Submarine signature modeled with ModelVision Pro. Courtesy of Fugro Airborne Surveys.

is reduced to roughly 6nT. An example profile with the expected level sets is shown below in Figure 3.5.

Currently, the submarine signature, $h()$, is only a function of the (x, y) coordinate ($h : \Re^2 \rightarrow \Re$); a more sophisticated model would take depth and altitude into account. The function would then be a function of four parameters. This formulation is easily incorporated into the target identification algorithm described in this chapter. The magnetic signature of the target (an idealized submarine) is modeled as a simple two dimensional Gaussian distribution, shown in Figure 3.5(a). In reality, the magnetic signature of the target is a function of many variables, namely depth of target, sensor altitude, etc. For current purposes, the target is assumed to be stationary and at a fixed depth, thereby rendering the magnetic signature static. Assuming that the magnetic signature of the target simply adds to the total magnetic intensity of the local region in a linear fashion [32], anomalies can easily be identified by simply subtracting the magnetometer reading from the local reference map which is stored on the ground station.

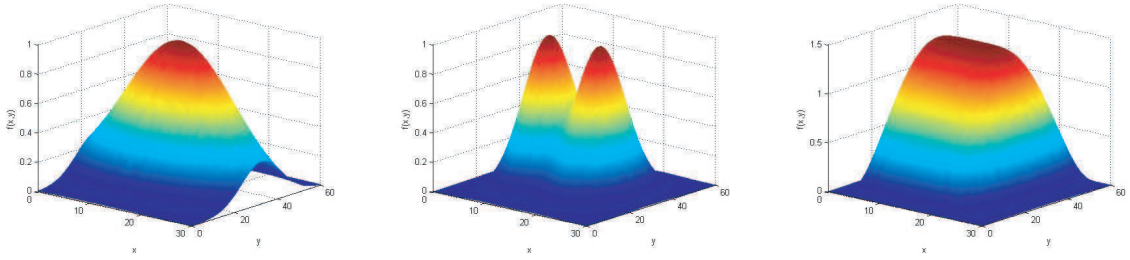


(a) Magnetic intensity of sub $h(x,y) = ||\bar{b}_{tgt}||$ (b) Level sets of magnetic profile

Figure 3.5: Magnetic signature of submarine.

3.2.2 False Target Models

The magnetic anomaly is either the true target with magnetic signature as described previously in Figure 3.5(a) or a false anomaly. To make the problem challenging, the false anomalies are made to appear similar to the true anomaly. Example of three false anomaly functions are shown below in Figure 3.6.



(a) $\tilde{h}_1(x,y)$ (b) $\tilde{h}_2(x,y)$ (c) $\tilde{h}_3(x,y)$

Figure 3.6: Possible false anomaly signatures.

3.3 Generating Data

With the sensor model and true and false target signatures in place, data can be generated by simulating encounters with these magnetic anomalies.

3.3.1 Agent Model

In practice, it is assumed that the agent's orientation in the earth frame is known via GPS. Because the anomaly's location and orientation is not known, the anomaly's frame of reference with respect to the earth frame is unknown. In practice, the agent will fly over a region and encounter an unexpected sensor reading (Figure 3.7(b)). It can be inferred from this reading that an anomaly was encountered, but it is unknown which part of the anomaly the agent flew over to generate this spurious measurement. One problem in this situation is how to estimate the state of the agent with respect to the target's frame of reference. In this context, the agent's state and control vectors are given by

$$\overline{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \\ \psi(t) \end{bmatrix} = \begin{bmatrix} x^{tgt}_{uav/tgt}(t) \\ y^{tgt}_{uav/tgt}(t) \\ \psi_{uav/tgt}(t) \end{bmatrix} \qquad \overline{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} V_{uav}(t) \\ \Delta\psi_{uav/tgt}(t) \end{bmatrix} \tag{3.3}$$

The agent is modeled as a simple planar kinematic vehicle with dynamics given by

$$\overline{x}(t+1) = u_1(t)\Delta T \begin{bmatrix} \cos(x_3(t) + u_2(t)) + x_1(t) \\ \sin(x_3(t) + u_2(t)) + x_2(t) \end{bmatrix} \tag{3.4}$$

A simulation environment was developed that allowed an agent to fly over a magnetic anomaly. A single trajectory consists of an agent flying a path over an anomaly. As the agent flies over an environment, its sensor makes a scalar reading ($z(k) \in \Re$) at each time step. Because only the differential measurement is desired, the magnetic contribution by the environment is neglected. The sensor is modeled as

$$z(t) = \begin{cases} h(x_1(t), x_2(t)) + g(N(0, \sigma^2_{sensor})) & \text{if true target encountered} \\ \tilde{h}_i(x_1(t), x_2(t)) + g(N(0, \sigma^2_{sensor})) & \text{if false anomaly encountered} \end{cases} \tag{3.5}$$

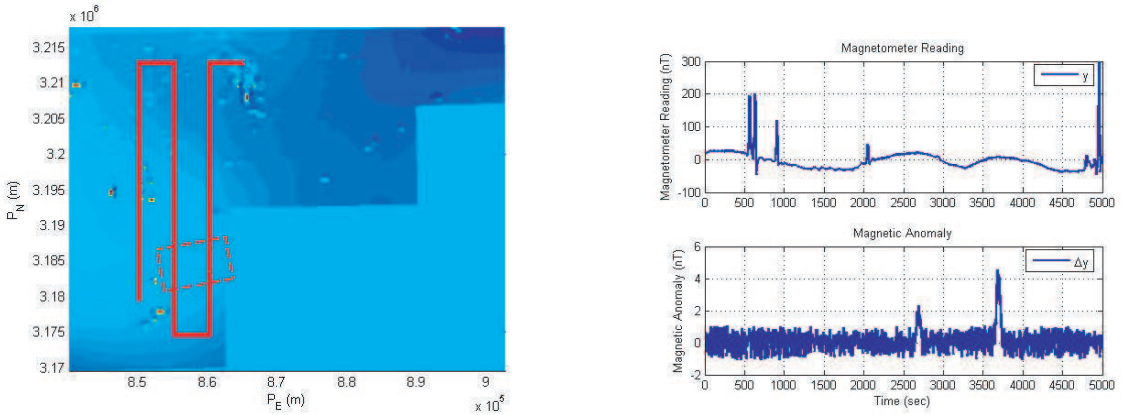
In Eq. 3.5, $g()$ is a sampling function that simply chooses a sample from a probability density function (in this case, a Gaussian distribution with zero mean and variance σ^2_{sensor}). The noise parameterized by σ_{sensor} is used to model the variation in the magnetic field and also to model sensor noise. The anomaly function $h()$ or $\tilde{h}_i()$ is either the true target or one of the false anomalies described previously in Section 3.2.1 and 3.2.2, respectively.

3.3.2 Simulated Trajectories

The framework we have described can be used to generate a simulation environment which recreates an agent flying through an environment and making measurements with a low dimensional sensor such as a scalar magnetometer. Large differential measurements imply the presence of a new magnetic anomaly and possible target. If the agent does not fly over any targets, the magnetic anomaly should be near zero. Small non-zero anomaly encounters can be attributed to temporal variations in magnetic field and sensor noise. A simple grid search pattern is shown in Figure 3.7(a).

In Figure 3.7(a), the location of the target is shown as a dashed red box and the trajectory of the agent is shown in the solid red line (starting in the lower left corner). The associated total magnetic intensity trace and differential measurement trace is shown in Figure 3.7(b). The total magnetic intensity reading as the agent flies over this trajectory is shown in the upper trace and the differential measurement is shown in the lower trace. As the agent flies this search trajectory, the sensor measurement is constantly compared to the reference data set to generate a differential measurement. As can be seen in Figure 3.7(b), given the differential magnetometer reading, the anomaly encounter can be easily detected (two spikes at approximately 2700 and 3700 seconds) even though the actual range of absolute measurements may be large.

Magnetic anomalies can be caused by many factors such as temporal variations in the



(a) Agent trajectory over TMI map.

(b) Associated magnetic traces.

Figure 3.7: The total magnetic intensity map and trajectory over area with corresponding magnetometer readings.

magnetic field or false anomaly encounters (i.e. boats or vessels). Once a magnetic anomaly is encountered, it must be identified and classified. On a simplistic level, the overall goal is to either classify the anomaly as the desired target or a false reading. Obviously, it would be simple to identify the anomaly if the entire magnetic signature of the anomaly is obtained (the UAV flies over the entire boxed region in Figure 3.7(a)). However, this requires many passes over a potential target, and significant time to take the necessary measurements. If the anomaly is moving or evading, this approach may not be feasible. To further exacerbate the problem, if the agent is equipped with a simplistic sensor (like a simple scalar magnetometer as in this case), the data obtained from a single pass may be limited. The question now becomes, given only a single pass over the anomaly, is it possible to correctly identify or provide a probability that this anomaly is indeed the target being sought after using this limited sensor data? This fits the mold of a binary classification problem.

The machine learning approach to solving the the binary classification problem requires generating a series of training data which is then used to train an adaptive learning algorithm. A Monte Carlo simulation is run with random, linear trajectories over the target. This generates many examples of trajectories and measurements of the agent flying over

both the true target and the false anomalies. Of these runs, only ones that meet a certain requirement are selected to be used as training examples.

The agent does not need to be simulated over the entire trajectory as shown in Figure 3.7(a) because for the majority of the time, the agent is not encountering an anomaly. The anomaly is only encountered when the agent flies over the dashed red box. Therefore, the Monte Carlo simulation only generates encounters of the agent with either a true target or false anomaly (essentially just flying the agent over the dashed red box). Anomaly encounters are generated by placing M points uniformly around the boundary of the anomaly. At each of these points, an angle is chosen from a uniform distribution between the ranges which correspond to an inward pointing angle. This angle is then used to generate a straight trajectory for the agent over the anomaly. This procedure is shown in Figure 3.8.

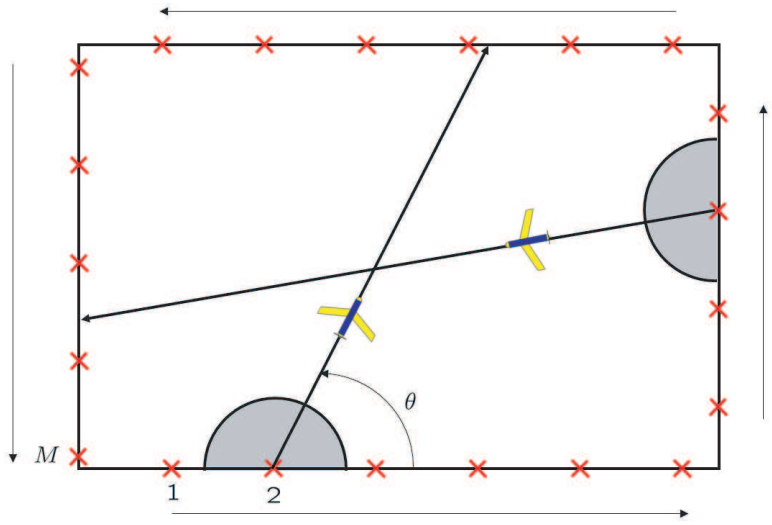


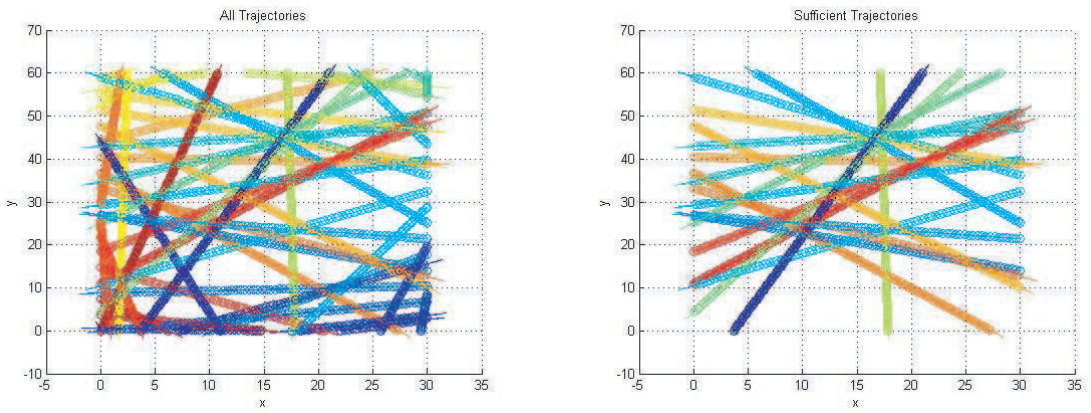
Figure 3.8: Generating approximately uniformly distributed anomaly encounters.

3.3.3 Selecting Sufficient Runs

The rate of convergence of most learning algorithms is dependent on the quality of the training data. It is desirable to train a classifier using examples which are distinguishable and are good representations of actual trajectories and to eliminate ambiguous examples

that may confuse the classifier. In this application, a run is deemed sufficient if it satisfies several criteria. It is unlikely that a measurement trace over a false anomaly will be discernable from a trace over the true anomaly if there is only a small number of measurements made. Therefore, a run is deemed sufficient only if the agent makes a minimum number of measurements as it flies over an anomaly. Similarly, a run is deemed sufficient only if the magnitude of the differential sensor measurement exceeds a certain threshold. This eliminates runs where the agent flies over the edge of the anomaly parallel with the axis of the anomaly (where the anomaly magnetic signature is nearly zero).

The runs that are used as training examples must meet both requirements. This process helps ensure that the training examples fed to the learning algorithm are distinctive enough to train a decent classifier. Examples of the total output of the Monte Carlo simulation and the selection of sufficient runs is shown in Figure 3.9.



(a) All trajectories (b) Sufficient trajectories used as training examples

Figure 3.9: Removing insufficient trajectories from Monte Carlo simulation.

Despite being selective about the runs used as training data, at this point, the raw sensor measurements from these trajectories do not provide enough distinction to differentiate the class of anomaly. An example of the raw sensor measurements from both a true and false anomaly encounter are shown in Figure 3.10. In this example it can be seen why it is difficult to discern the class of the anomaly based solely on these raw sensor readings because the traces are so similar. It is also difficult to determine the relevant features to look for in

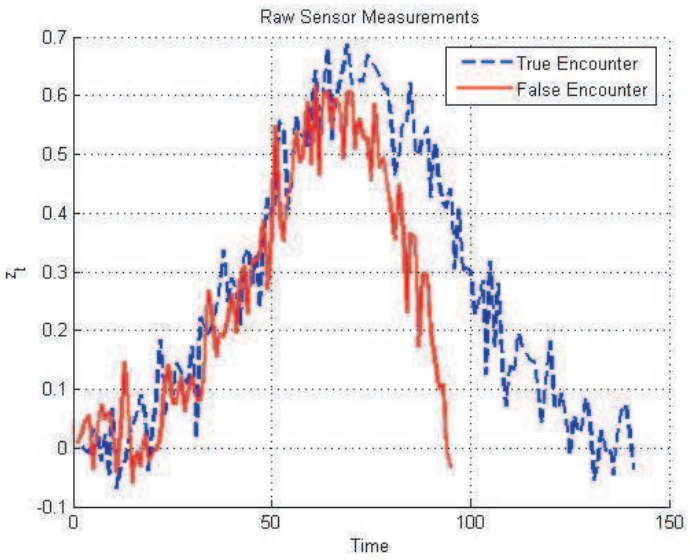


Figure 3.10: Raw sensor measurements from both a true and false anomaly encounter.

order to classify the anomaly. The task of extracting the relevant features from the sensor measurements is addressed in the next section.

3.4 Feature Extraction

This section discusses processing the raw sensor measurements in order to generate a feature vector which can be used to train a classification algorithm.

As evidenced by Figure 3.10, the raw sensor measurements offer limited amounts of useful data when it comes to classifying the anomaly. One option is to simply use the sensor measurement at each time step as an individual training example. Theoretically, it should be possible to create a classifier that is more successful than random guessing using this simple feature. For example, if the maximum sensor reading exceeds the maximum of the target signature, one can be fairly certain that the anomaly encountered is a false anomaly. However, this allows for multiple classifications within the same trace. Although this method provides an abundance of training data, it is virtually impossible to generate a reliable classifier using this simple a feature set.

Improvements to the accuracy can be made by using the assumption that during an

anomaly encounter, the agent only encounters the true target or a false anomaly (it cannot encounter both together). This means that an entire trace should be classified as one or the other. Instead of using the sensor value at every time step, one can extract features like the maximum/minimum sensor values over the entire trace. This yields a single training example for each trajectory and helps incorporate some temporal smoothness to the classification as this yields only a single classification per anomaly encounter.

Basic features such as maximum/minimum sensor readings and averages can be extracted fairly easily. These basic features help discern the classes somewhat. However, more ambiguous cases require additional features.

3.4.1 Particle Filter Features

The previously mentioned, simple features based on the raw sensor measurements still prove insufficient to train an accurate classifier. Further information can be gleaned by considering both the sensor model (Eq.3.5) and the dynamic motion model (Eq.3.8) of the agent. Since the data traces are generated using these models, a method that considers these temporal aspects has a much greater potential to yield useful features.

Typically, temporal smoothness is incorporated using Hidden Markov Models or Dynamic Bayesian Networks. These are useful to reduce classification noise in continuous traces. Some applications are classifying human activities [69], [96] where the classification can switch between different activities in a reasonable manner (for example, humans might transition from walking to sitting but typically do not rapidly transition between sitting and riding a bike). In effect, this makes use of the agent's motion model to predict the state and thereby augment the classification. Although this method can be used in this application to smooth the data, it does not explicitly use information regarding how the sensor measurements are gathered (the sensor model in Eq. 3.5). In this application, the temporal aspects of both the motion and sensor model are addressed using particle filters [62].

A particle filter is a recursive, non-parametric Bayes filter technique that estimates the states of a system using a finite number of state hypotheses [112]. In this situation, the state

vector being estimated is the position of the agent with respect to the target, expressed in the target's frame of reference and the relative heading of the agent with respect to the target.

$$\overline{x}_t^{[m]} = \begin{bmatrix} x_{uav/tgt}^{tgt} \\ y_{uav/tgt}^{tgt} \\ \psi_{uav/tgt} \end{bmatrix} \tag{3.6}$$

Each individual state hypothesis, $\overline{x}_t^{[m]}$, is referred to as a particle, and together they make up the particle filter set, χ_t.

$$\chi_t = \bigcup_M \overline{x}_t^{[m]} = \left\{ \overline{x}_t^{[1]}, \overline{x}_t^{[2]}, \ldots, \overline{x}_t^{[M]} \right\} \tag{3.7}$$

Historically, particle filters have been employed in this manner to perform tasks such as localization [111] and state estimation [38]. In this situation, the goal of the filter is to estimate the state of the agent (position and orientation with respect to the target, expressed in the target's frame of reference). The particle filter performs this estimate using two main steps, a prediction and correction step.

Prediction

In the prediction step, each particle is propagated forward in time using a motion model of the individual agent.

$$\overline{x}_t^{[m]} = g\left(p\left(\overline{x}_t^{[m]} | \overline{u}_t, \overline{x}_{t-1}^{[m]} \right) \right) \tag{3.8}$$

Each new particle is created from the old particle and the current control (applied to transition particle at time $t-1$ to time t). The term $p\left(\overline{x}_t^{[m]} | \overline{u}_t, \overline{x}_{t-1}^{[m]} \right)$ is a multi-dimensional probability density function of the new state given the old state and current control. Notice that in this formulation, the state transition is not a deterministic process. This stochastic aspect actually has important implications regarding the robustness of the particle filter [112].

Although $p\left(\overline{x}_t^{[m]}|\overline{u}_t, \overline{x}_{t-1}^{[m]}\right)$ may be difficult to compute analytically, Eq. 3.8 is implemented in simulation by simply adding noise to the control and then propagating the state forward using a deterministic motion model in Eq. 3.8. In simulation, the noise added to each element of the control vector is obtained by sampling from a normal, Gaussian distribution with a variable standard deviation, σ. The standard deviation is a function of the actual control applied to the agent, $\overline{u}_t$. In effect, as $||\overline{u}_t||$ increases, so does σ. Physically, this translates into a model whose state transition becomes more uncertain as the agent moves faster or executes larger heading changes.

In addition to the control input at each time step, the actual sensor measurement observed by the agent, z_t, is made available to the particle filter. Each particle is then assigned a weight, $w_t^{[m]}$, based on how likely it is to make the same sensor measurement at its current state $\left(w_t^{[m]} \propto p\left(z_t|\overline{x}_t^{[m]}\right)\right)$. In effect, a higher weight should be assigned to particles whose states are close to the actual state, $\overline{x}_t$. Notice that this does not require a sampling function like Eq. 3.8 because z_t and $\overline{x}_t^{[m]}$ are known at this point. This is another description of the sensor model of the agent. It allows for the fact that even though a particle's state may be vastly different than the true state of the agent, if the sensor is poor or unreliable, it has the possibility of still making the same sensor reading as the agent.

The sensor model used in simulation calculates the weights by creating an error between the particle sensor measurement and the true sensor measurement and then using this as the argument of a Gaussian distribution.

$$w_t^{[m]} = (2\pi\sigma_{sensor}^2)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\frac{(z_t - z_t^{[m]})^2}{\sigma_{sensor}^2}\right) \tag{3.9}$$

In Eq. 3.9, $z_t^{[m]}$ is the predicted sensor measurement made by particle m computed using the same sensor model in Eq. 3.5. As stated previously, σ_{sensor} is a measure of the sensor's accuracy. A larger σ_{sensor} implies an unreliable sensor; therefore, particles that do not make the same measurement as the true agent still receive high weights. Note that the weight is not a probability, but this system still achieves the goal of assigning high weights to particles that are more likely to have states that are similar to the true agent state. Similar to the particle set, χ_t, the weight set at a given time step t is given by

$$W_t = \bigcup_M w_t^{[m]} = \left\{ w_t^{[1]}, w_t^{[2]}, \ldots, w_t^{[M]} \right\} \tag{3.10}$$

Correction

Now that each particle has been propagated forward and assigned a weight, it becomes necessary to correct the particle filter set so that it comes closer to representing the actual state of the agent. This process is known as resampling.

As stated before, the particle filter's estimate of the state is made up of all the particles. Currently, the particle filter set contains particles that have both high and low weights. As more and more sensor measurements are acquired, it is desired that high scoring particles are replicated and kept in the next generation population whereas low scoring particles are discarded. The important feature in this evolutionary process is that the particles are resampled with replacement so that the total number of particles remain constant at each cycle. Any type of evolutionary scheme, such as survival of the fittest, can be used to evolve the current population to the next.

In simulation, a roulette wheel method is used. This is a popular method in many computer science applications where M bins are created (one for each particle). The size of each bin is directly proportional to the weight of the associated particle. The bins are placed next to each other and a random number is then generated. The bin in which the random number falls then has its associated particle included in the next population. This process is repeated M times and is synonymous to spinning a roulette wheel M times where the number and size of the slots on the wheel are directly proportional to M and the weights, respectively.

Using the roulette wheel method yields resampling proportional to the weights. This allows for a particle to be copied multiple times in the next generation. This also generates a small probability that particles with low weights have the possibility to survive to the next generation as well.

One important feature of the particle filter is the ability to use different motion and sensor models. This allows for a team of agents to be comprised of different types of

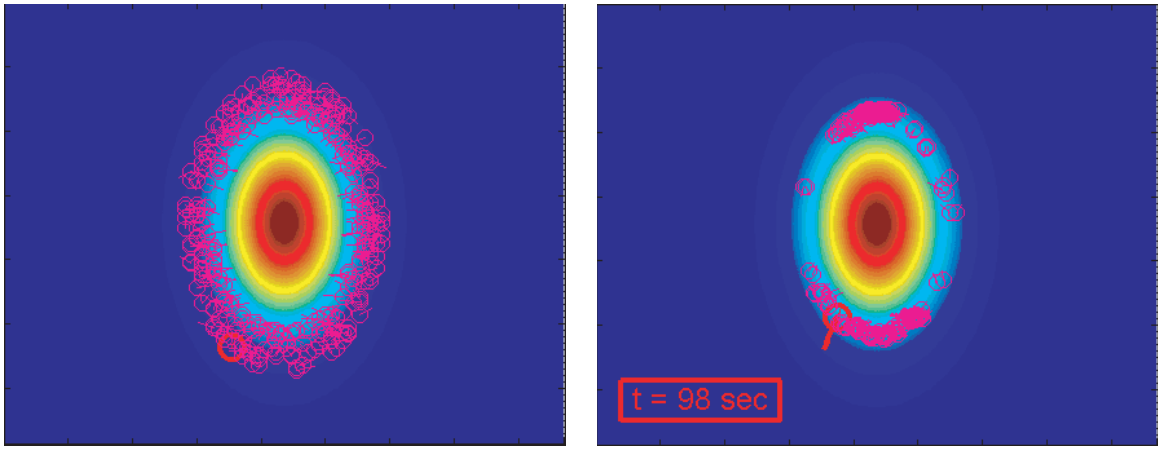
vehicles and sensors. This simply requires modifying the motion and sensor models of each particle filter for each member of the heterogeneous team.

Execution

When an agent encounters an anomaly whose magnitude exceeds the noise threshold (approximately 1nT in this case), the particle filter is started in an attempt to estimate the state of the agent with respect to the target. The particle filter's progression as the agent flies diagonally over the target is displayed over a top down view of the target signature (Figure 3.5) and is shown below in Figure 3.11.

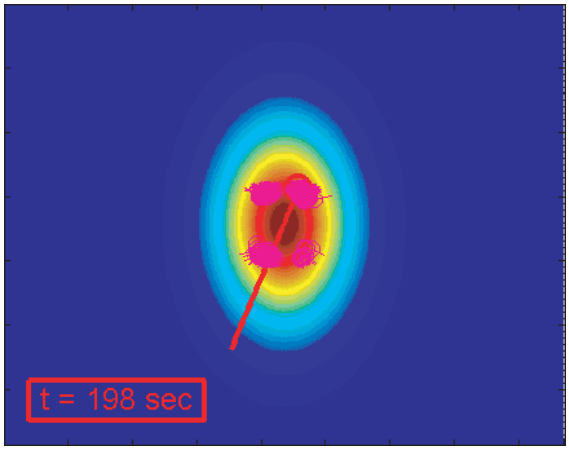
In this sequence, the large red circle represents the actual location of the agent and the solid red line represents the agent's trajectory over the target. The smaller magenta dots represent the particle filter's many different hypotheses of the possible state of the agent (North position, East position, and heading). The actual agent crosses over the target starting in the lower left corner and flies to the upper right corner. Also note that the initial distribution of particles is not simply random over the domain. Since the algorithm is recursive, the number of iterations before convergence is based on its initial condition. Incorporating a priori knowledge that the particle filter is started when the anomaly magnitude exceeds 1nT suggests that initially, the particles should be clustered along the level curves where the target signature is 1nT with inward pointing headings.

As the agent obtains more and more sensor measurements (at a simulated rate of 1Hz), the particle filter is able to eliminate particles that are inconsistent with the current measurement and resample these particles to regions that have a higher probability of producing the actual sensor reading, z_t. This is why as time progresses, the particles become concentrated around the actual UAV location. Near the end of the simulation, there are four distinct groups of particles. This is due to the symmetry of the underlying target signature. Each of these four groups of particles are equally likely because each group would produce the correct actual sensor readings. In effect, $z_t^{[m]} \approx z_t \, \forall m$. Due to this symmetry, the particle filter is not able to uniquely identify the position of the agent with respect to the target. This would require multiple passes over the target and more sensor measurements.

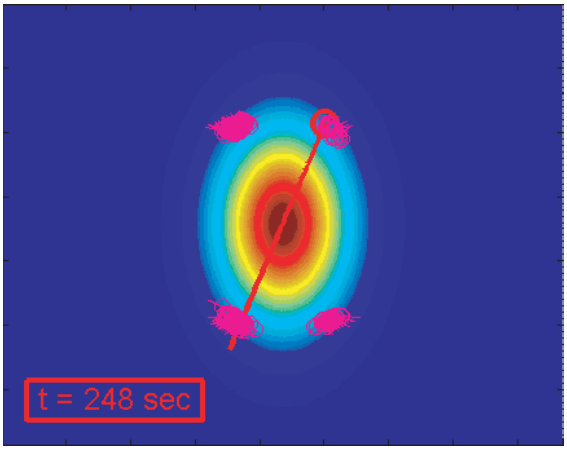


(a) Initial particle distribution, χ_0

(b) χ_{98}



(c) χ_{198}



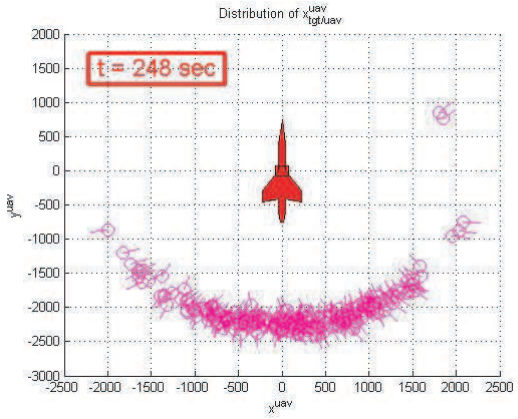
(d) χ_{248}

Figure 3.11: Particle filter progression during a target encounter. The solid line indicates actual aircraft position relative to target signature, while the particles concentrate about possible positions.

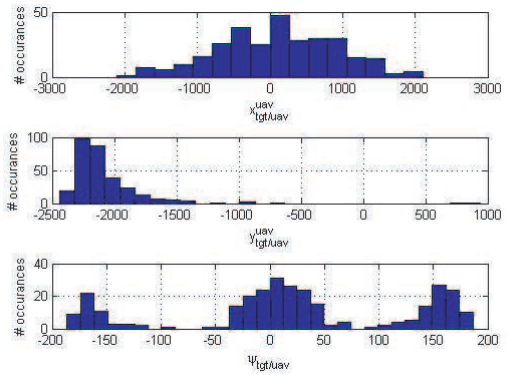
Although the goal of the particle filter is to estimate the position of the agent with respect to the target in the target frame of reference, in the larger picture, the location of the target with respect to the agent in the agent frame of reference is more useful because it then becomes simple to locate the target in the earth frame of reference (agent's position and orientation in the earth frame of reference is known via GPS). Each particle can be transformed using Eq. 3.11.

$$\begin{bmatrix} x^{uav}_{tgt/uav} \\ y^{uav}_{tgt/uav} \\ \psi_{tgt/uav} \end{bmatrix} = \begin{bmatrix} -\cos(\psi_{uav/tgt}) & \sin(\psi_{uav/tgt}) & 0 \\ -\sin(\psi_{uav/tgt}) & -\cos(\psi_{uav/tgt}) & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x^{tgt}_{uav/tgt} \\ y^{tgt}_{uav/tgt} \\ \psi_{uav/tgt} \end{bmatrix} \tag{3.11}$$

When each particle is transformed in this fashion, the distribution of the target location with respect to the agent in the agent's frame of reference becomes as shown in Figure 3.12.



(a) Distribution of transformed particles



(b) Histogram distribution of $x^{uav}_{tgt/uav}$, $y^{uav}_{tgt/uav}$, and $\psi_{tgt/uav}$

Figure 3.12: Transformed particles now representing position and orientation of target with respect to the agent in the agent's frame of reference.

In the first two plots in Figure 3.12(b), it appears that the particle filter now has a somewhat unique estimate of the location of the target relative to the agent as shown by an approximate unimodal distribution in $x^{uav}_{tgt/uav}$ and $y^{uav}_{tgt/uav}$ centered at approximately 0 and -2250, respectively. However, notice that the distribution of $\psi_{tgt/uav}$ is obviously a multimodal distribution. This distribution is actually the sum of four peaks which should ideally be centered at ± 11.3 degrees and ± 168.7 degrees. Since the number of particles was not large enough and since the motion and sensor models of the particle filter were not highly accurate, the two peaks centered at ± 11.3 degrees appear as a single peak at 0 degrees.

This multimodal distribution in $\psi_{tgt/uav}$ reflects the four distinct state hypotheses shown previously in Figure 3.11(d). However, if the orientation of the target is not desired, then by transforming the particles, it is possible to obtain a unique estimate of simply $x^{uav}_{tgt/uav}$ and $y^{uav}_{tgt/uav}$. Note that this is only the case when the agent happens to fly directly over the target as shown in this example. In a more general case where the agent passes over the target off-centered, then even with the transformation of the particles, the location of the target cannot be determined uniquely (but the number of possible locations may be reduced).

3.4.2 Generating a Feature Vector

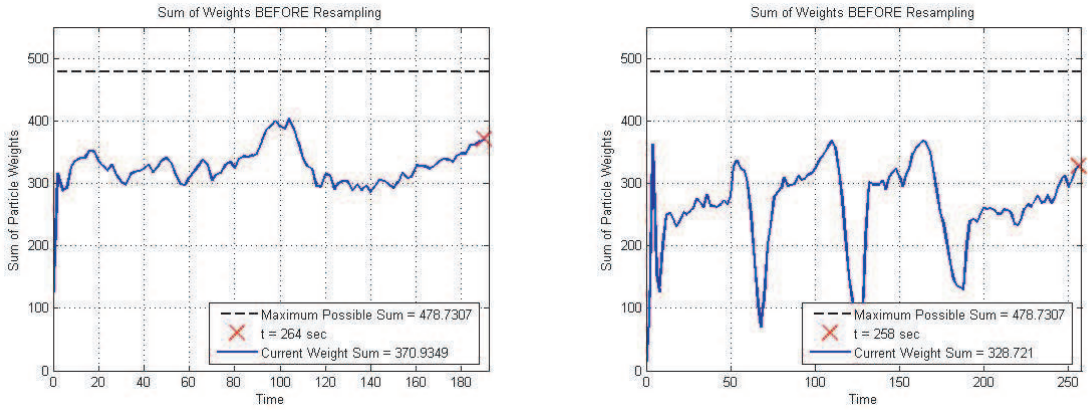
In the majority of this section, we discussed the state estimation problem using the particle filter method. Deviating from the standard usage of the filter, a closer look at the weights, $w_t^{[m]}$, is warranted in the context of target identification.

A scalar quantity which collectively measures the overall accuracy of the particle filter can be obtained by simply summing all the weights. If most of the particles are in locations that are similar to the true state, then the sum of the overall weights should be large.

$$C_t = \sum_{m=1}^{M} w_t^{[m]} \tag{3.12}$$

This trace of a C_t vs. t might be considered a side-effect of estimating $\overline{x}_t$, but as will be shown later, this is the primary piece of information that will be used to address the target identification problem.

The particle filter will attempt to estimate the agent's state regardless of whether the anomaly encountered is the actual target or a false anomaly. A method to identify the target is now required. The sum of all the particle weights, C_t, provides a quantitative measure of how confident the particle filter is that the anomaly encountered is the actual target. If all or most of the particles are resampled to areas that are near the actual state of the agent, then most of the weights will be fairly high. The sum of the particle weights for an encounter with the actual target and an encounter with a false anomaly is shown below in Figure 3.13.



(a) True target encounter

(b) False anomaly encounter

Figure 3.13: Sum of all particle weights during a true target encounter and a false anomaly encounter.

In Figure 3.13, the difference between a true target encounter and a false anomaly encounter is fairly clear. In the situation where the agent encounters the true target, the confidence measure increases initially as the particles are quickly resampled to locations that are consistent with the actual sensor measurements and then stays fairly constant. However, in the case where the agent encounters a false anomaly, the particle filter regularly "loses confidence" as inconsistent sensor measurements are obtained. This is characterized by the sharp drops in the sum of the particle weights.

The variance of the particle weights also is a good indicator of filter confidence. If the variance is high, this implies that the weight of the particles is spread out. Since a high weight indicates a good state estimate, if the filter is not very confident in its state estimate, the variance is high. The variance at each time step of a single trajectory is recorded to be used in generating features later.

$$V_t = \mathrm{Var}(W_t) \tag{3.13}$$

Using the raw sensor readings and the particle filter outputs, an eleven dimensional feature vector is generated.

$$f = \begin{pmatrix} T \\ M(2\pi\sigma^2)^{-1/2} \\ \max_{t=1,\ldots,T}(z_t) \\ \frac{1}{T}\sum_{t=1}^{T} z_t \\ \frac{1}{T}\sum_{t=1}^{T} V_t \\ \frac{1}{T}\sum_{t=1}^{T} C_t \\ \frac{1}{T}\sum_{t=1}^{T} \frac{C_t}{M(2\pi\sigma^2)^{-1/2}} \\ \max_{t=1,\ldots,T}(V_t) - \min_{t=1,\ldots,T}(V_t) \\ \max_{t=1,\ldots,T}(C_t) - \min_{t=1,\ldots,T}(C_t) \\ \max_{t=1,\ldots,T-1} |V_{t+1} - V_t| \\ \max_{t=1,\ldots,T-1} |C_{t+1} - C_t| \end{pmatrix} \tag{3.14}$$

The first two features are constants that are functions of the particle filter execution. The first feature is simply the number of measurements made by the agent. The second feature is the maximum sum of the weights possible at any point during the particle filter execution. The third feature is simply the maximum sensor reading encountered and is included for reasons discussed previously. The next three features are the average sensor reading, variance of weights, and sum of weights, respectively. The seventh feature is the average weight proportion which is somewhat redundant but helpful when visualizing data. The next two are the changes in variance of the weights and sum of the weights over the entire particle filter execution. Finally, the last two features are the maximum change in variance of the weight and sum of the weights in a single step, respectively.

A single feature vector is generated for each measurement trace and represents a single example for training a classifier. Using these features with various learning algorithms are discussed in the next section.

3.5 Autonomous Target Identification

As mentioned previously in Section 2.3, the field of autonomous target identification is a well studied discipline. A traditional classification scheme [121] is shown in Figure 3.14.

In this situation, the agent and sensor interacts with the environment. Feature are

Figure 3.14: Flow diagram for traditional classification system.

then extracted from the sensor measurements and passed to a trained classifier which then decides on the class based on these features. Typically, this signal is noisy since it has not taken into account how the measurements were generated. System consistency is added using some type of Bayes filter or HMM to smooth the signal and output the final anomaly class. Notice that in this system, information about the system models is incorporated once the classifier has decided on a class. An alternative to this strategy involves integrating the system consistency via a Bayes filter at the feature level and is discussed in the next section.

3.5.1 Training Classification Algorithms

The flow chart for the proposed classification system is shown in Figure 3.15. A simulated trajectory over an anomaly is generated using the specified sensor and motion model. If this trajectory is deemed sufficient, the particle filter is run using the controls and measurements. Features are then extracted from these controls and measurements and the outputs from the particle filter algorithm. This system offers benefits over the system shown in Figure 3.14 because the system models are able to influence the classifier instead of being applied to the output of the classifier since motion and sensor models are incorporated at the feature level.

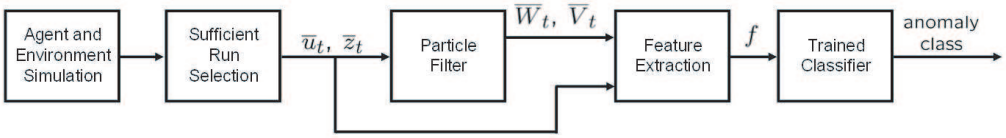


Figure 3.15: Flow diagram for proposed classification system.

To generate training examples, 1700 random trajectories were flown over both the true

target and various false anomalies (roughly equal proportions). Of these 1700 trajectories, 836 were deemed sufficient for feature extraction. Once the features are extracted from these sufficient runs, a set of 836 training/testing examples are generated. This data set is evaluated with several different learning schemes as described in the next section.

3.6 Simulated Results

The main focus of this chapter is generating an autonomous target identification algorithm. The actual classification algorithm is highly dependent on the features used to train it. Once discernable features are extracted (Eq. 3.14) they can be used to train any number of classifiers. To evaluate the effectiveness of these features, several different classifiers are trained using the Weka Toolkit [121].

3.6.1 Results

Each classifier was trained and tested using a stratified 10-fold cross-validation scheme. The results for four learning methods [121] are displayed in Table 3.1.

Table 3.1: Classification results for various learned algorithms.

	C4.5 Revision 8 Classified As		Decision Stump Classified As		Alternating Decision Tree Classified As		Multi-Layer Perceptron Classified As	
	True	False	True	False	True	False	True	False
Actual True	232	40	0	272	197	75	247	25
Actual False	70	494	0	564	57	507	59	505
Accuracy	86.84%		67.46%		84.21%		89.95%	

As can be seen from Table 3.1, most of the learning algorithms perform very well with an average accuracy of 87% (excluding the overly simplistic decision stump method). These results are encouraging considering that there are only 11 features in each example. Often times, hundreds of features are extracted for each example in order to obtain these accuracies. Of course, of these large feature sets, only several features are useful for the classification. This leads to the conclusion that the 11 features described in Eq. 3.14 are

discernable and therefore valuable in this application. Extracting more features might serve to increase accuracy but most likely not by a large margin.

The decision tree generated by the C4.5 Revision 8 algorithm [95] is shown in Figure 3.16. Although there is not much to be learned by looking at the individual nodes and leaves of the tree, the structure illustrates why this method of extracting a low number of features is desirable. The computational cost of executing the particle filter (and extracting the feature vector) is relatively high. However, once the features are extracted, the actual classification is fairly simple. The resulting decision tree is small enough to illustrate. Therefore, this tree can be implemented easily for a real-time application. Once the features are extracted from the data, the actual classification is not computationally intensive (in this case requiring at most eight if-statements).

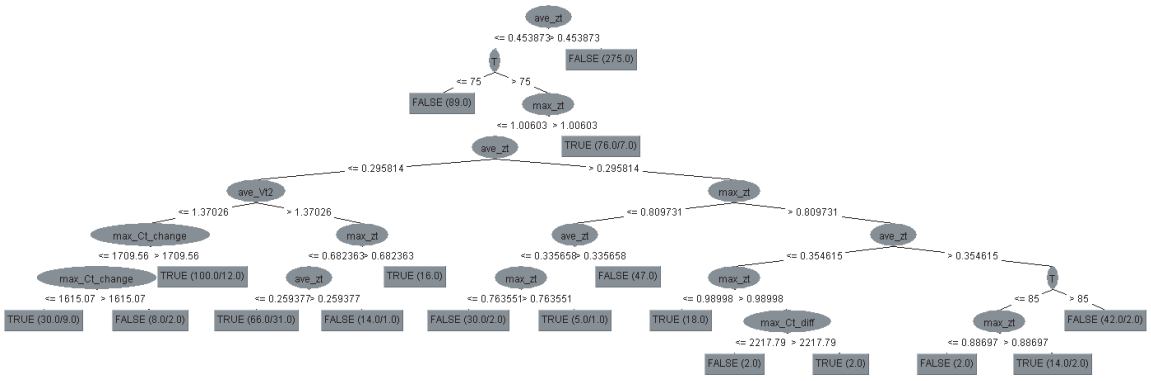


Figure 3.16: Decision tree generated by C4.5 Revision 8 algorithm.

3.6.2 Remarks

The main challenge in searching for a target in a noisy environment is identifying the anomalies that are encountered by the agent. This is challenging because in many cases, the agent is equipped with a simple sensor that captures a limited amount of data. The particle filter method gracefully captures the temporal aspects of how the data was obtained by using both the motion and sensor model of the agent and maximizes the amount of information gained from the sensor returns. The outputs from the particle filter and the

sensor data are then used to generate a low dimensional feature vector. This feature vector is able to distinguish between the true target and a false anomaly very effectively and can be used to train any number of machine learning algorithms for classification.

Training and learning a classifier off-line saves computational resources which can instead be directed towards feature extraction during a real time mission. Current research is directed toward optimizing the particle filter routine or finding other less computationally intensive methods for incorporating temporal smoothness and consistency into the feature vector. The high classification accuracies are very encouraging and efforts to find other features that may increase the accuracy even further are also being investigated. Previous research on this topic can be found in [72].

Chapter 4

BELIEF MAPS

In this chapter, we changes topics and look at some of the components that are required in order to address the more strategic problem of coordinating a multi-agent search mission. One of the most important aspects of any search mission is to create a representation of the environment that contains information about the target and world in which the agents operate. Section 2.4 detailed some representations such as two dimensional probability density functions and the like. This chapter focuses on an uncoupled, discrete representation of the world known as an occupancy based map.

4.1 Occupancy Based Maps

In order to effectively search a two dimensional domain for a target, the system must keep track of the state of the world in terms of possible target locations. To do this, an occupancy based map is employed. Recall that these constructs were originally formalized by Elfs [35], [34] but we add functionality such as Bayesian score updates and time varying models to the maps to accommodate our algorithm.

4.1.1 Occupancy Based Maps Definition

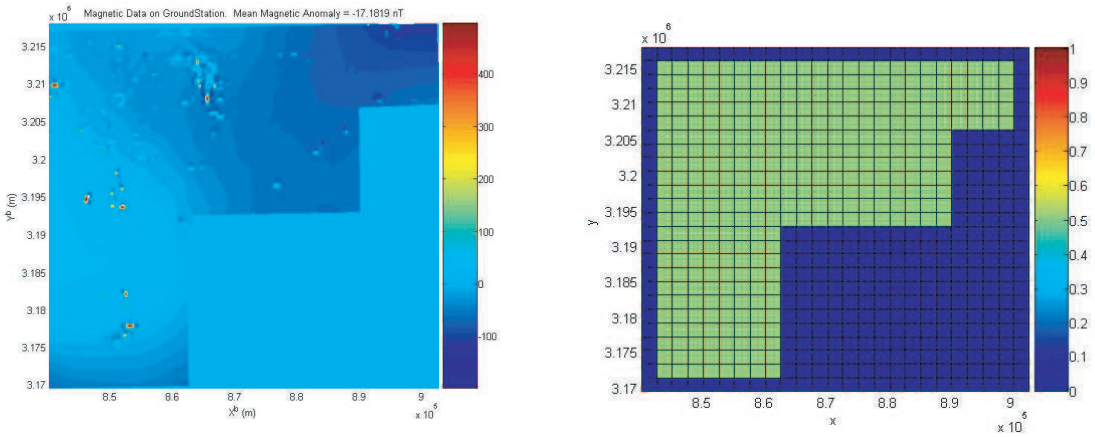
The search domain is discretized into rectangular cells. Each cell is assigned a score which is the probability that the target is located in that cell. This is similar to a two dimensional, discretized probability density function [21]. The spatial domain of the occupancy based map consists of a box where x is between x_{min} and x_{max} and similarly for the y dimension.

$$B = \left\{ \overline{z} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \mid z_1 \in [x_{min}, x_{max}], z_2 \in [y_{min}, y_{max}] \right\} \tag{4.1}$$

The occupancy based map is a function defined over the set $B \times \Re$ which assigns a score

in the range $[0, 1]$ to each element $\overline{z} \in B \subset \Re^2$ at a certain time step $k \in \Re$. In other words, $x_w : B \times \Re \rightarrow \Re$. The score of a given cell represents the probability that the target is located in that cell.

The occupancy based map is shared and updated by all agents involved in the search. At each time step, guidance decisions for each agent are computed based on this map. The state of the map at any time k is also referred to as the world state. This reflects the fact that the map represents the possible locations of targets and other objects in the environment. In essence, the system's belief of the state of the world is embedded in the state of the occupancy based map. A TMI map is shown in Figure 4.1(a) and an example of an occupancy based map over the same domain is shown below in Figure 4.1.



(a) Total magnetic intensity map of region to search

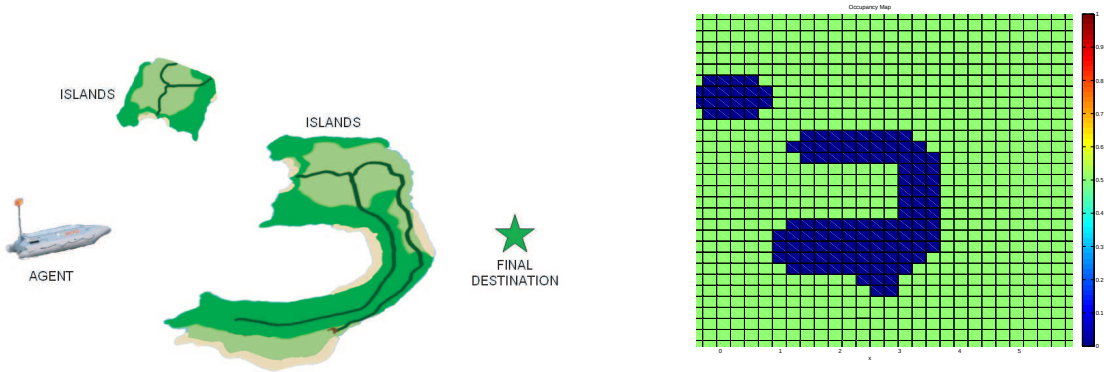
(b) Initial occupancy based map

Figure 4.1: Discretization of search region into an occupancy based map.

In Figure 4.1(b), the blue sections represent cells with zero scores whereas the green sections represent scores of 0.5. This is the initial state of the occupancy based map. It represents the situation where no a priori knowledge of the target's location exists other than it cannot be in a section where no real data exists (sections of uniform blue in Figure 4.1(a)).

The map can contain information about other aspects of the world rather than the presence or lack of magnetic data in that region. For example, the map can represent the world state with locations of obstacles and reward areas in the environment. The idea of

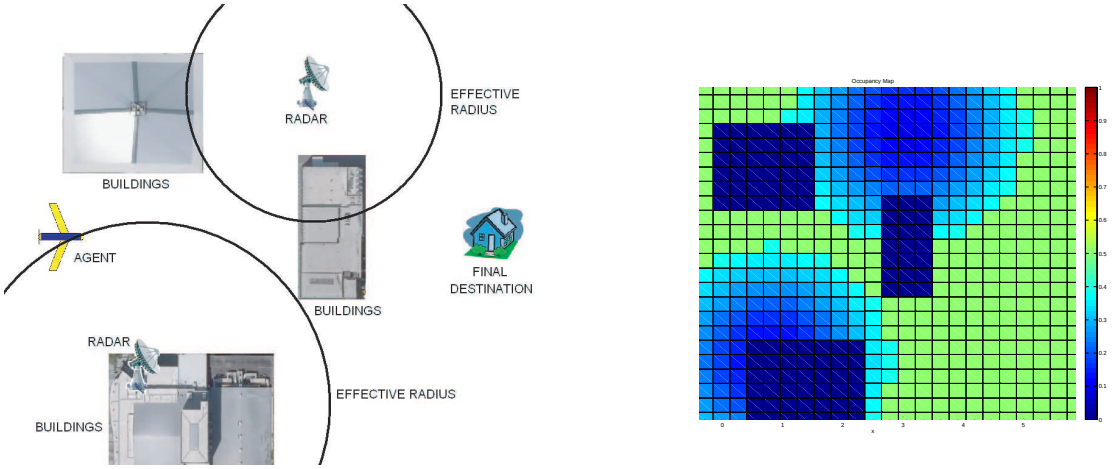
embedding obstacles in the map is similar to defining obstacles in a configuration space [65]. The agent's belief of the world state is embedded in the occupancy based map. In this application, the occupancy based map only provides information about two states of the agent's pose (i.e. the planar position). An example of embedded obstacles and reward areas in an occupancy based map is shown below in Figures 4.2 and 4.3.



(a) Physical environment with hard obstacles

(b) Occupancy based map representation of environment

Figure 4.2: Abstraction of marine environment using occupancy based maps.



(a) Physical environment with soft and hard obstacles

(b) Occupancy based map representation of environment

Figure 4.3: Abstraction of urban environment using occupancy based maps.

Figure 4.2(a) represents a marine environment that an agent may be forced to navigate through. In this scenario, there are two islands that are considered hard obstacles. This environment can be abstracted using an occupancy based map as shown in Figure 4.2(b). Here, the dark blue sections represent cells with zero scores (hard obstacles) and the green sections represent scores of 0.5 (neutral values).

An example of an urban environment with hard and soft obstacles is shown in Figure 4.3(a). The idea is the same except there are regions which correspond to soft obstacles that should be avoided if possible but entering these regions does not violate a constraint. These sections are represented by the lighter blue shades with scores ranging from 0 to 0.5. If there were regions that were beneficial to the agent, these could be assigned scores greater than 0.5.

It is useful to define the cell center set, $\tilde{B}$, as all values $\overline{z}$ which correspond to the center of a cell in the occupancy based map.

$$\tilde{B} = \left\{ \overline{z} \,\middle|\, \overline{z} = \begin{pmatrix} x_{min} + L_x(i - 1/2) \\ y_{min} + L_y(j - 1/2) \end{pmatrix}, i = 1, 2, \ldots, N_x, j = 1, 2, \ldots, N_y \right\} \tag{4.2}$$

The occupancy based map and its associated features provides a versatile framework from which to build a searching algorithm.

4.1.2 Combining with Geo-Referenced Images

The occupancy based maps can be combined with geo-reference imagery to create maps which correspond to real locations. Various software manipulation libraries such as OpenCV can be used to manipulate geo-referenced images and assign the domain of the occupancy map to correspond to locations on satellite images. Examples of these are shown in Chapter 8, Figure 8.4.

4.2 Updating Maps

Now that the occupancy based maps have been defined, it becomes useful to look at how these maps are updated. The world state is constantly changing and the maps must be able to reflect a dynamic environment.

4.2.1 Updating Map Cell Scores

The occupancy based map is dynamic and can be updated either by agents involved in the mission, external sources, or other means. The agents are able to modify the map to reflect their findings during the search mission. Each agent in the team is able to search the cell at its current location using its sensor. The discrete state space of the cell is simply $X_k = \{x_A, x_B\}$ where $X_k = x_A$ corresponds to the target not in the cell and $X_k = x_B$ meaning that the target is in the cell. In a similar fashion, the agent may make one of two sensor measurements, $Z_t = z_A$ (observe target not in cell) and $Z_t = z_B$ (observe target in cell). As mentioned previously, the score of a given cell in the occupancy based map reflects the scalar probability that the target is located in that cell at the current time step k. For convenience, the score of the cell at time step k is denoted $s_k = p(X_k = x_B)$.

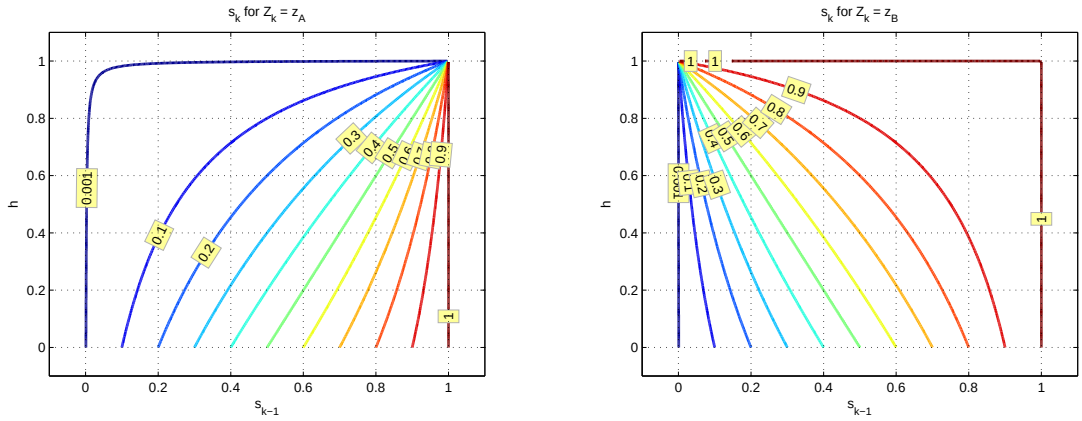
To model a heterogeneous team of agents with stochastic sensors, each agent's sensor is assigned a reliability factor $h \in [0, 1]$. A value of $h = 0$ implies that the sensor is completely unreliable and no information can be gained from this sensor. Conversely, $h = 1$ corresponds to a completely reliable sensor that can ascertain if the target is or is not located in the agent's current cell in a single measurement. The probabilistic sensor model can be formed as

$$\begin{aligned} p(Z_t = z_A | X_t = x_B) &= 1 - \tfrac{1}{2}(h+1) \\ p(Z_t = z_B | X_t = x_B) &= \tfrac{1}{2}(h+1) \end{aligned} \tag{4.3}$$

Assuming that the state of any given occupancy map cell score is not affected by the action of taking a measurement, the probabilistic score of a given occupancy based map cell can be updated using the sensor model of Eq. 4.3 which yields the following Bayesian update rule.

$$s_k = \begin{cases} \frac{s_{k-1}(1-h)}{1+(1-2s_{k-1})h} & \text{if } Z_k = z_A \\ \frac{s_{k-1}(1+h)}{1+(2s_{k-1}-1)h} & \text{if } Z_k = z_B \end{cases} \tag{4.4}$$

In Eq. 4.4, s_k represents the score of the occupancy map cell ($s_k = x_w(k, \overline{z})$ for $\overline{z}$ in some cell region). Contour plots of this function are shown in Figure 4.4.



(a) $s_k(s_{k-1}, h)$ for $Z_k = z_A$ (b) $s_k(s_{k-1}, h)$ for $Z_k = z_B$

Figure 4.4: Contour plots of the $s_k()$ function.

This update rule has several interesting properties. We show that the scores of each cell either monotonically increase or decrease with each sensor measurement for the majority of values of h and s_{k-1}. It is trivial to see that if $s_{k-1} = 1$ then $s_k = 1$ and if $s_{k-1} = 0$, then $s_k = 0$ for either case of $Z_k = z_A$ or $Z_k = z_B$ and for all values of h. The physical significance of this is that if the target is absolutely certain to either be in the cell or not, then no further updates will change this fact. Similarly, for $Z_k = z_A$, if $h = 0$ then $s_k = s_{k-1}$ and if $h = 1$ then $s_k = 0$. This implies that if the sensor is completely unreliable ($h = 0$), making a measurement with this sensor will not change the state of the cell. Conversely, if the sensor is completely reliable with $h = 1$, then only a single measurement of $Z_k = z_A$ is all that is required to change the score of the cell to 0. A similar situation arises for $Z_k = z_B$ where if $h = 0$ then $s_k = s_{k-1}$ but if $h = 1$ then $s_k = 1$. This is because z_B with $h = 1$ is the event of detecting the target with a 100% reliable sensor. The more interesting cases are when s_{k-1} and h are in the interval $(0, 1)$.

Theorem 4.2.1. *Under the update rule of Eq. 4.4,* $s_k < s_{k-1}$ *if* $Z_k = z_A$ *and* $s_k > s_{k-1}$ *if* $Z_k = z_B$ $\forall s_{k-1}, h \in (0, 1)$.

Proof. For the case of $Z_k = z_A$ the burden is to now show that $s_k < s_{k-1} \forall k$ for the cases where $s_{k-1}, h \in (0, 1)$.

One can examine the denominator of Eq. 4.4 and note that $\forall s_{k-1} \in (0,1)$ the term $1-2s_{k-1} > -1$. Therefore, the denominator term must be greater than $1-h$. Noting that for all $h \in (0,1)$ the term $1-h > 0$. Therefore, the denominator term must always be positive.

Furthermore, it is easy to see that $1-s_{k-1} > 0$ and $2h > 0$ for these conditions. So one can write

$$\begin{aligned} 0 &< 2h(1-s_{k-1}) && (4.5)\\ 1-h &< 1+h-2hs_{k-1} && (4.6)\\ \frac{1-h}{1+h-2hs_{k-1}} &< 1 && (4.7)\\ & && (4.8) \end{aligned}$$

Multiplying both sides by s_{k-1} yields the final result

$$\frac{s_{k-1}(1-h)}{1+(1-2s_{k-1})h} = s_k < s_{k-1} \;\; \forall h, s_{k-1} \in (0,1) \tag{4.9}$$

It is trivial to show that $s_k > 0 \; \forall k$. This in conjunction with Eq. 4.9 shows that with enough sensor measurements of $Z_k = z_A$, the score of a given cell will proceed monotonically towards 0.

A similar proof can be applied for the case of $Z_k = z_B$ with the result showing that with enough sensor measurements of $Z_k = z_B$, the score of a given cell will proceed monotonically towards 1. □

Although the occupancy map cells scores are assumed to be independent, a common operation with the map is to increase or decrease scores within a certain radius according to a given function. This may be used when the map cells are smaller than the size of the anticipated target or the target identification process is not highly accurate or certain. In this case, it is desired to add some coupling to model the fact that if an agent identifies that a target may be at a certain location, it is also likely that it may also be located nearby. The update rule of Eq. 4.4 can be applied with $Z_k = z_B$ and h set to be a function of radial distance to increase scores around the point $\overline{\mu}$. Similarly, if the scores are desired to be

decreased within the radius, the update rule may be used with $Z_k = z_A$. Eq. 4.10 is used to compute the corresponding value of h to use in this radial update function.

$$h = \begin{cases} \xi(1 - \frac{1}{r^2}(\overline{z} - \overline{\mu})^T(\overline{z} - \overline{\mu})) & \text{if } ||\overline{z} - \overline{\mu}|| \leq r \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

The update equation given in Eq. 4.4 can model an agent with a single point sensor. The radial update function of Eq. 4.10 is useful for modeling a wide area sensor that can search several cells simultaneously such as a wide angle camera similar to one shown in Figure 5.2(a).

A standard Markov assumption is used to remove the dependence of the next score on past scores, and therefore only one version of the occupancy based map must be maintained at any given time step.

4.2.2 Time Varying Maps

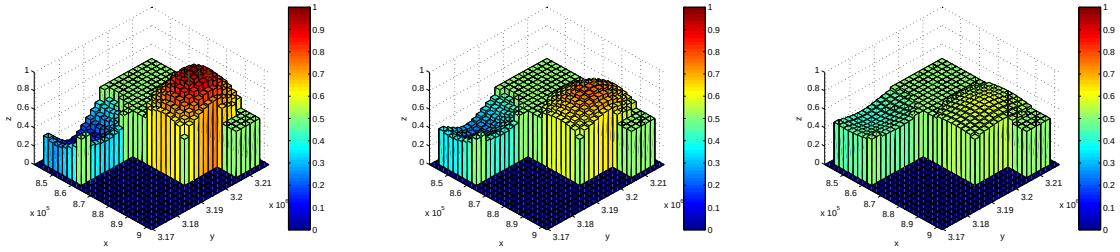
In addition to updating individual occupancy map cell scores or regions of scores, the entire map can be updated simultaneously. This can be used for situations where the map is time varying. The map can be time varying to model the fact that target location estimates become more uncertain as time progresses [14]. One modeling choice is to use a simple linear dynamic model of the form

$$x_w(k+1, \overline{z}) = \tau x_w(k, \overline{z}) + (1 - \tau)x_{w,nom} \quad \text{for } \overline{z} \in B \tag{4.11}$$

In Eq. 4.11, $x_{w,nom}$ is simply the nominal score (typically 0.5) and $\tau \in [0, 1]$ is the time constant governing how fast the score decays back to the nominal. The other stable region of $\tau \in [-1, 0]$ for the discrete system in Eq. 4.11 is excluded because although $\tau \in [-1, 0]$ does lead to a system that decays to the nominal score, it does so in an oscillatory manner, which does not make physical sense in this application.

The time varying effect using Eq. 4.11 is shown below in Figure 4.5. There is no estimate of target velocity so the regions of high probability do not translate in the x and y directions. However, as time progresses, the estimates return to their nominal values to model the

phenomenon that old measurements cannot be relied upon to judge if the target is still located in a certain cell or not.



(a) $x_w(k, \overline{z})$ (b) $x_w(k+1, \overline{z})$ (c) $x_w(k+2, \overline{z})$

Figure 4.5: World estimates decaying back to the nominal score over time.

Note that the results shown in Figure 4.5 have some minor heuristics included. Namely, if the score of the cell is zero, then Eq. 4.11 is not applied. If it was applied, these scores would decay to the nominal value as well. Notice that Eq. 4.4 naturally incorporates this behavior. Therefore, another option for implementing a time varying map is to use Eq. 4.4 to define the probabilistic update rule of

$$x_w(k+1, \overline{z}) = \begin{cases} \max\left(x_{w,nom}, \frac{s_k(1-h)}{1+(1-2s_k)h}\right) & \text{if } x_w(k, \overline{z}) > x_{w,nom} \\ \min\left(x_{w,nom}, \frac{s_k(1+h)}{1+(2s_k-1)h}\right) & \text{if } x_w(k, \overline{z}) < x_{w,nom} \\ x_w(k, \overline{z}) & \text{otherwise} \end{cases} \tag{4.12}$$

When Eq. 4.12 is applied for all $\overline{z} \in \tilde{B}$, the effect is similar to that shown in Figure 4.5 but no additional heuristics are required and the result is consistent with the Bayes probabilistic rule used to update the map.

One final aspect of system modeling that concerns updating the map is how the agents update the map when they encounter a target. If the agent searches the cell where the target is located and successfully finds the target, it can update the scores of its current cell and those around it using the radial update equation of Eq. 4.10. The sensor reliability factor is also used to model a sensor which may miss a positive target identification. If h is low, there is a high probability that the agent will not find the target even if it searches

the correct cell. This behavior affects several performance metrics such as the average time to target detection, which is discussed later in Chapter 6, Section 6.2.4.

Chapter 5

MULTI-AGENT SEARCHING

With the occupancy map framework in place, the strategic searching strategy can be addressed in this chapter. The problem of searching an area using a team of possibly heterogeneous, autonomous agents is still an open problem. Many of the sources listed in Section 2.2.1 provide methods to perform this mission with varying degrees of success. For example, randomized coverage methods [42], [68] are unable to provide guarantees regarding map coverage and target detection. Other approaches such as Bayesian searching [21], [20], [122] have difficulties embedding information about a complex environment into the algorithm. In terms of sub-tasks such as path planning, methods such as evolutionary computation [92], [25] may prove to be prohibitively expensive in terms of computational resources. This chapter focuses on addressing these issues and developing a modular and scalar algorithm that can be applied to this situation [76].

The search algorithm is modular in the sense that it is comprised of three main steps. Each step is somewhat independent of the others and different algorithms can be used to accomplish the same goal within any of the three main steps. The first step involves propagating the world state (the occupancy map) forward in time. By providing a predictive aspect to the problem, each agent can then make control decisions based on the predicted future state of the world rather than only using the current information. Once the system generates a predicted future world state, each agent determines a desirable coordinate to visit in the future. The desirability of a location is measured using a complex objective function. This formulation allows for each agent in the team to have a different set of parameters and therefore, each agent can have its own notion of desirability. Finally, once this desirable coordinate is determined, a path through the environment that transitions the agent from its current location to the desirable coordinate can be applied.

5.1 Algorithm Overview

It is desired that agents in the team exhibit certain behaviors and achieve several goals during a mission. One desired behavior is that the agent is to converge on regions of high score (a high probability that the target is located in a given location). Once an agent locates an anomaly, it should loiter there until a positive identification can be made by techniques such as discussed in Chapter 3. While this occurs, other agents who are farther away will continue searching. A search strategy for a single agent is developed with these goals in mind and then applied to each agent in the team. The overall flow of the search strategy for a single agent is shown in Figure 5.1.

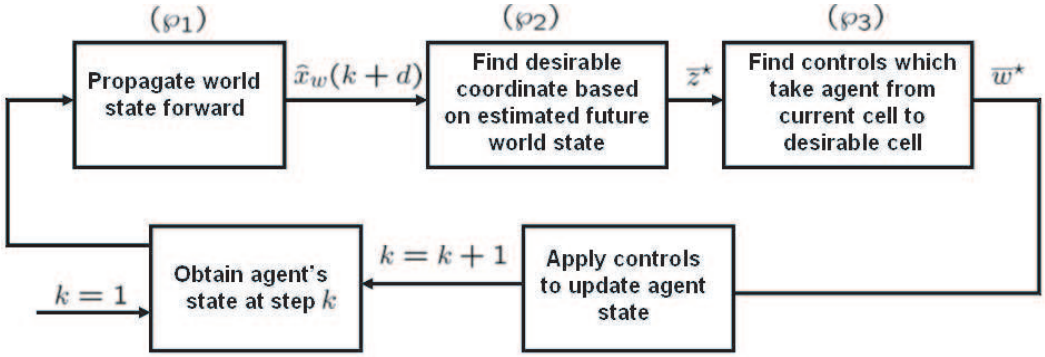


Figure 5.1: Flow diagram for single agent search strategy.

The single agent search strategy is comprised of three subproblems which are referred to as $(\wp_1)$, $(\wp_2)$, and $(\wp_3)$. The process starts by finding the agent's state at the current time, $\overline{x}_{agt}(k)$. Next, $(\wp_1)$ is solved to obtain the estimated world state at time $k+d$. Next, $(\wp_2)$ is solved to find a desirable coordinate, $\overline{z}^\star$, which the agent will visit within the next d steps. Finally, $(\wp_3)$ consists of finding an optimal set of waypoints/controls, $\overline{w}^\star$, which will take the agent from its current location to the location of the desirable coordinate found in $(\wp_2)$.

Each agent in the team follows this same policy without information or explicit knowledge of other members of the team. This approach has several advantages and disadvantages.

The major benefit of this strategy is that the computational resources needed for this

algorithm grow linearly with the number of agents. Each agent does not require explicit knowledge of other agents to compute its own control law. This allows the algorithm to be scalable to a large number of agents. Furthermore, although the algorithm is centralized in the sense that there must be a star communication topology where each agent is in communication with a centralized base, the computation for the algorithm can be performed in a decentralized manner in the sense that each agent can make its own decisions based on information from the base without consulting other agents in the team.

The primary disadvantage of this policy is that there is no explicit cooperation between agents. As shown in many situations, explicit cooperation has the potential to improve performance of the system at the cost of increased complexity and computational resources [90], [60], [122]. Some modifications to allow for explicit cooperation are presented in Section 5.5.

5.2 ($\wp_1$) ***Predictive World Model***

The main goal of problem ($\wp_1$) is to provide an estimate of the world state at a future time. This gives the algorithm an aspect of model predictive control [26], [57]. If an agent comes close to a target, it makes an estimate of the target state and the estimated target state is then propagated forward in time. Scores of the map are updated to reflect the future state of the world based on this target state estimate.

5.2.1 Target State Estimation

The estimated future world state is based on the estimated future state of the target. The target is assumed to have simple, planar kinematic dynamics. The estimated target state dynamics is assumed to have the form of

$$\hat{x}_{tgt}(k+1) = A_{tgt}\hat{x}_{tgt}(k) + B_{tgt}\hat{u}_{tgt}(k) \tag{5.1}$$

In Eq. 5.1, $\hat{x}_{tgt} = (\hat{x}_{tgt}\ \hat{y}_{tgt}\ \hat{\chi}_{tgt})^T$. In other words, it is assumed that the agent is able to estimate the position and course angle of the target. The estimated control vector is $\hat{u}_{tgt} = (\hat{V}_{x,tgt}\ \hat{V}_{y,tgt})^T$, meaning that the agent is able to estimate the planar velocity of the

target. Situations where these are not realistic assumptions are beyond the scope of this research and it is assumed that a secondary, tactical level algorithm (perhaps similar to that described in Chapter 3) has already be realized and implemented on the agents in order to accomplish this goal. For simulation implementation, the estimated target input, $\hat{u}_{tgt}$, is assumed to be the true value of the control plus additive Gaussian noise. Each agent has an effective radius for target state estimation. This is used to model agents which may be equipped with wide area sensors such as visual or infrared cameras. If the target is located within this radius, the agent is able to make an estimate of the target's state and control vectors. The estimated target state and control vectors are given as

$$\hat{x}_{tgt} = \begin{pmatrix} g(N(x_{tgt}, \sigma_x^2)) \\ g(N(y_{tgt}, \sigma_x^2)) \\ g(N(\chi_{tgt}, \sigma_\chi^2)) \end{pmatrix} \tag{5.2}$$

$$\hat{u}_{tgt} = g(N(\overline{u}_{tgt}, \sigma_V^2)) \tag{5.3}$$

It is assumed that the maximum velocity of the target, $V_{max,tgt}$ is known. This is a realistic assumption given some basic pre-mission intelligence. With this knowledge, the standard deviations for the target state and control noise are computed as

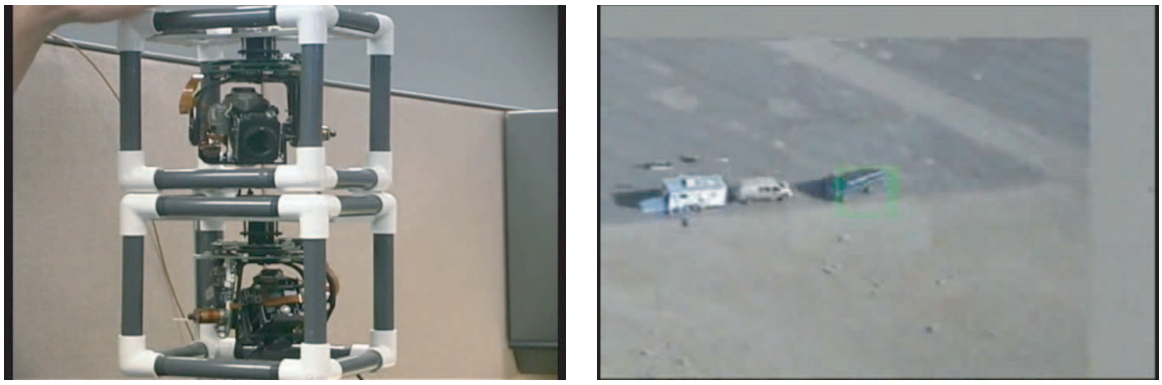
$$\sigma_x = \frac{(1-\lambda)}{2\hat{r}}\varphi \tag{5.4}$$

$$\sigma_V = \frac{(1-\lambda)V_{max,tgt}}{2\hat{r}}\varphi \tag{5.5}$$

$$\sigma_\chi = \frac{(1-\lambda)\pi}{2\hat{r}}\varphi \tag{5.6}$$

This model is useful for modeling a wide area sensor such as a camera system [50]. For example, the inertially stabilized camera system on the ScanEagle UAV is shown in Figure 5.2(a). A screen shot from the associated video feed is shown in Figure 5.2(b). Notice that the edges of the screen are blurry from the vibrations in the camera and the resulting

software stabilization. This provides motivation for using a model where the accuracy of the sensor readings deteriorate as the distance from the center of the camera focal point increases.



(a) ScanEagle camera system (b) Screen shot with software stabilization

Figure 5.2: Gimballed camera system on ScanEagle and associated screen shot of video feed.

Here, $\varphi = ||\overline{z}_{agt} - \overline{z}_{tgt}||$. Therefore, the estimated target state and control noise are a function of the distance from the agent to the target. Increasing the standard deviation linearly with distance models the fact that the sensor measurements are less reliable as distance increases. The parameter $\lambda \in [0, 1]$ is used to model the sensor reliability. A value of $\lambda = 0$ implies an unreliable sensor that cannot be heavily trusted. In this case, the error in the position estimate of the target can vary with 95.45% of the samples falling within $\pm\varphi$. This means that the variation in position estimate may be of the same magnitude as φ. A similar behavior is exhibited for the variation in the control velocity and course angle. These will most likely vary up to the maximum velocity of the agent and $\pm$ 180 degrees, respectively. An example of this estimation is shown in Figure 5.3.

In Figure 5.3 the blue 'x' represents the location of the agent and the triangles are locations of targets. The dashed red circle represents the agent's radius of target estimation, $\hat{r}$. The red airplane represents the estimated target state, $\hat{x}_{tgt}$ and the red arrow represents the estimated target control vector, $\hat{u}_{tgt}$. Figure 5.3 shows that when the agent is closer to the target, the estimates are in general more accurate (due to smaller standard deviations

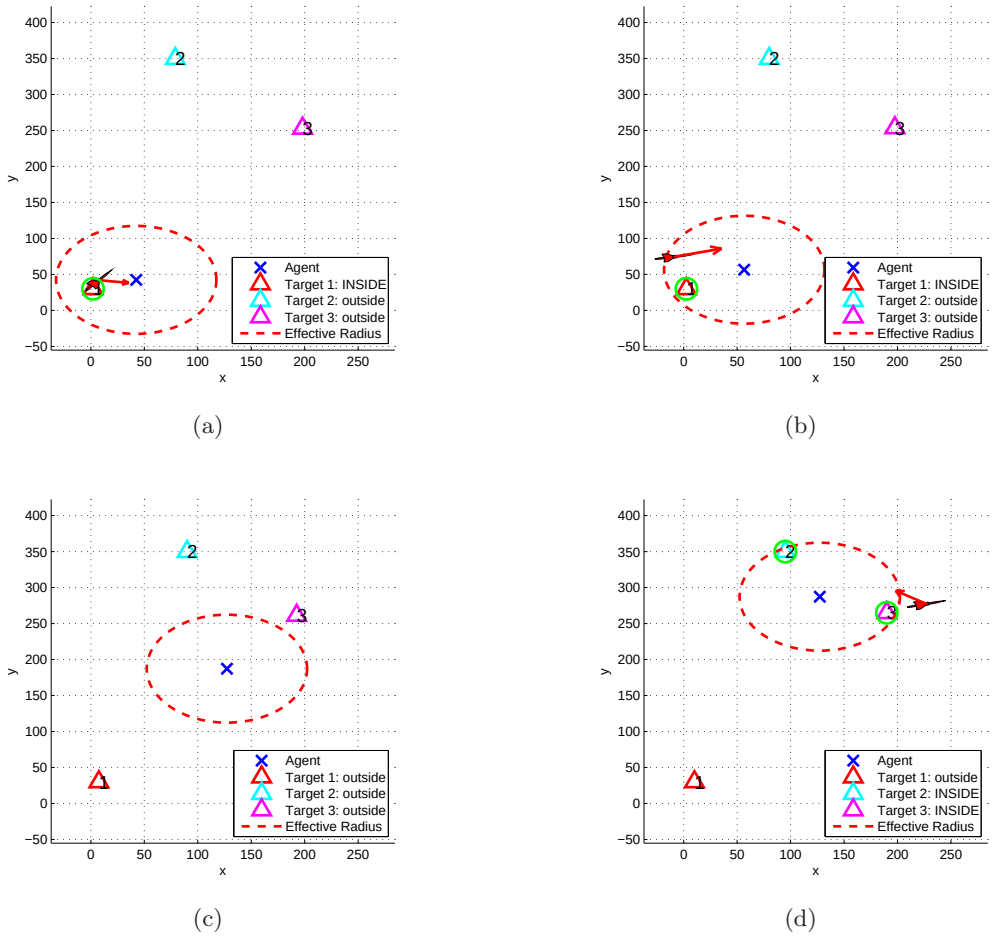


Figure 5.3: Estimated target state and control vectors for $\lambda = 0.15$.

on the noise). Furthermore, if there are no targets within the radius of target estimation, then no estimates of the target state or controls are made (Figure 5.3(c)).

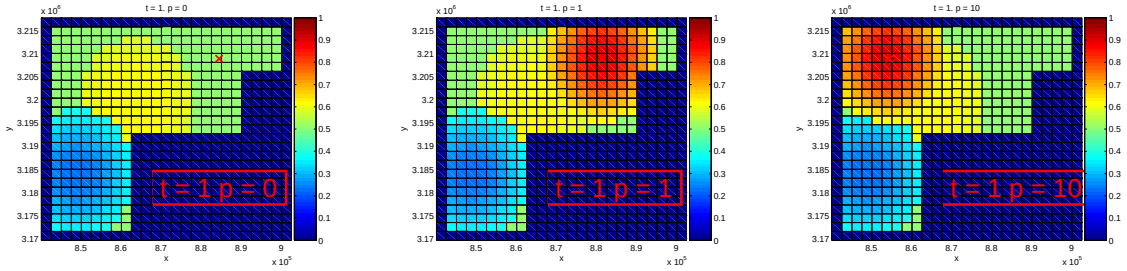
5.2.2 World State Estimation

The target state and control vector are estimated using the method described in Section 5.2.1. These estimates can be used to propagate the estimated target state forward to an arbitrary time in the future. Using this information, the estimated state of the world can be updated as well. This is done by taking the current state of the world, $x_w(k, \overline{z})$, and then modifying it using the estimated future state of the target. The world estimate

at time $k+p$ is then a function of the estimated target state at time $k+p$ and the world state at the original time k.

$$\hat{x}_w(k+p,\overline{z}) = \Gamma(\hat{x}_{tgt}(k+p), x_w(k,\overline{z})) \quad \text{for } p = 0, ..., d \tag{5.7}$$

Eq. 5.7 is a fairly general representation of the estimated future world state. $\Gamma()$ can be implemented using several methods. One method is to simply add a two dimensional Gaussian centered about $\hat{x}_{tgt}(k+d)$ to $x_w(k,\overline{z})$. An example of this is shown below in Figure 5.4 when the estimated target state is observed to be moving to the left at a constant velocity.



(a) $\hat{x}_w(k,\overline{z}) = x_w(k,z)$ (b) $\hat{x}_w(k+1,\overline{z})$ (c) $\hat{x}_w(k+10,\overline{z})$

Figure 5.4: Estimated world states at different times for estimated target moving to the left.

A more robust alternative is to update the scores of the map using the radial update function previously described in Eq. 4.10. The estimate of the future world state is similar to that shown in Figure 5.4 except it incorporates the current world scores. In other words, the scores of the region near $\hat{x}_{tgt}$ are increased but they are done so in a probabilistic fashion instead of simply replacing the scores with a given value.

In the event that the agent is not able to estimate the target state, the estimated future world state is the same as the current world state, $\hat{x}_w(k+p,\overline{z}) = x_w(k,\overline{z})\ \forall p$.

5.2.3 ($\wp_1$) *Remarks*

This strategy benefits the agent in several ways. The main purpose of ($\wp_1$) is to add an element of model predictive control to the algorithm. The most obvious is that the agent is choosing control actions which will benefit it d steps in the future rather than "only thinking one step ahead." This effect is illustrated later in Section 6.2.1, Figure 6.4.

Another advantage is that external sources other than agents may influence the agent's behavior. For example, human operators who may have access to additional mission intelligence can easily update the state of the world through ($\wp_1$). They can thereby influence the autonomous team while remaining relatively removed from the low level planning algorithms.

Now that the state of the world can be estimated at step $k+d$, the system attempts to find a coordinate that has desirable properties and is within the agent's reachable set. This is addressed in ($\wp_2$).

5.3 ($\wp_2$) ***Desirable Location Selection***

The agent now needs to choose a location that it must travel to based on the estimated future state of the world. In this section we describe several modular methods that can be used to choose a desirable cell to search.

5.3.1 Defining the Reward Function

In ($\wp_1$), the state of the world is propagated forward in time by d steps. The subproblem ($\wp_2$) concerns finding a desirable location (a desirable $\overline{z}$ value) for the agent to travel to in d steps. The desirability of a location is determined by a reward function of the following form

$$J_0(\overline{z}) = \alpha \hat{x}_w(k+d, \overline{z}) + \eta \left(\beta f_\chi(\overline{z}) + \gamma f_d(\overline{z})\right) + \delta f_h(\overline{z}) \tag{5.8}$$

The reward function is a combination of terms which model both the environment and agent states. For example, the term $\hat{x}_w(k+d, \overline{z})$ measures the estimated state of the world d steps into the future. The function $f_\chi()$ is a function which penalizes heading changes.

In a similar fashion, $f_d()$ rewards locations which are farther away from the agent. Finally, $f_h()$ is an indicator function which serves to drive the agent towards the highest score in the map. This describes the general behavior of the cost function, but a more detailed look at its construction is necessary.

With the cost function defined, the most desirable location is then found via an optimization scheme using Eq. 5.8 as an objective function. Typically, the set over which one optimizes Eq. 5.8 is the set of all locations that the agent can reach in d steps. This set is determined by parameters such as its maximum velocity and time step. The set of all locations that the agent can reach in d steps is referred to as the agent's reachable set, $B_R \subseteq B$. This application defines B_R as

$$B_R = \{\overline{z} \in B \mid ||\overline{z} - \overline{z}_{agt}|| \leq R_{max}\} \tag{5.9}$$

Eq. 5.9 assumes that the agent has no turn rate limits and the agent has simple planar kinematics. In a practical application where there may be saturation concerns, it is possible that B_R may not be a perfect circle as described in Eq. 5.9. In this case, it simply becomes more difficult to define and compute B_R but the following analysis is not affected by the geometry of B_R.

It is useful to also define the set of cell centers that the agent can reach in d steps. This is simply

$$\tilde{B}_R = B_R \bigcap \tilde{B} \tag{5.10}$$

In Eq. 5.8, the function $f_\chi()$ is given by

$$f_\chi(\overline{z}) = \begin{cases} 0 & \text{if } \overline{z} \text{ in same cell as current agent} \\ 1 - \frac{q\left(\chi_{agt}, \pi/2 - \text{atan2}(\overline{z}_2 - y_{agt}, \overline{z}_1 - x_{agt})\right)}{\pi} & \text{otherwise} \end{cases} \tag{5.11}$$

In Eq. 5.11, the function $q(a, b)$ computes the absolute angular difference between the two angles, a and b. A simple absolute value of the difference of a and b is not sufficient and

some simple heuristics are included in the function $\eta()$ of Eq. 5.8 to take care of situations such as where $a = 1 \cdot \pi/180$ radians and $b = 359 \cdot \pi/180$ radians. A simple absolute value of the difference would return an angle of $358 \cdot \pi/180$ radians, which is incorrect. However, the function $\eta()$ returns the correct angular difference of $2 \cdot \pi/180$ radians. Note that the range of $f_\chi()$ is $[0, 1]$. An example of this function is shown in Figure 5.6(c).

The function $f_d()$ of Eq. 5.8 is given by

$$f_d(\overline{z}) = \begin{cases} \frac{||\overline{z} - \overline{z}_{agt}||}{R_{max}} & \text{if } \overline{z} \in B_R \\ 0 & \text{otherwise} \end{cases} \tag{5.12}$$

The function $f_d()$ effectively rewards locations that are farther away from the current agent position until the distance R_{max} is reached; past this point, the function returns 0. An example of this function is shown in Figure 5.6(d).

Finally, the function $f_h()$ of Eq. 5.8 is an indicator function. To define this function, it becomes necessary to first define some intermediate variables. The first of these variables is given by

$$\tilde{B}_{max} = \left\{ \overline{z} \in \tilde{B} \mid \hat{x}_w(k+d, \overline{z}) \text{ is maximum } \forall \overline{z} \in \tilde{B} \right\} \tag{5.13}$$

The set $\tilde{B}_{max}$ is simply the set of cell centers that have the highest score in the map. Note that this set may have more than one cell center. For example, in Figure 4.1(b), the set $\tilde{B}_{max}$ would include all the centers of the green cells since they all have the highest score in the map. For further clarification, the relationship between the sets $\tilde{B}_{max}$, B, B_R, and $\tilde{B}_R$ are shown later in Section 5.5.1 Figure 5.25.

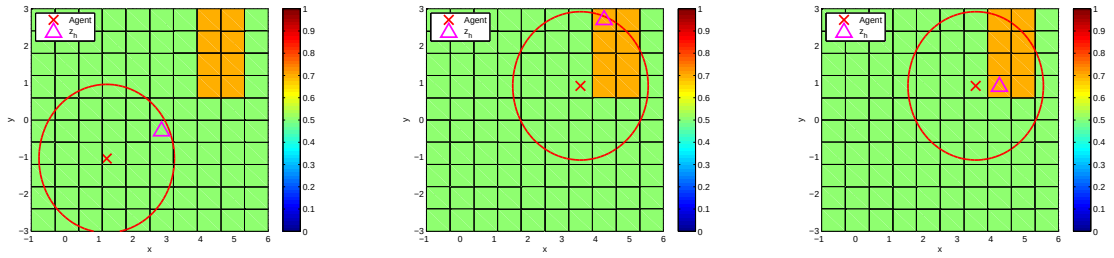
A single cell center from $\tilde{B}_{max}$ can be chosen using one of two methods. The first method involves choosing the point in $\tilde{B}_{max}$ which is closest to the agent. This location is designated as $\overline{z}_H$

$$\overline{z}_H \in \arg \underset{\overline{z} \in \tilde{B}_{max}}{\text{minimize}} ||\overline{z} - \overline{z}_{agt}|| \tag{5.14}$$

An alternative method for choosing $\overline{z}_H$ is to use

$$\overline{z}_H \in \begin{cases} \underset{\overline{z} \in \tilde{B}_{max}}{\arg\ \text{minimize}} \ ||\overline{z} - \overline{z}_{agt}|| & \text{if } \tilde{B}_{max} \bigcap \tilde{B}_R = \emptyset \\ \underset{\overline{z} \in \tilde{B}_{max} \bigcap \tilde{B}_R}{\arg\ \text{maximize}} \ ||\overline{z} - \overline{z}_{agt}|| & \text{otherwise} \end{cases} \tag{5.15}$$

The method for choosing $\overline{z}_H$ in Eq. 5.15 entails first checking if the set $\tilde{B}_{max} \bigcap \tilde{B}_R = \emptyset$. If this is true, this implies that there are no cells that have the highest score of the map in the agent's reachable set. If this is the case, Eq. 5.14 and Eq. 5.15 will choose the same point for $\overline{z}_H$. This can be seen in Figure 5.5(a). The two methods deviate if there is a cell with the highest score of the map within the agent's reachable set. In this case, Eq. 5.14 would still choose the cell in $\tilde{B}_{max}$ which is closest to the agent as shown in Figure 5.5(c). On the other hand, Eq. 5.15 would instead choose the cell with the maximum score that is farthest from the agent, but is still in the agent's reachable set as shown in Figure 5.5(b). Either method can be used and each has advantages and disadvantages that will be discussed in Section 5.3.3.



(a) $\overline{z}_h$ using either Eq. 5.14 or 5.15 (b) $\overline{z}_h$ using Eq. 5.15 (c) $\overline{z}_h$ using Eq. 5.14

Figure 5.5: Differences between using Eq. 5.14 and 5.15 for choosing $\overline{z}_H$.

With the point $\overline{z}_H$ defined, the point in the agent's reachable cell centers that is closest to $\overline{z}_H$ is given by

$$\overline{z}_h \in \underset{\overline{z} \in \tilde{B}_R}{\arg\ \text{minimize}} \ ||\overline{z} - \overline{z}_H|| \tag{5.16}$$

With $\overline{z}_H$ and $\overline{z}_h$ defined, the function $f_h()$ is simply given as

$$f_h(\overline{z}) = \begin{cases} 1 & \text{if } \overline{z} \text{ is in cell containing } \overline{z}_h \\ 0 & \text{otherwise} \end{cases} \tag{5.17}$$

An example of the $f_h()$ function is shown in Figure 5.6(e).

The final parameter in the reward function is the variable η which is the maximum score within the agent's reachable set.

$$\eta = \max_{\overline{z} \in \tilde{B}_R} \hat{x}_w(k+d, \overline{z}) \tag{5.18}$$

An example of the various functions that make up the reward function are shown below in Figure 5.6.

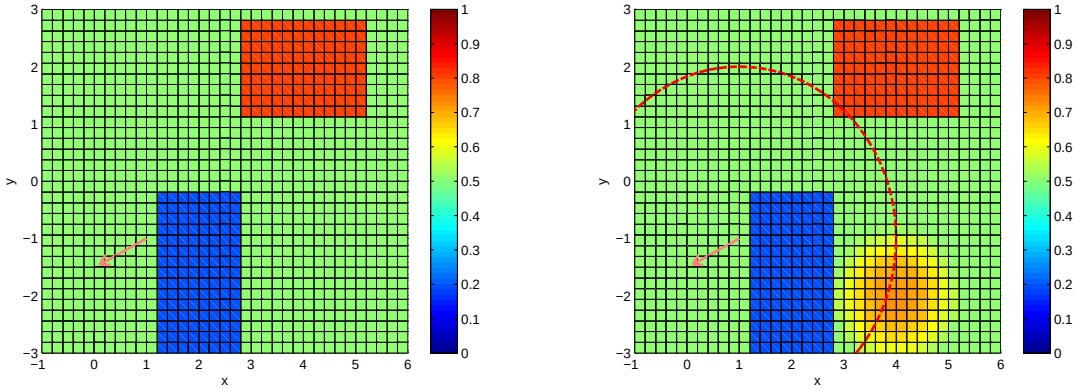
Figure 5.6(a) shows the state of the world at the current time. This is the system's belief of the world without making any estimates of the target location.

Figure 5.6(b) shows the estimated state of the world in d steps. In this case, an external source has provided information that the target is likely located in the lower right corner. This estimate can be made by a team member or another source in the system. The dashed red line shows the maximum distance the agent can reach in d steps. Maximizing this function over $\tilde{B}_R$ requires choosing the cell with the highest score within the dashed red line.

Figure 5.6(c) shows the $f_\chi()$ function for the particular case of the agent having a course angle of 243°. Maximizing this function requires picking a location $\overline{z}$ which minimizes the course change required to visit this location ($\overline{z}$ locations which are in line with the pink arrow).

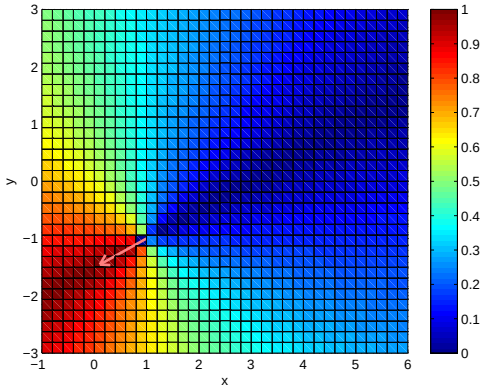
Figure 5.6(d) shows the $f_d()$ function. Maximizing this function corresponds to choosing a $\overline{z}$ location which is farthest away from the current location.

Figure 5.6(e) shows the indicator function $f_h()$. In this scenario, the set $\tilde{B}_{max}$ is the set of cell centers which maximize $\hat{x}_w(k+d, \tilde{B})$. These correspond to the cell centers of the dark red rectangle in Figure 5.6(b). From the set $\tilde{B}_{max}$, $\overline{z}_H$ is chosen as the cell that is closest to the agent (the lower left corner point). Finally, the point $\overline{z}_h$ is determined as the point within the agent's reachable cell centers which is closest to $\overline{z}_H$. As can be seen,

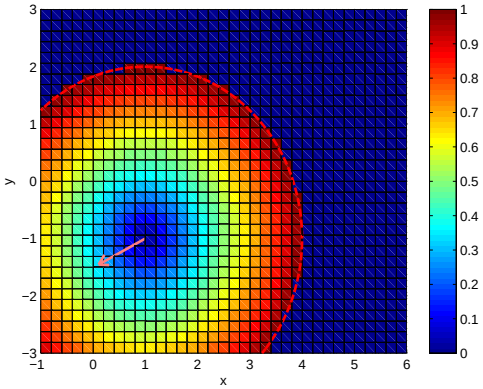


(a) $x_w(k, \overline{z})$

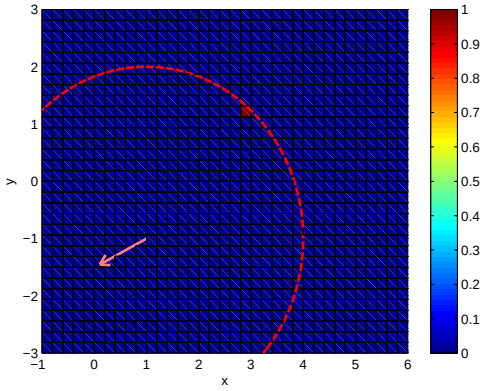
(b) $\hat{x}_w(k+d, \overline{z})$. $\alpha = 0.5$



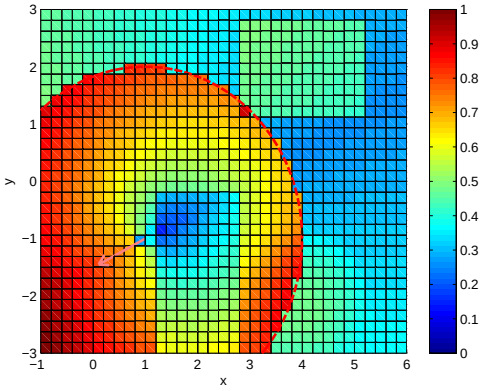
(c) $f_\chi(\overline{z})$.$\chi_{agt} = 243°$. $\beta = 0.5$



(d) $f_d(\overline{z})$. $\gamma = 0.5$



(e) $f_h(\overline{z})$. $\delta = 0.05$



(f) $J_0(\overline{z})$

Figure 5.6: Various functions which compose the reward function, $J_0()$.

maximizing $f_h()$ requires simply choosing $\overline{z}$ in the same cell as $\overline{z}_h$.

Figure 5.6(f) shows the normalized total reward function. The reward function can typically exceed 1 but it is normalized so that the max value is 1 for plotting purposes.

The parameters α, β, γ, and δ can be chosen differently to reflect different agent capabilities. For example, small α values yield agents which will not be adverse to large course changes. This may be appropriate for more agile agents such as small UAVs. Conversely, large α values correspond to agents which may deem a coordinate more desirable if it has a somewhat low score but is in line with its current course. This may be appropriate for cumbersome agents like large boats. One important aspect of Eq. 5.8 is that α, β, γ, δ, $\hat{x}_w()$, $f_\chi()$, $f_d()$, and $f_h()$ are all in the range $[0, 1]$. This is explained further in Section 5.3.3.

5.3.2 Quasi-Optimal Solution Via Adaptive Sampling

With the reward function $J_0()$ fully defined, the problem of finding the most desirable coordinate to search within the agent's reachable set may be posed as a standard optimization problem where the goal is to maximize $J_0()$ (Eq. 5.8) over the agent's reachable set.

$$(\wp_2) \quad \overline{z}^\star \in \arg \underset{\overline{z} \in B_R}{\text{maximize}} \; J_0(\overline{z}) \tag{5.19}$$

Although this formulation is straightforward, its solution is not. $(\wp_2)$ is far from a well posed, convex program. Although the feasible set B_R is typically a convex subset of $\Re^2$ (a perfect circle), the objective function is not as agreeable. The objective function $J_0()$ is interesting for several reasons. First, the function is the combination of a numerical, discontinuous function (the world state, $\hat{x}_w()$) and a continuous, functional representation of potential fields (i.e. the course penalty, $f_\chi()$). This combination of numerical lookup tables and analytical functions creates a unique situation requiring a numerical algorithm for solutions. In this situation, it may not be possible to find the true optimal to $(\wp_2)$. Instead, the primary objective becomes finding a feasible solution and then adding some notion of optimality as a secondary goal. This defines the concept of quasi-optimality. We now present a general method which provides a quasi-optimal solution to the problem of optimizing an arbitrary cost function over a box set. This mixes the ideas of probability

collectives [48] and particle filters [40]. Recall the particle filter in the context of target identification was detailed in Section 3.4.1. This approach innovates from these methods by applying the particle filter/adaptive sampling method to a general optimization problem over a box set while guaranteeing feasible solutions despite the stochastic nature of filter.

Theory

In this situation, we consider finding a feasible minimizer to the following general optimization problem.

$$(\wp) \text{ minimize } f_0(x) \text{ over } x \in X = \text{a box} \tag{5.20}$$

Recall that a box is defined by each element x_k of $x \in X \subseteq \Re^n$ being in a certain interval $I_k = [l_k, u_k]$.

$$X = \left\{ x \left| \begin{array}{c} x_1 \in I_1 = [l_1, u_1] \\ x_2 \in I_2 = [l_2, u_2] \\ \vdots \\ x_n \in I_n = [l_n, u_n] \end{array} \right. \right\} \tag{5.21}$$

The difficulty in solving Eq. 5.20 arises from the fact that the objective function is arbitrary and may not be well behaved (i.e. non-convex, non-differentiable, etc.). This is especially true in $(\wp_2)$. It may be difficult or impossible to find an optimal solution. An algorithm to find a quasi-optimal, feasible solution is now proposed.

1. Generate M particles (instances of $x \in X \subseteq \Re^n$) distributed over X in some fashion.

2. Assign weights to each particle based on its objective function value.

3. Resample the particles proportional to the weights.

4. Repeat step 2 and 3 until some stopping criterion is reached.

This algorithm can be interpreted as a type of evolutionary algorithm that employs adaptive sampling. Each particle represents a sample and each generation of particles is adaptively moved to locations where they matter the most.

Initial Particle Distribution

To find a quasi-optimal minimizer of $f_0()$, a finite set of possible minimizers is used. Each individual guess of a minimizer, $x^{[m]}(t)$ is called a particle and together the particles make up the particle set, $\chi(t)$.

$$\chi(t) = \bigcup_{M} x^{[m]}(t) = \left\{ x^{[1]}(t), x^{[2]}(t), \ldots, x^{[M]}(t) \right\} \tag{5.22}$$

To initialize the algorithm, it is necessary to assign actual values to the initial particle set. Since there is no a priori knowledge regarding the minimizer of $f_0()$, the initial distribution of the particles is chosen as a uniform distribution over the set X.

$$x_k^{[m]}(0) = \text{rand}(l_k, u_k) \quad \text{for} \begin{matrix} m = 1, \ldots, M \\ k = 1, \ldots, n \end{matrix} \tag{5.23}$$

Assign Particle Weights

A weight is now assigned to each particle based on its objective function value.

$$w^{[m]}(t) = -f_0(x^{[m]}(t)) \quad \text{for } m = 1, \ldots, M \tag{5.24}$$

Note that this assigns a higher weight to particles which yield a smaller objective function value.

Resample Particles

The next particle set is generated by first sampling from the current particle set proportional to the weights.

$$\tilde{x}^{[m]}(t) = g(\chi(t), w(t)) \quad \text{for } m = 1, \ldots, M \tag{5.25}$$

Here, $g()$ is a sampling function which samples elements from the particle set, $\chi(t)$, proportional to the weights, $w(t)$. One popular method is the roulette wheel method described previously in Section 3.4.1.

As with many evolutionary-type algorithms, a mutation process must be performed when evolving one population to another. This is true here as well and the mutation operation is simply adding noise to each sample $\tilde{x}^{[m]}$. Note that in order for each minimizer to be feasible, it is required that $x^{[m]}(t) \in X \ \forall t$. Care must be taken so that the noise added to each particle does not "push the particle out of X". The noise must therefore be in the interval

$$n^{[m]}(t) \in [l - \tilde{x}^{[m]}(t), u - \tilde{x}^{[m]}(t)] \quad \text{for } m = 1, \ldots, M \tag{5.26}$$

Finally, the new particle set is determined by

$$x^{[m]}(t+1) = \tilde{x}^{[m]}(t) + n^{[m]}(t) \quad \text{for } m = 1, \ldots, M \tag{5.27}$$

Eq. 5.26 and Eq. 5.27 ensure that each particle is feasible and therefore, this optimization scheme can be classified as an interior point method. In other words, this formulation guarantees that each particle $x^{[m]}(t) \in X \ \forall \ t$ (each particle represents a feasible solution to $(\wp)$).

This scheme has the feature that as the particle set evolves from generation to generation, the particles with a higher weight (i.e. lower objective function value) are more likely to continue on to the next population.

Stopping Criterion

Steps 2 and 3 are repeated until some stopping criterion is reached. An example stopping criterion could be: "terminate when the variance of the particles is reduced below some threshold." In this case, the process is simply repeated for T steps. The quasi-optimal minimizer is then computed from the average of the final particle set.

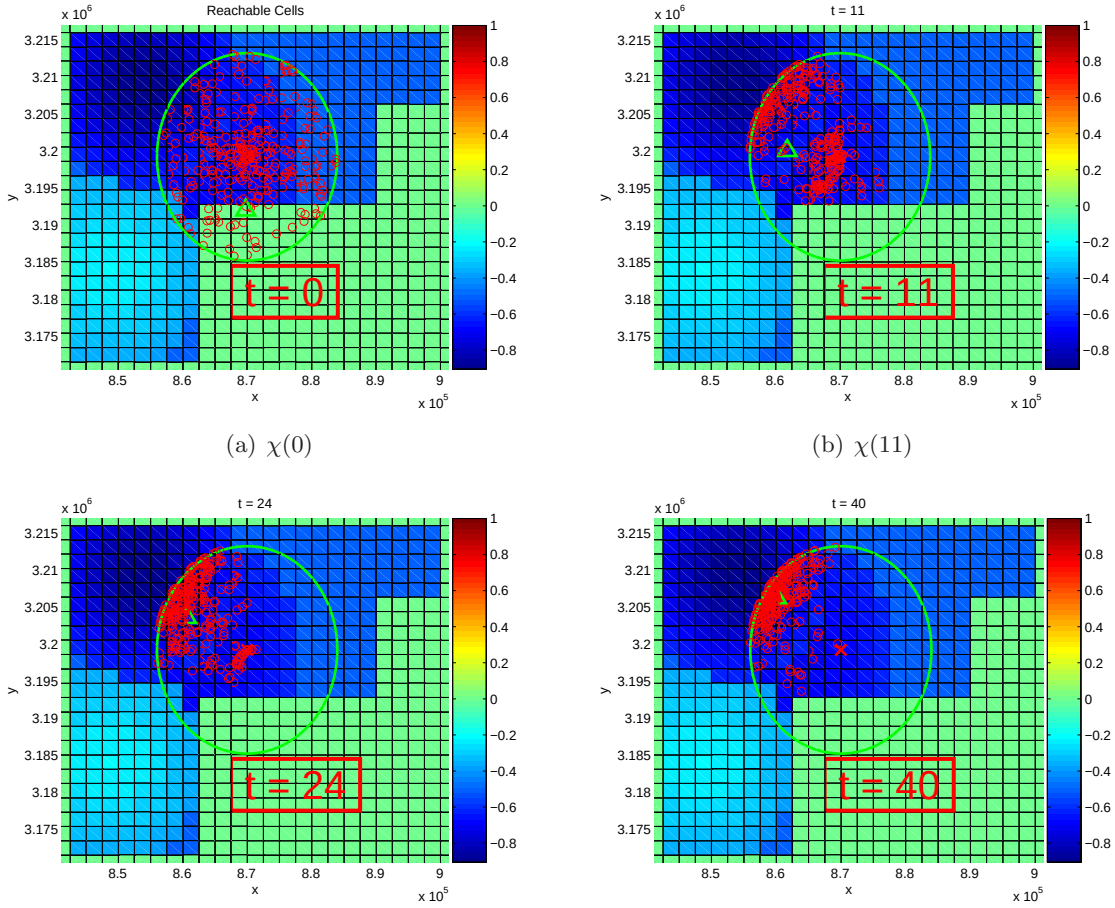
$$\bar{x}^{\star} = \frac{1}{M} \sum_{m=1}^{M} x^{[m]}(T) \tag{5.28}$$

Application to $(\wp_2)$

The above described method can be applied to the searching problem by parameterizing the reachable set B_R using a radius and an angle [70]. By placing an upper and lower bound on the radius and angle, the set B_R can be represented as a box set. An example of applying the probability collective/particle filter method to solve $(\wp_2)$ is shown in Figure 5.7 for $\beta = \gamma = \delta = 0$.

The individual particles are shown as red circles and the centroid of the particles is shown as a green triangle. After 40 iterations, the quasi-optimal minimizer settles near the true optimal solution. Note that it does not achieve the true optimal but it does achieve a feasible solution which is near the true optimal solution.

The above method is suitable for solving a general optimization problem over a box. Although $(\wp_2)$ can be solved using this method, it is computationally intensive due to the relatively large number of particles required and the necessity to evolve the population through several generations. The benefit of this method is that it is able to produce a quasi-optimal solution $\overline{z}^{\star} \in B_R$. This method may be suitable for slower agents which may have more time to plan/compute between waypoints. But in light of the computational intensity of the above method, an alternative approximation is desired for faster agents which operate at a higher bandwidth. An approximation is to use an exhaustive search of the objective function over the set $\tilde{B} \bigcap B_R$ (the occupancy based map cell center set intersected with the reachable set). This amounts to evaluating $J_0()$ at the center of each cell inside in B_R (i.e. inside the green circle in Figure 5.7) and choosing the argument that produces the largest objective function value as $\overline{z}^{\star}$. Although this method is less computationally intensive, the resulting approximate solution, $\overline{z}^{\star}$ is at the center of an occupancy map cell. If the cells are large, this may not be desirable since the resolution is greatly reduced using this method. This exhaustive searching method is described in the following section.



(a) $\chi(0)$ (b) $\chi(11)$

(c) $\chi(24)$ (d) $\chi(40)$

Figure 5.7: Progression of probability collective process. True minimum is located in upper left corner of reachable cells. Note that figure shows minimizing $-J_0()$ which is equivalent to maximizing $J_0()$.

5.3.3 Exhaustive Searching

The previous section investigated adaptive sampling methods to provide quasi-optimal solutions over the feasible set B_R [75]. Although this method worked well for most cases, it was nondeterministic. The stochastic nature when selecting $\overline{z}^\star$ makes analysis difficult. Therefore, instead of optimizing $J_0()$ over the compact set B_R, the problem is reduced to optimizing over the feasible set of $\tilde{B}_R$. With this approximation, the most desirable cell is then given by

$$(\wp_2) \quad \overline{z}^\star \in \arg\underset{\overline{z}\in\tilde{B}_R}{\text{maximize}}\ J_0(\overline{z}) \tag{5.29}$$

The problem formulated in Eq. 5.29 differs in that the feasible set is manageably finite. Therefore, the optimization is reduced to an exhaustive search over the set $\tilde{B}_R$. An additional benefit of this method is that it can be easily implemented by evaluating the reward function at each cell center in the set $\tilde{B}_R$.

The primary goal of a search mission is to find one or more targets which are located somewhere in the domain. One would be concerned if there are conditions where the target might be able to hide in the environment and avoid detection by the agents. We will show that under a reasonable set of assumptions, the agents are guaranteed to visit all cells in the map with non-zero score sufficiently often to drive the cell scores to zero. In other words, they will exhaustively search and cover the entire area of interest. The assumptions are

$$\begin{aligned}
\text{Assumptions:} \quad & h \in (0,1) && (A.1)\\
& \delta \in (0,1] && (A.2)\\
& R_{max} \geq \ \max(L_x, L_y) && (A.3)\\
& Z_k = z_A \ \ \forall k && (A.4)\\
& \hat{x}_w(k+d,\overline{z}) = x_w(k,\overline{z}) \ \ \forall k && (A.5)\\
& x_w(0,\overline{z}) \in [0,1) \ \ \forall \overline{z} && (A.6)\\
& \delta > \beta + \gamma && (A.7)
\end{aligned}$$

The physical meaning of assumption A.1 is to ensure that each agent has a sensor that can be relied upon to some degree. Eliminating the possibility that $h = 0$ ensures that with each sensor measurement, the score of the searched cell decreases. Note that the case of $h = 1$ implies an infinitely reliable sensor which is actually the best scenario. However, this requires rewording several following theorems so it is simply excluded using this set of assumptions. All of the following analysis can apply for the case of $h = 1$ with slight changes.

Recall from Section 5.3.1, the scalars α, β, γ, and δ are in the range of $[0, 1]$. Assumption A.2 ensures that $\delta \neq 0$. Its significance will become clear in Section 5.3.3.

Assumption A.3 effectively specifies a minimum size of the set $\tilde{B}_R$. This states that there must be more than one cell center in $\tilde{B}_R$. Furthermore, this guarantees that $\tilde{B}_R$ includes the cells centers to the North, East, South, and West of the current agent location and ensures that the agent can move in any direction and eventually reach any other cell under an appropriate control law.

Assumptions A.4, A.5, and A.6 deal with the nature of the complete coverage of the map. Complete coverage implies the agents will search each cell in the map sufficiently to drive its score to zero. In a scenario involving complete coverage of the map, the target is located in the last cell that the agents would search or is simply not located in the map at all. These assumptions ensure that the scores of the map are never increasing and can in fact decrease under Eq. 4.4.

Assumption A.7 is crucial for the proof of complete coverage and is explained in Section 5.3.3.

All of these assumptions are reasonable and can be implemented easily. Under these assumptions we will show that the agents operating under this strategy will exhaustively search the map and drive all cell scores towards zero given sufficient time.

Decreasing Scores

As the agents search a cell, it is desired that the score of the cells decrease. We show that the scores of each cell monotonically decrease with each sensor measurement.

Theorem 5.3.1. *Under assumptions A.1, A.4, A.5, and A.6 and the previously described and search strategy, the score of any given occupancy map cell with a score not equal to* 0 *or* 1 *will monotonically decrease towards 0.*

Proof. Recall that the score of the cell is updated via Eq. 4.4 with $Z_k = z_A \ \forall k$. It is trivial to see that if $s_{k-1} = 1$ then $s_k = 1$ and if $s_{k-1} = 0$, then $s_k = 0$. The burden is to now show that $s_k < s_{k-1} \ \forall k$ for the cases where $s_{k-1} \in (0, 1)$. This was already proved in Theorem 4.2.1

It is trivial to show that $s_k > 0 \ \forall k$. This, in conjunction with Eq. 4.9, shows that with enough sensor measurements of $Z_k = z_A$, the score of a given cell will proceed monotonically

towards 0. □

Guaranteed Coverage

Theorem 5.3.1 shows that if a cell is searched a sufficient number of times, its score will decrease towards zero. In order to show that the agents exhaustively search the map, the burden is to show that under the given control law, the agent will visit each cell enough times to decrease the scores to zero.

Theorem 5.3.2. *Under assumptions A.3, A.4, A.5, and the previously described search strategy, if an agent is not currently in the cell containing $\overline{z}_H$, the agent must move to a new cell once the quantity $\alpha x_w(k, \overline{z}_{agt})$ decreases below the value δ.*

Proof. If the agent is not in the cell containing $\overline{z}_H$, then it cannot be at the location $\overline{z}_h$ either since A.3 $\Leftrightarrow |\tilde{B}_R| > 1$ and by definition of $\overline{z}_h$, $\overline{z}_h$ must be closer to $\overline{z}_H$ than the current agent position. This implies that $f_h(\overline{z}_{agt}) = 0$. Similarly, by definition in Eq. 5.11, $f_\chi(\overline{z}_{agt}) = 0$. The maximum possible reward gained by choosing the same cell as the current agent (denoted $\overline{z}_s$) is

$$J_0(\overline{z}_s) = \alpha x_w(k, \overline{z}_s) + \eta\gamma f_d(\overline{z}_s) \tag{5.30}$$

By definition, $\overline{z}_h \in \tilde{B}_R$. It is possible that $x_w(k, \overline{z}_h) = f_\chi(\overline{z}_h) = 0$ (if the score is already zero and the cell is located directly behind the agent) so the minimum possible value of the reward function at this point is

$$J_0(\overline{z}_h) = \eta\gamma f_d(\overline{z}_h) + \delta \tag{5.31}$$

If the control solution from ($\wp_3$) performs correctly, the agent should be located at the point $\overline{z}^\star$ from ($\wp_2$) and $f_d(\overline{z}_s) = 0$. However, the proof can be applied for the case where the agent is not necessarily located on a cell center at all times. From a geometric perspective, the maximum value of $f_d(\overline{z}_s) = (L_x^2 + L_y^2)^{1/2}/R_{max}$ which corresponds to the agent located in the corner of the cell. The distance to any other cell in $\tilde{B}_R$ must be at least that if not

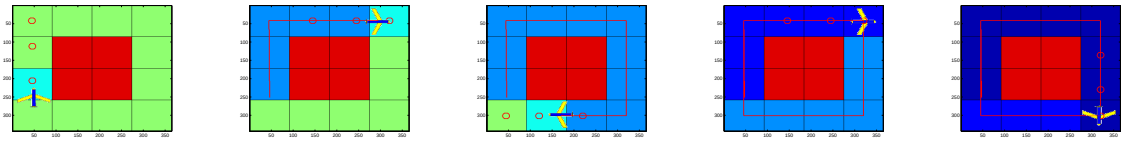
greater, so $f_d(\overline{z}_h) \geq f_d(\overline{z}_s)$. This result, combined with Eq. 5.30 and Eq. 5.31, yields the result that

$$\{\alpha x_w(k, \overline{z}_s) < \delta\} \Rightarrow \{J_0(\overline{z}_h) > J_0(\overline{z}_s)\} \tag{5.32}$$

□

This shows the existence of a point with a higher score than the agent's current cell. This gives a lower bound of the score when the agent must choose to leave the current cell. It is possible that the agent will chose a different cell before $\alpha x_w(k, \overline{z}_s) < \delta$.

Theorems 5.3.1 and 5.3.2 ensure that an agent searching a non-zero score cell will decrease the score consistently and an agent cannot remain at a given cell indefinitely. Combined, these two ensure movement of the agent to different cells but does not guarantee that the map is covered. In fact, there are counter examples where an agent can constantly choose cells of nearly zero score and never search all of the map, thereby ignoring certain cells which have non-zero scores. So far, the previous theorems only required assumptions A.1 - A.6. The example in Figure 5.8 shows that although assumptions A.1- A.6 may be satisfied, this is not sufficient to guarantee exhaustive map coverage.



(a) $t = 0$ (b) $t = 195$ (c) $t = 325$ (d) $t = 520$ (e) $t = 845$

Figure 5.8: Example with $\alpha = 0.15$, $\beta = 0.95$, $\gamma = 0.85$, $\delta = 0.10$ showing a map not being covered due to Assumption A.7 violation.

In this example, the agent continues to fly around the perimeter of the map and never explores the interior. This is due to the fact that the cost of changing heading to turn into the interior of the map is too high relative to the other terms in the cost function. In order to guarantee complete coverage of the map, the restriction of A.7 must be enforced.

Theorem 5.3.3. *Under assumption A.7 and the previously described search strategy, the agent must choose either a cell which has non-zero score, contains $\overline{z}_h$, or both as the solution to $(\wp_2)$.*

Proof. Assume there exists a point $\overline{z}_0$ which has a score of zero and is not $\overline{z}_h$. In this case $x_w(k, \overline{z_0}) = f_h(z_0) = 0$. The maximum value of the reward function at this point is

$$J_0(\overline{z}_0) = \eta(\beta + \gamma) \tag{5.33}$$

The minimum value of the reward function obtained by choosing $\overline{z}_h$ is

$$J_0(\overline{z}_h) = \delta \tag{5.34}$$

Since $\eta \in [0, 1]$, these two results along with the assumption $\delta > \gamma + \beta$ ensures that $J_0(\overline{z}_h) > J_0(\overline{z}_0)$. This guarantees that the agent will not choose a cell if it has zero score and is not $\overline{z}_h$. Therefore, the only other options are to choose a cell which has a non-zero score, contains $\overline{z}_h$, or both. □

Theorem 5.3.4. *Under assumptions A.4, A.5 and the previously described search strategy, if Eq. 5.14 is used to choose $\overline{z}_H$ and the agent chooses $\overline{z}_h$ as the solution to $(\wp_2)$ and $\overline{z}_h \neq \overline{z}_H$, then at the next time step the point $\overline{z}_H$ will remain unchanged unless it is searched by another agent.*

Proof. Consider the distance between $\overline{z}_H$ and $\overline{z}_{agt}$ at a given time step k ($||\overline{z}_H - \overline{z}_{agt}||$). By definition, $\overline{z}_h \neq \overline{z}_H \Leftrightarrow \overline{z}_H \notin \tilde{B}_R$. So if the agent chooses $\overline{z}_h$ as the solution to $(\wp_2)$ and $\overline{z}_h \neq \overline{z}_H$, the agent must move closer to the point $\overline{z}_H$ in the sense that $||\overline{z}_H - \overline{z}_{agt}||$ cannot increase. Since the point $\overline{z}_h$ is not equal to $\overline{z}_H$ then the agent cannot search the cell containing $\overline{z}_H$ and score at $\overline{z}_H$ will not change. Therefore, at the next time step, the same point $\overline{z}_H \in \tilde{B}_{max}$. Furthermore, since the distance between the agent and the same point $\overline{z}_H$ has decreased, it will be chosen again as $\overline{z}_H$ using Eq. 5.14.

Of course, if the point $\overline{z}_H$ is searched first by another agent, its score may decrease and it is possible that at the next step the same point is no longer in $\tilde{B}_{max}$ due to the fact that the score decreased. □

A similar theorem can be stated for the case where $\overline{z}_H$ is chosen using Eq. 5.15.

Theorem 5.3.5. *Under assumptions A.4, A.5 and the previously described search strategy, if Eq. 5.15 is used to choose $\overline{z}_H$ and the agent chooses $\overline{z}_h$ as the solution to $(\wp_2)$ and $\overline{z}_h \neq \overline{z}_H$, then at the next time step the point $\overline{z}_H$ will either remain unchanged or be located in the agent's reachable set, $\tilde{B}_R$ unless it is searched by another agent.*

Proof. Recall that if $\tilde{B}_{max} \bigcap \tilde{B}_R = \emptyset$ (there are no cells with maximum score within the agent's reachable set), then both Eq. 5.14 and Eq. 5.15 choose the same cell center for $\overline{z}_H$, so the proof of Theorem 5.3.4 applies and shows that as long as $\tilde{B}_{max} \bigcap \tilde{B}_R = \emptyset$, the point $\overline{z}_H$ will remain unchanged under these assumptions. The difference between Eq. 5.14 and Eq. 5.15 occurs when $\tilde{B}_{max} \bigcap \tilde{B}_R \neq \emptyset$.

Assume at step k, the $\tilde{B}_{max} \bigcap \tilde{B}_R = \emptyset \Rightarrow \overline{z}_H \notin \tilde{B}_R$. Assuming that the agent chooses $\overline{z}_h$ as the solution to $(\wp_2)$ and $\overline{z}_h \neq \overline{z}_H$, the previous analysis shows that $\overline{z}_H$ does not change. However, at the next step $k+1$, if the same point $\overline{z}_H \in \tilde{B}_R$, then it is possible that Eq. 5.15 will not choose the same point $\overline{z}_H$ again because there might be another member of $\tilde{B}_{max} \bigcap \tilde{B}_R$ which is farther away from $\overline{z}_{agt}$. However, it is guaranteed that $\tilde{B}_{max} \bigcap \tilde{B}_R \neq \emptyset$ since the same point $\overline{z}_H$ from step k is in $\tilde{B}_{max}$ and $\tilde{B}_R$ under these assumptions, thus ensuring that $\overline{z}_H \in \tilde{B}_R$.

Of course, if the point $\overline{z}_H$ is searched first by another agent, its score may decrease and it is possible that at the next step the same point is no longer in $\tilde{B}_{max}$ due to the fact that the score decreased. □

Figure 5.5 can be used as a visual aid for understanding the proofs of both Theorems 5.3.4 and 5.3.5. These theorems can be used to guarantee that the team performs an exhaustive search of the map given the proper conditions.

Theorem 5.3.6. *Under the previously described assumptions and search strategy, $x_w(k, \overline{z}) \rightarrow 0 \ \forall \overline{z} \in B$ (the scores of all cells in the map will approach 0).*

Proof. Theorem 5.3.2 guarantees that an agent cannot remain in a single cell indefinitely and Theorem 5.3.3 ensures that it must choose either a cell of non-zero score, $\overline{z}_h$, or both

as the solution to ($\wp_2$) at any given time step. If the agent chooses a cell of non-zero score, Theorem 5.3.1 ensures that the score of that cell is monotonically decreased towards 0.

If the agent does not choose a cell of non-zero score, the only alternative scenario allowed by Theorem 5.3.3 is that the agent chooses $\overline{z}_h$ and $x_w(k, \overline{z}_h) = 0$. By definition of $\overline{z}_H$, if $x_w(k, \overline{z}_H) = 0$, then the map has been completely covered since all scores are at most 0. Therefore, assuming that the map has not been entirely searched yet, $x_w(k, \overline{z}_h) = 0 \Rightarrow \overline{z}_h \neq \overline{z}_H$.

Theorem 5.3.4 and Theorem 5.3.5 ensure that if $\overline{z}_h \neq \overline{z}_H$, choosing $\overline{z}_h$ as the solution to ($\wp_2$) does not change the value of $\overline{z}_H$ at the next time step or if $\overline{z}_H$ does change, $\overline{z}_H \in \tilde{B}_R$ at the next time step.

At this next time step, the agent once again must choose a cell with non-zero score, $\overline{z}_h$, or both. If the agent continues to choose $\overline{z}_h$, eventually $\overline{z}_H \in \tilde{B}_R$ and at this point, choosing a cell with non-zero score, $\overline{z}_h$, or both guarantees that a cell of non-zero score is chosen. Therefore, the score of some cell will be ensured to decrease and given sufficient time, $x_w(k, \overline{z}) \to 0 \ \ \forall \overline{z} \in B$.

□

Note that the results of Theorem 5.3.6 yield a stronger result than simply ensuring that each cell is visited during the mission. In the specified framework, visiting a cell once may not be enough to guarantee that the target is not located in that cell. In the case where the agent has an unreliable sensor, the amount of information obtained from a single visit to a cell may not decrease the score of that cell sufficiently. Theorem 5.3.6 and its associated proof show that the map will be exhaustively searched for the target using any number of agents in the sense that each cell is visited a sufficient number of times to drive the scores of all cells in the map to zero.

Note that only one agent must be constricted by Assumption A.7 in order to guarantee an exhaustive search by the team. Performance may be increased by relaxing Assumption A.7 for some agents but it cannot be guaranteed that the map will be covered by the agents who do not satisfy Assumption A.7

Exhaustive Searching Results

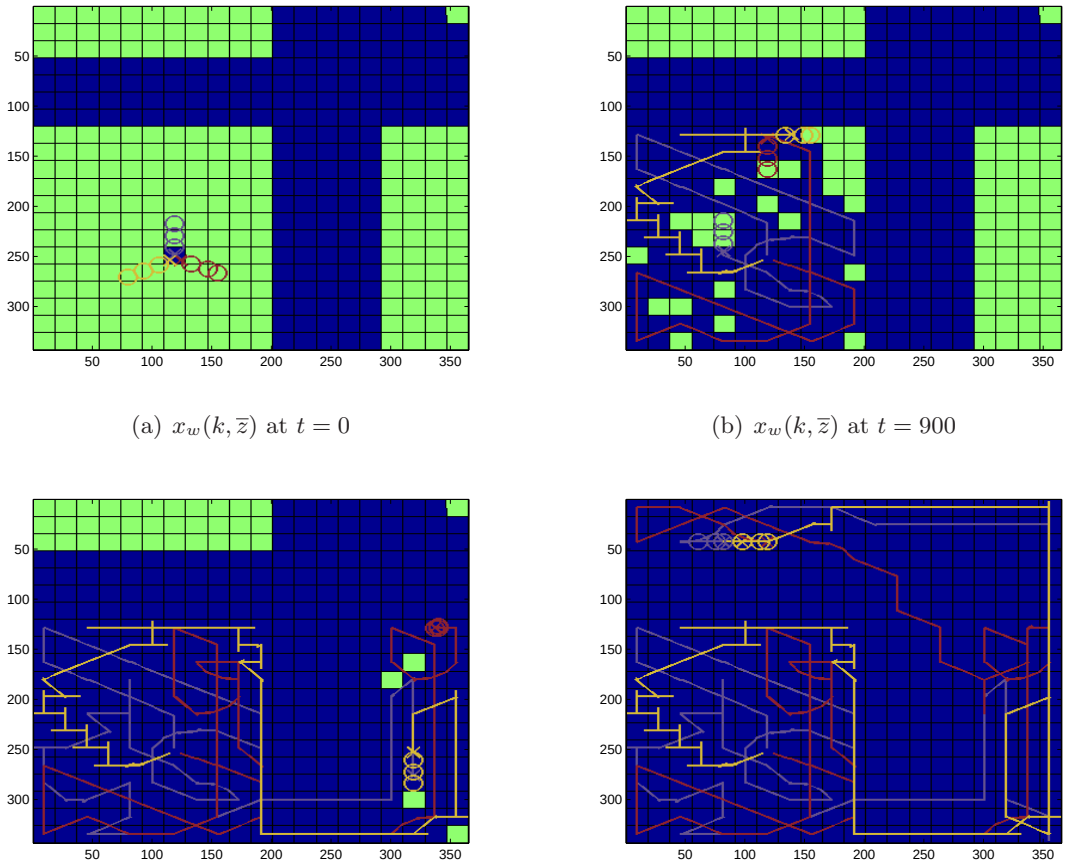
A heterogeneous team of agents can be simulated in different environments using the framework we have described. A heterogeneous team can be modeled by varying parameters such as α, β, γ, and δ. The parameters such as ΔT and d can also be varied but are held constant for these simulations. A table summarizing the parameters used for the different agents in two scenarios are listed below in Table 5.1.

Table 5.1: Parameters of agents in team during search missions ($d = 3$ for all agents).

Scenario	Agent	α	β	γ	δ	h	A.7	$\overline{z}_H$ method
A	1 (purple)	0.75	0.40	0.95	0.15	0.99	No	Eq. 5.14
A	2 (red)	0.90	0.50	0.90	0.05	0.99	No	Eq. 5.14
A	3 (gold)	0.10	0.05	0.25	0.35	0.99	Yes	Eq. 5.14
B	1 (purple)	0.75	0.40	0.95	0.15	0.99	No	Eq. 5.15
B	2 (red)	0.90	0.50	0.90	0.05	0.99	No	Eq. 5.15
B	3 (gold)	0.10	0.05	0.25	0.35	0.99	Yes	Eq. 5.15
C	1 (purple)	0.75	0.40	0.95	0.15	0.25	No	Eq. 5.14
C	2 (red)	0.90	0.50	0.90	0.05	0.35	No	Eq. 5.14
C	3 (gold)	0.10	0.05	0.25	0.35	0.55	Yes	Eq. 5.14
D	1 (purple)	0.75	0.40	0.95	0.15	0.25	No	Eq. 5.15
D	2 (red)	0.90	0.50	0.90	0.05	0.35	No	Eq. 5.15
D	3 (gold)	0.10	0.05	0.25	0.35	0.55	Yes	Eq. 5.15

The first scenario involves three agents equipped with near perfect sensors ($h \approx 1$). This implies that the score of the cell can be reduced to effectively 0 in a single sensor measurement. The results of the simulation are shown in Figure 5.9.

The 'x' shows the location of the agent. The 'o' marks illustrate the agent's path, $\overline{w}^\star$. Recall that these are the waypoints which are determined to be the solution to ($\wp_3$). In this situation, the prediction horizon is $d = 3$, so there are 3 'o' marks for each agent. Recall that the only constraint is that the last location of $\overline{w}^\star$ must be equal to $\overline{z}^\star$. In scenario A, the occupancy map is initialized so that there are 4 distinct regions that need to be searched. These green areas are initialized with score of 0.5. The dark blue areas correspond to cells with scores of 0. Notice that in this situation, R_{max} (the distance the agent can travel in d



(a) $x_w(k, \overline{z})$ at $t = 0$

(b) $x_w(k, \overline{z})$ at $t = 900$

(c) $x_w(k, \overline{z})$ at $t = 2100$

(d) $x_w(k, \overline{z})$ at $t = 3100$

Figure 5.9: Scenario A: 3 agents with near perfect sensors ($h \approx 1$) using Eq. 5.14 to choose $\overline{z}_H$.

steps) is less than the width of the dark blue bands which run North/South and East/West across the entire map. Since the agents are initially located in the lower left green quadrant, at no point does the set $\tilde{B}_R$ include cells in the other three green quadrants as long as the agent remains in the lower left quadrant. In this situation, the behavior of the agents is to search the lower quadrant exhaustively before Theorem 5.3.3 forces the agent to choose the point z_h and therefore begin to migrate to the other quadrants. A simple algorithm such as a gradient climb algorithm would not be able to guarantee coverage in a situation like this. Note that only agent 3 (gold) meets the requirement that $\delta > \beta + \gamma$. Having one such agent

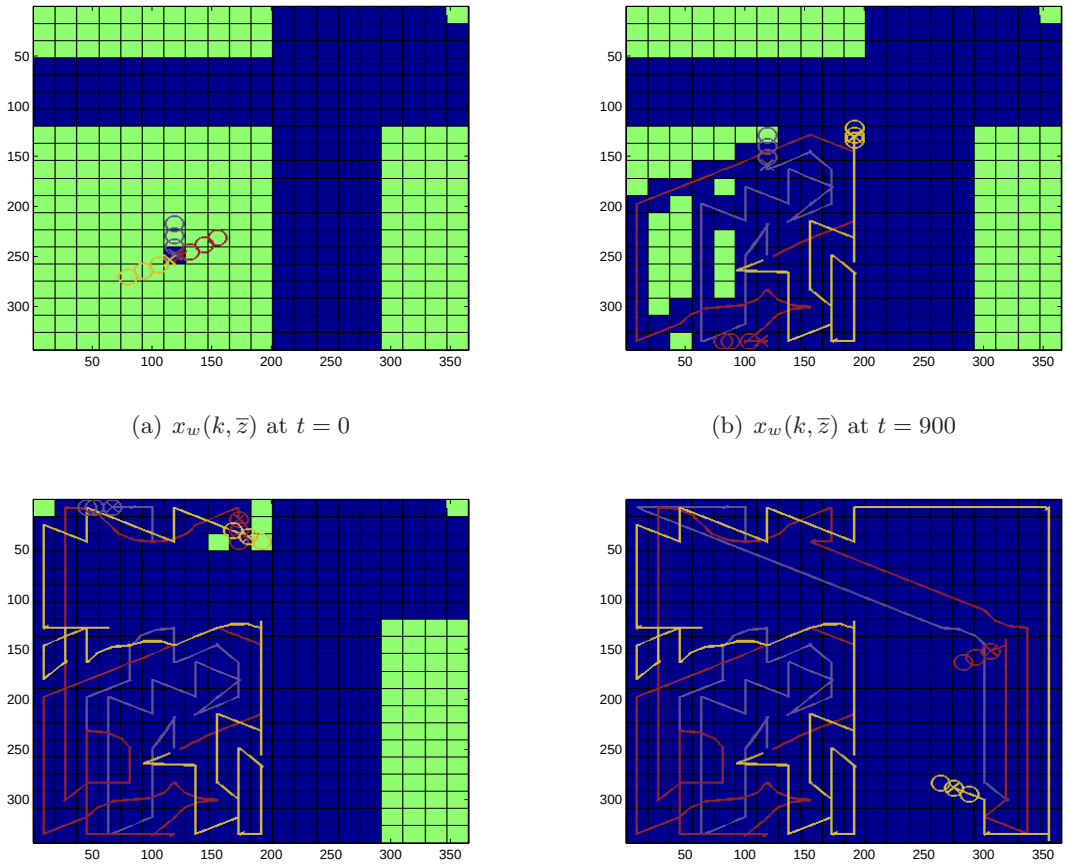
is sufficient to guarantee complete coverage and eventually the team of agents exhaustively searches the entire map.

The effect of using Eq. 5.14 to choose $\overline{z}_H$ is most evident when looking at the trajectories for agent 3. Notice that the map is initialized as all green cells have a value of 0.5 which happens to be the highest score in the map. This means that at any given time step, it is likely that $\overline{z}_H \in \tilde{B}_R$. Furthermore, Eq. 5.14 chooses $\overline{z}_H$ as the point with maximum score which is closest to the agent, so it is likely that the point $\overline{z}_H = \overline{z}_h$ is the cell center directly next to the current agent's location. Depending on the situation, this can be a benefit or detractor. Since $\delta > \beta + \gamma$ for agent 3, it is more likely to choose $\overline{z}^\star = \overline{z}_h$ as the solution to ($\wp_2$). In the event that $\overline{z}^\star$ is the cell directly adjacent to the agent, the agent then has d steps to move only a single cell. This gives the agent a large amount of freedom in choosing how it transitions from the current cell to $\overline{z}^\star$. Ultimately, this transition problem is addressed in ($\wp_3$). Notice that in Figure 5.9, agent 3 does not cover much ground quickly. In fact, the stair stepping and back tracking patterns that are generated by the agent are due to a suboptimal solution of ($\wp_3$), not of ($\wp_2$). Solutions to ($\wp_3$) are addressed in previous work [73] and are covered in Section 5.4.

The alternative is to use Eq. 5.15 to choose $\overline{z}_H$. This is shown with the identical scenario in Figure 5.10.

In this situation, although it is likely that $\overline{z}_H \in \tilde{B}_R$ at each time step, Eq. 5.15 chooses $\overline{z}_H$ as the point in the reachable set which is farthest away from the agent. The result is that agent 3 must move farther from its current location in d steps than what was shown in Figure 5.9. In this situation, the agents search the map faster than previously achieved in Figure 5.9. Once again, it should be reiterated that the fact that agent 3 searches faster is because Eq. 5.15 is used rather than Eq. 5.14. The results are further influenced depending on how ($\wp_3$) is solved. In this case, the solution to ($\wp_3$) does not make use of the fact that if $\overline{z}^\star$ is close to the current agent location (which happens frequently when using Eq. 5.14 to solve ($\wp_2$)), it has a large degree of freedom in choosing its path. For example, it could use longer, more indirect paths which search the neighboring cells before arriving at $\overline{z}^\star$ at the end of d steps.

The next scenario involves three agents equipped with less robust sensors ($h << 1$).



(a) $x_w(k, \overline{z})$ at $t = 0$

(b) $x_w(k, \overline{z})$ at $t = 900$

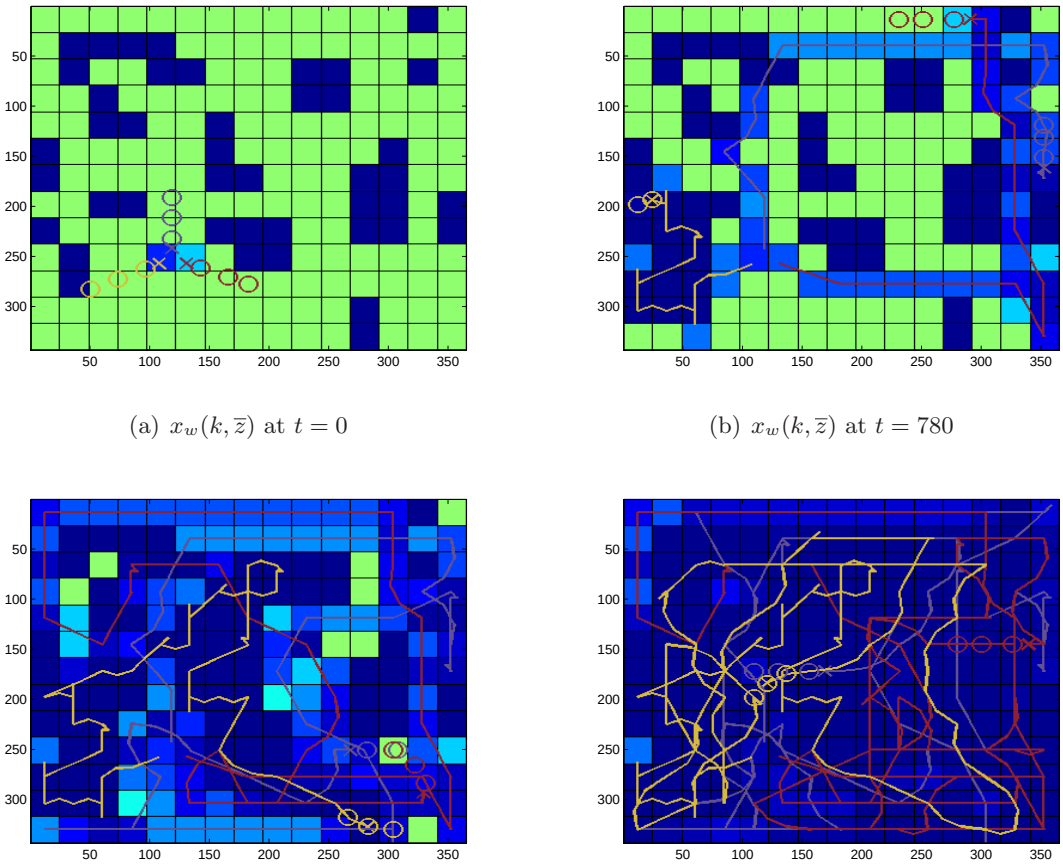
(c) $x_w(k, \overline{z})$ at $t = 1900$

(d) $x_w(k, \overline{z})$ at $t = 2800$

Figure 5.10: Scenario B: 3 agents with near perfect sensors ($h \approx 1$) using Eq. 5.15 to choose $\overline{z}_H$.

This implies that a cell must be visited more than once in order to drive the score to zero. The results of the simulation are shown in Figure 5.11.

This scenario is similar to the first scenario except the agent's sensors are considerably degraded. Due to their decreased reliability, a single visit to a cell only decreases the score to slightly less than the original score. This can be seen since the cell colors change from green to a light blue instead of a dark blue. Multiple visits to the same cell are required to change the color to dark blue (a score of near 0). The algorithm routes the agents to initially search the cells of high probability and then revisits cells as required to drive the



(a) $x_w(k, \overline{z})$ at $t = 0$

(b) $x_w(k, \overline{z})$ at $t = 780$

(c) $x_w(k, \overline{z})$ at $t = 2340$

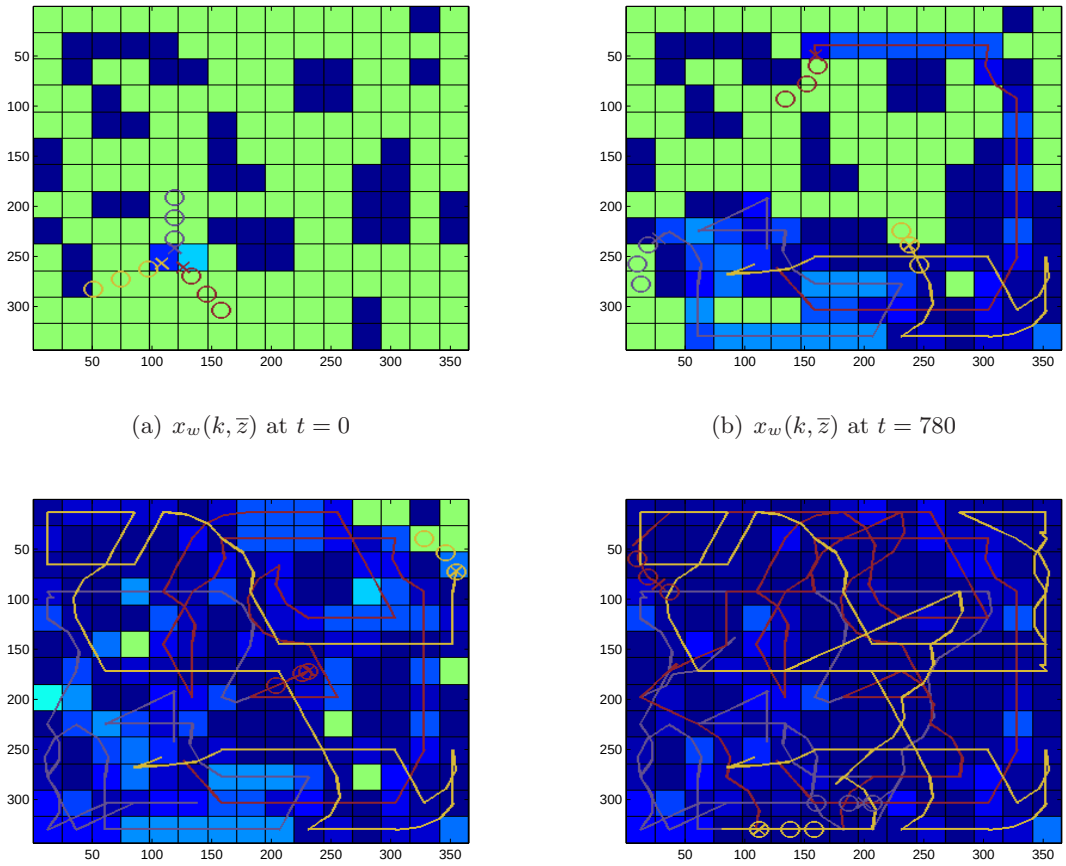
(d) $x_w(k, \overline{z})$ at $t = 5070$

Figure 5.11: Scenario C: 3 agents with typical sensors using Eq. 5.14 to choose $\overline{z}_H$.

scores to zero, guaranteeing an exhaustive search of the map.

5.3.4 ($\wp_2$) *Remarks*

The centralized occupancy based map represents the system's belief of the state of the world at a given time. The map can be propagated forward in time to provide the estimate of the future state of the world at step $k + d$. Each agent then decides which coordinate is the most desirable to search in the next d steps. The team can be comprised of different types of agents with different capabilities. This formulation allows each agent to determine what is desirable for its individual capabilities. Each agent then computes control decisions based



(a) $x_w(k, \overline{z})$ at $t = 0$

(b) $x_w(k, \overline{z})$ at $t = 780$

(c) $x_w(k, \overline{z})$ at $t = 2080$

(d) $x_w(k, \overline{z})$ at $t = 3640$

Figure 5.12: Scenario D: 3 agents with typical sensors using Eq. 5.15 to choose $\overline{z}_H$.

on the predicted future state of the world. Although these actions may not be optimal in a single step, they will benefit the agent in the future.

Although there is no explicit cooperation between agents in the team, the agents are implicitly coupled through the centralized occupancy map. This portion of the modular algorithm remains scalable because each agent does not need to explicitly know about the existence of other agents. Each agent executes the searching algorithm and the emergent behavior is that the team performs a coordinated search. Explicit cooperation can be included in ($\wp_2$) and is discussed in Section 5.5.

The modularity of the algorithm allows the user to tailor specific parts of the algorithm

to address individual agent capabilities. This section focused primarily on the solution for ($\wp_2$) and showed that under the proposed solution, the agents are shown to perform an exhaustive search of the map regardless of the methods used to provide solutions to ($\wp_1$) and ($\wp_3$).

As mentioned previously, it has been observed that using parameters which violate Assumption A.7 may yield paths that cover more ground quicker and in a seemingly more efficient manner. Although an exhaustive search of the map is guaranteed under the current conditions, Assumption A.7 appears to hinder performance of the system. Performance may be increased by tailoring the reward function and finding additional constraints which may be more relaxed than Assumption A.7 and still guarantee coverage. The issue of system performance is tied tightly to the way that ($\wp_3$) is solved. A closer look at the relationships between ($\wp_2$) and ($\wp_3$) is the focus of the following section. An additional challenge is that the target may be moving or evading the searching agents. Therefore, the assumption of a static environment may not be valid.

5.4 ($\wp_3$) ***Path Planning***

The final subproblem involved in the single agent search strategy can be posed as a path planning problem. The algorithm in Section 5.3 details how to find a location for the agent to search in the next d steps. It does not specify how to transition the agent from its current location, $\overline{z}_{agt} = \overline{z}_0$, to the desirable cell location found in ($\wp_2$). The general path planning problem is a well studied field and previous work in this area is detailed in Section 2.2.2. The current section investigates several methods which can be used to plan paths for the agent within the given framework.

5.4.1 Path Planning Using Convex Optimization

The final subproblem, ($\wp_3$), concerns finding feasible waypoints which take the agent from the current location, $\overline{z}_0$, to the desirable cell location found in ($\wp_2$). This may be formed as a convex optimization problem. From an optimization standpoint, the corresponding decision vector $\bar{w} \in \Re^{2 \cdot d}$ is

$$\bar{w} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ \vdots \\ w_{2\cdot d-1} \\ w_{2\cdot d} \end{pmatrix} = \begin{pmatrix} \overline{z}_1(k+1) \\ \overline{z}_2(k+1) \\ \overline{z}_1(k+2) \\ \overline{z}_2(k+2) \\ \vdots \\ \overline{z}_1(k+d) \\ \overline{z}_2(k+d) \end{pmatrix} \tag{5.35}$$

The goal is to move the agent from its current position to the desired position $\overline{z}^{\star}$, so the objective function for ($\wp_3$) is formulated as

$$f_0(\overline{w}) := \sum_{t=1}^{d} ||\bar{z}(k+t) - \bar{z}^{\star}||^2 \tag{5.36}$$

In Eq. 5.36, each coordinate $\bar{z}(k)$ is a waypoint which dictates where the agent should be located at time k. Obviously, the minimum of this function is zero. This corresponds to all the waypoints being placed at $\bar{z}^{\star}$. However this in general would not be feasible. The agent can only travel a distance $r_{max} = \Delta T \cdot V_{max}$ in a single step. So in order for the waypoints to be feasible, it becomes necessary to introduce constraints of the form

$$f_1(\overline{w}) := ||\overline{z}(1) - \overline{z}_0||^2 - r_{max}^2 \leq 0 \tag{5.37}$$

$$f_i(\overline{w}) := ||\overline{z}(i) - \overline{z}(i-1)||^2 - r_{max}^2 \leq 0 \text{ for } i = 2, \ldots, d \tag{5.38}$$

Physically, these constraints enforce that each waypoint must be within a distance r_{max} of the previous waypoint. This ensures that the path generated is a feasible one. The problem can now be formally stated as

$$\begin{aligned} (\wp_3) \text{ minimize } \ & f_0(\overline{w}) \text{ over } \overline{w} \in \Re^{2\cdot d} \\ \text{subject to } & f_i(\overline{w}) \leq 0 \text{ for } i = 1, \ldots, d \end{aligned} \tag{5.39}$$

In order to analyze what type of optimization problem this is, a closer look at the objective function and constraint functions is required. The objective function can be written as

$$f_0(\overline{w}) = \frac{1}{2}\overline{w}^T H \overline{w} + f^T \overline{w} + r \tag{5.40}$$

$$\begin{aligned} \text{where} \quad H &= 2I_{d\times d} \\ f^T &= 2\begin{pmatrix} \bar{z}_1^\star & \bar{z}_2^\star & \bar{z}_1^\star & \bar{z}_2^\star & \dots \end{pmatrix} \\ r &= d||\bar{z}^\star||_2^2 \end{aligned}$$

This is a strictly convex function since it is in a quadratic form and the Hessian is equal to H which is positive definite (all eigenvalues are equal to 2).

The constraint functions can be analyzed in a similar fashion. The first constraint $f_1(\overline{w})$ can be written as

$$f_1(\overline{w}) = \frac{1}{2}\overline{w}^T H_1 \overline{w} + f_1^T \overline{w} + r_1 \tag{5.41}$$

$$\begin{aligned} \text{where} \quad H_1 &= \text{diag}(2I_{2\times 2}, 0_{d-2\times d-2}) \\ f_1^T &= \begin{pmatrix} -2\overline{z}_{1,0} & -2\overline{z}_{2,0} & 0 & \dots & 0 \end{pmatrix} \\ r_1 &= -r_{max}^2 + ||\overline{z}_0||_2^2 \end{aligned}$$

And the constraint functions $f_2(\overline{w})$ through $f_d(\overline{w})$ can be written as

$$f_i(\overline{w}) = \frac{1}{2}\overline{w}^T H_i \overline{w} + f_i^T \overline{w} + r_i \text{ for } i = 2, \dots, d \tag{5.42}$$

$$
\begin{aligned}
\text{where} \quad H_i &= \text{diag}(N_i, \text{diag}(A, M_i)) \\
f_i^T &= \begin{pmatrix} 0 & \dots & 0 \end{pmatrix} \\
r_i &= -r_{max}^2 \\
N_i &= \text{zeros}(2(i-2)) \\
M_i &= \text{zeros}(2d - 4 - 2(i-2)) \\
A &= \begin{pmatrix} 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \end{pmatrix}
\end{aligned}
$$

In Eq. 5.41 and 5.42, $\text{diag}(x, y)$ represents a block diagonal matrix with submatrix x in the upper left corner and submatrix y in the lower right corner. Similarly, $\text{zeros}(p)$ represents a square zero matrix of size $p \times p$. Although the formulation appears complicated, note that only H_i changes with each i. The formulation describes that H_2 is a block diagonal matrix with A in the upper left corner and zeros elsewhere. H_3 is a block diagonal matrix where the submatrix A moves two columns to the right and two rows down. This process of moving the A matrix by 2 rows and columns with each i is described by the N_i and M_i submatrices.

One can now see that the constraint functions $f_i(\overline{w})$ for $i = 1, \dots, d$ are convex functions because they are in quadratic forms and their respective Hessians are all positive semi-definite.

Therefore, $(\wp_3)$ consists of a strictly convex objective function over a convex set, implying that this is a convex programming problem. It can also be shown that it is well posed and the feasible set is non-empty, so a unique optimal solution exists. In a similar fashion to $(\wp_1)$ and $(\wp_2)$, $(\wp_3)$ can be packaged nicely into a system with inputs of the vehicle capabilities and desirable coordinate and then process the optimization problem to obtain an optimal set of waypoints or controls to take the agent to the desired cell. An example of the solution for the example situation (with $d = 10$) is shown in Figure 5.13.

In Figure 5.13, the red 'x' represents the current location of the agent, the green triangle is the desired destination $\bar{z}^\star$ from $(\wp_2)$, and the red circles represent the optimal waypoints $\bar{w}^\star$. Notice that although $d = 10$, there are only 8 visible waypoints. This is because

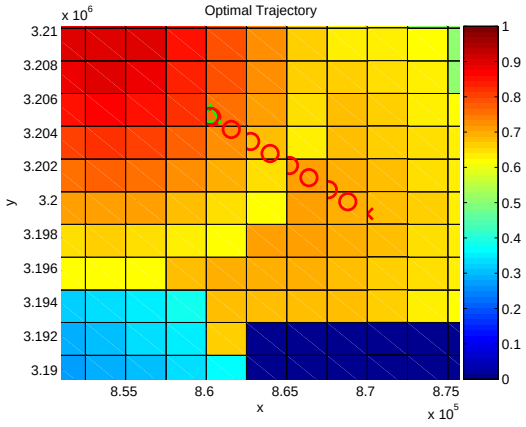


Figure 5.13: Optimal solution $\overline{w}^{\star}$ to $(\wp_3)$ zoomed into area of interest with $d = 10$.

waypoints 8, 9, and 10 are overlapping and all are equal to desired destination ($\overline{z}_8 = \overline{z}_9 = \overline{z}_{10} = \overline{z}^{\star}$). Furthermore, constraints $f_1(\overline{w})$ through $f_7(\overline{w})$ are active. This shows that the formulation of the objective function yields waypoints that place the agent at the optimal solution in the shortest possible time. This is the desired behavior during a searching type application where it is desired that the agent moves to the location $\overline{z}^{\star}$ and verifies the target as soon as possible.

5.4.2 Graph Based Path Planning

The path planning method proposed in Section 5.4.1 provides an efficient and guaranteed solution to the path planning problem for the agent. The drawback is that it does not make use of additional information about the world in its solution. The current section investigates a graph-based path planning technique that also provides a guaranteed feasible path for the agent and combines this graph with the occupancy based map to introduce a notion of environmental optimality.

Generating Primary Paths: The Spatial Network Algorithm

As stated previously, a feasible path for the agent consists of a sequence of waypoints ($x_i \in \Re^2$) where each consecutive waypoint is no more than a distance r_{max} away from

the previous one. The first problem to address involves finding a sequence of points that take the agent from its current point to the end point while obeying the constraints. This problem is referred to as ($\wp_{3,a}$) and can be stated as

Given:	x_0	Agent's current location (starting point)	
	x_d	Desired final location (ending point)	
	r_{max}	Distance agent can travel in one step	
	d	Number of steps	
Assumptions:	$\|\|x_0 - x_d\|\| \leq d \cdot r_{max}$		$(A.1)$
	$r_{max} > 0$		$(A.2)$
	$d \in \mathbb{Z}, d > 1$		$(A.3)$
Goal:	Find a sequence $\{x_i\} : \|\|x_j - x_{j-1}\|\| \leq r_{max}$		
	for $i = 1, ..., d-1$ and $j = 1, ..., d$		

The problem can be visualized as having a beaded necklace where each bead (represented by a waypoint) is connected to the next with a string of length r_{max} or less. One end of the necklace is fastened to x_0 and the other is fixed at x_d. Different paths correspond to different ways the necklace can be stretched while the two end points remain fixed. A stable algorithm to find the sequence $\{x_i\}$ called the Spatial Network Algorithm is presented in this dissertation and in [73]. A flow diagram of the Spatial Network Algorithm is shown in Figure 5.14.

As can be seen from the flowchart, the algorithm works by defining three sets: P_i, Q_i, and R_i. The set P_i makes up a circle (if $x \in \Re^2$) or a sphere (if $x \in \Re^3$) centered around the point x_{i-1} with radius of r_{max}. The set Q_i is also a circle or sphere centered around x_d (the end point) where the radius of these sets change as i changes. Namely, as i increases by 1, the radius decreases by r_{max} until at the step $i = d - 1$, the radius is r_{max}. Finally, the set R_i is simply the intersection of the the sets P_i and Q_i.

The algorithm is a recursive procedure. Choosing the next point x_i requires knowledge of the previous point x_{i-1}. Some parts of the algorithm do not have this coupled behavior.

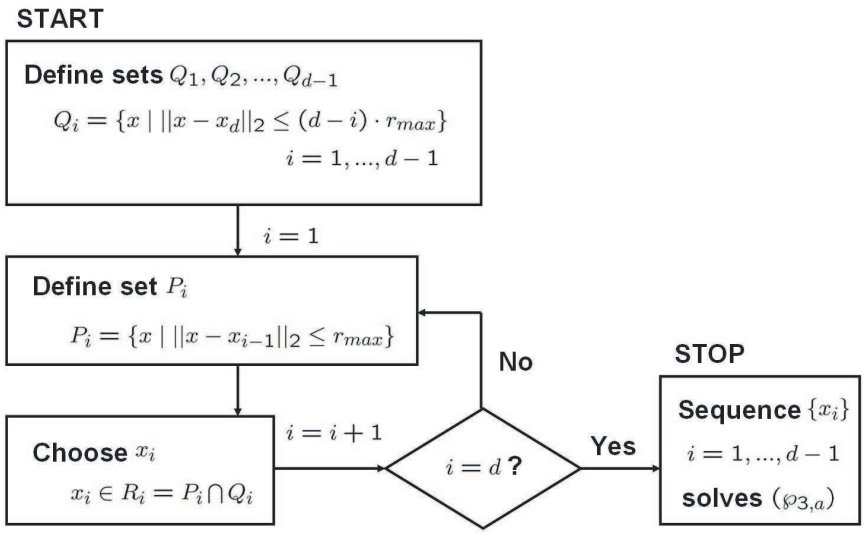


Figure 5.14: Spatial Network Algorithm for solving the $(\wp_3, a)$ problem.

Namely, the sets $Q_1, Q_2, ..., Q_{d-1}$ can be defined a-priori outside of the main algorithmic loop. Inside the main loop, the additional two sets P_i and R_i are computed. The next waypoint x_i is then simply chosen from the set R_i. The natural question becomes, is there a situation where $R_i = \emptyset$? The set R_i is all the points which satisfy the following two inequalities.

$$R_i = \left\{ x \;\middle|\; \begin{array}{l} ||x - x_{i-1}|| \leq r_{max}, \\ ||x - x_d|| \leq (d-i) r_{max} \end{array} \right\} \tag{5.43}$$

Provided that the problem adheres to the assumptions stated previously, we show that $R_i \neq \emptyset$ for $i = 1, ..., d-1$.

Theorem 5.4.1. *Under assumptions A.1, A.2, A.3, and the previously described Spatial Network Algorithm, the set $R_i \neq \emptyset$ for $i = 1, ..., d-1$*

Proof. The proof is started for the case of $i = 1$. For the set R_1 to be non-empty, there must be a point which satisfies both inequalities in Eq.5.43. Consider the point

$$\tilde{x}_i = x_{i-1} + \frac{x_d - x_{i-1}}{||x_d - x_{i-1}||} r_{max} \tag{5.44}$$

At a given step i, the point $\tilde{x}_i$ is generated by taking the vector directed from the previous point x_{i-1} towards x_d and traveling along this vector a distance r_{max} (this would be the point to go to if the agent is headed towards trying to reach x_d as quickly as possible). From construction, $||\tilde{x}_i - x_{i-1}|| = r_{max} \leq r_{max}$ for all i, so the candidate point $\tilde{x}_i \in P_i$.

For the case of $i = 1$, it is now shown that this point satisfies the second inequality (and is therefore in the set Q_1).

$$\begin{aligned} ||\tilde{x}_1 - x_d|| &= ||x_0 + \tfrac{x_d - x_0}{||x_d - x_0||} r_{max} - x_d|| \\ &= ||(1 - \tfrac{r_{max}}{||x_d - x_0||})(x_0 - x_d)|| \end{aligned}$$

From assumption A.1, A.2, and A.3 in the problem statement, $||x_0 - x_d|| \leq d \cdot r_{max}$, $r_{max} > 0$, and $d > 1$ respectively.

$$\begin{aligned} &\leq ||(1 - \tfrac{r_{max}}{d \cdot r_{max}})(x_0 - x_d)|| \\ &= \tfrac{d-1}{d} ||x_0 - x_d|| \\ &\leq \tfrac{d-1}{d}(d \cdot r_{max}) \\ ||\tilde{x}_1 - x_d|| &\leq (d-1) r_{max} \end{aligned}$$

Therefore the candidate point $\tilde{x}_1 \in Q_1$. Since this candidate point satisfies both inequalities, it is in the set R_1 and therefore R_1 is non-empty. This demonstrates that the algorithm cannot fail at step $i = 1$. As the algorithm progresses for $i = 2, ..., d-1$, the set R_i can be shown to always be non-empty using a similar proof as outlined below.

Once again, $R_i \neq \emptyset$ for $i = 2, ..., d-1$ is proven by showing that the candidate point $\tilde{x}_i \in R_i$. As stated previously, $\tilde{x}_i \in P_i \ \forall i$ by construction of $\tilde{x}_i$. Showing that $\tilde{x}_i \in Q_i$ for $i = 2, ..., d-1$ relies on the fact that the algorithm is recursive and therefore the previous

point $x_{i-1} \in R_{i-1}$. More specifically,

$$\{x_{i-1} \in R_{i-1}\} \Rightarrow \{x_{i-1} \in Q_{i-1}\} \Leftrightarrow \{||x_{i-1} - x_d|| \leq (d-i+1)r_{max}\} \tag{5.45}$$

The proof that $\tilde{x}_i \in Q_i$ proceeds in an identical fashion to the proof $\tilde{x}_1 \in Q_1$, except instead of overbounding the right hand side using assumption A.1 from the problem statement, Eq.5.45 is used instead.

$$\begin{aligned} ||\tilde{x}_i - x_d|| &= ||x_{i-1} + \tfrac{x_d - x_{i-1}}{||x_d - x_{i-1}||} r_{max} - x_d|| \\ &\leq ||(1 - \tfrac{r_{max}}{(d-i+1)r_{max}})(x_{i-1} - x_d)|| \\ &\leq \tfrac{d-i}{d-i+1}(d-i+1)r_{max} \\ ||\tilde{x}_i - x_d|| &\leq (d-i)r_{max} \end{aligned} \tag{5.46}$$

This shows that $\tilde{x}_i \in Q_i$. Together with the previous proof that $\tilde{x}_i \in P_i$ shows that $\tilde{x}_i \in R_i$ and therefore, $R_i \neq \emptyset$ □

Showing that the set R_i is non-empty at each step of the process ensures that the algorithm will terminate. It now becomes necessary to show that once the algorithm terminates, the sequence $\{x_i\}$ solves $(\wp_{3,a})$ and is feasible for the agent.

Theorem 5.4.2. *The sequence $\{x_i\}$ obtained via the Spatial Network Algorithm solves $(\wp_{3,a})$ and is feasible for the agent in the sense that $||x_j - x_{j-1}|| \leq r_{max}$ for $j = 1, ..., d$.*

Proof. The point x_i is chosen from the set $R_i = P_i \bigcap Q_i$. Therefore, $x_i \in P_i \Leftrightarrow ||x_i - x_{i-1}|| \leq r_{max}$. This ensures that each subsequent waypoint for the agent is reachable in a single step.

By construction, the set Q_i is the set of all points from which the agent can reach the goal location x_d in $d-i$ steps. Therefore, choosing $x_i \in Q_i$ implies that the agent can reach x_d in $d-i$ steps. This ensures that the agent eventually reaches the point x_d.

This along with Theorem 5.4.1 ensures that the sequence $\{x_i\}$ solves $(\wp_{3,a})$ and is feasible for the agent. □

The algorithm is shown to always generate feasible paths because the set R_i is never empty at any step of the process. Since the algorithm always generates feasible paths, it can now be computationally implemented and evaluated. Flexibility in the algorithm comes from the freedom to choose $x_i \in R_i$. For example, by choosing $x_i = \tilde{x}_i$, a flight path from x_0 to x_d is achieved where the agent always heads towards x_d and moves a distance r_{max} at each step. There exists literature regarding how to intelligently choose these points $x_i \in R_i$ [14]. An example would be to choose $x_i \in R_i$ such x_i maximizes the reward from the occupancy based map (i.e. $x_i \in \arg\max_{x_i \in R_i} x_w(k, x_i)$). In application, the point x_i is chosen differently each time the Spatial Network Algorithm is run. Each time the algorithm is executed, the points are chosen so that they are distributed throughout $\Re^2$ (or $\Re^3$) as much as possible. This yields a set of paths which are a good approximation of the space of all possible paths. An example where $x_i \in R_i$ is chosen to generate longer paths is shown in Figure 5.15.

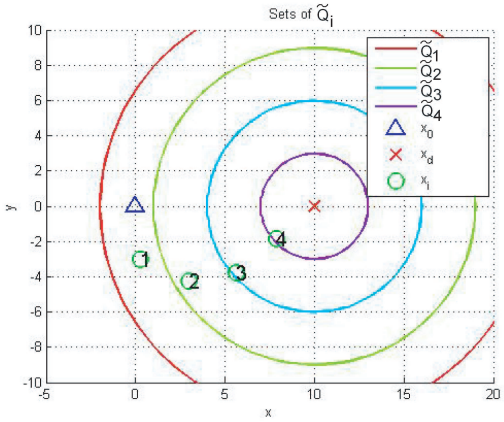


Figure 5.15: A single path generated using the Spatial Network Algorithm by choosing $x_i \in R_i$ where x_i has the minimum possible y value. Situation shown for $d = 5$, $r_{max} = 3$.

The Spatial Network Algorithm generates a sequence $\{x_i\}$ which represents the way-points for a path from start to end point. By executing the Spatial Network Algorithm

multiple times and indexing the corresponding waypoints, the resulting set of paths can be represented as a structured network (directed graph).

Waypoints form the nodes in the network and paths between them are referred to as arcs or edges. Each node embeds information about its spatial location (the coordinates x_i). The Spatial Network Algorithm generates a sequence $\{x_i\}$ for $i = 1, ..., d-1$. Notice that when running the algorithm multiple times, only the coordinates for nodes in the middle of the path will change; the start and end points remain the same for all paths. The nodes associated with start and end points are numbered as nodes i_1 and i_2 respectively. After running the algorithm once, an additional $d-1$ nodes are generated. These are labeled as nodes i_3 through i_{3+d-2}. The second time the algorithm is run, it adds another $d-1$ nodes to the network. These nodes are labeled nodes i_{d+2} through nodes $i_{2 \cdot d}$. This process of sequential numbering continues for as many times as the Spatial Network Algorithm is run.

The arcs are defined in a similar manner. For the first path, the agent goes from node i_1 to node i_3. To represent this motion, an arc j_1 is added that originates at i_1 and terminates at i_3 (so $j_1 \sim (i_1, i_3)$). The agent then goes from i_3 to i_4, so $j_2 \sim (i_3, i_4)$ is added to the network. This continues until finally, the arc $j_d \sim (i_{d+1}, i_2)$ is added. When the Spatial Network Algorithm is run again, this numbering scheme of arcs continues in a similar fashion.

An example for $d = 3$ (number of time steps for agent to reach end point) and $N = 3$ (number of times the Spatial Network Algorithm is run) is shown in Figure 5.16.

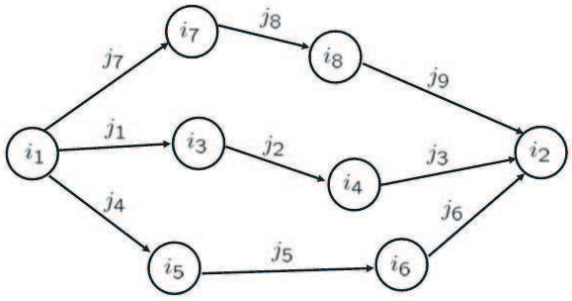


Figure 5.16: An example network with $d = 3$ and $N = 3$.

Each time the algorithm is executed another path is generated from i_1 to i_2. For example, the first path is given by $P_1 : j_1, j_2, j_3$. Figure 5.16 is drawn specifically to draw attention to the fact that each node has coordinate data associated with it and each node represents a waypoint in the path. Simple heuristics can be added to the algorithm to ensure that these paths do not share any nodes. These individual paths are called primary paths. The resulting network has a one-to-one mapping with its incidence matrix. The incidence matrix becomes a useful way to represent the network. The incidence matrix is defined as

$$[E]_{ij} = \begin{cases} +1 & \text{if } i \text{ is initial node of arc } j \\ -1 & \text{if } i \text{ is terminal node of arc } j \\ 0 & \text{in all other cases} \end{cases} \tag{5.47}$$

The benefit of the proposed numbering system is that the network composed of N primary paths may now be represented with the convenient form of Eq. 5.48.

$$E = \begin{pmatrix} J & J & \dots & J \\ K & L & \dots & L \\ L & K & \dots & \vdots \\ \vdots & \vdots & \ddots & L \\ L & \dots & L & K \end{pmatrix} \tag{5.48}$$

$$\text{where} \quad J = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & -1 \end{pmatrix}$$

$$K = \begin{pmatrix} -1 & 1 & 0 & \dots & 0 \\ 0 & -1 & 1 & 0 & \vdots \\ \vdots & 0 & -1 & 1 & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -1 & 1 \end{pmatrix}$$

$$L = \text{zeros}(d-1, d)$$

The incidence matrix is composed of three submatrices. The matrix J is a $2 \times d$ matrix composed of all zeros except for a 1 in the top left entry and a -1 in the bottom right entry. The matrix K is a $d-1 \times d$ matrix. The first row has -1 and +1 as the first two entries and zeros elsewhere. This pattern of a -1 followed by a +1 moves down the diagonal of the matrix. The matrix L is simply a $d-1 \times d$ matrix of zeros. The overall incidence matrix is a matrix of size $2 + N(d-1) \times Nd$ with non-zero entries along the top two rows and in a somewhat block diagonal shape.

Generating Secondary Paths: The Progressive Frontier Algorithm

All of the forward paths in the network are feasible paths for the agent and transition it from its current location to the desired end location in d steps or less. However, the number of feasible paths is exactly equal to the number of times the Spatial Network Algorithm is run. In order to generate another feasible path, the Spatial Network Algorithm must be executed again.

In Figure 5.16, notice that a path from i_1 to i_3 to i_8 to i_2 would also take the agent to the desired final coordinate in d steps. However, an arc from i_3 to i_8 does not exist. To represent a feasible path, an arc must satisfy several requirements. First, the arc must join two nodes that are separated by a distance r_{max} or less. Second, traversal of the arc must take the agent closer (not necessarily in a Euclidean sense) to the desired end state. Third, the complete sequence of arcs must place the agent at the end state in d steps or less. The question becomes, where can arcs be added so that positive paths in the network are still feasible paths for the agent and places the agent at the end state in d steps or less? This problem is referred to as $(\wp_{3,b})$. To solve $(\wp_{3,b})$, the node subsets C_s are defined as follows.

$$C_s = \{i_{2+s+t(d-1)} \text{ for } t = 0, 1, \ldots, N-1\} \text{ for } s = 1, 2, \ldots, d-1 \tag{5.49}$$

Using Figure 5.16 as an example, the sets defined by Eq. 5.49 yield $C_1 = \{i_3, i_5, i_7\}$, $C_2 = \{i_4, i_6, i_8\}$. The set C_1 represents the nodes which are 1 arc away from the initial point (therefore $d-1$ away from the final point). Similarly, the set C_2 represents the nodes that are 2 arcs away from the initial point (therefore $d-2$ arcs away from the final point).

An algorithm for solving $(\wp_{3,b})$ is shown in Figure 5.17. We refer to this algorithm as the Progressive Frontier Algorithm.

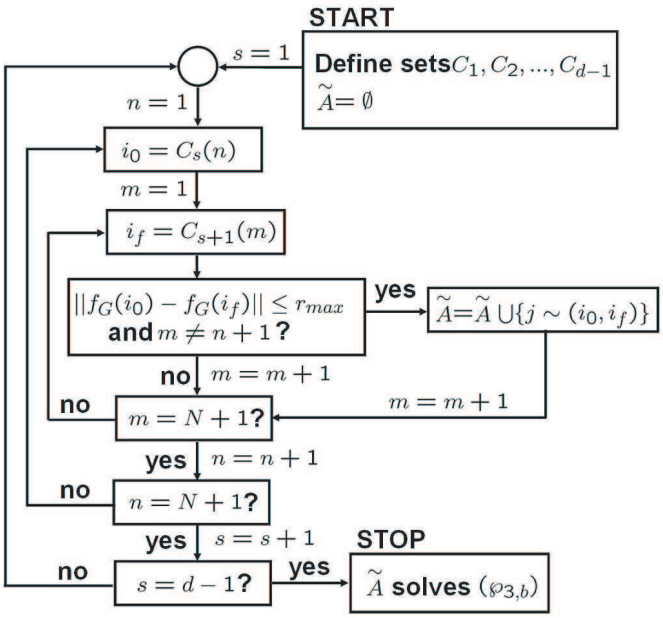


Figure 5.17: Progressive Frontier Algorithm flowchart for solving $(\wp_{3,b})$.

The Progressive Frontier Algorithm can be implemented as a series of three loops. At a given s, the algorithm functions by choosing a node $i_0 \in C_s$ and then comparing it with every node in $i_f \in C_{s+1}$ (except for the next consecutive node because an arc already exists between them). If the distance is less than or equal to r_{max}, an arc from i_0 to i_f is added to the set $\tilde{A}$. Once this is completed for all nodes in C_s, s is incremented by one and the process is repeated.

Using Figure 5.16 as an example, the algorithm first chooses $i_0 = i_3$ and compares the distance between this node and nodes i_6 and i_8 (but not with i_4 because $j \sim (i_3, i_4)$ already exists). Whenever the distance is less than or equal to r_{max} an arc is added to $\tilde{A}$. Then i_0 is changed to i_5 and its distance to nodes i_4 and i_8 is checked. Finally, i_0 is changed to i_7 and its distance to nodes i_4 and i_6 is checked.

The Progressive Frontier Algorithm performs the logical test of $||f_G(i_0) - f_G(i_f)|| \leq r_{max}$

exactly $M = (d-2)N^2 + (2-d)N$ times. If a combinatorial approach is taken and the distance between a given node is compared with every other intermediate node in the network, the logical test must be performed exactly $M' = d(d-2)N^2 + N^2 + 3(d-1)N + 2$ times. Assuming that $N >> d$, the ratio of the two quantities is approximately $M'/M \approx d$. The advantage of this approach instead of a combinatorial approach is immediately evident by the significant computational savings.

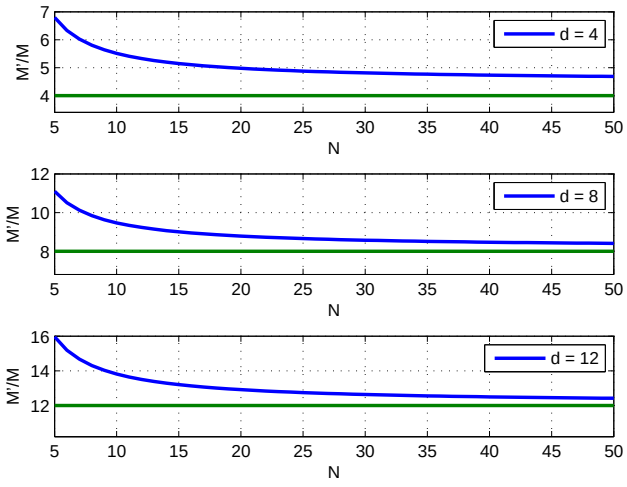


Figure 5.18: Ratio of M'/M approaching d for d =4, 8, and 12.

When the Progressive Frontier Algorithm terminates, the network is updated by adding the new arcs, $\tilde{A}$, to the current arc set, A, to obtain a new set of arcs, A'.

$$A' = A \bigcup \tilde{A} \tag{5.50}$$

The resulting graph preserves the feature that all forward paths through the graph place the agent at the desired location in d steps.

An example of the Progressive Frontier Algorithm run on a network with $d = 5$, $N = 3$ is shown in Figure 5.19. The original, primary paths are shown as black arcs. The algorithm is run with $r_{max} = 1.5$ and the new arcs which make up $\tilde{A}$ are shown in green.

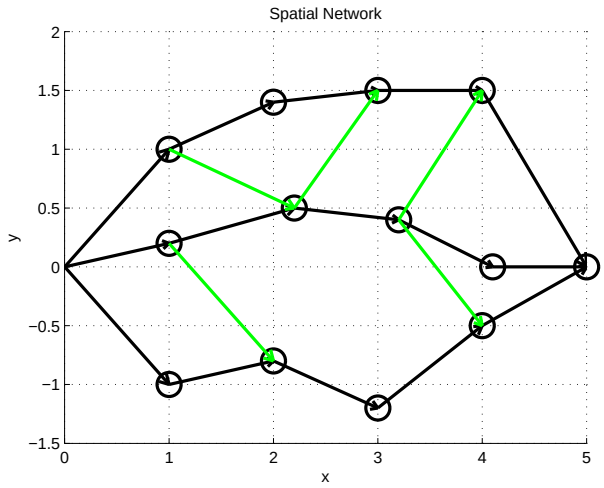


Figure 5.19: Arcs added for $d = 5$, $N = 3$, and $r_{max} = 1.5$.

Network Optimization

Recall that $(\wp_{3,a})$ and $(\wp_{3,b})$ only concerned feasibility of paths. The remaining goal is to find an optimal path among all the generated feasible paths from node i_1 to node i_2. This problem is referred to as $(\wp_{3,c})$.

To solve problem $(\wp_{3,c})$, the notion of span intervals are introduced. The span interval is a a nonempty real interval $D(j)$ assigned to each arc $j \in A'$ [98].

$$D(j) = [d^-(j), d^+(j)] \tag{5.51}$$

The value $d^-(j)$ is the lower span interval and in the context of $(\wp_{3,c})$, this value represents the cost of traversing the arc j in the reverse direction. Similarly, $d^+(j)$ is the upper span interval and represents the cost of traversing the arc j in the forward direction. Information about the environment can be embedded into the network by combining the location of the nodes with the scores of the occupancy based map. One possible mapping of $d^-(j)$ and $d^+(j)$ is

$$\begin{aligned} d^-(j) &= -\infty \\ d^+(j) &= 1 - x_w(k, f_G(i')) \end{aligned} \quad \forall j \sim (i, i') \in A' \tag{5.52}$$

The lower span intervals are set to $-\infty$ for all arcs. The upper span intervals are set using information from the occupancy based map. For the arc $j \sim (i, i')$, the terminal node's coordinates are represented by $f_G(i')$. The function $x_w()$ then maps these coordinates to the corresponding score in the occupancy based map. Since the range of $x_w()$ is $[0, 1]$, the upper span interval ranges from 0 (when the terminal node is located in a region with score equal to 1) to +1 (when the terminal node is located in a region with score equal to 0).

An alternative method to assign the span intervals is to use

$$\begin{aligned} d^-(j) &= -\infty \\ d^+(j) &= \frac{1}{f_M(x_w(k, \overline{z}), j)} \end{aligned} \quad \forall j \sim (i, i') \in A' \tag{5.53}$$

This uses the same lower span interval of $-\infty$ for all arcs. The upper span interval is similar to the previously described method except instead of using the score of the occupancy map cell located at the terminal node as was done in Eq. 5.52, the function $f_M()$ instead returns the minimum occupancy map score between the initial and terminal node of the arc j. This is done using a modified version of Bresenham's line algorithm [23].

A third alternative method to assign span intervals is to use

$$\begin{aligned} d^-(j) &= -\infty \\ d^+(j) &= \nu - f_C(x_w(k, \overline{z}), j) \end{aligned} \quad \forall j \sim (i, i') \in A' \tag{5.54}$$

Once again, the lower span interval is set to $-\infty$ for all arcs. The function $f_C()$ returns the cumulative score of all the cells which the arc j covers. The parameter ν is defined as $\nu = \max\ f_C(x_w(k, \overline{z}, j))$ for all j. In practice, this term is not needed since its existence only offsets the cost function in Eq. 5.55 by a constant. Its purpose is to ensure that $d^+(j) \geq 0$ since this is the traditional form of an upper span interval.

Each of the methods of assigning span intervals using either Eq. 5.52, Eq. 5.53, or Eq. 5.54 have scenarios where they are valid and yield different behavior.

Using Eq. 5.52, an arc is given an upper span interval of +1 (the maximum value under

this assignment function) if the terminating node is located in a region of zero score and an upper span interval of 0 if the terminating node is located in a region of +1 score. Therefore, an arc that passes through but does not terminate in a region of zero score would receive an upper span interval of less than the maximum possible value of +1. In some situations, this may be unacceptable because the agent will intersect with a hard obstacle in the environment. An example illustrating the differences in agent behavior is shown in Figure 5.20. In this example, the red arc has its terminal node in an occupancy map cell with score of 0.75. Eq. 5.52 would assign this arc an upper span interval of $d^+(j) = 1 - 0.75 = 0.25$. Since one can physically interpret the upper span interval as the cost of traversing this arc in the forward direction, this arc will be seen as somewhat desirable to traverse when an optimal path through the graph is found via the Min Path algorithm [98]. The problem arises if low occupancy map scores are meant to represent obstacles or areas to be avoided. In this case, Eq. 5.52 does not account for the fact that in order to reach the terminal node with score of 0.75, it must first cross a series of cells with scores of 0. If these represented hard obstacles, this path would not be feasible. On the other hand, Eq. 5.53 would recognize this fact and set the span intervals of the red arc to $+\infty$. However, the upper span intervals of the two magenta arcs would be set to +2. Therefore, using Eq. 5.53, it would be more desirable to traverse the two magenta arcs rather than travel the shorter distance (in Euclidean space) of the single red arc. This method ensures that the upper span interval will be infinite if any part of the arc passes through a region of zero score. This significantly increases computational complexity but guarantees that all paths are feasible in the sense that there are no intersections with hard obstacles.

Finally, using Eq. 5.54 to assign the span intervals yields yet another type of behavior. As the agent traverses a given arc j, it will search the cells covered by this arc. Therefore, the cumulative score of the cells covered by the arc j provides a measure of the amount of information to be gained by traversing this arc. Note that this is not equivalent to the amount of reduction in score that will result from traversing this arc due to the fact that $f_C()$ returns the linear combination of all the cell scores whereas the score reduction equation given by Eq. 4.4 is non-linear. Since $f_C()$ returns the amount of information to be gained by

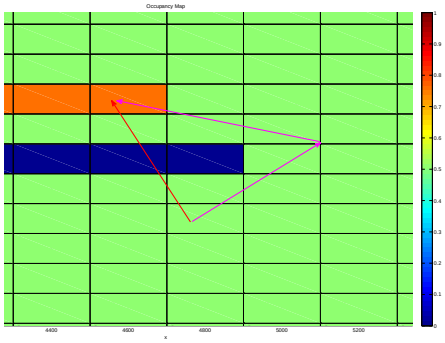


Figure 5.20: Example showing where differences in Eq. 5.52 and Eq. 5.53 affect span interval.

traversing the arc j, when using Eq. 5.54, $d^+(j)$ is minimal when the amount of information gained by traversing arc j is maximal, and vice versa. Note that this method cannot enforce the avoidance of hard obstacles (behavior which is inherent when using Eq. 5.53).

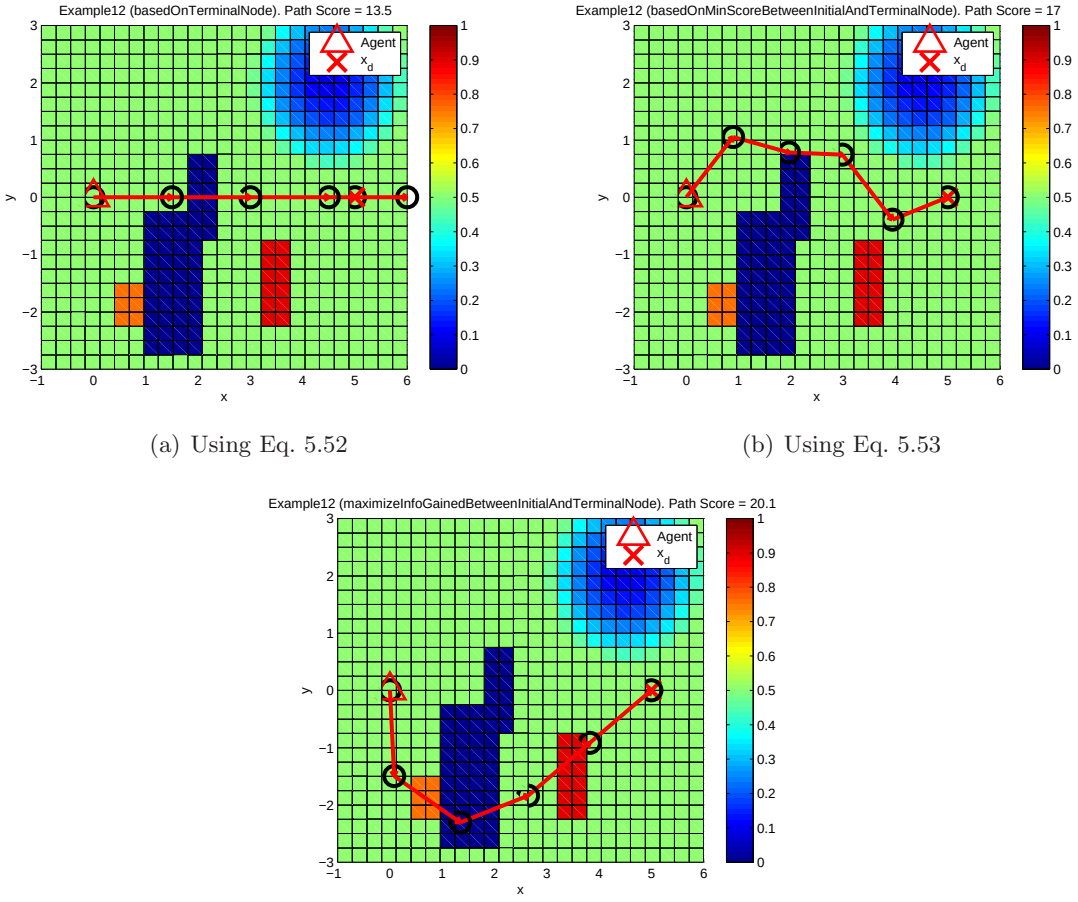
The same scenario using these different span assignment functions is shown in Figure 5.21.

One final detail must be addressed in order to formulate the $(\wp_{3,c})$ as a network optimization problem. This concerns the initial potential u_0 that assigns a scalar value to each node $i \in I$. The differential induced by this potential must be feasible with respect to the span intervals. In other words, the differential $v_0 = -u_0 E$ must be in the span interval $D(j)$ for all $j \in A$. Note that since $0 \in D(j)\ \forall j \in A$, the initial potential can always be taken as $u_0 = 0\ \forall i \in I$ [98].

With the span intervals and initial potential set, $(\wp_{3,c})$ is now in the form of a network optimization problem. A path P is a signed, ordered set of arcs that can be further decomposed into sets P^+ and P^- (arcs which are traversed in the positive and negative direction, respectively). The cost function below evaluates the cost of traversing a certain path P.

$$d^+(P) = \sum_{j \in P^+} d^+(j) - \sum_{j \in P^-} d^-(j) \tag{5.55}$$

The well known Min Path Problem consists of minimizing $d^+(P)$ over all paths $P : N^+ \rightarrow$



(a) Using Eq. 5.52

(b) Using Eq. 5.53

(c) Using Eq. 5.54

Figure 5.21: Effects of choosing different functions to assign span intervals.

N^-. In the context of this path planning algorithm, $N^+ = i_1$ and $N^- = i_2$. The Min Path Problem is solved efficiently using the well known Min Path/Max Tension algorithm [98] or the Bellman-Ford algorithm [15]. With some problem reformulation, Dijkstra's Method may also be used to further increase computational efficiency [17]. Details for the Min Path/Max Tension algorithm are included in Appendix B.

Assuming that assumption $(A.1)$ is satisfied in $(\wp_{3,a})$, a forward path $P : N^+ \rightarrow N^-$ must exist. By setting $d^-(j) = -\infty \; \forall j$, the resulting cost function has the behavior that $\{P^- \neq \emptyset\} \Rightarrow \{d^+(P) = \infty\}$. Therefore, this formulation ensures that only forward paths can solve the min path problem. In some situations, it may be optimal to traverse an arc

in a backwards direction. However, if this is done, it is not possible to guarantee that the path will take the agent to the desired final position in d steps or less. Furthermore, the careful selection of $d^+(j)$ can be used to tailor the solution as described previously. Applying the Min Path/Max Tension algorithm to the network constructed for the marine situation depicted in Figure 4.2 yields an optimal path P^* shown in Figure 5.23 in red.

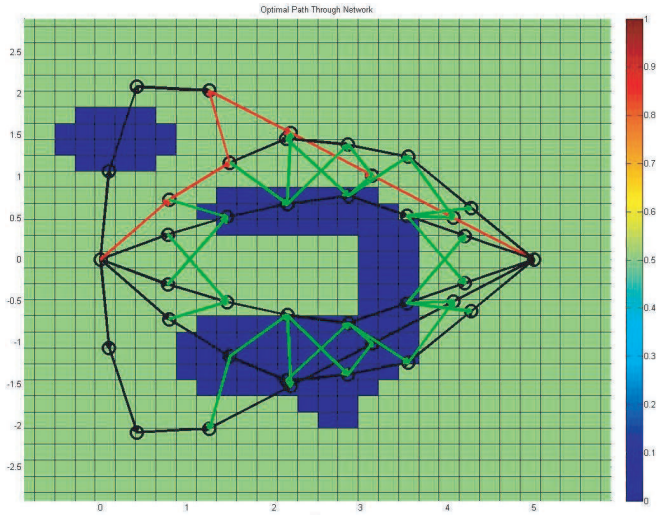


Figure 5.22: Optimal path through environment with $d = 7$, $r_{max} = 1.06$, $N = 6$. Optimal path P^* shown in red.

Figure 5.22 is generated using the methods described in the previous sections. First, the Spatial Network Algorithm is executed six times to generate the six primary paths which are shown in black. Then, the Progressive Frontier Algorithm is run to solve $(\wp_{3,b})$ which adds the green arcs. The span intervals of each arc (both black and green arcs) are calculated by combining the network with the occupancy based map using Eq. 5.52. In this example, the islands that the agent must navigate around represent hard obstacles which must be avoided. Therefore, the occupancy based map cells corresponding to the islands are given a score of zero (the dark blue regions). Once the span intervals are set, the initial potential is set to zero for all nodes and the Min Path/Max Tension algorithm is run to generate the optimal path, which is shown in red. The agent successfully plans a path that circumnavigates the islands on its way to the final destination. Notice that this problem

is one of simply finding a feasible path since all the scores of each cell are either 0 or 0.5. Any of the feasible paths yield the exact same score under the cost function of Eq. 5.55. This is why the optimal path appears to go in the backwards direction from waypoint 2 to waypoint 3. Despite this, the agent is still placed at the desired end goal in d steps or less. This issue can be addressed by incorporating the concept of Euclidean distance into the cost function (Eq.5.55) by changing the way the span intervals are defined in Eq. 5.52. Namely, adding a term of $+\alpha||f_G(i) - f_G(i')||$ to the definition of $d^+(j)$ where α is a parameter to trade off between occupancy map score and distance traveled.

Note that the middle island has a box canyon shape. Many methods that plan paths using a receding horizon type algorithm may fail with this geometry. The agent enters into the canyon and does not realize until it is too late that it has effectively entered a dead-end. This algorithm avoids this pitfall by planning waypoints over the entire path.

Path planning in the urban environment depicted in Figure 4.3 is computed using the same methods. The optimal path through this environment is shown in Figure 5.23.

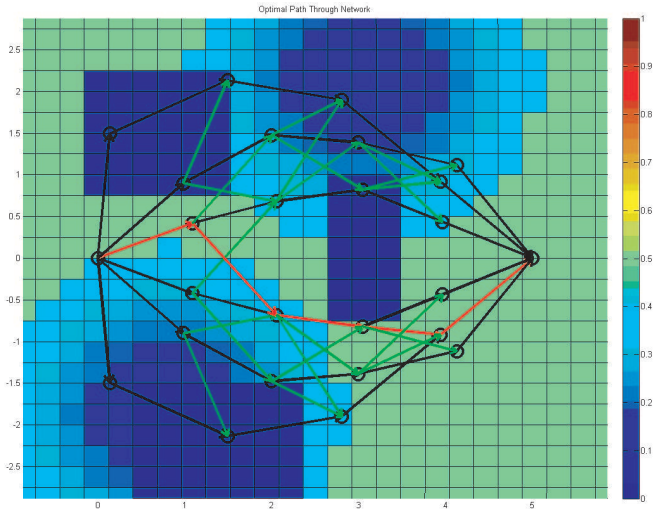


Figure 5.23: Optimal path through environment with $d = 5$, $r_{max} = 1.5$, $N = 6$. Optimal path P^* shown in red.

The main difference between this example and the marine example is that this environment contains soft obstacles (the radar stations). Therefore, in addition to finding feasible

paths, an optimal path can be selected. As can be seen, the agent's first waypoint avoids the areas of low score but the second waypoint must enter the area so that the third waypoint can avoid the region with zero score. The fourth waypoint also avoids the obstacle and the final, fifth waypoint brings the agent to the desired final position.

5.4.3 ($\wp_3$) Remarks

The path planning methods described in this section generate paths that are parameterized in terms of discrete waypoints. A single path is made up of d waypoints. Each of these waypoints is a coordinate $x_i \in \Re^2$. Furthermore, these waypoints are time stamped. There are specific times when the agent must reach a specific waypoint. The agent's capabilities are taken into account when these waypoints and arrival times are generated and therefore, each path is feasible for the agent.

This section presented a computationally efficient approach to obtain feasible paths through complex environments. The approach formulates path planning as a network optimization problem. A network of feasible paths is generated first. Next, the network is combined with occupancy based maps to embed information about the environment. The well known Min Path Algorithm generates optimal paths through the network. Although these algorithms are not able to claim a bounded error from a true optimal path, the network formulation ensures that a quasi-optimal path avoids hard obstacles and reaches the goal in the required number of discrete steps.

Although the examples presented were two dimensional, the algorithms and proofs presented can be directly extended to three dimensions. This may have applications in path planning for underwater vehicles or unmanned aerial systems which can maneuver in three dimensions.

5.5 Modifications for Explicit Cooperation

The search strategy described in Section 5.1 and detailed in Sections 5.2 through 5.4 was derived as a single agent search strategy, each agent in the team follows this same policy without information or explicit knowledge of other members of the team. This section

investigates some modifications to the single agent search strategy to allow for explicit cooperation between agents in the team. This has the potential to increase performance and change the behavior of the system although it requires additional complexity and computational resources and may diminish the scalability of the algorithm and possibly make it impractical for large numbers of agents.

5.5.1 ($\wp_2$) with Explicit Cooperation

Recall that solving ($\wp_2$) involved picking a desirable location for the agent to search within the agent's reachable set, B_R. Without explicit cooperation, there are scenarios where two or more agents might choose the same location $\overline{z}^\star$. This is undesirable for several reasons. One is that the agents must now have some type of reactive collision avoidance [63], [83] since collision free trajectories are not guaranteed at the strategic level. Furthermore, the performance of the system may suffer if there are multiple agents searching the same cells. One method to alleviate these problems is to partition the search space. In essence, this is a divide and conquer approach. The search space can be partitioned using a Voronoi diagram which is a tessellation of a Euclidean space and is described in a purely mathematical sense in Appendix D. This section investigates applying them to ($\wp_2$). In this sense, the location of the agents are considered to be generators for the Voronoi polygons

$$P = \{p_1, p_2, ..., p_n\} = \{\overline{z}_{agt_1}, \overline{z}_{agt_2}, ..., \overline{z}_{agt_n}, \} \tag{5.56}$$

The Voronoi diagram embeds information about an agent's position relative to the other agents. Each agent now has an influence on the diagram and in turn each agent will have an influence on the algorithm. The degradation of algorithm scalability is immediately visible when employing this type of cooperation as the Voronoi diagram requires at least $O(n\ log\ n)$ at best [86] where n is the number of generators (or agents). Previously, the single agent search strategy scaled linearly with n.

Redefining $\overline{z}_{h_i}$ with Voronoi Diagrams

The point in $\tilde{B}_{max}$ that is closest to agent i is given as

$$\overline{z}_{H_i} \in \underset{\overline{z} \in \tilde{B}_{max}}{\arg\,\text{minimize}} ||\overline{z} - \overline{z}_{agt_i}|| \tag{5.57}$$

With $\overline{z}_{H_i}$ defined, the distance between agent i and $\overline{z}_{H_i}$ is given by

$$d_i = ||\overline{z}_{H_i} - \overline{z}_{agt_i}|| \tag{5.58}$$

An example of these distances and locations is shown in Figure 5.24. In this figure, the black dots represent $\tilde{B}$ (the cell centers of the occupancy map). The black dots circled in purple represent $\overline{z} \in \tilde{B}_{max}$. These are the cell centers that correspond to cells which have the highest score in the map. As defined previously, locations inside the dashed red circles represent B_{R_i} (agent i's reachable set). The black dots within the dashed red circles represent $\tilde{B}_{R_i}$ (the discrete approximation of agent i's reachable set).

The distance d_i is computed by checking the distance between agent i and each $\overline{z} \in \tilde{B}$ with the minimum value assigned as d_i. The corresponding location $\overline{z} \in \tilde{B}$ which yields d_i for agent i is denoted as $\overline{z}_{H_i}$.

Note that $\overline{z}_{H_i}$ may not be in the agent's reachable set (either the compact set B_{R_i} or its discrete approximation $\tilde{B}_{R_i}$). In other words, it is possible that $\overline{z}_{H_i} \notin B_{R_i}$ and $\overline{z}_{H_i} \notin \tilde{B}_{R_i}$. This is the case with both agent 1 and agent 2 but not agent 3 in Figure 5.24.

The Voronoi edges are shown as the solid black lines. The Voronoi polygon associated with each agent is enclosed by the solid black lines. Note that it is possible that $\overline{z}_{H_i} \notin V(\overline{z}_{agt_i})$. In Figure 5.24, this is the case for agent 3 where the point $\overline{z}_{H_3}$ is not in agent 3's Voronoi polygon. It is also possible that $\overline{z}_{H_i} = \overline{z}_{H_j}$ for some $i \neq j$ as shown with agent 2 and agent 3 in Figure 5.24.

In general, $a \notin A$ does not imply that $a \notin B$ if $A \subset B$. However, in the case of $\overline{z}_{H_i}$ it can be shown that $\overline{z}_{H_i} \notin \tilde{B}_R \Rightarrow \overline{z}_{H_i} \notin B_R$

Theorem 5.5.1. *If $\overline{z}_{H_i}$ is computed using Eq. 5.57, $\overline{z}_{H_i} \notin \tilde{B}_R \Rightarrow \overline{z}_{H_i} \notin B_R$.*

Proof. Using the definitions of B and $\tilde{B}$ from Section 4.1.1 and the definitions of B_{max}, B_{R_i}, $\tilde{B}_{R_i}$ from Section 5.3.1, the relationships of the sets can be drawn as shown in Figure 5.25.

From Eq. 5.57 it can be seen that $\overline{z}_{H_i} \in \tilde{B}_{max}$. $\tilde{B}_{max}$ is denoted by vertical lines in

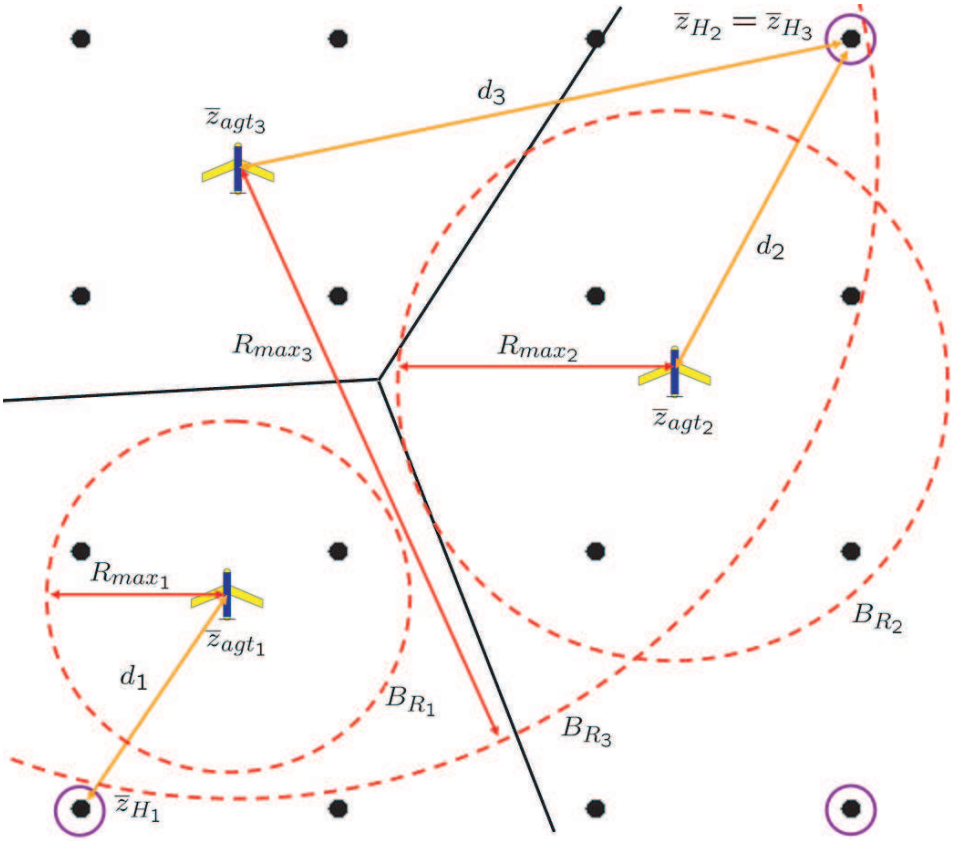


Figure 5.24: Example with 3 agents showing d_i and $\overline{z}_{H_i}$.

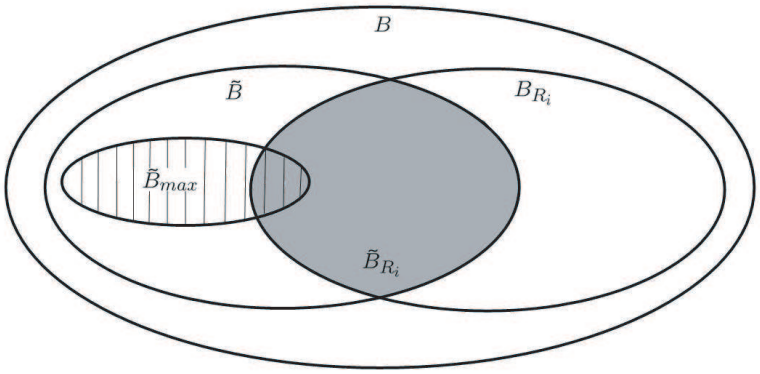


Figure 5.25: Relationships between sets B, $\tilde{B}$, $\tilde{B}_{max}$, B_{R_i}, and $\tilde{B}_{R_i}$.

Figure 5.25. The set $\tilde{B}_{R_i}$ is shown as the grey shaded area in Figure 5.25. Therefore, $\overline{z}_{H_i} \in \tilde{B}_{max}$ and $\overline{z}_{H_i} \notin \tilde{B}_R$ is only satisfied if $\overline{z}_{H_i}$ is in the subset of $\tilde{B}_{max}$ which is marked with the vertical lines with no grey shading. The intersection of this set and B_{R_i} is empty since the sets $\tilde{B}_{max}$ and $\tilde{B}_{R_i}$ are closed. Therefore, $\overline{z}_{H_i} \notin \tilde{B}_R \Rightarrow \overline{z}_{H_i} \notin B_R$ □

Most of the following analysis requires identifying the agent which is closest to the set $\tilde{B}_{max}$. The index of the agent closest to the set $\tilde{B}_{max}$ is denoted as I. The index I is given by (recall that $I_n = \{1, 2, ..., n\}$)

$$I \in \arg\underset{i \in I_n}{\text{minimize}}\ d_i \tag{5.59}$$

Therefore, $\overline{z}_{H_I}$ denotes the location of the point in $\tilde{B}_{max}$ that is closest to any agent.

As stated previously, in general it is possible that $\overline{z}_{H_i} \notin V(\overline{z}_{agt_i})$. However it is true that $\overline{z}_{H_I} \in V(\overline{z}_{agt_I})$ always.

Theorem 5.5.2. *Given any Voronoi diagram* $\overline{V} = \{V(\overline{z}_{agt_1}), ..., V(\overline{z}_{agt_n})\}$, $\overline{z}_{H_I} \in V(\overline{z}_{agt_I})$.

Proof. As stated previously, $\overline{z}_{H_I}$ denotes the location of the point in $\tilde{B}_{max}$ which is closest to any agent. Denote this agent's index as I and its location as $\overline{z}_{agt_I}$.

Note that the definition of the Voronoi polygon generated by agent I is given as $V(\overline{z}_{agt_I}) = \{\overline{z} : ||\overline{z} - \overline{z}_{agt_I}|| \leq ||\overline{z} - \overline{z}_{agt_i}|| \text{ for } i \in I_n, i \neq I\}$ (Eq. D.1). In order for $\overline{z}_{H_I} \in V(\overline{z}_{agt_I})$, the following inequality must be satisfied

$$||\overline{z}_{H_I} - \overline{z}_{agt_I}|| \leq ||\overline{z}_{H_I} - \overline{z}_{agt_i}|| \text{ for } i \in I_n, i \neq I \tag{5.60}$$

By definition, $d_I \leq d_i$ for $i \in I_n$. Therefore, $\overline{z}_{H_I}$ has the property that

$$||\overline{z}_{H_I} - \overline{z}_{agt_I}|| \leq ||\overline{z}_{H_i} - \overline{z}_{agt_i}|| \text{ for } i \in I_n \tag{5.61}$$

From Eq. 5.57, it can be seen that $\overline{z}_{H_i}$ is the point in $\tilde{B}_{max}$ which is closer to agent i than any other point in $\tilde{B}_{max}$. In other words, $\overline{z}_{H_i}$ has the property that $||\overline{z}_{H_i} - \overline{z}_{agt_i}|| \leq ||\overline{z} - \overline{z}_{agt_i}||$ for all $\overline{z} \in \tilde{B}_{max}$. Choosing $\overline{z}_{H_I} \in \tilde{B}_{max}$ in the previous expression yields the relationship

$$||\overline{z}_{H_i} - \overline{z}_{agt_i}|| \leq ||\overline{z}_{H_I} - \overline{z}_{agt_i}|| \text{ for } i \in I_n \tag{5.62}$$

Overbounding Eq. 5.61 with Eq. 5.62 yields

$$||\overline{z}_{H_I} - \overline{z}_{agt_I}|| \leq ||\overline{z}_{H_I} - \overline{z}_{agt_i}|| \text{ for } i \in I_n \tag{5.63}$$

The inequality given in Eq. 5.60 only needs to hold for $i \in I_n \setminus \{I\}$ whereas the result in Eq. 5.63 is valid for $i \in I_n$. Therefore, this shows that $\overline{z}_{H_I} \in V(\overline{z}_{agt_I})$. □

Remark 5.5.3. *Theorem 5.5.2 can be argued intuitively as follows. The quantity* $d_i = ||\overline{z}_{H_i} - \overline{z}_{agt_i}||$ *is the shortest distance between* $\overline{z}_{agt_i}$ *and the set* $\tilde{B}_{max}$*. Without loss of generality, consider there to be only a single point in the map which has maximum score. In other words,* $\tilde{B}_{max} = \overline{z}_H = \overline{z}_{H_i} \forall i$*. In this case,* d_i *is the distance between this point and generator* $\overline{z}_{agt_i}$*. In general,* $\overline{z}_{H_i}$ *is not necessarily in* $V(\overline{z}_{agt_i})$ *(i.e.* $\overline{z}_{H_3}$ *in Figure 5.24). For the case of* $i = I$*, the generator* $\overline{z}_{agt_I}$ *is closer to* $\overline{z}_H$ *than any other generator by definition. Therefore, by definition of the Voronoi polygon* $V(\overline{z}_{agt_I})$*, the point* $\overline{z}_H = \overline{z}_{H_I} \in V(\overline{z}_{agt_I})$*.*

Recall that the function $f_h()$ (Eq. 5.17) served to indicate the direction the agent could travel in the set $\tilde{B}_R$ in order to move towards a cell of maximum score. This required definition of the point $\overline{z}_{h_i}$. The presence of the Voronoi polygon complicates matters slightly. In this context, the definition of $\overline{z}_{h_i}$ is through several intermediate variables. The first of which is

$$\overline{z}'_{h_i} \in \begin{cases} \arg \underset{\overline{z} \in \tilde{B}_{R_i} \bigcap V(\overline{z}_{agt_i})}{\text{minimize}} ||\overline{z} - \overline{z}_{H_i}|| & \text{if } \tilde{B}_{R_i} \bigcap V(\overline{z}_{agt_i}) \neq \emptyset \\ \overline{z}_{agt_i} & \text{otherwise} \end{cases} \tag{5.64}$$

Although it is guaranteed that $V(\overline{z}_{agt_i})$ has finite area, it is not guaranteed that $\tilde{B}_{R_i} \bigcap V(\overline{z}_{agt_i}) \neq \emptyset$. An example of this is shown in Figure 5.26. In this situation, the set $\tilde{B}_{R_i}$ are all the black dots within the dashed red circle. Due to the proximity of other agents, the Voronoi polygon $V(\overline{z}_{agt_i})$ may be so small and not include any of these points. Despite this fact, using Eq. 5.64, it is guaranteed that $\overline{z}'_{h_i} \in B_{R_i} \bigcap V(\overline{z}_{agt_i})$.

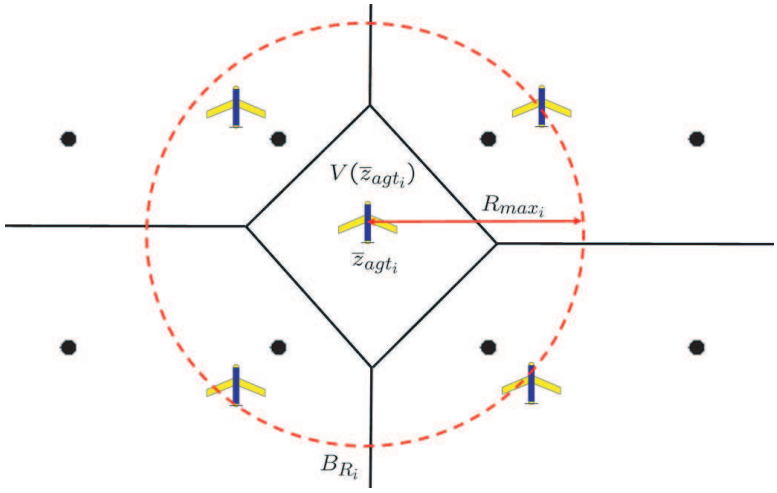


Figure 5.26: Situation showing $\tilde{B}_{R_i} \bigcap V(\overline{z}_{agt_i}) = \emptyset$.

Theorem 5.5.4. *Given any Voronoi diagram* $\overline{V} = \{V(\overline{z}_{agt_1}), ..., V(\overline{z}_{agt_n})\}$, $\overline{z}'_{h_i} \in B_{R_i} \bigcap V(\overline{z}_{agt_i})$ *for all* $i \in I_n$.

Proof. If the set $\tilde{B}_{R_i} \bigcap V(\overline{z}_{agt_i}) \neq \emptyset$ then the feasible set of the minimization problem in Eq. 5.64 is $\tilde{B}_{R_i} \bigcap V(\overline{z}_{agt_i})$ and therefore, $\overline{z}'_{h_i} \in \tilde{B}_{R_i} \bigcap V(\overline{z}_{agt_i})$. Since $\tilde{B}_R \subset B_{R_i}$, $\overline{z}'_{h_i} \in \tilde{B}_{R_i} \bigcap V(\overline{z}_{agt_i}) \Rightarrow \overline{z}'_{h_i} \in B_{R_i} \bigcap V(\overline{z}_{agt_i})$.

If the set $\tilde{B}_{R_i} \bigcap V(\overline{z}_{agt_i}) = \emptyset$ then $\overline{z}'_{h_i} = \overline{z}_{agt_i}$. By definition of a Voronoi polygon, $\overline{z}'_{h_i} = \overline{z}_{agt_i} \in V(\overline{z}_{agt_i})$ since $\overline{z}_{agt_i}$ is the generator for the Voronoi polygon $V(\overline{z}_{agt_i})$. By definition of B_{R_i} (Eq. 5.9), $\overline{z}'_{h_i} = \overline{z}_{agt_i} \in B_{R_i}$ for all possible values of R_{max} (since negative R_{max} values are not allowed in this framework). Since $\overline{z}'_{h_i} \in V(\overline{z}_{agt_i})$ and $\overline{z}'_{h_i} \in B_{R_i}$, then $\overline{z}'_{h_i} = \overline{z}_{agt_i} \in B_{R_i} \bigcap V(\overline{z}_{agt_i})$. □

Remark 5.5.5. *It should be noted that the reachable set in Theorem 5.5.4 is* B_{R_i}, *not* $\tilde{B}_{R_i}$.

With $\overline{z}'_{h_i}$ defined, it is convenient to define a flag ζ_i as

$$\zeta_i = \begin{cases} 1 & \text{if } ||\overline{z}'_{h_i} - \overline{z}_{H_i}|| \geq ||\overline{z}_{agt_i} - \overline{z}_{H_i}|| \\ 0 & \text{otherwise} \end{cases} \tag{5.65}$$

Physically, $\zeta_i = 1$ if the point $\overline{z}'_{h_i}$ is further away from $\overline{z}_{H_i}$ than the agent's current position of $\overline{z}_{agt_i}$. An example showing when $\zeta_i = 1$ is shown in Figure 5.27. In this situation,

the set $\tilde{B}_{R_i} \bigcap V(\overline{z}_{agt_i})$ is a single point and therefore, Eq. 5.64 chooses it as $\overline{z}'_{h_i}$ (denoted by the black dot enclosed by the orange circle). In this case, $\overline{z}'_{h_i}$ is obviously farther away from $\overline{z}_{H_i}$ than $\overline{z}_{agt_i}$, so $\zeta_i = 1$.

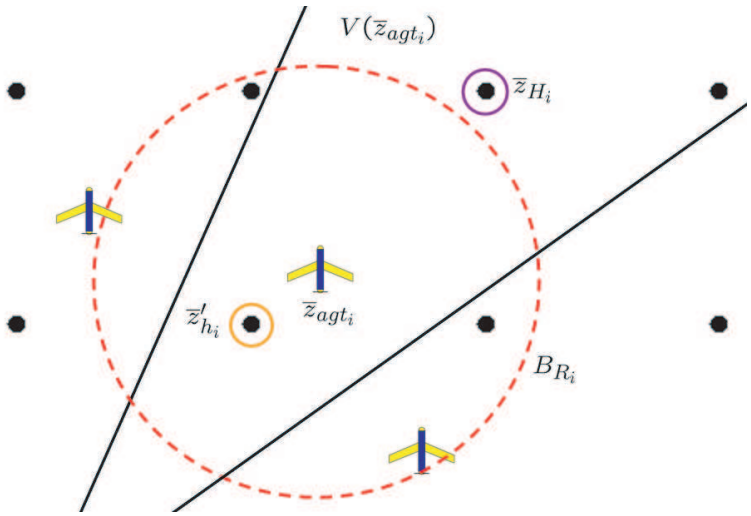


Figure 5.27: Situation showing $\zeta_i = 1$.

With the flag ζ_i defined, the point $\overline{z}_{h_i}$ is given by

$$\overline{z}_{h_i} = \begin{cases} \overline{z}'_{h_i} & \text{if } \zeta_i = 0 \\ \overline{z}_{agt_i} & \text{if } \zeta_i = 1 \text{ and } i \neq I \\ \overline{z}_{S_I} & \text{if } \zeta_i = 1 \text{ and } i = I \end{cases} \tag{5.66}$$

where

$$\overline{z}_{S_I} = \overline{z}_{agt_I} + R_{max_I} \frac{\overline{z}_{H_I} - \overline{z}_{agt_I}}{||\overline{z}_{H_I} - \overline{z}_{agt_I}||} \tag{5.67}$$

The point $\overline{z}_{h_i}$ has several interesting properties

Theorem 5.5.6. *The point $\overline{z}_{h_i}$ is always reachable by agent i and remains in the agent's Voronoi polygon, $\overline{z}_{h_i} \in B_R \bigcap V(\overline{z}_{agt_i})$.*

Proof. Eq. 5.66 states that $\overline{z}_{h_i}$ can be equal to either $\overline{z}'_{h_i}$, $\overline{z}_{agt_i}$, or $\overline{z}_{S_I}$, depending on the situation. Showing $\overline{z}_{h_i} \in B_R \bigcap V(\overline{z}_{agt_i})$ can be achieved by showing that each of the three

previously mentioned values are in this set as well (depending on the situation).

Theorem 5.5.4 showed that $\overline{z}'_{h_i} \in B_R \bigcap V(\overline{z}_{agt_i})$. The proof section of Theorem 5.5.4 also showed how $\overline{z}_{agt_i} \in B_R \bigcap V(\overline{z}_{agt_i})$.

The last case to check is if $\overline{z}_{h_i} = \overline{z}_{S_I}$. From Eq. 5.66, $\overline{z}_{h_i} = \overline{z}_{S_I}$ if and only if $i = I$ and $\zeta_I = 1$. For the case where $i = I$ and $\zeta_I = 1$, from Eq. 5.65, $\zeta_I = 1 \Rightarrow ||\overline{z}'_{h_I} - \overline{z}_{H_I}|| \geq ||\overline{z}_{agt_I} - \overline{z}_{H_I}||$. Looking at the equation for $\overline{z}'_{h_I}$ (Eq. 5.64), it can be seen that $\overline{z}'_{h_I} \neq \overline{z}_{agt_I}$ (or else ζ_I would equal 0 and this cannot happen in this case). Therefore, $\overline{z}'_{h_I}$ must be obtained by solving the minimization problem in Eq. 5.64. Furthermore, in order for $\zeta_I = 1$, the solution to the minimization problem must be farther away from the point $\overline{z}_{H_I}$ than the current agent location so $\overline{z}_{H_I}$ cannot be in the feasible set of $\tilde{B}_{R_I} \bigcap V(\overline{z}_{agt_I})$ (otherwise it would be chosen as the minimizer since it would yield a cost of 0 in the cost function). Theorem 5.5.2 showed that $\overline{z}_{H_I} \in V(\overline{z}_{agt_I})$ so $\overline{z}_{H_I} \notin \tilde{B}_{R_I}$. Theorem 5.5.1 can then be applied to show that $\overline{z}_{H_I} \notin \tilde{B}_{R_I} \Rightarrow \overline{z}_{H_I} \notin B_{R_I}$. Now note that $\overline{z}_{H_I} \notin B_{R_I} \iff ||\overline{z}_{H_I} - \overline{z}_{agt_I}|| > R_{max_I}$. Since $R_{max_I} > 0$, Eq. 5.67 can be written as

$$\overline{z}_{S_I} = \overline{z}_{agt_I} + \theta(\overline{z}_{H_I} - \overline{z}_{agt_I}) \in V(\overline{z}_{agt_I}) \text{ for some } \theta \in (0, 1) \text{ when } \zeta_I = 1 \tag{5.68}$$

Theorem 5.5.2 gave one point that is always in $V(\overline{z}_{agt_I})$ by showing that $\overline{z}_{H_I} \in V(\overline{z}_{agt_I})$. The generator of the Voronoi polygon is a second point which is always in the Voronoi polygon, so $\overline{z}_{agt_I} \in V(\overline{z}_{agt_I})$. Using the fact that all Voronoi polygons are convex sets (Eq. D.6 of Appendix D) and that the line segment between any two points in a convex set is in the convex set as well, it can be seen that the line segment $\overline{z}_{agt_I} + \theta(\overline{z}_{H_I} - \overline{z}_{agt_I}) \in V(\overline{z}_{agt_I})$ for some $\theta \in [0, 1]$. Comparing this to the result in Eq. 5.68, it can be seen that

$$\overline{z}_{S_I} \in V(\overline{z}_{agt_I}) \text{ when } \zeta_I = 1 \tag{5.69}$$

Physically, the point $\overline{z}_{S_I}$ is obtained by moving a distance R_{max_I} towards $\overline{z}_{H_I}$ from the point $\overline{z}_{agt_I}$. So by construction, $\overline{z}_{S_I} \in B_{R_I}$. This, combined with the result in Eq. 5.69, shows that $\overline{z}_{S_I} \in B_{R_I} \bigcap V(\overline{z}_{agt_I})$ when $\zeta_I = 1$.

The combined analysis shows that $\overline{z}_{h_i} \in \overline{z}_{h_i} \bigcap V(\overline{z}_{agt_i})$ for all $i \in I_n$. □

The point $\overline{z}_{h_i}$ has the property that it is guaranteed to no be farther away from the point $\overline{z}_{H_i}$ than the agent's current location.

Theorem 5.5.7. *The point $\overline{z}_{h_i}$ is not farther away from the point $\overline{z}_{H_i}$ than the agent's current location in the sense that $||\overline{z}_{h_i} - \overline{z}_{H_i}|| \leq ||\overline{z}_{agt_i} - \overline{z}_{H_i}|| \; \forall i \in I_n \backslash \{I\}$ and $||\overline{z}_{h_I} - \overline{z}_{H_I}|| < ||\overline{z}_{agt_I} - \overline{z}_{H_I}||$ (the inequality is strict for the case of $i = I$).*

Proof. From Eq. 5.66 if $\zeta_i = 0$, then $\overline{z}_{h_i} = \overline{z}'_{h_i}$. From Eq. 5.65, $\zeta_i = 0 \iff ||\overline{z}'_{h_i} - \overline{z}_{H_i}|| < ||\overline{z}_{agt_i} - \overline{z}_{H_i}||$ so

$$||\overline{z}_{h_i} - \overline{z}_{H_i}|| < ||\overline{z}_{agt_i} - \overline{z}_{H_i}|| \text{ if } \zeta_i = 0 \text{ and } \overline{z}_{h_i} = \overline{z}'_{h_i} \tag{5.70}$$

For the case when $\zeta_i = 1$ and $i \neq I$, then $\overline{z}_{h_i} = \overline{z}_{agt_i}$. In this situation it is obvious that

$$||\overline{z}_{h_i} - \overline{z}_{H_i}|| = ||\overline{z}_{agt_i} - \overline{z}_{H_i}|| \text{ if } \zeta_i = 1 \text{ and } \overline{z}_{h_i} = \overline{z}_{agt_i} \tag{5.71}$$

Finally, for the case when $\zeta_i = 1$ and $i = I$, then $\overline{z}_{h_I} = \overline{z}_{S_I}$. The proof section of Theorem 5.5.6 showed that in this situation, $\overline{z}_{H_I} \notin B_{R_I}$ and Eq. 5.68 showed that $\overline{z}_{S_I}$ must be closer to $\overline{z}_{H_I}$ than $\overline{z}_{agt_I}$. Therefore,

$$||\overline{z}_{h_I} - \overline{z}_{H_I}|| < ||\overline{z}_{agt_I} - \overline{z}_{H_I}|| \text{ if } \zeta_I = 1 \text{ and } \overline{z}_{h_I} = \overline{z}_{S_I} \tag{5.72}$$

□

Remark 5.5.8. *Theorem 5.5.7 can also be written as $||\overline{z}_{h_i} - \overline{z}_{H_I}|| = ||\overline{z}_{agt_i} - \overline{z}_{H_I}||$ if $\zeta_i = 0$ and $i \neq I$. Otherwise, $||\overline{z}_{h_i} - \overline{z}_{H_I}|| < ||\overline{z}_{agt_i} - \overline{z}_{H_I}||$.*

The above analysis shows that the point $\overline{z}_{h_i}$ has some interesting properties:

1. $\overline{z}_{h_i} \in B_{R_i} \bigcap V(\overline{z}_{agt_i})$

2. $||\overline{z}_{h_i} - \overline{z}_{H_i}|| \leq ||\overline{z}_{agt_i} - \overline{z}_{H_i}||$ for $i \in I_n \setminus \{I\}$

3. $||\overline{z}_{h_I} - \overline{z}_{H_I}|| < ||\overline{z}_{agt_I} - \overline{z}_{H_I}||$ for $i = I$

These properties can be used to develop the control law with explicit cooperation between agents.

Control Law with Explicit Cooperation

The control law for choosing $\overline{z}^\star$ was previously described in Section 5.3.3, Eq. 5.29. This involved maximizing the reward function $J_{0_i}()$ over the set $\tilde{B}_{R_i}$. The added complexity of the Voronoi diagram requires a slight redefinition of the reward function

$$\hat{J}_{0_i} = \alpha \hat{x}_w(k+d, \overline{z}) + \eta \left(\beta f_\chi(\overline{z}) + \gamma f_d(\overline{z})\right) \tag{5.73}$$

In Eq. 5.73 it is understood that all of the parameters α, β, and γ and all the functions η, $f_\chi()$, and $f_d()$ are specific to agent i. The variable η is defined slightly differently as

$$\eta = \max_{\overline{z} \in \tilde{B}_R \bigcap V(\overline{z}_{agt_i})} \hat{x}_w(k+d, \overline{z}) \tag{5.74}$$

Notice that this reward function omits the term that contains the indicator function $f_h()$. Previously, this was used to guarantee coverage of the map and it is instead incorporated in the algorithm in a heuristic fashion.

In order to maintain the Voronoi partitioning, the modified control law to incorporate explicit cooperation is now of the form

$$\overline{z}_i^{\star\prime} \in \begin{cases} \arg \underset{\overline{z} \in \tilde{B}_{R_i} \bigcap V(\overline{z}_{agt_i})}{\text{maximize}} \hat{J}_{0_i}(\overline{z}) & \text{if } \tilde{B}_{R_i} \bigcap V(\overline{z}_{agt_i}) \neq \emptyset \\ \overline{z}_{agt_i} & \text{otherwise} \end{cases} \tag{5.75}$$

$$\overline{z}_i^\star \in \begin{cases} \overline{z}_{h_i} & \text{if } \alpha \hat{x}_w(k+d, \overline{z}_i^{\star\prime}) < \delta \\ \overline{z}_i^{\star\prime} & \text{otherwise} \end{cases} \tag{5.76}$$

Agent i chooses $\overline{z}_i^\star$ as the next location to search and the process repeats once the agent reaches $\overline{z}_i^\star$.

Theorem 5.5.9. *The point $\overline{z}_i^\star$ is always reachable by agent i and remains in the agent's Voronoi polygon, $\overline{z}_i^\star \in B_{R_i} \bigcap V(\overline{z}_{agt_i})$.*

Proof. In the case where $\overline{z}_i^\star = \overline{z}_i^{\star'}$, it is obvious From Eq. 5.75 that $\overline{z}_i^{\star'} \in B_{R_i} \bigcap V(\overline{z}_{agt_i})$ since the feasible set of the maximization problem is precisely $B_{R_i} \bigcap V(\overline{z}_{agt_i})$ and it was previously shown that $\overline{z}_{agt_i} \in B_{R_i} \bigcap V(\overline{z}_{agt_i})$.

In the case where $\overline{z}_i^\star = \overline{z}_{h_i}$, Theorem 5.5.6 can be applied to show that $\overline{z}_{h_i} \in B_{R_i} \bigcap V(\overline{z}_{agt_i})$. □

If the same assumptions in Section 5.3.3 are satisfied, several guarantees about algorithm performance can be made. However, unlike the results in Section 5.3.3 where guarantees are shown for all agents in the team, most of the guarantees for the case of explicit cooperation can be shown only for the case of $i = I$.

Theorem 5.5.10. *Under assumptions A.3, A.4, A.5, and the previously described search strategy with explicit cooperation, if agent I is not currently at the point $\overline{z}_{H_I}$ the agent must move to a point other than $\overline{z}_{agt_I}$ once the quantity $\alpha x_w(k, \overline{z}_{agt_I})$ decreases below the value δ.*

Proof. Assuming that $\overline{z}_i^{\star'} = \overline{z}_{agt_I}$, if the quantity $\alpha x_w(k, \overline{z}_{agt_I})$ decreases below the value δ, then Eq. 5.76 will assign $\overline{z}_i^\star = \overline{z}_{h_i}$. Theorem 5.5.7 can be applied to show that $\overline{z}_i^\star$ is strictly closer to $\overline{z}_{H_i}$ than $\overline{z}_{agt_I}$. Since the agent is not at the point $\overline{z}_{H_I}$, the agent must move to a point other than $\overline{z}_{agt_I}$. □

Theorem 5.5.10 shows that the agent cannot remain at the same location indefinitely. The control law in Eq. 5.76 differs from that in Section 5.3.3. One consequence of this is that Assumption A.7 is not required to ensure exhaustive searches. This is illustrated in the next several theorems.

Theorem 5.5.11. *Under the previously described search strategy, agent I must choose either a cell center which has non-zero score, $\overline{z}_{h_I}$, or both as the point $\overline{z}_I^\star$.*

Proof. The point $\overline{z}_i^\star$ is assigned according to Eq. 5.76. If the point $\overline{z}_i^{\star'}$ has a score of zero, then $\alpha \hat{x}_w(k+d, \overline{z}_i^{\star'}) = 0$ which is less than the value of δ for any $\delta \in (0, 1]$ so Eq. 5.76 will assign $\overline{z}_i^\star = \overline{z}_{h_i}$.

Alternatively, if $\alpha\hat{x}_w(k+d, \overline{z}_i^{\star'}) \geq \delta$, then Eq. 5.76 will assign $\overline{z}_i^{\star} = \overline{z}_i^{\star'}$. Since $\delta, \alpha \in (0, 1]$, $\alpha\hat{x}_w(k+d, \overline{z}_i^{\star'}) \geq \delta \iff \hat{x}_w(k+d, \overline{z}_i^{\star'}) > 0$ which shows that the point associated with $\overline{z}_I^{\star}$ has a non-zero score. □

Theorem 5.5.12. *Under assumption A.4, A.5 and the previously described search strategy, if agent I chooses $\overline{z}_{h_I}$ as $\overline{z}_I^{\star}$ and $\overline{z}_{h_I} \neq \overline{z}_{H_I}$, then at the next time step the point $\overline{z}_{H_I}$ will remain unchanged and some agent $i \in I_n$ will move closer to the point $\overline{z}_{H_I}$.*

Proof. Theorem 5.5.7 showed that $||\overline{z}_{h_I} - \overline{z}_{H_I}|| < ||\overline{z}_{agt_I} - \overline{z}_{H_I}||$ so if the agent chooses $\overline{z}_I^{\star} = \overline{z}_{h_I}$, at the next time step, it be closer to the point $\overline{z}_{H_I}$ than it was before. If $\overline{z}_{h_I} \neq \overline{z}_{H_I}$, then the agent cannot search that cell so the score will not decrease and at the next step, $\overline{z}_{H_I}$ will still be in the set $\tilde{B}_{max}$. Unlike Theorem 5.3.4, another agent cannot search this same location due to fact that $\overline{z}_i^{\star} \in V(\overline{z}_{agt_i})$ and therefore, $\overline{z}_{H_I} \in V(\overline{z}_{agt_I})$.

Furthermore, since the distance between the agent and the same point $\overline{z}_{H_I}$ is decreased, this same point will be chosen by Eq. 5.57 as the point $\overline{z}_{H_i}$. Note that the index is i, not I in the last sentence. This is because it is possible that the index of the agent that is closest to the set $\tilde{B}_{max}$ may change at the next time step. However, it will change only if another agent moves closer to the set $\tilde{B}_{max}$ than the current agent located at $\overline{z}_{agt_I}$. This does not affect the proof in the sense that the point $\overline{z}_{H_I}$ does not change, simply the index of whichever agent happens to be closest to $\overline{z}_{H_I}$ at the next step. □

Using these results, we can show that the control law with explicit cooperation yields an exhaustive map search

Theorem 5.5.13. *Under the previously described assumptions and search strategy with explicit cooperation, $x_w(k, \overline{z}) \to 0 \ \forall \overline{z} \in B$ (the scores of all cells in the map will approach 0).*

Proof. Theorem 5.5.10 guarantees that agent I cannot remain in a single cell indefinitely and Theorem 5.5.11 ensures that it must choose either a cell of non-zero score, $\overline{z}_{h_I}$, or both as the point $\overline{z}_I^{\star}$ at any given time step. If agent I chooses a cell of non-zero score, Theorem 5.3.1 ensures that the score of that cell is monotonically decreased towards 0.

If the agent does not choose a cell of non-zero score, the only alternative scenario allowed by Theorem 5.5.11 is that the agent chooses $\overline{z}_{h_I}$ and $x_w(k, \overline{z}_{h_I}) = 0$. By definition of $\overline{z}_{H_I}$, if $x_w(k, \overline{z}_{H_I}) = 0$, then the map has been completely covered since all scores are at most 0. Therefore, assuming that the map has not been entirely searched yet, $x_w(k, \overline{z}_{h_I}) = 0 \Rightarrow \overline{z}_{h_I} \neq \overline{z}_{H_I}$.

Theorem 5.5.12 ensures that if if $\overline{z}_{h_I} \neq \overline{z}_{H_I}$, choosing $\overline{z}_{h_I}$ as the point $\overline{z}_I^\star$ does not change the value of $\overline{z}_{H_I}$ at the next time step.

At this next time step, the agent once again must choose a cell with non-zero score, $\overline{z}_{h_I}$, or both. If the agent continues to choose $\overline{z}_{h_I}$, eventually $\overline{z}_{H_I} \in \tilde{B}_{R_I}$ and at this point, choosing a cell with non-zero score, $\overline{z}_{h_I}$, or both guarantees that a cell of non-zero score is chosen. Therefore, the score of some cell will be ensured to decrease and given sufficient time, $x_w(k, \overline{z}) \to 0 \ \ \forall \overline{z} \in B$. □

Remark 5.5.14. *Theorem 5.5.13 differs from Theorem 5.3.6 in a few subtle ways. In this situation, the explicit cooperation makes the proof valid only for agent I instead of all agents (Theorem 5.3.6). However, this still guarantees coverage for the team since at any point, there must be an agent in the team which has the properties that allow it to have the index I.*

5.5.2 ($\wp_3$) with Explicit Cooperation

Recall that ($\wp_3$) involved finding a path from the agent's current location or $\overline{z}_{agt_i}$ to the optimal point or $\overline{z}_i^\star$. Section 5.4.1 proposed a method that plans paths by formulating the problem as a convex optimization scheme. This method can be applied without modification to this situation with explicit cooperation. Recall that the waypoints returned by solving Eq. 5.39 all lie on a line between $\overline{z}_{agt_i}$ and $\overline{z}_i^\star$. Showing that these waypoints are all within the set $B_{R_i} \bigcap V(\overline{z}_{agt_i})$ can be done by applying Theorem 5.5.9 and its supporting proofs. Therefore, this path will remain in the agent's Voronoi polygon.

The Voronoi partitions can be easily incorporated into the graph based path planning by carefully assigning the span intervals. Using a method similar to that described in Eq. 5.54, the upper span interval can be set to $+\infty$ if the arc travels outside the agent's Voronoi

polygon. This guarantees that any forward path through the graph with cost less than $+\infty$ is guaranteed to be both feasible for the agent and also remain within the agent's Voronoi polygon.

Using these modifications for explicit cooperation, each agent in the team makes decision based on the existence of the other agents in the team. Under this control law, agents remain partitioned according to the Voronoi tessellation. Although this increases the computational resource requirements and complexity, it has several benefits such as guaranteed collision avoidance at the strategic planning level and increased team performance.

Chapter 6

SIMULATION IMPLEMENTATION

The algorithms and results described in the previous chapters were implemented as a stand-alone, object oriented, C++ application. This monolithic application is used for both simulation and real-time data collection purposes. This chapter details pure simulation results and the architecture of the application. Human-in-the-loop simulation is detailed in Chapter 7 and real-time flight testing and hardware results are described in Chapter 8.

6.1 Simulation Architecture

Due to the complexity of the simulation, it is implemented as an object oriented application coded in C++. Each relevant object is implemented as its own class. Some classes have the capability to interface with external hardware such as GPS devices and serial ports. A list of some of the major classes in the simulation are listed in Table 6.1.

Table 6.1: Function of major classes in monolithic simulation.

Class	**Function**
Agent	• Simulate agent dynamics • Interface with DataHub and serial port • Record agent state and control history • Update occupancy based map based on agent measurements
OccupancyMap	• Maintain occupancy based map • Interact with occupancy based map

continued on next page...

Table 6.1 – Continued

Class	**Function**
Path	• Represents variable number of waypoints that makeup the path • Interface with DataHub to publish path information
AgentController	• Inner loop controller for agents • Waypoint navigation
StrategicController	• Outer loops controller for agents • Compute strategic, single agent search strategy
MyGSLMatrix	• Wrapper for GSL library functions • Implement linear algebra functions (matrices, vectors, etc.)
MySerial	• Interact with serial port • Record/interpret data from GPS device
DataHubInterface	• Initialize DataHub server/connection • Publish/Subscribe information from DataHub server

6.1.1 Code Flow

The system is implemented as an object-orientated application in C++. The total application exceeds 100,000 custom, hand-coded lines and countless libraries. A high level, general flow diagram of the code is shown in Figure 6.1.

Figure 6.1, Line 1 creates many default objects that are required for the simulation. These include objects which represent the agents, targets, occupancy maps, and various support classes such as the inner and outer loop controller for the agents. These objects are then initialized in Line 2 of the code. This process involves setting initial conditions for the target and agents. Also, the occupancy map is initialized by interfacing with a geo-referenced image of the search area (described previously in Section 4.1.2). The DataHub server and connection to said server must be initialized as well. In addition to the DataHub

```
1   Create default objects
2   Initialize Simulation
3   while (1)
4       Recompute expired paths
5       Propagate target objects
6       Propagate agent objects
7       Update occupancy map based on agent and target states
8       Save loop variables
9       Check termination
10      Delay process
11  end while
12  Save simulation variables
```

Figure 6.1: High Level code flow of simulation application.

connection, a serial port connection to the GPS unit is established here as well. The DataHub is described later in Section 7.2.1. If the agent is a real agent, its initial condition is set via data from the GPS.

After setup and initialization, the main loop of the algorithm is executed. The system first checks to see if any of the planned paths for the agents are expired or not. If they are expired, new paths are recomputed using the methods described in Chapter 5. Pseudo-code which shows a more detailed explanation of Figure 6.1, Line 4 is shown in Figure 6.2.

Once the paths are computed for all agents, the target states are updated using an external control signal. In most situations, the target is stationary so Line 5 in Figure 6.1 involves simply updating the target state with a control vector of zeros.

The agent's states are updated in Line 6 in Figure 6.1. Each agent has an inner loop controller (AgentController object) which is responsible for tactical tasks such as path following

```
1  for all expired paths
2      Estimate target's state and control vector
3      Propagate world state forward by d steps (℘1)
4      Find desirable location for agent to search (℘2)
5      Plan path from agent's current position to desirable location (℘3)
6  end for
7  return new paths
```

Figure 6.2: Pseudo code for process which recomputes expired paths.

and navigation. This controller computes controls which will allow the agent to follow the desired path by calculating trajectories which take the agent to the active waypoint in the path. These controls are then applied to the agent to propagate its state forward. The dynamical model used to simulate the agent at this point can be user specified to model a possibly heterogeneous team.

With the target and agent states updated, the agents make measurements using their sensor models and update the occupancy map in Line 7. This involves incorporating both the sensor measurements from the agents and also applying the time varying score decay model described in Eq. 4.11.

The pertinent simulation variables are then written to standard ASCII text files. These are saved in a standardized form so that they may be imported into other applications such as Matlab for visualization and data analysis purposes.

6.1.2 Timing

The application is a monolithic simulation of N agents. In the case where the application is purely simulating the scenario, real time constrains are not necessary and timing is not an issue. Each agent can request as much processing time as needed in order to compute

its required algorithms. However, when the application is used for gathering real-time data, timing and processor sharing becomes an issue. A major obstacle is the fact that the application is monolithic rather than distributed, and therefore all calculations for all N agents are performed on a single processor. The application is not a true real time application in the sense that interrupts and event handlers are not used so processor sharing is not inherent in the code. Instead, in an attempt to ameliorate the problem of N agents utilizing a single processor, a careful timing structure is implemented to allow the N agents to share processor time. A timing diagram of this procedure is shown in Figure 6.3.

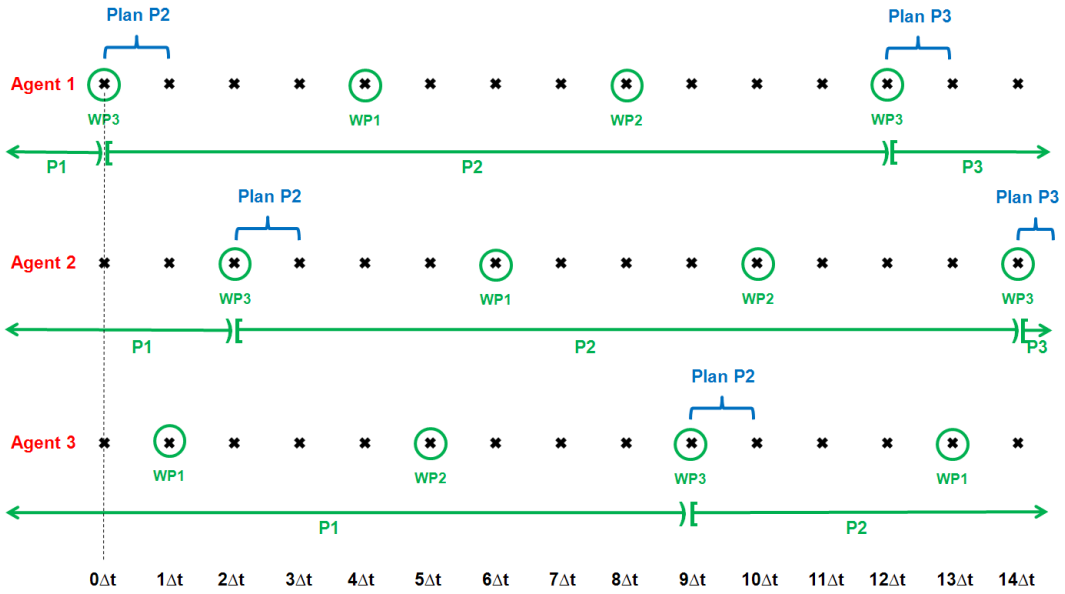


Figure 6.3: Timing diagram with $N = 3$, $d = 3$, $K = 4$.

Each black 'x' in Figure 6.3 represents a single iteration of the main loop of the algorithm (shown in Figure 6.1). A single iteration of the main loop represents Δt seconds. The main loop executes K times between each waypoint in the path and therefore, the time between waypoints is $K \cdot \Delta t$. Since there are d waypoints in a single path, a single path is valid for $d \cdot K \cdot \Delta t$ seconds.

Recall that a path is comprised of both waypoints and desired arrival times for each waypoint (Section 5.4.3). A path is expired if the simulation time is greater than the arrival

time of the dth waypoint in the path (all waypoints for the given path occur in the past). As soon as a path expires, the system has at most Δt seconds to compute a new path. Computing a new path requires executing the code previously described in Figure 6.2. If m paths expire at the same time, the single agent search strategy must be executed m times. This process may require significant computational resources and therefore, it is undesirable to have more than one path expire at any given time. The system performance is achieved by simply staggering paths so that no two paths expire at the same time. An example of this staggering is shown in Figure 6.3.

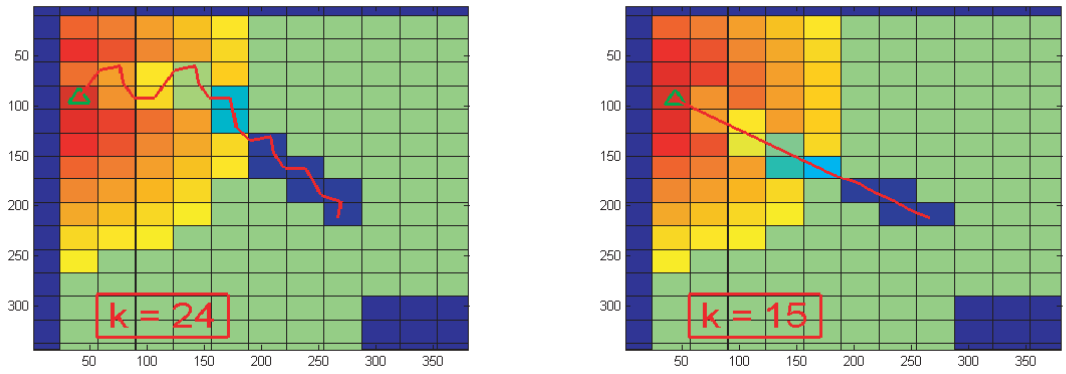
In this example, the system is initialized so the first path for agent 1 expires at the start of the simulation. At the very first iteration through the main loop, the system executes the single agent search strategy code for agent 1 to replan its path. This planning occurs between $0 \cdot \Delta t$ and $1 \cdot \Delta t$. Once this path is computed, agent 1 is committed to this path for the next d waypoints (or $K \cdot d$ loop iterations which is equivalent to $K \cdot d \cdot \Delta t$ seconds). At the next time step (between $1 \cdot \Delta t$ and $2 \cdot \Delta t$), all agents have active paths that have not expired, so the only tasks required are updating the target, agent, and occupancy map states. At the third loop iteration (between $2 \cdot \Delta t$ and $3 \cdot \Delta t$), the path for agent 2 expires and a new path must be computed for agent 2. If the planning horizon is constant for all agents (d is same for all agents), then if there are at most $K \cdot d$ agents in the team, a staggering pattern can be established so that no two paths expire simultaneously. Utilizing this staggering pattern allows for the duty cycle for the single processor to be as high as possible before affecting real time timing constraints.

6.2 Simulation Results

After analyzing and implementing the autonomous algorithms as discussed in the previous chapters, the algorithms can be verified in simulation. Results and performance of this strategy in simulation is the focus of the current section.

6.2.1 Emergent Behavior

The agents following the control law described in the previous chapter exhibit some interesting behaviors. For example, the agents choose control decisions which will benefit them



(a) Greedy algorithm

(b) Planning algorithm

Figure 6.4: Difference between greedy algorithm and planning algorithm showing smooth convergence to a moving target.

d steps in the future rather than doing what is best in the next time step. This leads to desirable behavior such as smooth convergence to a moving target as shown in Figure 6.4.

At step $k = 0$, the system receives an estimate of the target state, $\hat{x}_{tgt}(0)$. This notifies it that the target is directly north of the agent's current position and is moving west with a constant velocity. The trajectory in Figure 6.4(a) is generated when the agent employs a greedy algorithm where at each time step ($k = 0, 1, \ldots, 24$) the agent chooses a control that transitions it to the most desirable coordinate that it can reach in 1 step. The result is that the agent is constantly trying to "catch up" up with the target. In contrast, the trajectory in Figure 6.4(b) is generated when the agent employs the above described search strategy with a look ahead window of $d = 10$. The agent is now choosing control actions which will benefit it in 10 steps. This yields a much smoother and faster convergence with the target.

Another emergent behavior of the agents is the tendency to search areas which have been previously explored as time progresses. This stems from the fact that the occupancy map scores can be time varying. This models the fact that estimates become less certain with time or for the possibility of a moving target. This is particularly useful in missions where the agent is tasked to patrol a region for a possibly evading target. And example of this mission scenario is shown in Figure 6.5. In this situation, the agent (a ScanEagle) is

patrolling a harbor in New York. Initially, it checks the harbor for the target (a submarine) but does not find it (Figure 6.5(b)). It then decides to investigate a possible target to the Southeast (Figure 6.5(c)). When it leaves the harbor, the submarine takes advantage of the agent's departure and moves into the harbor (Figure 6.5(d)). However, the agent eventually decides to return to the harbor to ensure that no targets moved in during its absence and it successfully finds the target (Figure 6.5(f)).

6.2.2 Scenarios and Performance

To evaluate the performance of the various algorithms, several test scenarios are created. These scenarios are used to emulate situations and environments where a general search algorithm might be used.

Testing Scenarios

The algorithm can be tested using several different scenarios. Three representative scenarios are shown in Figure 6.6. Scenario 1 (S1) is used to represent a possible urban scenario where there are hard obstacles such as buildings to avoid. Scenario 2 (S2) represents a maritime environment where the agents might be searching for a lost ship on the water. Finally, scenario 3 (S3) is another urban environment where there is some a priori knowledge regarding the target's possible location.

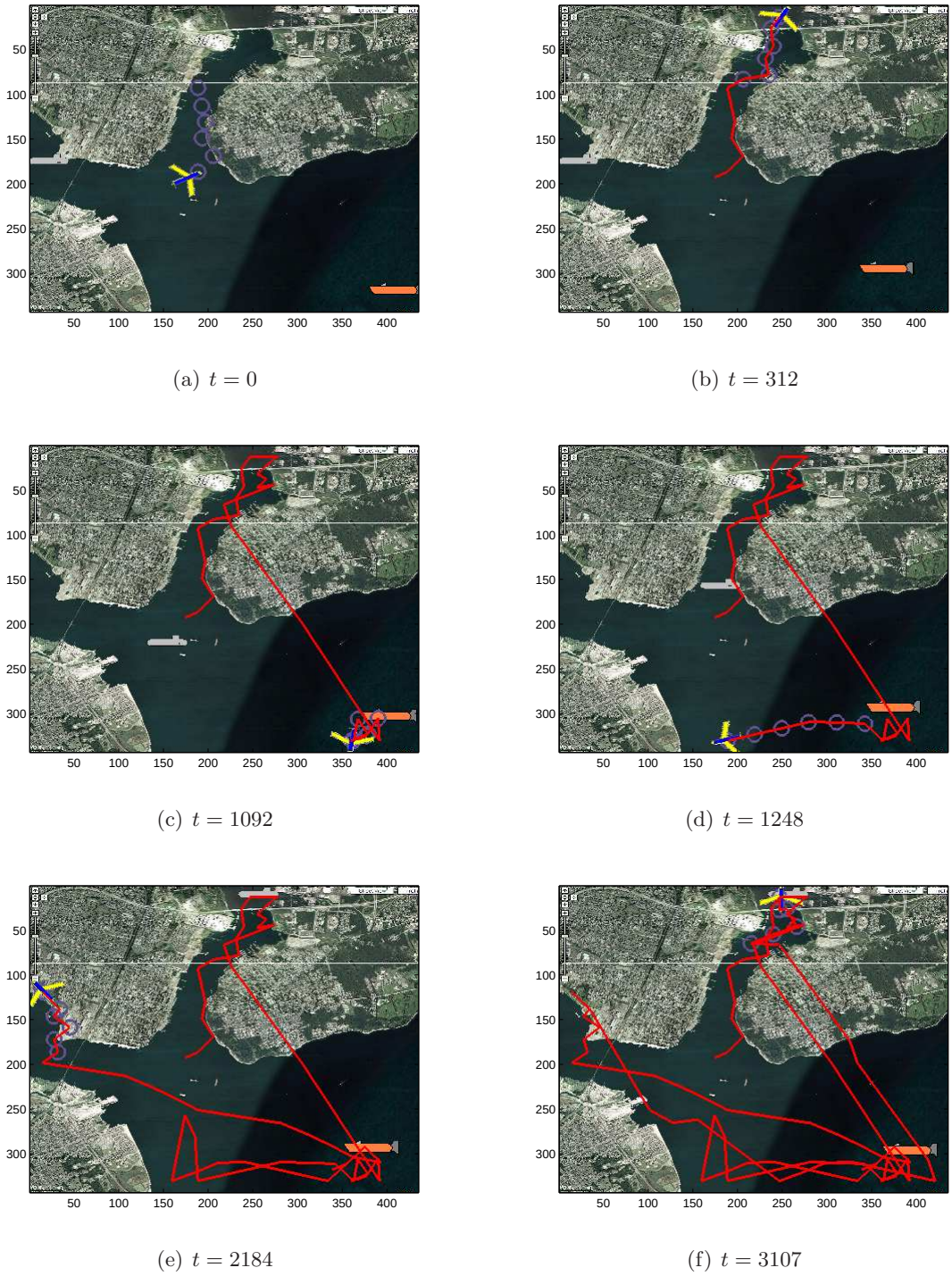
Performance Metrics

In order to gauge performance, several metrics are used. Perhaps the most intuitive is to sum the scores of all the cells. This is directly proportional to the mean of the cell scores. The cumulative map score for a run i at time step k is denoted

$$S(i,k) = \sum_{\overline{z} \in \tilde{B}} x_w(k, \overline{z}) \tag{6.1}$$

Along a similar vein, the variance of the map scores is defined as

$$V(i,k) = \mathrm{Var}(x_w(k, \overline{z})) \text{ for } \overline{z} \in \tilde{B} \tag{6.2}$$



(a) $t = 0$

(b) $t = 312$

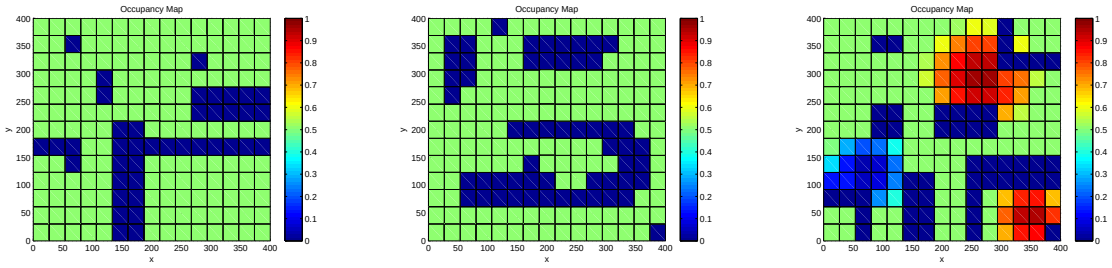
(c) $t = 1092$

(d) $t = 1248$

(e) $t = 2184$

(f) $t = 3107$

Figure 6.5: Single agent in a harbor patrol mission showing revisiting of locations.



(a) Scenario 1 (S1) (b) Scenario 2 (S2) (c) Scenario 3 (S3)

Figure 6.6: Several test scenarios used to verify algorithm performance.

The average values across n runs are simply

$$S_{ave}(k) = \frac{1}{n} \sum_{i \in I_n} S(i,k) \tag{6.3}$$

$$V_{ave}(k) = \frac{1}{n} \sum_{i \in I_n} V(i,k) \tag{6.4}$$

In terms of map coverage, the best and worse case scenarios can be given by

$$S_{max}(k) = \max_{i \in I_n} S(i,k) \tag{6.5}$$

$$S_{min}(k) = \min_{i \in I_n} S(i,k) \tag{6.6}$$

The occupancy map at a given time can also be thought of as a function over the domain B which maps to a scalar in a certain range. In this context, $x_w : B \rightarrow [0,1]$, and the range of the function can be partitioned into a series of finite regions according to

$$\Delta = \{\delta_j,\ j = 1, 2, ..., M\} \tag{6.7}$$

Where each element of the range partition, δ_j, corresponds to a region in the range of $x_w()$

$$\delta_j = \begin{cases} [s_{j-1}, s_j) & j = 1, 2, ..., M-1 \\ [s_{j-1}, s_j] & j = M \end{cases} \tag{6.8}$$

The points $s_j \in \Re$ are most easily chosen as linearly spaced over the range of x_w, but they can be chosen in any fashion that has them monotonically increasing with $s_0 < s_1 < ... < s_M$. With the range partition Δ defined, the domain of the function can be partitioned according to which element of δ_j the image corresponds to. In this context, the inverse of the function with respect to an element of the range partition, δ_j can be defined as

$$x_w^{-1}(\delta_j) \triangleq \{\overline{z} \in B : x_w(\overline{z}) \in \delta_j\} \tag{6.9}$$

Using Eq. 6.9, one possible partitioning of the domain is given as [44]

$$Q(\Delta) = \{Q_j = x_w^{-1}(\delta_j),\ j = 1, 2, ..., M\} \tag{6.10}$$

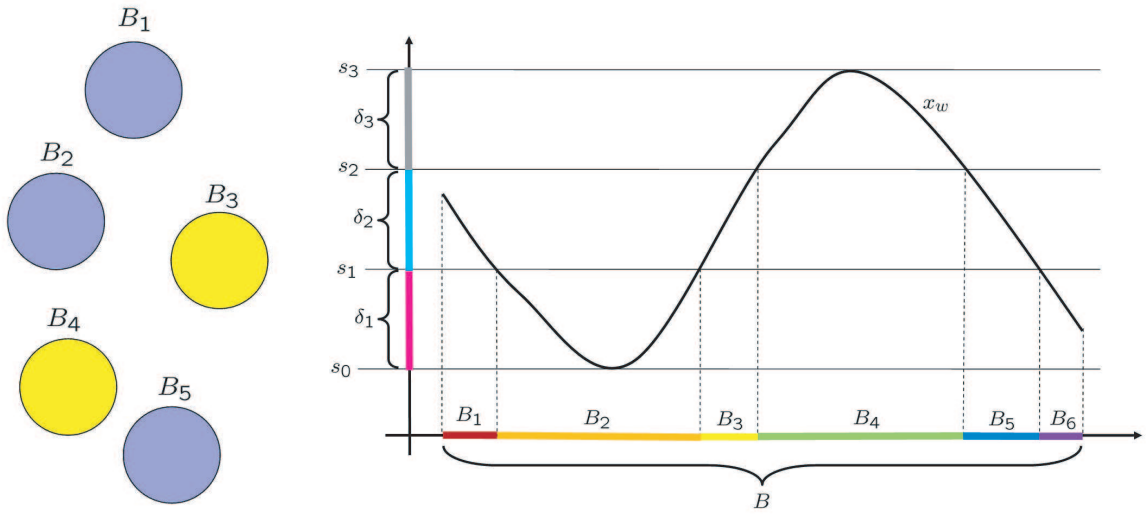
Using Eq. 6.10, each element in the partition $Q(\Delta)$ consists of areas of the domain which map to the same range partition element δ_j. Regardless of the function $x_w()$, using Eq. 6.10 the cardinality of the domain partition equals the cardinality of the range partition in the sense that $|Q(\Delta)| = |\Delta| = M$.

A more complex partitioning of the domain can be obtained by looking at the connected components of $x_w^{-1}(\delta_j)$ [14].

$$\nu(\Delta) = \{\nu_i,\ i = 1, ..., N\} = \bigcup_{j=1,2,...,M} cc\{x_w^{-1}(\delta_j)\} = \left\{cc\{x_w^{-1}(\delta_1)\}, ..., cc\{x_w^{-1}(\delta_M)\}\right\} \tag{6.11}$$

With Eq. 6.11, the $cc\{\}$ function returns the connected components of the argument set. Using Figure 6.7(a) as an example, if a set A is defined as the locations which are yellow and the set B was defined as locations which are purple, then the set $cc\{A\} = \{B_3, B_4\}$ and $cc\{B\} = \{B_1, B_2, B_5\}$.

Each set $cc\{x_w^{-1}(\delta_j)\}$ is a set that corresponds to connected components of $x_w^{-1}(\delta_j)$. Each element in the set $cc\{x_w^{-1}(\delta_j)\}$ is a continuous set of locations which map to δ_j under the



(a) Connected components

(b) Range and domain partitioning

Figure 6.7: Examples showing connected components and range/domain partitioning of $x_w()$ function.

function $x_w()$. The union operator in Eq. 6.11 signifies that the set $\nu(\Delta)$ is comprised of all the sets $cc\{x_w^{-1}(\delta_j)\}$ for $j = 1, ..., M$ but each element retains its individual partitioning (clarified by the second half of the equation).

Figure 6.7(b) provides an example where $x_w : \Re \rightarrow \Re$ but the idea can easily be extended to the case considered in this work where $x_w : \Re^2 \rightarrow \Re$. In this example, the domain of the function is denoted as B and the range of the function over B is partitioning into $M = 3$ components. The points $s_0, ..., s_3$ are chosen to be linearly distributed between the minimum and maximum of the range and define the range components δ_1, δ_2, and δ_3 which are shown as the vertical magenta, cyan, and grey lines, respectively. The locations in the domain of $x_w()$ that correspond to these range partitions are described by $x_w^{-1}(\delta_j)$. For example, $x_w^{-1}(\delta_1)$ has a single element in the set which corresponds to $\{B_2 \bigcup B_6\}$. In a similar fashion, $x_w^{-1}(\delta_2) = \{B_1 \bigcup B_3 \bigcup B_5\}$ and $x_w^{-1}(\delta_3) = \{B_4\}$. Note that each set $x_w^{-1}(\delta_j)$ has only a single element, but this element may be comprised of several disconnected components. Using the partitioning in Eq. 6.11 simply uses $Q_j = x_w^{-1}(\delta_j)$ and therefore, the domain partition has 3 elements as well. However, if Eq. 6.11 is used to partition the domain, then the cardinality

of domain partition is greater than or equal to that of the range partition, $|\nu(\Delta)| \geq |\Delta|$. Using the example, $\nu_1 = cc\{x_w^{-1}(\delta_1)\} = \{B_2, B_6\}$, $\nu_2 = cc\{x_w^{-1}(\delta_2)\} = \{B_1, B_3, B_5\}$, and $\nu_3 = cc\{x_w^{-1}(\delta_3)\} = \{B_4\}$ so the total partition is given by $\nu = \{B_1, B_2, ..., B_6\}$.

With the domain partition Q (or ν) defined , the entropy of the function can be defined using the standard form

$$H(Q) = -\sum_{i=1}^{N} \frac{\mu(Q_i)}{\mu(B)} \log_2 \frac{\mu(Q_i)}{\mu(B)} \tag{6.12}$$

where $\mu(Q_i)$ is the standard Lebesgue measure of the set Q_i. In this case, this is the area of the set Q_i. To compute the entropy of the partition ν, the term Q_i is simply replaced with ν_i. This closely parallels the information theoretic version of entropy and has many of the same properties. The most significant is that if the domain partition contains only a single element (which occurs if the function is constant) then $H(Q) = 0$ [29].

The joint entropy between two partitions Q^o and Q^s is similarly defined as

$$H(Q^o, Q^s) = -\sum_{i=1}^{N^s} \sum_{j=1}^{N^o} \frac{\mu(Q_i \bigcap Q_j)}{\mu(B)} \log_2 \frac{\mu(Q_i \bigcap Q_j)}{\mu(B)} \tag{6.13}$$

The joint entropy has the property that $H(Q^o, Q^s) \leq H(Q^o) + H(Q^s)$. The inequality becomes equality when the two partitions maximally differ (from an information theoretic perspective, this corresponds to the two probabilistic variables being independent). A measure of convergence, deemed the perspectives mutual information [14], between the two partitions is given as

$$I(Q^o, Q^s) = H(Q^o) + H(Q^s) - H(Q^o, Q^s) \tag{6.14}$$

The perspective mutual information has the property that if the two partitions maximally differ, then $I(Q^o, Q^s) = 0$ and if the two partitions are the same (meaning that the two functions are similar since they induce the same partition), then $I(Q^o, Q^s) = H(Q^o) = H(Q^s)$. This provides a useful metric to determine if the current state of the world (embedded in the function $x_w()$) is converging to the desired function.

Physically, the entropy measures the minimum amount of bits (if a base 2 logarithm is

used) required to represent the function on average [29]. This is the minimum level to which the function can be compressed.

In addition to coverage and information metrics, performance metrics which measure expected time to target detection can be defined. The number of agents which find the target in run i is given by $\tilde{N}(i)$. The average number of agents which find the target in n runs can be defined as

$$\tilde{N}_{ave} = \frac{1}{n} \sum_{i \in I_n} \tilde{N}(i) \tag{6.15}$$

The time when the m^{th} agent finds the target is given by $T_m(i)$. Note that by definition, $T_1(i) \leq T_2(i) \leq ... \leq T_M(i)$. The average time to target detection for the encounters is slightly more complicated. This is because in some runs, $T_m(i)$ is undefined (in the case that the m^{th} agent does not find the target). Therefore, the values of $T_m(i)$ which contribute to the average are only those when it is defined

$$T_{m,ave} = \frac{1}{\hat{n}(m)} \sum_{i \in I_{\hat{n}(m)}} T_m(i) \tag{6.16}$$

Where $\hat{n}(m)$ is the number of runs where at least m agents find the target and $I_{\hat{n}(m)}$ corresponds to the indices where the times occur. The ramifications of this definition are explained in the context of the simulation in Section 6.2.4.

6.2.3 Alternative Search Strategies for Comparison

The searching algorithm can be compared with other common search strategies to evaluate its performance. One of the simplest search strategies is the simple raster scan. This involves the agents moving in a north/south or east/west lines until the boundary of the search is reached. The agent then moves over by one row and then turns around. And example of a raster trajectory is shown in Figure 6.8.

Another heuristic search strategy that is related to the raster scan is the "lawn mower" algorithm. It is referred to as a lawn mower algorithm because the agents follow a set of heuristics which might be similar to a person mowing a lawn. This follows a simple strategy

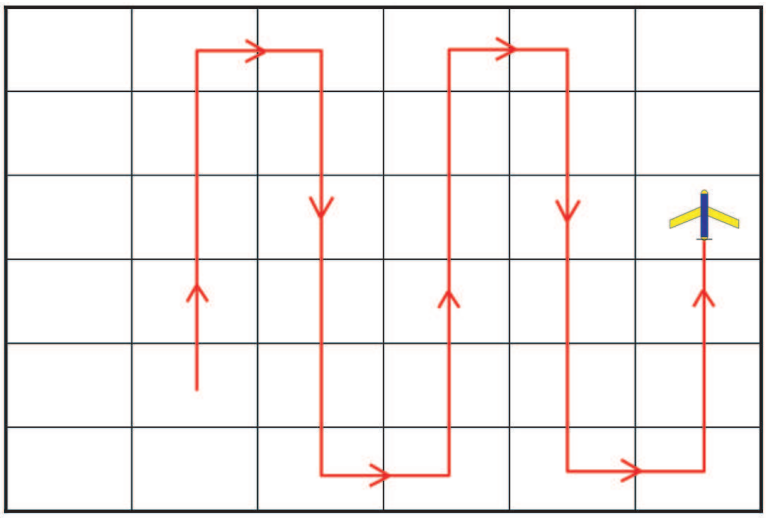


Figure 6.8: Example raster scan trajectory.

outlined in Figure 6.9.

Another simple search strategy is the gradient climb algorithm. In this case, the algorithm evaluates the scores of the cells surrounding it (to the north, east, south and west) and then chooses the cell which has the highest score. If two or more cells have the same maximum score, the algorithm chooses the cell which requires the smallest course change to visit. This is similar to a steepest ascent algorithm [22], [99] where the feasible directions are only the four cardinal directions.

Finally, the Voronoi partitioning method has been used by several groups to generate coverage algorithms which can be applied in this situation [42], [68]. In this case, the Voronoi diagram $\overline{V}$ is generated using the agent's positions as generators. Then the point for the agent to search within the next d steps is chosen to be within the agent's Voronoi polygon and its reachable set. This algorithm is outlined in Figure 6.10.

For purposes of comparison, the full algorithm refers to the algorithm described in Section 5.1 (no explicit cooperation between agents) and the full algorithm with Voronoi partitioning refers to the algorithm described in Section 5.5 (algorithm with explicit cooperation between agents).

```
1   if cell in front is not obstacle
2       move forward
3   else if cell to right is not obstacle
4       move right
5   else if cell to left is not obstacle
6       move left
7   else if cell to rear is not obstacle
8       move backwards
9   else
10      do not move
11  end
```

Figure 6.9: Pseudo code for lawn mower algorithm.

```
1   Generate Voronoi diagram of system using agent locations as generators.
2   if V(z_agt_i) ⋂ B_Ri ≠ ∅
3       Choose z_i* ∈ V(z_agt_i) ⋂ B_Ri
4   else
5       Choose z_i* = z_agt_i
6   end
7   Generate linear path from z_agt_i to z_i*
```

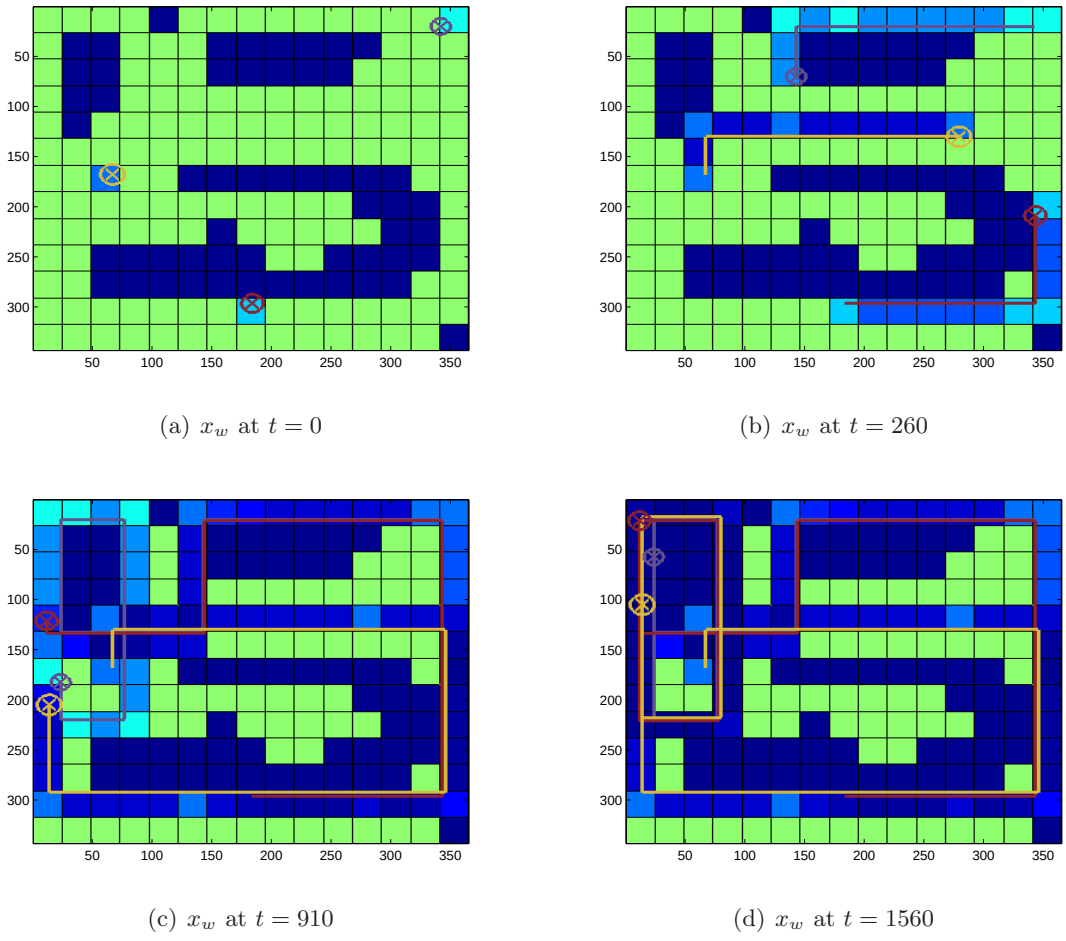
Figure 6.10: Pseudo code for randomized Voronoi partitioning algorithm.

6.2.4 Comparison Results

The various comparison algorithms are used in the three different scenarios. The previously mentioned performance metrics can then be used to judge each algorithm's efficiency.

Map Coverage

When evaluating the map coverage by the different strategies, the relevant quantities to analyze are the cumulative map scores and other related metrics. These scenarios involve multiple agents in the team searching the map with no targets. For example, the lawn mower trajectories for scenario 2 are shown in Figure 6.11.



(a) x_w at $t = 0$

(b) x_w at $t = 260$

(c) x_w at $t = 910$

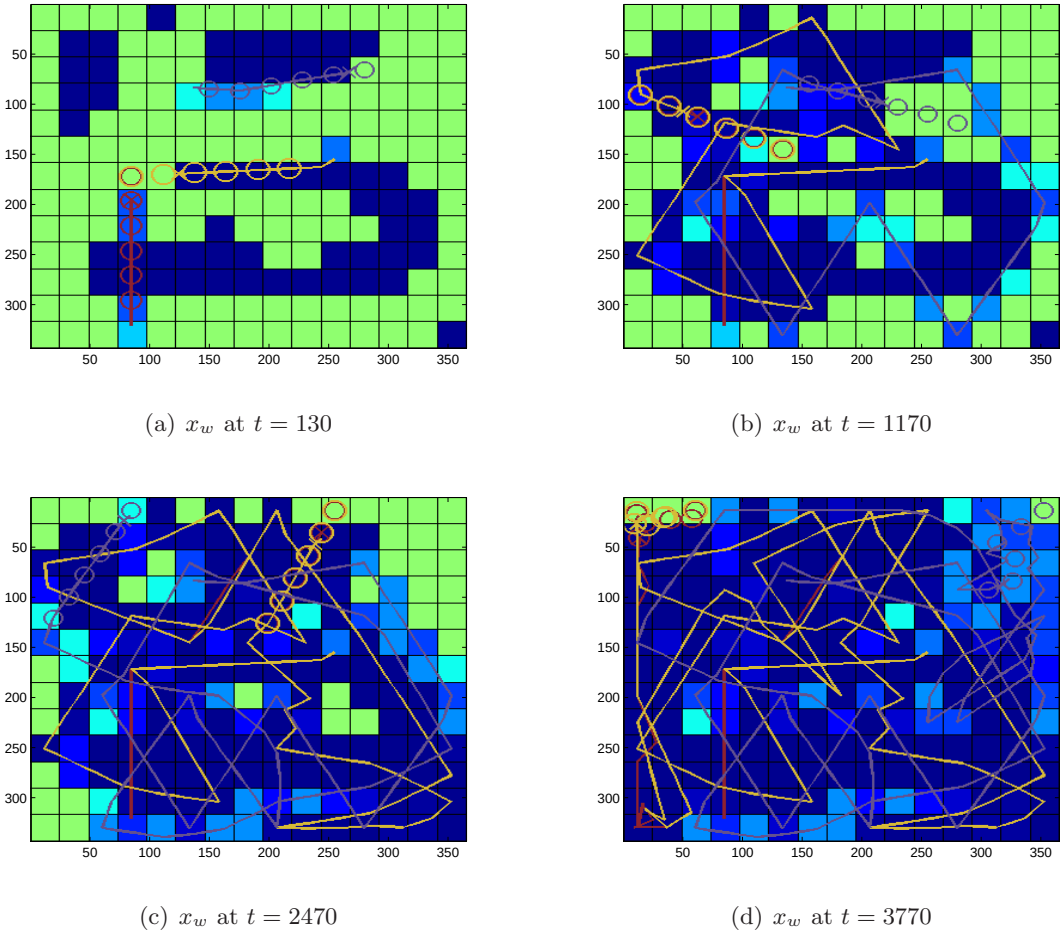
(d) x_w at $t = 1560$

Figure 6.11: Lawn mower trajectories for 3 agents with scenario 2.

The deficiency in this algorithm becomes immediately clear. The map is not exhaustively searched and furthermore the agents tend to become stuck in limit cycles where they continually search the same cells over and over again. No information about the map or

the state of the environment is taken into account and therefore, the algorithm performs poorly.

The same agents in the same scenario are instead controlled by the full algorithm and the results are significantly improved as shown in Figure 6.12.



(a) x_w at $t = 130$

(b) x_w at $t = 1170$

(c) x_w at $t = 2470$

(d) x_w at $t = 3770$

Figure 6.12: Full algorithm trajectories for 3 agents with scenario 2.

As can be seen, the map is covered completely in this scenario. The improvement of the full algorithm over the lawn mower algorithm in this scenario is evident when looking at traces of $S(i,k)$ and $V(i,k)$ as shown in Figure 6.14. From this one can see that the agents under the lawn mower strategy become stuck in a limit cycle at around $t = 1500$ and no new information is gained in the search. Furthermore, the value of $S(i,k)$ is not driven

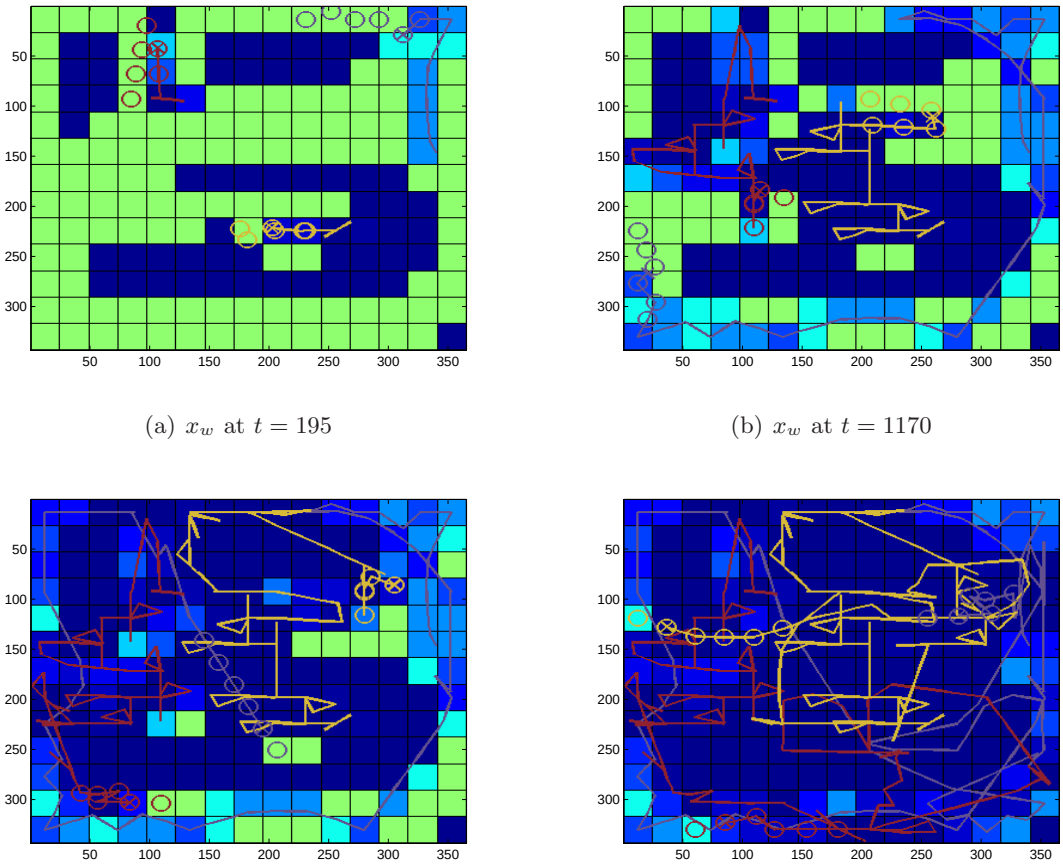
towards zero using the lawn mower strategy as it converges to a value of approximately 37. Conversely, under the full algorithm policy, the cumulative map scores are driven towards zero and if the simulation had be allowed to run for a longer period of time, they would approach zero, thus showing that the map is exhaustively searched. Also note that the variances of the scores are also driven to zero under the full algorithm.

An interesting phenomenon occurs with the full algorithm in this situation. Notice that in Figure 6.12(b), both the red and yellow agent choose the same location $\overline{z}^{\star}$ as the solution to ($\wp_2$). They both plan paths to this location and then after these paths expire, they continue to choose the same point as the optimal cell to search in the next d steps. Since they both use the same path planning algorithm, they both continue to stay on top of each other, effectively acting as one agent. Theoretically, since they all have different values of α, β, γ, and δ, it is possible that at some later point, they may choose different points and split apart from each other (although this does not occur in this example). This problem is exacerbated by the fact that the staggered timing method shown previously in Figure 6.3 is not used. This phenomenon provides motivation for the modifications to the algorithm which allow for explicit cooperation between the agents as described in Section 5.5. The usage of the Voronoi partitioning does not allow the agents to occupy the same space as each other and therefore, this type of behavior will not occur as seen in Figure 6.13.

The cumulative map scores and their variance for the different algorithms for scenario 2 for these selected runs are shown below in Figure 6.14.

Notice that in this scenario, it appears that the gradient climb algorithm outperforms the other methods. This occurs for several reasons. One reason being that the areas to be searched are connected and also because these are single runs and do not give a good representation of the behavior of the respective algorithms in general. In order to judge the general behavior of the algorithms, a series of Monte Carlo simulations are used. In these simulations, the performance of each algorithm is gauged over a series of 20 runs and then averaged using Eq. 6.3. The best and worst case scenarios for the series of runs are also computed using Eq. 6.6 and Eq. 6.5, respectively. The results for scenarios 1, 2, and 3 are presented in Figures 6.15, 6.16, and 6.17, respectively.

In Figures 6.15 through 6.17, the solid line represents $S_{ave}(k)$ and the dashed line repre-



(a) x_w at $t = 195$

(b) x_w at $t = 1170$

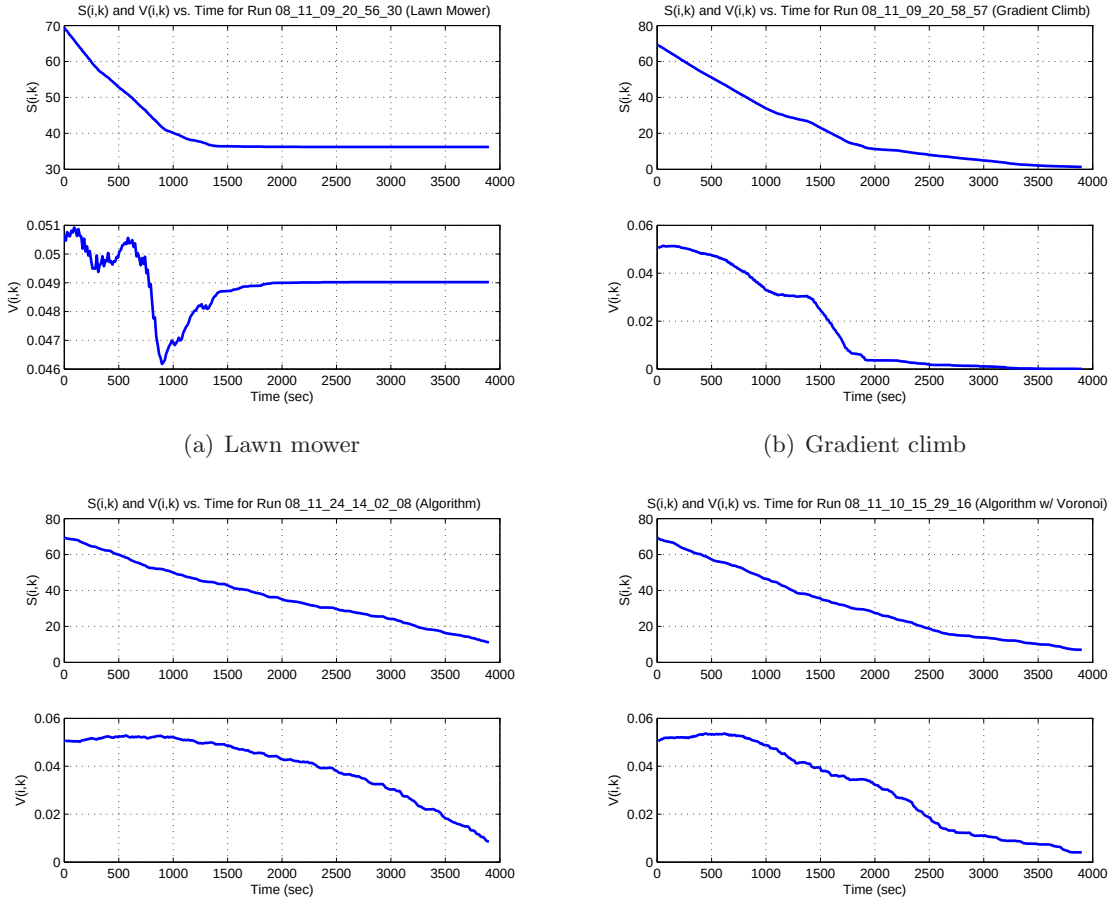
(c) x_w at $t = 2145$

(d) x_w at $t = 3705$

Figure 6.13: Full algorithm with Voronoi partitioning trajectories for 3 agents with scenario 2.

sents $S_{max}(k)$ for the corresponding search strategy. The performance of the various search strategies using $S_{ave}(k)$ and $S_{max}(k)$ as metrics is summarized in Table 6.2. In this table, 1 corresponds to the best performance and 6 corresponds to the worst performance.

Looking at average performance measured by $S_{ave}(k)$, it can be seen that in scenarios 1 and 3, the performance of the algorithms from worst to best appears to be, lawn mower, randomized Voronoi, raster scan, gradient climb, full algorithm, then full algorithm with Voronoi partitioning. This is the expected result and shows that the best performance and guarantee of map coverage is achieved with the full algorithm. Furthermore, it shows



(a) Lawn mower

(b) Gradient climb

(c) Full algorithm

(d) Full algorithm w/ Voronoi partitioning

Figure 6.14: $S(i,k)$ and $V(i,k)$ for scenario 2.

Table 6.2: Rankings of search strategies using $S_{ave}(k)$ and $S_{max}(k)$ as metrics (1 = best).

	$S_{ave}(k)$			$S_{max}(k)$		
Strategy	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**
Lawn Mower	6	6	6	6	6	5
Randomized Voronoi	5	5	5	4	4	4
Raster Scan	4	4	4	5	5	3
Gradient Climb	3	1	3	3	2	6
Full Algorithm	2	3	2	2	3	2
Full Algorithm Co-op	1	2	1	1	1	1

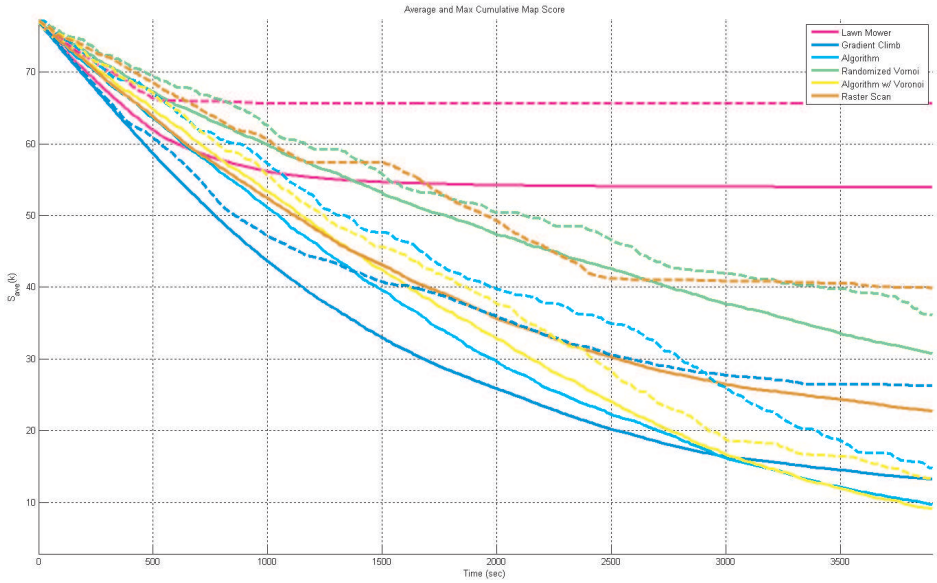


Figure 6.15: $S_{ave}(k)$ and $S_{max}(k)$ for scenario 1.

that the performance is further increased (and the coverage guarantee is preserved) when augmenting the full algorithm with explicit cooperation between agents through the Voronoi partitioning. It should be noted that if the simulation were run for a longer amount of time, it is expected that the raster scan algorithm will eventually outperform the gradient climb when using $S_{ave}(k)$ as the metric for map coverage. This is most evident by looking at Figure 6.17. Map coverage is guaranteed with the raster scan algorithm, but it is obvious that the performance is suboptimal. Note that in scenario 2, it appears that the gradient climb algorithm performs the best. As mentioned previously, this occurs because the areas to be searched are connected and the environment is fairly simple. If the environment was comprised of long, narrow corridors, the gradient climb algorithm would perform poorly due to the fact that it would not cross over areas of low score whereas the full algorithm would (see Figure 5.21).

The guarantees of map coverage are more evident when looking at the worst case scenario for map coverage. Recall that $S_{max}(k)$ is a measure of the worst case scenario possible over

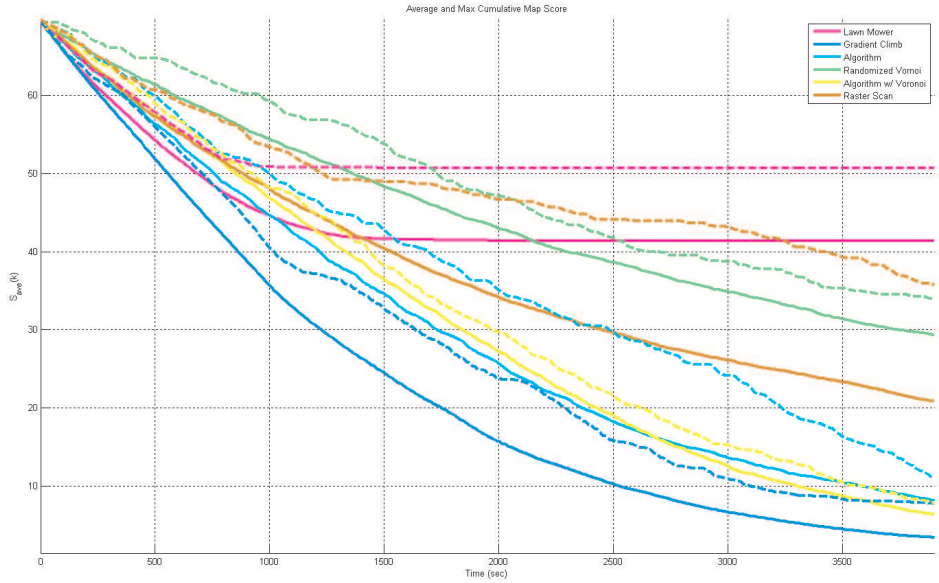


Figure 6.16: $S_{ave}(k)$ and $S_{max}(k)$ for scenario 2.

all test cases. In this case, it is obvious that the full algorithm with explicit cooperation is the best policy to use. Although the gradient climb strategy may work well for some situations, there are situations where it performs the worst out of all the possible strategies ($S_{max}(k)$ for scenario 3).

The average variance of the runs can be computed using Eq. 6.4 and the results are shown in Figure 6.18. These supply further proof that an exhaustive map search is guaranteed only using the raster scan, full algorithm, and full algorithm with explicit cooperation strategies. Of these guaranteed methods, the latter two provide superior performance.

A similar trend of performance is observed using the information theoretic metrics of $H(Q^s)$ and $H(\nu^s)$. For brevity, only the plots for scenario 3 are both partitions are shown in Figures 6.19 and 6.20. The performance results are tabulated in Table 6.3. Once again, it is clear that the full algorithm and the full algorithm with explicit cooperation yield the best performance since under both partitions, the information approaches zero as the agents exhaustively search the domain.

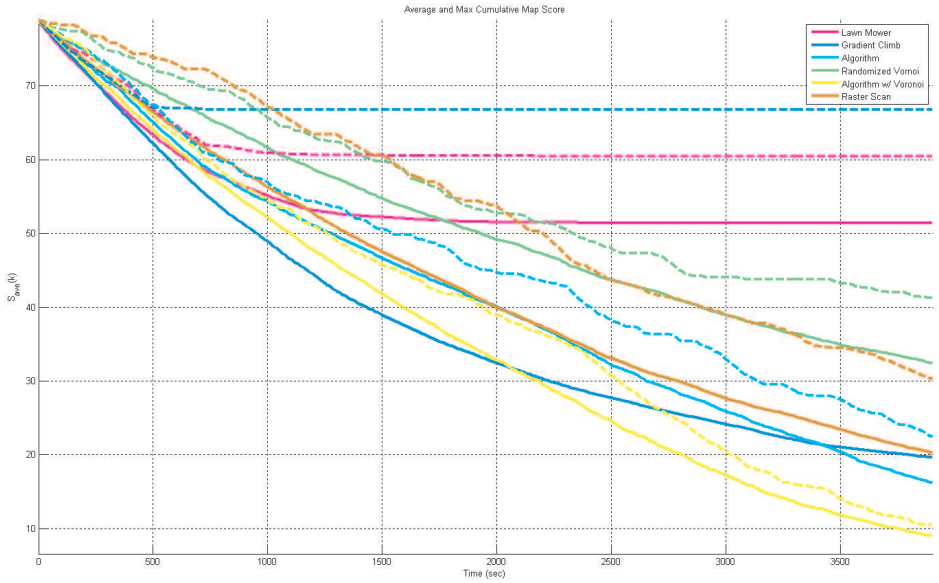
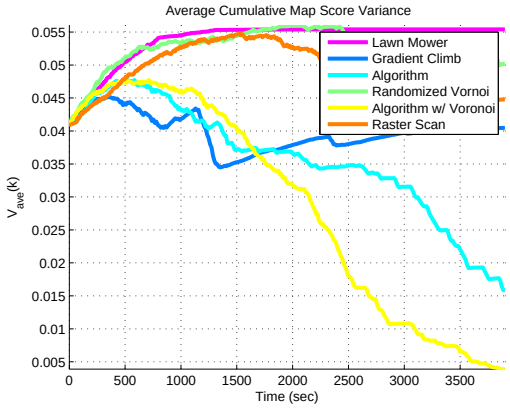


Figure 6.17: $S_{ave}(k)$ and $S_{max}(k)$ for scenario 3.

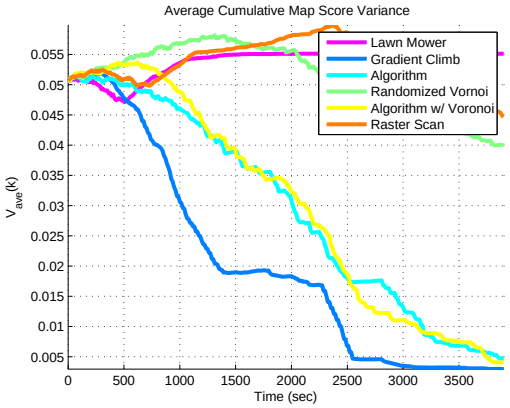
Table 6.3: Rankings of strategies using $H_{ave}(Q^s)$ and $H_{max}(Q^s)$ as metrics (1 = best).

	$H_{ave}(Q^s)$			$H_{ave}(\nu^s)$			$H_{max}(Q^s)$			$H_{max}(\nu^s)$		
Strategy	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**
Lawn Mower	6	6	6	6	6	6	6	6	6	6	6	6
Randomized Voronoi	5	5	5	5	5	5	4	4	4	5	5	5
Raster Scan	4	4	4	4	4	3	5	5	3	4	4	3
Gradient Climb	3	1	3	3	1	2	3	3	5	3	2	4
Full Algorithm	2	3	2	2	3	4	2	2	2	2	3	2
Full Algorithm Co-op	1	2	1	1	2	1	1	1	1	1	1	1

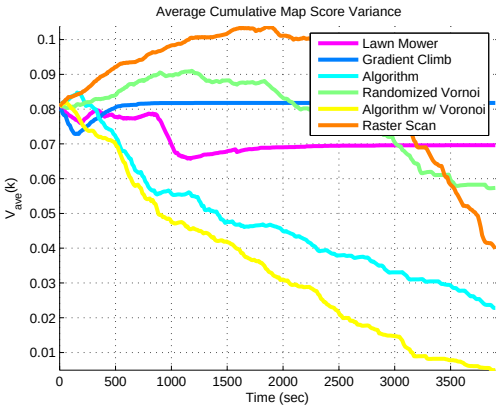
Improving performance in a searching mission typically involves increasing the number of agents involved in the search. The main challenge with current unmanned systems is that raising the number of agents greatly increases operator workload required to manage the team. If the team is comprised of heterogeneous agents with different capabilities, the mission management task becomes even more complicated. The benefits of the full algorithm



(a) Scenario 1



(b) Scenario 2



(c) Scenario 3

Figure 6.18: $V_{ave}(k)$ for scenarios.

both with and without explicit cooperation within this framework can be seen when looking at the coverage vs. time for varying number of agents. For example, the effects of varying the number of homogeneous agents with the full algorithm and full algorithm with explicit cooperation strategies are shown shown in Figure 6.21(a) and Figure 6.21(b), respectively.

Figure 6.21 displays several interesting phenomena. First, the guarantee of exhaustive map searching is reinforced. Also, the effect of increasing the number of agents involved in the search is evident since in both cases, the time to drive the map scores to zero decreases as the number of agents increase. The effect of increasing the number of agents in the team can be investigated by calculating the settling time for the coverage metric. In this context,

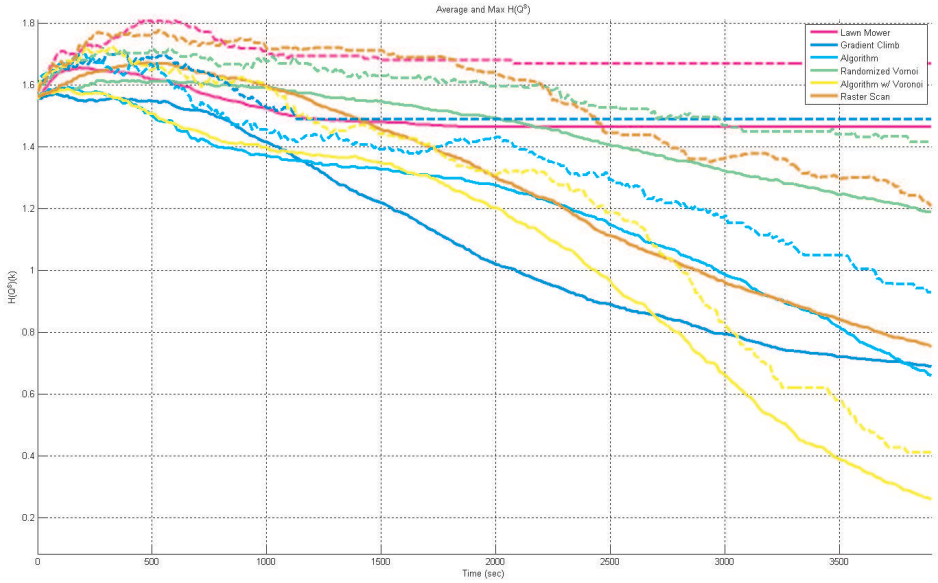


Figure 6.19: $H_{ave}(Q^s)$ and $H_{max}(Q^s)$ for scenario 3.

the settling time is defined as the time it takes for $S_{ave}(k)$ to drop below to 15% of the initial cumulative map score. The settling time vs. number of agents is shown in Figure 6.22.

As expected, the settling time decreases as the number of agents increases. Notice that the settling time does not decrease in a linear fashion. Instead, the relative increase in performance decreases with each successive addition of a team member. In other words, adding more agents to the team does not greatly increase performance after a certain point. The trend is roughly exponential and the fitted exponential function is shown with the data as the solid lines. Note that the exponential decay rate for the full algorithm with explicit cooperation is $\lambda = -0.274$ which decays faster than the exponential decay rate for the full algorithm of $\lambda = -0.235$. The modifications to the algorithm to accommodate explicit cooperation benefits the team more as the number of agents in the team increases. This can be attributed to the Voronoi partitioning which helps ensure that agents do not overlap and search the same cells as the mission progresses.

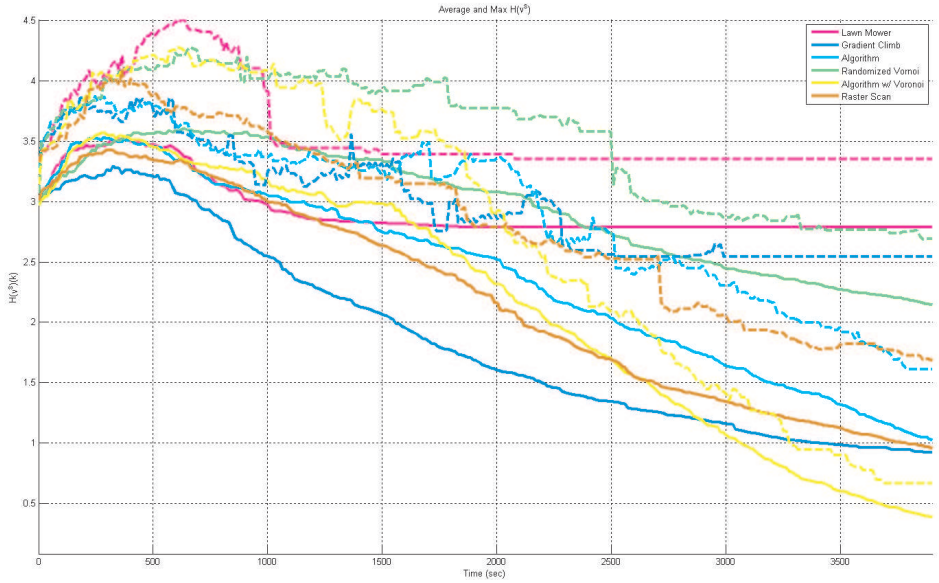
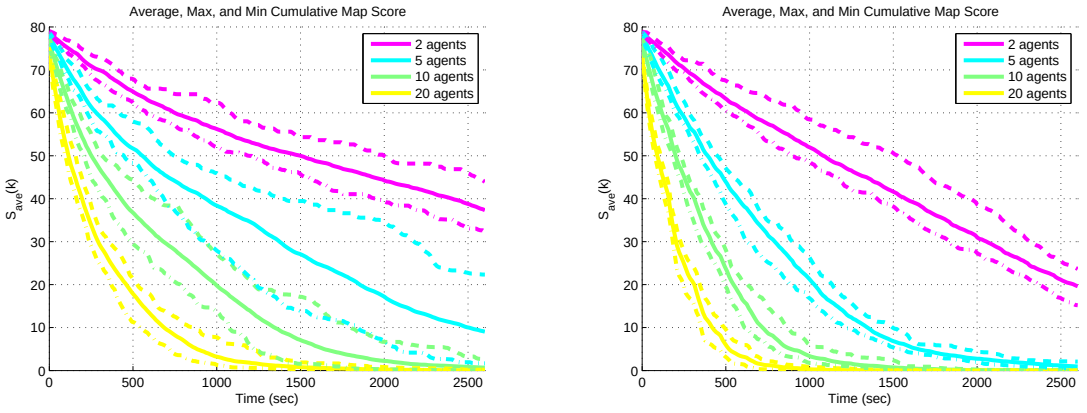


Figure 6.20: $H_{ave}(\nu^s)$ and $H_{max}(\nu^s)$ for scenario 3.



(a) Full algorithm

(b) Full algorithm with explicit cooperation

Figure 6.21: Coverage metrics for varying number of agent using scenario 3.

Time to Target Detection

There are several parameters that measure the efficiency of the algorithm in terms of target detection time. One metric is the average number of agents that find the target for a given

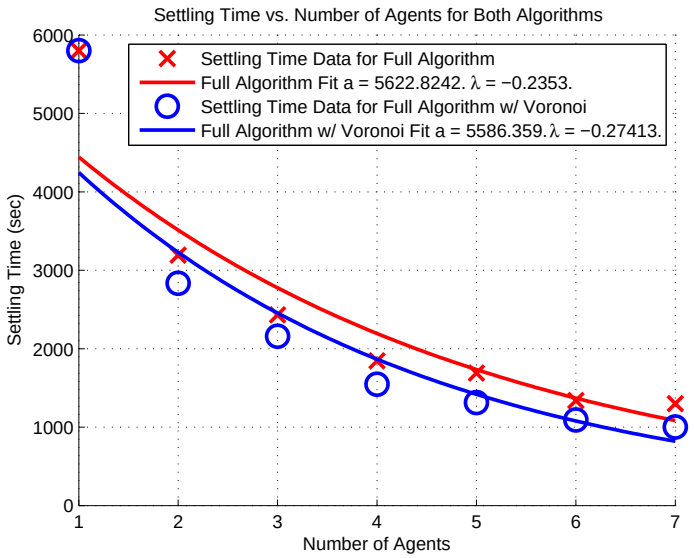


Figure 6.22: Settling time for various number of agents using full algorithm and full algorithm with explicit cooperation.

scenario. Related to this metric is the number of scenarios where at least 1, 2, or 3 agents find the target. All runs for the time to target detection Monte Carlo simulation use 200 time steps with 3 agents. For example, the results from scenario 2 using the full algorithm with explicit cooperation are shown in Figure 6.23.

As can be seen, for almost all of the runs, at least 1 agent finds the target (run 10 is the only run where the target was not found within 200 time steps and is represented by the 'x') and the times when the first agent finds the target are shown on the vertical axis of the first subplot in Figure 6.23. The scenarios where at least two agents find the target can be found in the second subplot of the same figure. The circles on this subplot correspond to the time when the second agent finds the target. The same interpretation is used to explain the meaning of the third subplot.

An interesting phenomenon occurs in this situation. Note that these results were generated with agents using the full search algorithm with explicit cooperation as described in Section 5.5. Recall from Theorem 5.5.6 that the solution to ($\wp_2$) for each agent is guar-

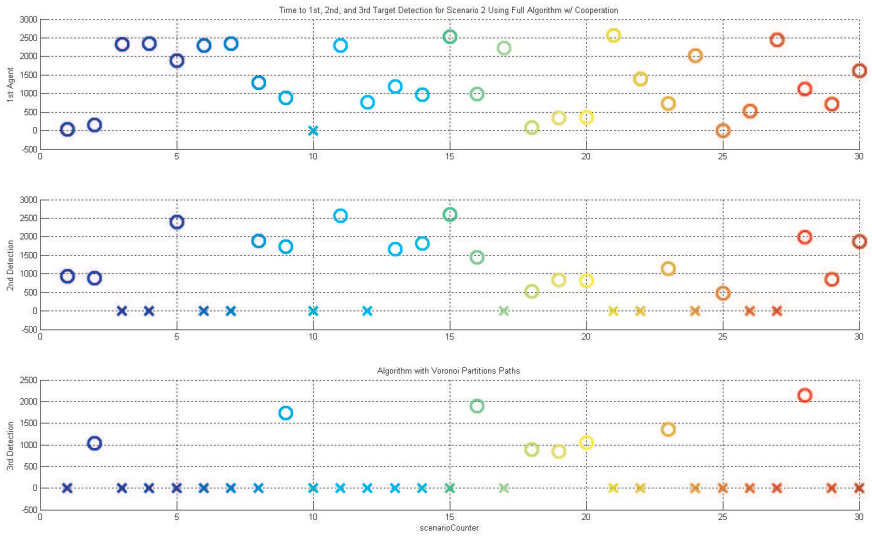


Figure 6.23: Time to 1st, 2nd, and 3rd target detection for scenario 2 using full algorithm with cooperation. 'x' = not applicable for this scenario.

anteed to be both reachable and is contained within its own Voronoi cell. When the first agent finds the target, it increases the scores locally around the target and stops at the cell center which contains the target. Therefore, it is impossible for another agent to choose this same cell as the solution to its corresponding ($\wp_2$). However, the graph-based path planning method described in Section 5.4.2 is used to solve ($\wp_3$) for all agents, and it is possible that the second agent will still find the target because the path may stretch to cover the target's cell (similar to the behavior shown in Figure 5.21(c)). This is why in 18 runs, at least 2 agents find the target. This also shows why in 8 runs, all 3 agents find the target. This can be contrasted with the randomized Voronoi search strategy. The same reasoning applied to this search scenario will show that it is nearly impossible for more than one agent to find the target because each agent remains inside its own Voronoi polygon and once a single agent finds the target and stops, it will create a repulsive effect for the other agents.

The values of $\tilde{N}_{ave}$, $\hat{n}(1)$, $\hat{n}(2)$, and $\hat{n}(3)$ for 30 runs are summarized in Table 6.4.

The easiest metric to interpret is the average number of agents that find the target.

Table 6.4: Average number of agents which find target and number of scenarios where at least 1, 2, or 3 agents finds the target (30 scenarios each).

	Average # Agents Which Find Target $\tilde{N}_{ave}$			# Runs w/ 1 Encounter $\hat{n}(1)$			# Runs w/ 2 Encounters $\hat{n}(2)$			# Runs w/ 3 Encounters $\hat{n}(3)$		
Strategy	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**
Lawn Mower	0.267	0.367	0.233	5	8	5	3	2	2	0	1	0
Randomized Voronoi	0.467	0.433	0.633	14	13	19	0	0	0	0	0	0
Raster Scan	1.000	1.033	1.200	20	23	23	9	8	11	1	0	2
Gradient Climb	1.400	2.067	1.500	23	27	19	13	24	16	6	11	10
Full Algorithm	2.567	2.400	1.500	28	24	16	26	24	15	23	24	14
Full Algorithm Co-op	1.833	0.433	0.867	24	29	14	17	18	10	4	8	2

This is shown in the first three columns of Table 6.4. As can be seen, the full algorithm performs the best in all scenarios in terms of getting the most agents to find the target. The full algorithm with explicit cooperation does not perform as well for the reasons previously stated. These same reasons explain why the randomized Voronoi search strategy has no runs where more than one agent finds the target.

It is also worth noting that the raster scan method does not appear to be affected by the different scenarios. This makes sense considering that information about the environment is not taken into account when using this method. It should be noted that in scenario 3, many of the algorithms which use information about the environment (gradient climb, full algorithm, full algorithm with explicit cooperation) show a decrease in performance. This is because the initial world map (shown previously in Figure 6.6(c)) has several regions of high score. These would correspond to a priori knowledge of possible target locations. Despite this being the initial state of the world, the locations of the targets are placed according to a uniform distribution across the map. In other words, the initial target location is not distributed according to the distribution shown in Figure 6.6(c). This simulates a situation where the agents are given inaccurate a priori knowledge before starting the mission, thus leading to a decrease in performance. To simulate a situation with accurate information, the initial target location would need to be distributed according to the distribution shown

in Figure 6.6(c) [112].

The columns for $\hat{n}(m)$ show the number of runs where m agents find the target. Once again, it is apparent that methods which use information about the environment perform the best.

Information regarding the average amount of time required to find the target are displayed in Table 6.5.

Table 6.5: Average time to target detection by first, second, and third agent (30 scenarios each).

	Average Time for 1st Encounter $T_{1,ave}$			Average Time for 2nd Encounter $T_{2,ave}$			Average Time for 3rd Encounter $T_{3,ave}$		
Strategy	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**
Lawn Mower	143	322	143	500	774	501	N/A	754	N/A
Randomized Voronoi	1114	845	1115	N/A	N/A	N/A	N/A	N/A	N/A
Raster Scan	1045	807	1045	1797	1412	1798	1989	N/A	1989
Gradient Climb	1115	814	1116	1222	1316	1223	1481	1715	1481
Full Algorithm	844	915	844	904	1143	904	1238	1264	1238
Full Algorithm Co-op	1054	1326	979	1526	1470	1791	1703	1370	2015

The results in Table 6.5 are not as good of a representation of performance as those shown in Table 6.4. The reason can be seen from the definition of $T_{m,ave}$ in Eq. 6.16. These averages are computed only for the situations where the agents find the target. For example, with the lawn mower strategy with scenario 1, from Table 6.4, it can be seen only 5, 3, and 0 runs out of 30 show at least 1, 2, or 3 agents finding the target, respectively. Therefore, the averages $T_{1,ave}$, $T_{2,ave}$, and $T_{3,ave}$ are computed using only 5, 3, and 0 samples (explaining why $T_{3,ave}$ = N/A). Using the lawn mower strategy, the agents tend to become stuck in limit cycles and therefore, if the agent is going to find the target, it is will happen very quickly or not at all. This shows why the values of $T_{1,ave}$ and $T_{2,ave}$ are low. This can be compared with the full algorithm where it is virtually guaranteed that the agents will find the target but it will take a longer amount of time to do so.

Furthermore, notice that it is not required that $T_{1,ave} \leq T_{2,ave} \leq ... \leq T_{M,ave}$. For

example, for the full algorithm with explicit cooperation using scenario 2, $T_{3,ave} < T_{2,ave}$. Once again, this is because the average is computed over a different number of samples. The reason why this occurs can be seen by examining Figure 6.23.

Information Gain

The final metric to apply is the information theoretic based metrics defined in Eq. 6.14. In order to calculate the information gained during a search mission, an objective perspective of the environment, denoted ν^o, is required. In this situation, ν^o represents the desired state of the world. In this situation, it is desired that the agents drive the state of the occupancy based map to a situation where the scores of all cells are 0 except for the score of the cell containing the target, which should have a score of exactly 1. This corresponds to a unimodal distribution where the location of the target is known exactly. A more realistic view of the world is one where the occupancy map is driven to a Gaussian distribution with mean at the location of the target and a variance similar to what the agents would produce via Eq. 4.10. This is used in simulation to generate ν^o.

With ν^o defined, the utility of Eq. 6.14 becomes evident when applied to a situation where the agents find the target. The information metric during a run using the full algorithm as the search policy is shown in Figure 6.24.

In Figure 6.24, the vertical dashed lines denote the time when the nth agent encounters the target. As can be seen, events of finding the target are directly reflected in the information gained by the team which is measured by $I(\nu^o, \nu^s)$.

Similar to the other scenarios, the average and minimum values of $I(\nu^o, \nu^s)$ can be determined over several runs. These represent the average and the worst case scenarios for information gain. The results of these are shown in Figure 6.25. As expected, the most information appears to be obtained using the full algorithm (both with and without explicit cooperation).

The information matric is useful because it can be used to measure the value of activities other than searching. For example, the information gained by remaining in one place to track a possible target using another type of algorithm [60], [103] can be determined and

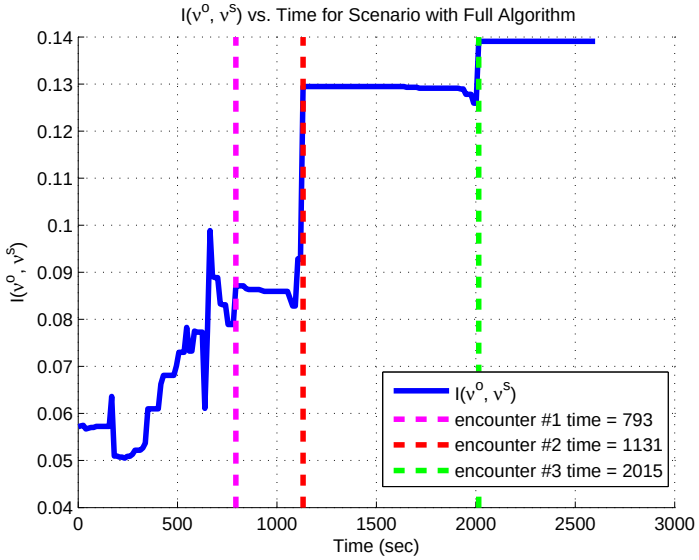


Figure 6.24: $I(\nu^o, \nu^s)$ vs. time for scenario with full algorithm strategy. Bars denote times that targets are detected.

compared to the amount of information being gained by performing the search. This leads to the explore vs. exploit paradigm [14] which can be evaluated in this context.

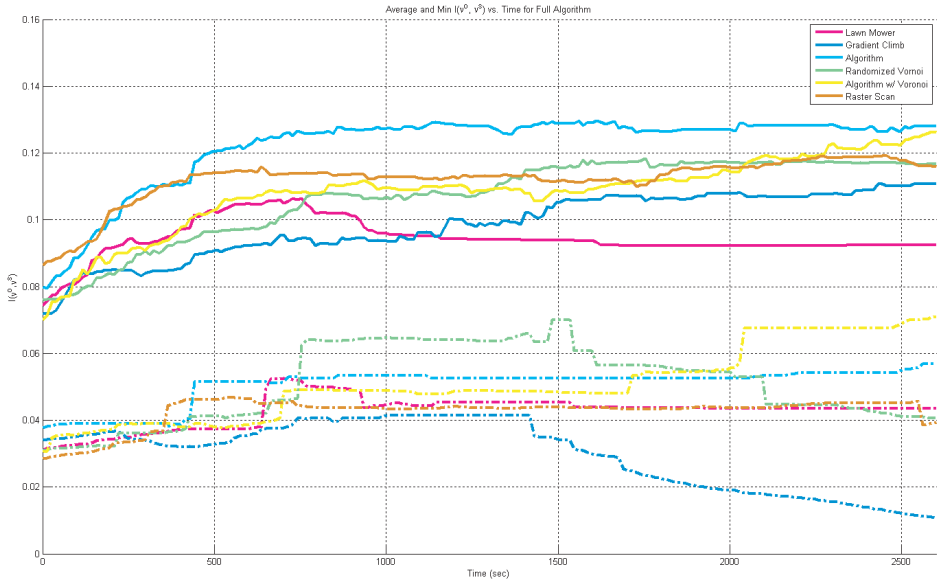


Figure 6.25: $I(\nu^o, \nu^s)_{ave}$ and $I(\nu^o, \nu^s)_{min}$ for scenario 3.

Chapter 7

HUMAN-IN-THE-LOOP SIMULATION

A large part of the research in this dissertation is directed toward the development of algorithms that govern the actions of a single UAV or a team of UAVs at an abstract level. These algorithms apply to other unmanned vehicles as well. These algorithms govern high level behaviors such as task and path planning, but we assume that low level algorithms already exist to handle tasks such as state stabilization and signal tracking.

The end goal for many of these algorithms is fully autonomous behavior without input from human operators. However, there are many benefits for allowing human interaction with the system. Verifying and validating a fully autonomous algorithm through an unmanned flight test requires significant logistical planning and development. Many other UAV subsystems that are not directly related to the core strategic algorithm must be developed in order to support the mission. Furthermore, these autonomous flight tests must be conducted in controlled airspaces and under strict supervision. All of these factors greatly increase complexity and development time of the overall system. Many of these problems can be addressed simply by introducing human decision making and interaction at very specific points in the system.

This chapter investigates a ground based, distributed testing environment which is used at the Autonomous Flight Systems Laboratory to test high level algorithms by using a human operator in place of several low level systems. In this fashion, the overall system operates in a manner very similar to the fully autonomous system. This approach offers many benefits. The main advantage is that the high level algorithms can be implemented and tested much faster and with significantly less effort. In addition, applications can be developed for a standard Windows based environment instead of embedded real time systems. Furthermore, the algorithms can be tested in unrestricted airspace due to the fact that they are operating as a pilot-assistance system which the pilot is free to ignore at any

point.

The main drawback with this architecture is the introduction of a human operator or pilot who may behave in a non-deterministic fashion. To alleviate this problem, a distributed ground based simulator is used to train potential pilots to interact efficiently and consistently with the system.

7.1 Distributed Simulator

The simulation application described in Chapter 6 is designed to interact with additional hardware and software components that make up a distributed simulator. This distributed simulator delegates low level control algorithms to a human operator while leaving the strategic and tactical algorithms as autonomous algorithms.

7.1.1 Fully Autonomous Architecture

For a fully autonomous system to operate successfully, control laws must be designed and implemented to address all levels of autonomy. An example of a typical setup for a fully autonomous system is shown in Figure 7.1.

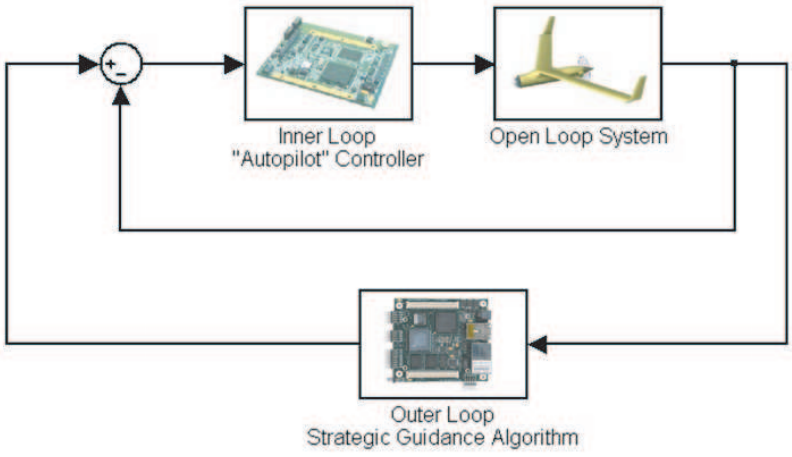


Figure 7.1: Desired system architecture for fully autonomous flight.

Low and middle level control tasks are handled by an inner loop controller and the

strategic algorithms are responsible for more abstract, higher level mission management tasks. This type of architecture works well to partition the workload and assign tasks to the controllers. Both the inner and outer loop control laws may be implemented using onboard microprocessors or embedded controllers [11].

Although this architecture is efficient, the main drawback is the fact that although the inner loop control laws can be designed and tested independently from the outer loop controller, the reverse is not true. A valid inner loop must be implemented in order to verify and validate the outer loop. Furthermore, the inner loop is vehicle specific and must be designed and implemented individually for each vehicle. To further complicate matters, often these inner loop applications must be developed to run on embedded controllers, thus requiring specialized training and development environments.

7.1.2 Human-in-the-Loop Architecture

Many consider low level control problems such as state stabilization and signal tracking to be mature technologies with limited academic research currently focused on these problems. However, verification and validation of strategic and tactical control algorithms is currently an active field of research [71].

Flight Test Architecture

The architecture used in this work for semi-autonomous flight tests that replaces the fully autonomous UAV system with a human pilot and flight vehicle is shown in Figure 7.2.

In this setup, the strategic control laws are implemented on a standard laptop PC (see Section 7.2.2). The tasks that are usually handled by the inner loop controller are instead managed by the human pilot. The outer loop relays information to the inner loop (pilot) by displaying pertinent information to a second laptop (called the visualization laptop). The visualization and strategic laptops are connected via software called OPC DataHub (see Section 7.2.1).

There are several advantages to this architecture. The most obvious is that this setup avoids the significant time and effort required to develop a viable inner loop controller for the

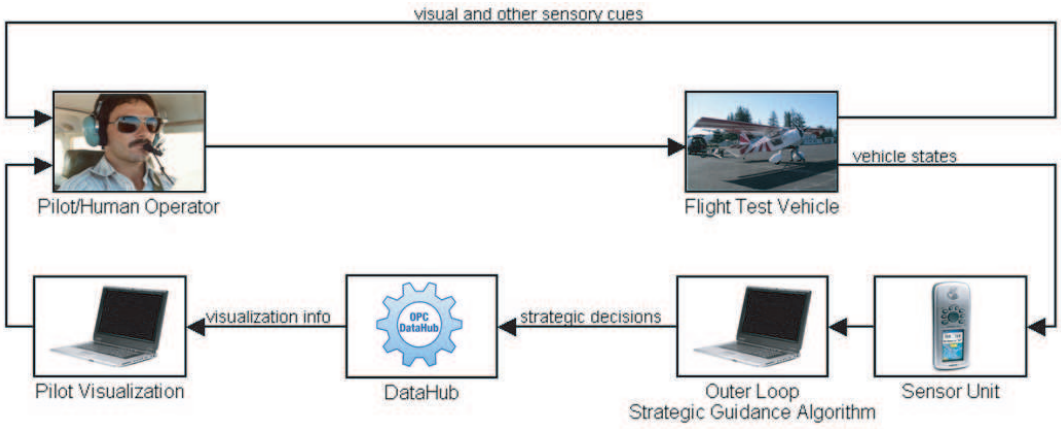


Figure 7.2: System architecture for human-in-the-loop flight test of strategic controller.

vehicle. Instead, the human pilot operates as the inner loop controller by taking commands from the outer loop (strategic guidance algorithms). This is similar to the difference between driving a car assisted by GPS navigation and developing a fully autonomous car capable of navigating via GPS.

This setup allows the strategic algorithms to be developed using a standard development environment such as Microsoft Visual Studio or Matlab since the application will be implemented on a laptop PC running a Windows operating system. This further saves time and effort because it is not necessary to port algorithms to an embedded real time operating system (RTOS).

In addition, this architecture allows vehicles using these strategic algorithms to operate in normal airspace with little or no safety problems (the pilot can choose to ignore commands from the strategic algorithm at any time).

Distributed Simulation Architecture

In order to accurately validate the outer loop controller, the inner loop controller must behave in a reliable, deterministic fashion. In other words, the human operator must interface with the system in a predictable manner. One popular method for designing inner loop controllers is to use neural networks [59]. Neural networks have been some of the earliest

and most studied models of human brain function [19]. The amount of error introduced by the human operator depends on the amount and type of practice that they receive (i.e. training sets for the network). The system architecture used in this work to train human operators for interfacing with the strategic algorithms is shown in Figure 7.3.

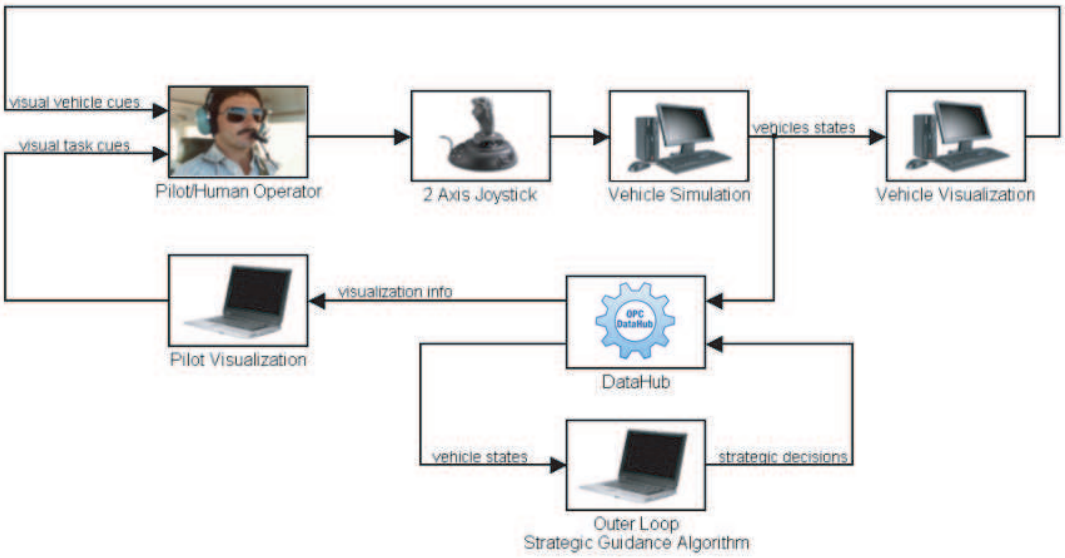


Figure 7.3: System architecture for ground based distributed human-in-the-loop simulation.

This setup mirrors the flight test architecture shown previously in Figure 7.2 except the flight test vehicle and sensor unit are replaced with their simulated counterparts. The vehicle is modeled as a 6 degree of freedom rigid body based on the Research Civil Aircraft Model [64]. The aircraft simulation is implemented on two separate desktop PCs. More information regarding the plant model implementation is found in Appendix C.

7.2 Hardware and Software

The human-in-the-loop distributed simulator is comprised of several software and hardware systems running in parallel. These systems and their functions are described below.

The physical setup of the distributed human-in-the-loop simulator is shown in Figure 7.5. In this system, the pilot interacts with the simulator via a three axis joystick and receives

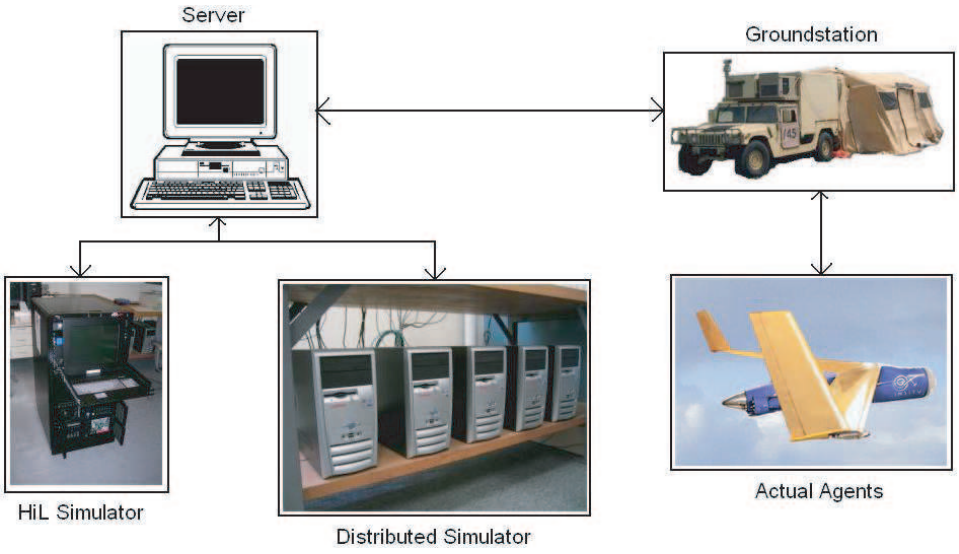
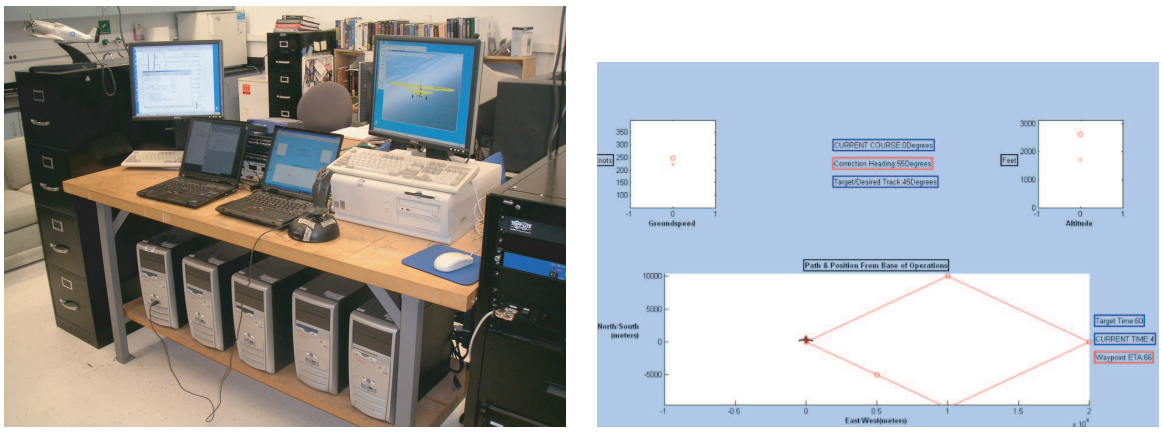


Figure 7.4: Multi-vehicle implementation using HiL and Distributed Computing Facility (DCF).



(a) Physical setup

(b) Pilot visualization

Figure 7.5: Physical setup of Distributed Human-in-the-Loop Simulator and screen shot of operator visualization.

visual cues regarding the state of the aircraft from the FlightGear [8] output. Information regarding the output of the strategic algorithm is displayed to the pilot via a dedicated laptop. A screenshot of the information conveyed to the pilot is shown in Figure 7.5(b). In this situation, the strategic algorithm is a path planner and outputs information regarding

the path that the pilot is required to fly.

The output consists of four separate screens. In the upper left, a display showing the desired groundspeed and the current groundspeed is displayed. In the upper right, a similar display is used for altitude tracking. The display on the bottom consists of a top view of the current path and the location and orientation of the agent. Directly above this wire frame drawing is a series of numbers indicating the current course angle, the desired course angle, and a correction course angle which will put the agent back on track if there is a non-zero cross track error [102].

7.2.1 Software

Software applications used in the distributed simulator are all implemented on Windows XP machines. The vehicle simulation is performed using Matlab Version 7.2.0.232 (R2006a) and Simulink 6. In addition, the AeroSim blockset version 1.2 [115] is used to interface to the two axis joystick with the Simulink model.

To distribute processing power, the vehicle visualization is handled by a separate desktop PC. This PC renders the vehicle in 3D using FlightGear v0.9.8. The AeroSim blockset is used to interface the Simulink model of the vehicle with this visualization tool.

Another crucial piece of software is the OPC DataHub. This is a centralized database with a publish/subscribe architecture where multiple applications can both write to and read from a centralized server. This software is used to transfer data between different applications. Many applications have tools for interfacing with the DataHub. The OPC Toolbox in Simulink allows signals from the Simulink model to be written to (or read from) the DataHub. The pilot visualization is also implemented using Matlab and Simulink and uses the OPC Toolbox to read relevant information from the DataHub.

The strategic algorithm is implemented as a standalone executable. In the Autonomous Flight Systems Laboratory, it is developed using the C++ language. This allows separate software objects to be dedicated to interfacing with the sensor and the DataHub.

Software applications used in the system are designed to be modular so that they can be developed and improved independently. Because multiple developers work on different

pieces of software at different times, the lab makes use of TortiseSVN to handle version control of code.

7.2.2 Hardware

The distributed simulator is made up of several hardware components as well. The main hardware is the series of networked desktop and laptop PCs. The various computers, the applications they host, and their functions are summarized in Table 7.1.

Table 7.1: Hardware and software components used by various machines in distributed simulator.

Type	**Applications**	**Function**
Desktop PC	Matlab Version 7.2.0.232 (R2006a), Simulink 6 with AeroSim version 1.2 blockset and OPC Toolbox, OPC DataHub version 6.3.14.166	Vehicle state and environment simulation and human interface system
Desktop PC	FlightGear v0.9.8	Vehicle visualization
Laptop PC	Matlab Version 7.2.0.232 (R2006a), Simulink 6 with OPC Toolbox, OPC DataHub version 6.1.9.133	Pilot Visualization and OPC DataHub server
Laptop PC	Strategic Algorithm	Strategic Algorithm carrier and sensor interface

The Desktop PC responsible for the vehicle simulation is also responsible for interfacing with the human operator. This is done via a three axis Microsoft Sidewinder joystick.

For flight tests, the vehicle and sensor model are replaced with an actual aircraft and sensor. These are described in Section 8.1.

7.3 Operator Results

Human pilots can be trained on this simulator to interact with the autonomous algorithms. The performance of the operators is also analyzed in this section using various scenarios and metrics.

7.3.1 Human Performance Metrics

The distributed simulator was developed in order to verify and validate the strategic team management algorithm described in Chapter 5. From an agent's perspective, the strategic algorithms generates paths for the agent to follow. For this example, the distributed simulator is used to train pilots to perform a path following task.

A feasible path for the agent consists of a sequence of waypoints ($x_i \in \Re^3$) where each consecutive waypoint is no more than a distance r_{max} away from the previous one. In addition, each of these waypoints specifies a time, t_i when the agent must arrive at the location x_i. An example of such a path was shown previously in Figure 5.23. In this example, the algorithm plans a path (shown in red) for the vehicle to navigate through the environment. Once the path is determined and displayed to the pilot, it now becomes the job of the pilot to follow this path to the best of their ability.

The skill and ability of the pilot is measured using a performance metric. The position of the simulated agent is recorded every Δt seconds and is denoted $x_{agt}(t)$. At each time t, the agent should ideally be located on the line segment joining waypoints x_i and x_{i+1} for some $i \in \{0, 1, \ldots, d-1\}$ (for the case of $i = 0$, $x_0 = x_{agt}(0)$). For convenience, the point x_i is referred to as x_A (the previous waypoint) and the point x_{i+1} is referred to as x_B (the next waypoint). x_A and x_B have respective time stamps t_i and t_{i+1}. If the agent is not on this line segment, the agent is off the path and the point $x_p(t)$ can be found which is the point on the line segment with minimum Euclidean distance from the agent's current location, $x_{agt}(t)$. The point $x_p(t)$ at each time step is given by

$$x_p(t) = x_A + \kappa^\star(x_B - x_A) \tag{7.1}$$

In this situation, $\kappa^\star$ is a scalar in the range of $[0, 1]$ which denotes how far from x_A to x_B the point $x_p(t)$ is. It is obtained by solving a minimization problem of $\kappa^\star \in \arg\min f_0(\kappa, t) = \frac{1}{2}||x_{agt}(t) - [x_A + \kappa(x_B - x_A)]||^2$ over all $\kappa \in [0, 1]$. The solution can be analytically found to be

$$\kappa^\star = \begin{cases} 1 & \text{if } \varphi > 1 \\ 0 & \text{if } \varphi < 0 \\ \varphi & \text{otherwise} \end{cases} \tag{7.2}$$

$$\text{where } \varphi = \frac{x_A^T(x_A - x_B) + (x_B^T - x_A^T)}{x_A^T x_A - 2x_A^T x_B + x_B^T x_B} x_{agt}(t)$$

So the instantaneous cost at time step t is given by

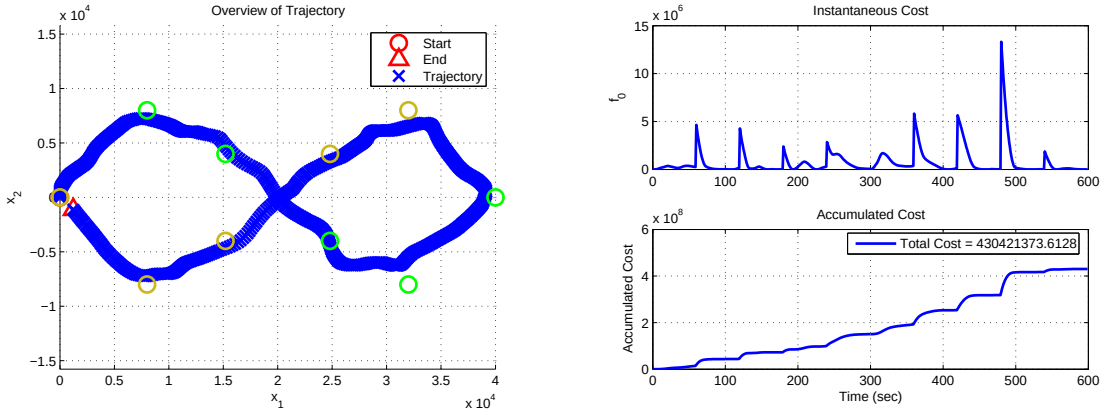
$$f_0(\kappa^\star, t) = \frac{1}{2}||x_{agt}(t) - [x_A + \kappa^\star(x_B - x_A)]||^2 \tag{7.3}$$

The accumulated cost up to time t is simply the sum of all the instantaneous costs up to the current time

$$J(t) = \sum_{\tau=0}^{\tau=t} f_0(\kappa^\star, \tau) \tag{7.4}$$

7.3.2 Path Following Example

An example of an unskilled pilot flying through two paths and the instantaneous and accumulated cost is shown in Figure 7.6.



(a) Overview of trajectory

(b) Instantaneous and accumulated cost

Figure 7.6: Simulator results from human-in-the-loop simulation.

In this situation, the pilot is tasked with flying a figure eight pattern. The pattern is broken up into two distinct paths. The first path consists of five green waypoints (p_i for $i = 1, ..., 5$), each of which have desired arrival times of t = 60, 120, 180, 240, and 300 seconds, respectively. At t = 300, a second path is generated which consists of the five brown waypoints which return the agent to the starting point.

The pilot's performance is measured and displayed in Figure 7.6(b). The instantaneous cost is a measure of how far off the desired path the pilot is. Notice that the discontinuities in instantaneous cost occur when the active waypoint changes from x_i to x_{i+1} (for path 1, this occurs at t =60, 120, 180, 240, and 300 seconds). These jumps are due to the fact that at time $t < t_i$, the agent has not reached waypoint x_i but is roughly on the desired path between x_{i-1} and x_i. This results in a low cost (most likely that $\kappa^\star \in [0, 1]$). However at $t > t_i$, the next active waypoint becomes x_{i+1} and since the agent has not yet reached point x_i, the cost becomes large (most likely that $\kappa^\star = 0$).

The performance of the pilot can be judged by the instantaneous cost trace. Skilled pilots will have a low average value with minimal discontinuities. For identical paths and times, the performance can also be judged by $J(t_f)$ which provides a type of score for the run.

The results of a human pilot flying 4 runs over a single path are shown in Figure 7.7. As can be seen in this figure, the human operates as a nonlinear, adaptive, quickly learning inner loop controller and is able to obtain very good results within just 4 training runs with the system.

7.3.3 Remarks

This chapter presented an architecture for verifying and validating the performance and output of strategic control algorithms with a high degree of accuracy while minimizing time between simulation and flight test. This was done by introducing human interaction at specific points in the system which preserves the autonomous contributions of the strategic algorithm by reducing the human to a simple inner loop controller. This architecture is used in the distributed ground based simulator to simulate flight test conditions in a controlled

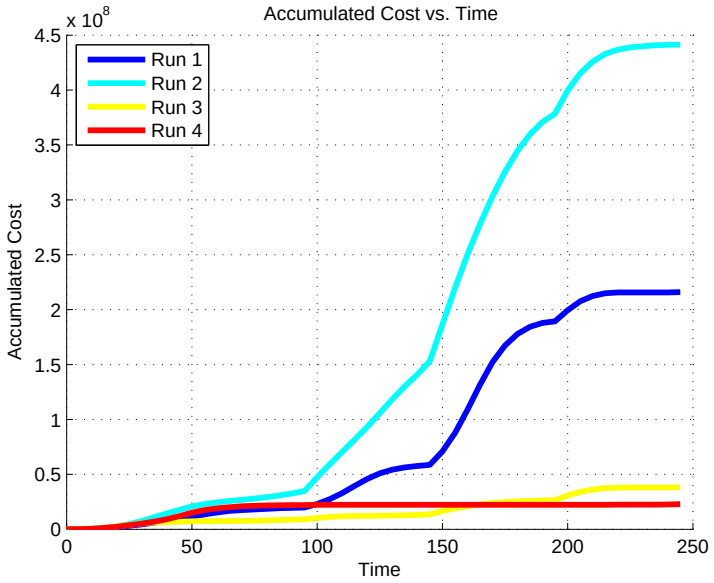


Figure 7.7: Improvement of human operator performance over 4 runs.

environment. The ground based distributed simulator is used to verify and validate the strategic control algorithms and also to familiarize pilots with the interface and to train them before actually performing a flight test.

During the development of the distributed simulator, many different methods of allowing human interaction were experimented with. Significant development effort was focused on developing a pilot visualization system which relays the pertinent path information to the pilot. By interviewing pilots after they had used the system, it was discovered that most of them only rely on the overview of the path (the bottom graph in Figure 7.5(b)). The operator workload became too high when they were forced to constantly scan all four displays and process the information. Current research is directed towards developing a single display which efficiently conveys all the relevant information to the pilot.

The other major interface between the simulator and the pilot involves the joystick. The current joystick is only a three axis device. The two principal axes were mapped to elevator and aileron inputs. Initially, the third axis was mapped to the throttle input and the rudder was fixed to zero. This design was selected because many commercial pilots

expressed that rudder input is typically reserved for a yaw damping system. However, it was discovered that path following was easier for the pilot if sideslipping and coordinated turns were allowed. To facilitate these maneuvers, the third axis was mapped to the rudder input. The throttle was instead controlled using the buttons and an integrator scheme.

Finally, the current joystick automatically re-centers the two main axes to zero when the pilot takes their hands off the stick. Many pilots were displeased by this because it made it difficult to trim the aircraft during a run. The next generation simulator will include a more sophisticated joystick with more than three axes and trim features.

All of the functionality described previously is readily ported to 3 dimensional examples with minimal changes.

Chapter 8

FLIGHT TESTING

The last step in the verification and validation process of the autonomous algorithms involves a real time flight test. This chapter describes the various components necessary for the flight test. Section 8.1 describes the hardware used during the setup and how it integrates with the overall system architecture described previously in Section 7.1.2. The setup and experimental methodology of the test are described in Section 8.2. The results and analysis of the successful flight test are detailed in Section 8.3.

8.1 Hardware

The main hardware for the flight test involves equipment used to support the autonomous algorithm and the flight test vehicle itself.

8.1.1 Algorithm Hardware

Two laptop computers make up the main hardware components of the flight test. These machines and their functions were described previously in Section 7.2.2. One computer is the strategic algorithm computer and is responsible for computing the high level, mission management algorithms for each agent. It is also responsible for interfacing with the GPS device and publishing relevant data to the DataHub. The second computer is called the visualization computer and responsible for maintaining the DataHub server and displaying the pertinent information to the human operator.

One major difference between the distributed simulator described in Section 7.2.2 and the flight test is that the position of the aircraft is not simulated. During the flight test, the position of the aircraft is measured using a Garmin GPSMap76CSx device connected to the Strategic Algorithm laptop via a serial cable. This GPS device outputs NMEA sentences over a standard RS-232 port at a rate of 1 Hz. The simulation application actively polls

the COM port and interprets GPGLL sentences in real time to obtain agent position. The latitude and longitude encoded in the GPGLL sentence is translated into the North and East positions from the specified set of coordinates (the base) using the Vincenty Formula [116] using the WGS-84 Earth model [2]. GRMZ sentences are interpreted to obtain the agent's altitude in real time. The serial port is configured for a 9600 Baud rate. The various hardware and their respective connections are shown in Figure 8.1.

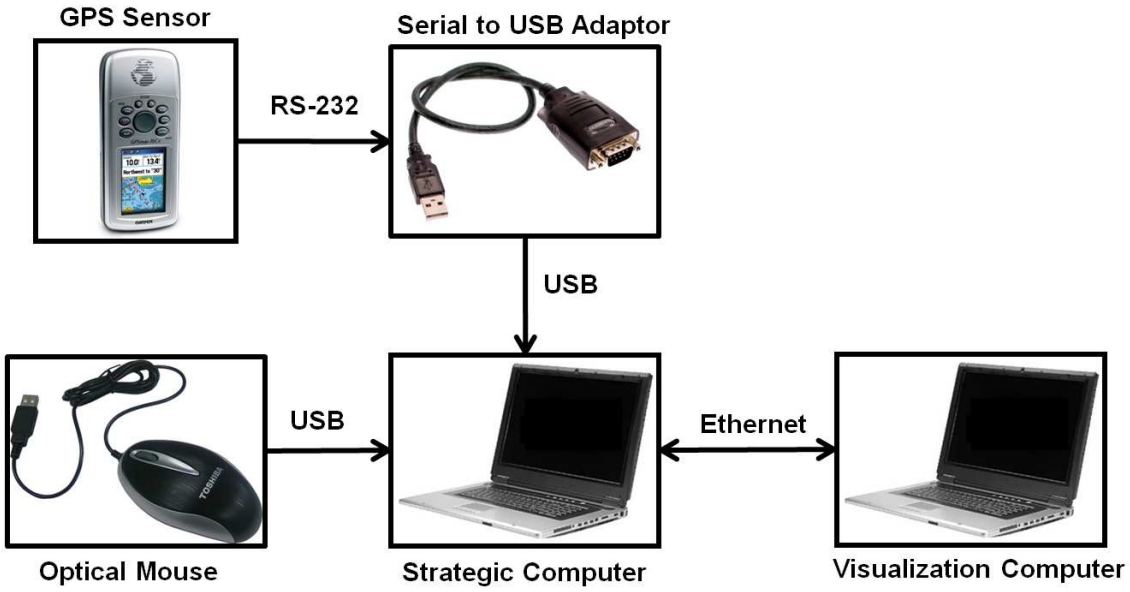


Figure 8.1: Flight test hardware connection diagram.

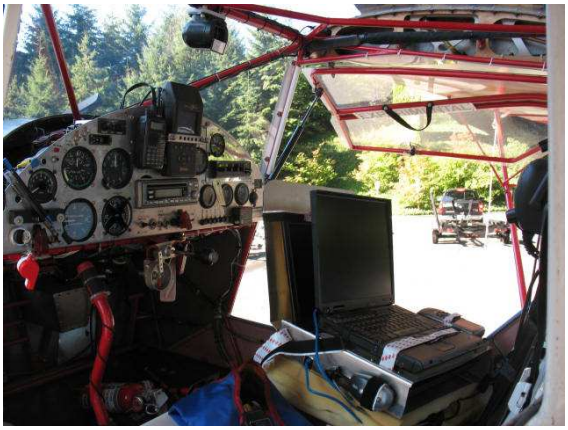
The components are enclosed in a custom made case designed to isolate the hardware components from the aircraft vibrations (typically 5500-6000 RPM at 65 MPH). The hardware carried in the vehicle during flight tests is shown in Figure 8.2.

8.1.2 Flight Test Vehicle

The flight test was conducted using a 1993 Kitfox Classic IV. This is an experimental, two-seater, single engine aircraft with the following specifications.

The aircraft can be equipped with either wheels or floats for amphibious operation. A

(a) Flight test hardware in protective carrier.



(b) Hardware situated in co-pilot seat.

Figure 8.2: Flight test hardware in test vehicle.

Table 8.1: Kitfox Classic IV specifications.

Specification	**Value**	**Specification**	**Value**
Manufacturer	Skystar	Fuel Capacity	26 gallons
Model	Kitfox Classic IV	Fuel Burn	5.2 gal/hr @ 5600 RPM
Year	1993	Cruise Speed	75 MPH
Tail Number	N328ML	Powerplant	Rotax 582-LC (65 Hp)
Serial Number	DCU033	Propeller	Ivoprop adjustable pitch
Top Speed	125 MPH	MAC	51.1”
Range	300 Miles	Max Rate of Climb	600 ft/min
MGTOW	1200 lbs	Service Ceiling	14,000 ft
Dry Weight	769 lbs	V_y (max dy/dt)	63 MPH @1089 lbs
Wingspan	32 feet	V_x (max dy/dx)	56 MPH @1089 lbs
Wing Area	130.5 ft^2	V_{stall}	51 MPH
Aspect Ratio	7.85		

picture of the aircraft is shown in Figure 8.3.

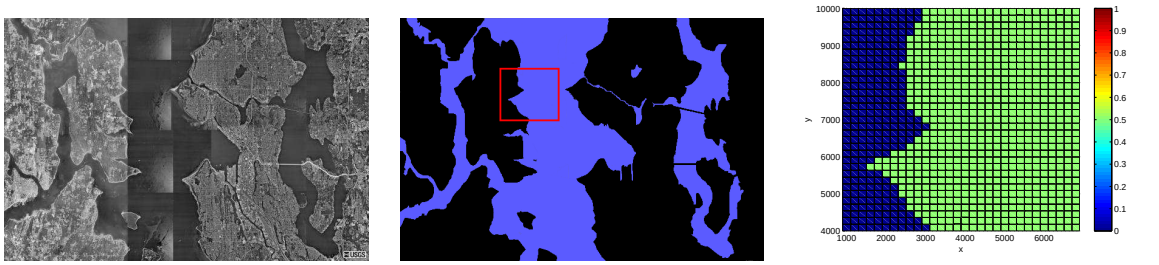
8.2 Mission Description

The flight test mission is modeled after a standard maritime search and rescue operation. A single target is assumed to be lost somewhere at sea [21] and there are multiple agents searching the area for this target. In this scenario, the target test area is situated off the

Figure 8.3: 1993 Kitfox Classic IV flight test vehicle.

north east coast of Bainbridge Island, WA. A georeferenced image which covers the test area is obtained as shown in Figure 8.4(a). From this, the areas of water and land are assigned different colors as shown in Figure 8.4(b). The base of operations is designated as 47° 36' 1.46"N and -122° 32' 17.10"E. From the base of operations, a 6km by 6km test area is projected with part of the area over land and part over water. The test area is shown in Figure 8.4(b) as the red outline. Figure 8.4(b) is used to create an initial occupancy map representation of the test area.

The mission involves searching the test area for the target using five agents. Four of the agents are simulated and one is the real aircraft. Two of the simulated agents have parameters used to emulate the ScanEagle unmanned aerial vehicle and two have parameters to emulate the SeaFox unmanned surface vehicle. Some of the relevant parameters for the agents are shown below in Table 8.2. For all agents in the heterogeneous team, parameters of



(a) Georeferenced image of Puget Sound

(b) Simplified image with test area in red

(c) Initial occupancy map of flight test area

Figure 8.4: Flight test area of interest.

$d = 5$, $K = 3$, $\Delta T = 4$ seconds are used. The paths are staggered as previously described in Section 6.1.2 to allow efficient multi-agent operations within a single threaded application.

Table 8.2: Parameters of agents in team during Flight Test.

Agent	α	β	γ	δ	h	V_{max}	$\overline{z}_H$ method
1 (purple)	0.75	0.85	0.35	0.15	0.02	26 m/s	Eq. 5.15
2 (red)	0.90	0.50	0.90	0.05	0.35	30 m/s	Eq. 5.15
3 (gold)	0.30	0.05	0.45	0.51	0.55	30 m/s	Eq. 5.15
4 (pink)	0.70	0.25	0.65	0.10	0.45	20 m/s	Eq. 5.15
5 (yellow)	0.50	0.15	0.85	0.10	0.45	20 m/s	Eq. 5.15

8.3 Flight Test Results

The aircraft is launched from a public boat launch and makes it way to the test area. Once the aircraft enters the test area, the algorithm is started to guide the aircraft and the four simulated agents in the search mission. The mission time limit is set for 20 minutes. Each agent's path is updated once every $d \cdot K \cdot \Delta T$ seconds and the visualization output to the operator of the aircraft is updated once every ΔT seconds. In this flight test, the co-pilot is in charge of monitoring the output of the autonomous algorithm and then relaying the information to the pilot. During the flight test, the next waypoint was conveyed to the pilot by communicating a heading angle and distance to next waypoint (there was no wind

during the flight test so heading angle was approximately equivalent to course angle). The trajectory for the entire mission for the real agent is show in Figure 8.5. In this figure, the red crosses represent the actual position of the agent as recorded by the GPS. The purple circles show the agent's position projected onto the test domain. In order to ensure that subproblem ($\wp_2$) and ($\wp_3$) generate feasible paths, the agent's position is projected onto the search area. The only difference between the trajectories is when the agent leaves the test area. Theoretically, this will never happen as the algorithm will only generate paths which are within the search area. This error only occurs when the pilot does not follow the desired path accurately enough.

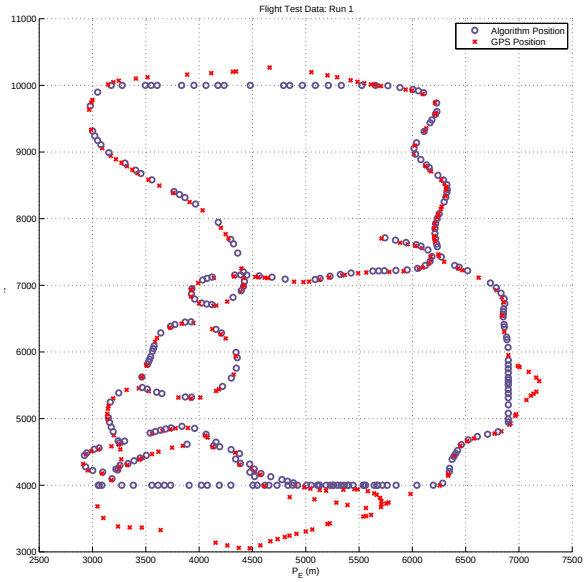


Figure 8.5: GPS and recorded agent position during flight test.

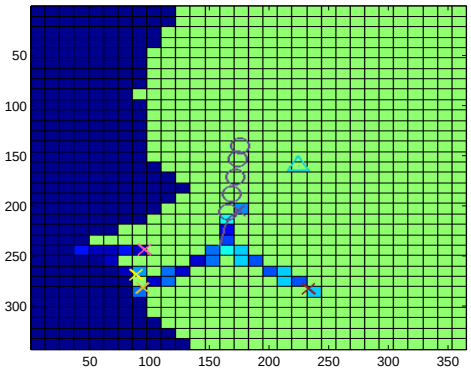
The same run and the associated occupancy based map and the other agents is shown in Figure 8.6.

In Figure 8.6, the purple cross represents the location of the real agent. The red, gold, yellow, and magenta crosses represent the location of the simulated agents. The purple circles are the paths planned for the real agent by ($\wp_3$). The solid purple line is the real

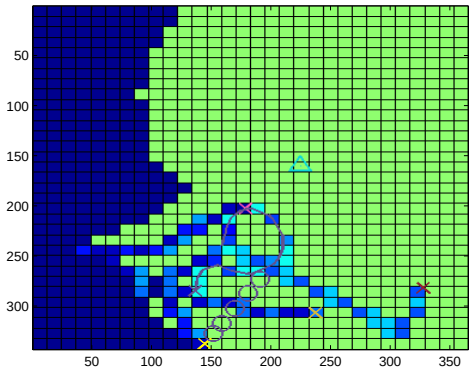
agent's trajectory. To avoid cluttering the figure, the associated paths and trajectories for the simulated agents are omitted. The target is assumed to be static (i.e. a boat that has lost power in standing waters) and its location is represented by the teal triangle. It is worth mentioning that the flight test in Figure 8.6 is the first time this particular pilot had used this system besides practice runs on the distributed simulator. This is evident in Figure 8.6(b) which shows some initialization errors on the part of the pilot as he becomes accustomed to the interface (initially, the pilot is unable to track the path). Figure 8.6(d) shows the pilot successfully tracking the path specified by the algorithm. Eventually, one of the agents locates the target and updates cells in the local area. This causes nearby agents, including the real agent, to converge on the target location (Figure 8.6(e)) and terminate a successful mission. Note that at this point, the strategic searching mission is terminated and control can be transferred to a tactical level autonomous algorithm which might be in control of tasks such as target tracking and observation [60], [120], [103], [43].

Note that the consequence of introducing a human operator to act as an inner loop controller is evident in the real agent's trajectory. Using an architecture where the pilot is responsible for dynamics and control level tasks (Figure 7.2) instead of the fully autonomous system (Figure 7.1) yields a decrease in performance. Many times during the actual flight test, the real agent was not on the path specified by the algorithm. This was due to several factors. The most significant of these was the fact that the parameter ΔT was relatively large. Although the path for each agent was valid for 60 seconds, the agent's position and the pilot visualization screen were only updated at 0.25 Hz. Furthermore, since $K = 3$, there were only three updates between waypoints. If the agent was off course, the pilot would first be made aware of the error at the first update. The pilot would then only have two further feedback updates to correct the problem. The 4 second delay between updates proved problematic because the cross track error may grow to a significant value before the pilot is first made aware of the error. The dynamics of the aircraft were slow enough to exacerbate this problem. In other words, the maximum yaw rate was not sufficient to correct the cross track error quickly. The overall effect was that the pilot was constantly trying to correct errors and would not converge to the desired waypoint until near the end of the path. Despite this, the algorithm is robust in the sense that it gracefully handles these

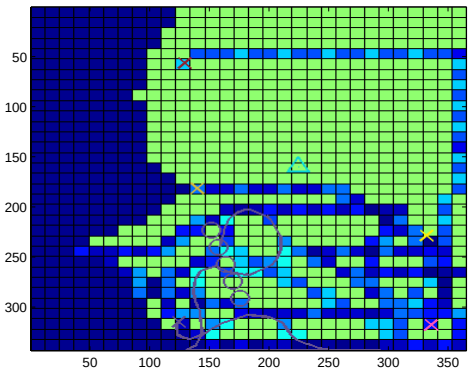
errors and still performs the search successfully. It is able to replan based on the agent's current position even though this position may not be the desired location specified in ($\wp_2$).



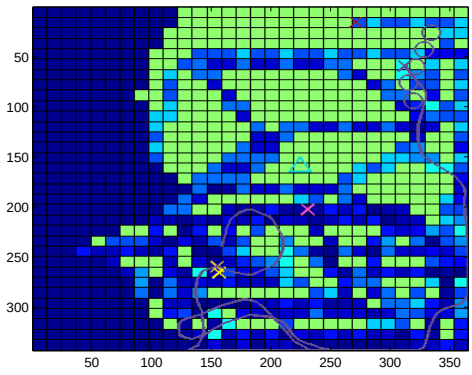
(a) $x_w(k, \overline{z})$ at $t = 40$



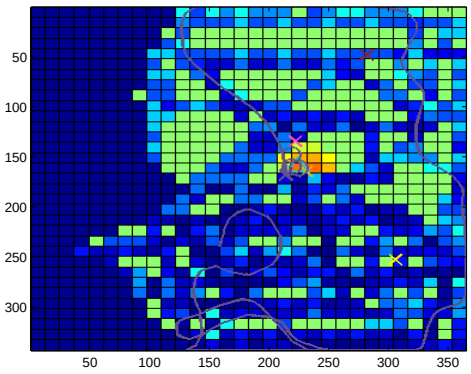
(b) $x_w(k, \overline{z})$ at $t = 120$



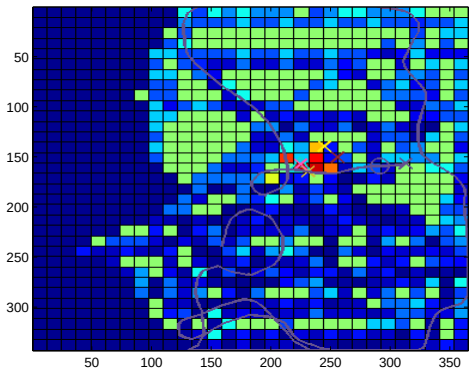
(c) $x_w(k, \overline{z})$ at $t = 400$



(d) $x_w(k, \overline{z})$ at $t = 800$



(e) $x_w(k, \overline{z})$ at $t = 1040$



(f) $x_w(k, \overline{z})$ at $t = 1160$

Figure 8.6: Flight test results with 4 simulated agents and 1 real agent.

Chapter 9

CONCLUSIONS

9.1 Concluding Remarks

This dissertation presented research and development of several algorithms which are used to increase the autonomous abilities of a team of heterogeneous vehicles involved in ISR type missions. The modular strategy is comprised of individual algorithms that can be used to coordinate a group of agents in a search mission in an efficient manner.

As mentioned previously, an autonomous system involves more than a theoretical algorithm which is capable of making autonomous decisions. It encompasses the infrastructure and processes used to support the agent during a mission. A practical autonomous system includes mathematical analysis, software development, numerical simulation, verification and validation procedures, hardware-in-the-loop testing, flight tests, and many other subjects. The research contained in this dissertation documents the development of a viable autonomous system starting from theoretical ideas and ending with hardware deployment.

This dissertation began by developing an autonomous target identification system which can be used to extract discernable features from noisy, low dimensional sensors in Chapter 3. The constructs of occupancy based maps and operations to interact with them were then considered in Chapter 4. With the ability to accurately classify anomalies and the establishment of the occupancy based map framework, the theoretical development of a modular strategy for autonomous searching using groups of heterogeneous vehicles was then investigated in Chapter 5. Implementation of this strategy in simulation along with metrics for measuring performance were then discussed in Chapter 6. The challenges and steps between pure software simulation and hardware deployment were investigated and addressed in Chapter 7. Finally, the transition from theoretical algorithm to viable product and flight test hardware was discussed in Chapter 8.

9.2 Future Research

Although the work described in this dissertation has been successfully deployed as a viable product, some improvements to the existing system can implemented. In addition, open questions still remain regarding various aspects of the research.

For example, with respect to the autonomous target identification, this tactical algorithm has yet to see hardware verification and validation. Implementing this on currently existing Georangers or other UAV systems would require significant effort in terms of logistics and hardware interfaces but would have significant commercial and military applications.

The most mature part of this research is the autonomous search strategy. It provides provably correct and efficient behavior in many real world scenarios. There are several improvements, however, that can be used to increase performance even further. One major disadvantage of the current architecture is that the algorithm requires a centralized topology. This limits the flexibility and robustness of the overall algorithm. Current research activities are directed towards modifying the algorithm for decentralized deployment. In addition, there are situations where different policies work better for different phases of the mission. For example, it was shown that map coverage performance is maximized using the full algorithm with explicit cooperation. However, once an agent finds a target, it is difficult for the same strategy to plan efficient paths for the other agents in the team to converge on the same spot. In this situation, switching modes to use the full algorithm by itself would increase performance. Questions such as when is the best time to switch modes and which modes should be used can be investigated in future research. On this note, studies in terms of the explore vs. exploit paradigm may be conducted. An information theoretic framework has already been established and may be used to determine when operating in an explore mode (standard search) or an exploit mode (target tracking and identification) yields more information.

In terms of hardware development and deployment, the human-in-the-loop distributed simulator can currently support only one human operator. All other simulated agents are required to be simulated on computers located in the same laboratory. Furthermore, the actual agent is limited to being simulated by a full software simulation rather than

a hardware-in-the-loop simulation. Current research is directed towards allowing the distributed simulator to interface with hardware-in-the-loop simulators and also with agents simulated at other facilities or deployed in the field.

9.3 Final Remarks

Despite the large amount of research in the field of autonomous systems, many consider the subject to be in its infancy. The study of autonomous systems is a relatively young field and there are many opportunities and research to be done before it can be considered a mature technology. Many state of the art systems have basic autonomous capabilities but still require a large amount of human interaction and infrastructure to manage multiple vehicles. The primary limitation to the concurrent operation of multiple vehicles remains the lack of autonomy of these vehicles. The field is at the brink of a revolution. The demand for autonomous systems with ever increasing aptitudes will most likely continue to rise. As long as this occurs, technologies such as those described in this dissertation that serve to increase the autonomous abilities of teams of agents must continue to be developed in order to meet this demand.

BIBLIOGRAPHY

[1] Boeing-insitu scaneagle uav achieves 10,000 flight hours in support of australian army operations. Insitu Press Release. http://www.insitu.com/index.cfm?navid=20&cid=2305.

[2] Department of defense world geodetic system 1984, its definition and relationships with local geodetic systems 3rd edition. Technical Report TR8350.2, National Geospatial-Intelligence Agency.

[3] International geomagnetic reference field. Public Information. http://www.ngdc.noaa.gov/IAGA/vmod/igrf.html.

[4] Puget sound aeromagnetic maps and data. Public Information. http://pubs.usgs.gov/of/1999/of99-514/.

[5] Scaneagle uas flies with heavy fuel in iraq. Insitu Press Release. http://www.insitu.com/index.cfm?navid=20&cid=2582.

[6] U.s. air force fact sheet mq-1 predator unmanned aircraft system. Public Information. http://www.af.mil/factsheets/factsheet.asp?fsID=122.

[7] U.s. air force fact sheet rq-4 global hawk unmanned aircraft system. Public Information. http://www.af.mil/factsheets/factsheet.asp?fsID=13225.

[8] Flightgear flight simulator. Public Information, 2006. http://www.flightgear.org/.

[9] R. Alexander and N. Rowe. Path planning by optimal-path-map contruction for homogeneous-cost two-dimensional regions. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 1990.

[10] Jayesh N. Amin, Jovan D. Boskovic, and Raman K. Mehra. A fast and efficient approach to path planning for unmanned vehicles. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Keystone, CO, 2006.

[11] David E. Anderson and Alejandro C. Pita. Geophysical surveying with georanger uav. In *Proceedings of the 2005 Infotech@Aerospace Conference*, Arlington, VA, September 2005. The Insitu Group.

[12] P. J. Antisaklis and Kevin M. Passino. Towards intelligent autonomous control systems: Architecture and fundamental issues. *Journal of Intelligent and Robotic Systems*, 1(4):315–342, 1989.

[13] Alessandro Arsie and Emilio Frazzoli. Efficient routing of multiple vehicles with no communication. In *Proceedings of the 2007 American Control Conference*, New York City, New York, 2007.

[14] Dimitar Baronov and John Baillieul. Search decisions for teams of automata. In *Proceedings of the 47th Conference on Decision and Control*, Cancun, Mexico, 2008.

[15] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, pages 87–90, April 1958.

[16] Stanley J. Benkoski, Michael G. Monticino, and James R. Weisinger. A survey of the search theory literature. *Naval Research Logistics*, 38:469–494, 1991.

[17] Dimitri P. Bertsekas. *Network Optimization: Continous and Discrete Models*. Athena Scientific, Belmont, Mass, 1st edition, 1998.

[18] Lars Blackmore. A probabilistic particle control approach to optimal, robutst predictive control. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Keystone, CO, 2006.

[19] Donald Borrett, Sean Kelly, and Hon Kwan. Phenomenology, dynamical neural networks and brain function. *Philosophical Psychology, Vol. 13, No. 2*, pages 213–228, 2000.

[20] Frederic Bourgault and Hugh F. Durrant-Whyte. Communication in general decentralized filters and the coordinated search strategy. In *Proceedings of the 7th International Conference on Information Fusion*, Stockholm, Sweden, 2004. Australian Centre for Field Robotics.

[21] Frederic Bourgault, Tomonari Furukawa, and Hugh Durrant-Whyte. Coordinated decentralized search for a lost target in a bayesian world. In *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Las Vegas, NV, October 2003. Australian Centre for Field Robotics.

[22] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004.

[23] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal, Vol. 4, No. 1*, pages 25 – 30, January 1965.

[24] Ming Cao, Andrew Stewart, and Naomi Ehrich Leonard. Integrating human and robot decision-making dynamics with feedback: Models and convergence analysis. In *Proceedings of the Conference on Decision and Control*, Cancun, Mexico, 2008.

[25] Brian J. Capozzi and Juris Vagners. Navigating annoying environments through evolution. In *Proceedings of the 40th IEEE Conference on Decision and Control*, Orlando, FL, 2001. University of Washington.

[26] John M. III Carson and Behcet Ackmese. A model predictive control technique with guaranteed resolvability and required thruster silent times for small-body proximity operations. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Keystone, CO, 2006.

[27] Bruce T Clough. Metrics, schmetrics! how the heck do you determine a uav's autonomy anyway? In *Proceedings of the Performance Metrics for Intelligent Systems Workshop*, Gaithersbug, MD, 2002.

[28] Jorge Cortes, Sonia Martinez, Timur Karatas, and Francesco Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20:243–255, 2004.

[29] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory 2nd Edition*. John Wiley and Sons, 2006.

[30] Robert L. Dollarhide, Arvin Agah, and Gary J. Minden. Evolving controllers for autonomous robot search teams. *Artificial Life and Robotics Journal*, 5:178–188, 2002.

[31] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the Association for Computing Machinery*, pages 1048–1066, November 1993.

[32] Mark Dransfield, Asbjorn Christensen, and Guimin Liu. Airborne vector magnetics mapping of remanently magnetised banded iron-formations at rocklea, western australia. In *Proceedings of the ASEG 16th Geophysical Conference and Exhibition*, Adelaide, Australia, February 2003.

[33] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *Society for Industrial and Applied Mathematics Review*, 41:637–676, 1999.

[34] Alberto Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, May 1989.

[35] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, pages 46–57, 1989.

[36] Charles A. Erignac. An exhaustive swarming search strategy based on distributed pheromone maps. Technical report, Boeing, Seattle, WA, 2004.

[37] Gregory L. Feitshans, Allen J. Rowe, Jason E. Davis, Michael Holland, and Lee Berger. Vigilant spirit control station (vscs) the face of counter. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Honolulu, HI, 2008.

[38] Brian Ferris, Dirk Hahnel, and Dieter Fox. Gaussian processes for signal strength-based location estimation. In *Proceedings of Robotics: Science and Systems*, 2006.

[39] Matthew Flint, Marios Polycarpou, and Emmanuel Fernandez-Gaucherand. Cooperative control for multiple autonomous uav's searching for targets. In *Proceedings of the 41st IEEE Conference on Decision and Control*, Las Vegas, NV, 2004. University of Cincinnati.

[40] Dieter Fox, Jeffrey Hightower, Lin Liao, and Dirk Schulz. Bayesian filtering for location estimation. *IEEE Pervasive Computing*, pages 23–33, July 2003.

[41] Dieter Fox, Jonathan Ko, Kurt Konolige, Benson Limketkai, Dirk Schulz, and Benjamin Steward. Distributed multi-robot exploration and mapping. In *Proceedings of the 2nd Canadian Conference on Computer and Robot Vision*, 2005.

[42] Emilio Frazzoli and Francesco Bullo. Decentralized algorithms for vehicle routing in a stochastic time-varying environment. In *Proceedings of the IEEE Conference on Decision and Control*, December 2004.

[43] Eric W. Frew and Dale A. Lawrence. Cooperative stand-off tracking of moving targets by a team of autonomous aircraft. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, San Francisco, California, 2005.

[44] Robert M. Gray. *Entropy and Information Theory*. Springer-Verlag, 1990.

[45] Greg Hodges. Notes on magnetic model for submarine. Technical report, Fugro Airborne Surveys, Toronto, 2004.

[46] Gabriel M. Hoffman, Steven L. Waslander, and Claire J. Tomlin. Distributed cooperative search using information-theoretic costs for particle filters, with quadrotor applications. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Keystone, CO, 2006.

[47] Greg J. Holland, Tad McGeer, and Harold Youngre. Autonomous aerosondes for economical atmospheric soundings anywhere on the globe. *Bulletin of the American Meteorological Society*, 73:1987–1998, 1992.

[48] Chien-Feng Huang, David H. Wolpert, Stefan Bieniawski, and Charlie E.M. Strauss. A comparative study of probability collectives based multi-agent systems and genetic algorithms. In *Proceedings of the GECCO 2005 Conference*, 2005.

[49] Hui-Min Huang, Elena Messina, Robert Wade, Ralph English, Brian Novak, and James Albus. Autonomy measures for robots. In *Proceedings of the International Mechanical Engineering Congress*, 2004.

[50] Ole C. Jakobsen and Eric N. Johnson. Control architecture for a uav-mounted pan/tilt/roll camera gimbal. In *Proceedings of the Infotech@Aerospace Conference*, Arlington, VA, 2005.

[51] James S. Jennings, Greg Whelan, and William F. Evans. Cooperative search and rescue with a team of mobile robots. In *Proceedings of the International Conference on Advanced Robotics*, 1997.

[52] Yan Jin, Yan Liao, Ali A. Minai, and Marios M. Polycarpou. Balancing search and target response in cooperative unmanned aerial vehicle (uav) teams. *IEEE Transactions on Systems, Man, and Cybernetics*, 36:571–587, 2006.

[53] Nidal Jodeh, Mark Mears, and David Gross. An overview of the cooperative operations in urban terrain (counter) program. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Honolulu, HI, 2008.

[54] Eric N Johnson, Nimrod Rooz, Jeong Hur, and Wayne Pickell. A concurrent testing process for research unmanned aerrial vehicles. In *Proceedings of the 25th AIAA Aerodynamic Measurement Technology and Ground Testing Conference*, San Francisco, CA, June 2006.

[55] Eric N Johnson, Daniel P. Schrage, and George Vachtsevanos. Software enabled control experiments with university-operated unmanned aircraft. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Arlington, VA, 2005.

[56] Yeonsik Kang, Derek S. Caveney, and J. Karl Hedrick. Probabilistic mapping for uav using point-mass target detection. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Keystone, CO, 2006.

[57] Yeonsik Kang and J. Karl Hedrick. Design of nonlinear model predictive controller for a small fixed-wing unmanned aerial vehicle. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Keystone, CO, 2006.

[58] George Kantor, Sanjiv Singh, Ronald Peterson, Daniela Rus, Aveek Das, Vijay Kumar, Guilherme Pereiera, and John Spletzer. Distributed search and rescue with robot sensor teams. In *Proceedings of the 4th International Conference on Field and Service Robotics*, 2003.

[59] Byoung S. Kim and Anthony J. Calise. Nonlinear flight control using neural networks. *Journal of Guidance, Control and Dynamics*, pages 26–33, January 1997.

[60] Daniel J. Klein. *Coordinated Control and Estimation for Multi-agent Systems: Theory and Practice*. PhD thesis, University of Washington, Seattle, WA, September 2008.

[61] Daisuke Kurabayashi, Hironori Tsuchiya, Ikki Fujiwara, Hajime Asama, and Kuniaki Kawabata. Motion algorithm for autonomous rescue agents based on information assistance system. In *Proceedings of the 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Kobe, Japan, 2003.

[62] Cody Kwok, Dieter Fox, and Marina Meila. Real-time particle filters. *IEEE Special Issue on Sequential State Estimation*, 2004.

[63] Emmett Lalish, Kristi A. Morgansen, and Takashi Tsukamaki. Decentralized reactive collision avoidance for multiple unicycle-type vehicles. In *Proceedings of the 2008 American Control Conference*, Seattle, WA, 2008.

[64] Paul Lambrechts, Samir Bennani, Gertjan Looye, and Anders Helmersson. Robust flight control design challenge problem formulation and manual: the reserach civil aircraft model (rcam). Technical report, Group for Aeronautical Research and Technology in Europe, Europe, 1997.

[65] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

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

[67] Steven M. LaValle and James J. Kuffner Jr. Randomized kinodynamic planning. *International Journal of Robotics Research*, pages 378–400, May 2001.

[68] Katie Laventall and Jorge Cortes. Coverage control by robotic networks with limited-range anisotropic sensory. In *Proceedings of the 2008 American Control Conference*, Seattle, Washington, 2008.

[69] Jonathan Lester, Tanzeem Choudhury, Nicky Kern, Gaetano Borriello, and Blake Hannaford. A hybrid discriminative/generative approach for modeling human activities. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland, 2005.

[70] Christopher W. Lum. A single agent search of a two dimensional space using probability collectives and convex optimization. Technical report, University of Washington, Seattle, WA, 2006.

[71] Christopher W. Lum, Matthew L. Rowland, and Rolf T. Rysdyk. Human-in-the-loop distributed simulation and validation of strategic autonomous algorithms. In *Proceedings of the 2008 Aerodynamic Measurement Technology and Ground Testing Conference*, Seattle, WA, June 2008.

[72] Christopher W. Lum and Rolf T. Rysdyk. Feature extraction of low dimensional sensor returns for autonomous target identification. In *Proceedings of the 2008 Guidance, Navigation, and Control Conference*, Honolulu, HI, August 2008.

[73] Christopher W. Lum and Rolf T. Rysdyk. Time constrained randomized path planning using spatial networks. In *Proceedings of the 2008 American Control Conference*, Seattle, WA, June 2008.

[74] Christopher W. Lum, Rolf T. Rysdyk, and Anawat Pongpunwattana. Autonomous airborne geomagnetic surveying and target identification. In *Proceedings of the 2005 Infotech@Aerospace Conference*, Arlington, VA, September 2005. AIAA.

[75] Christopher W. Lum, Rolf T. Rysdyk, and Anawat Pongpunwattana. Occupancy based map searching using heterogeneous teams of autonomous vehicles. In *Proceedings of the 2006 Guidance, Navigation, and Control Conference*, Keystone, CO, August 2006.

[76] Christopher W. Lum and Juris Vagners. A modular algorithm for exhaustive map searching using occupancy based maps. In *To appear in Proceedings of the 2009 Infotech@Aerospace Conference*, Seattle, WA, April 2009.

[77] Jason S. McCarley and Christopher D. Wickens. Human factors concerns in uav flight. Technical report, Institude of Aviation, Aviation Human Factors Division, University of Illinois aat Urbana-Champaign, 2004.

[78] Jason S. McCarley and Christopher D. Wickens. Human factors implications of uavs in the national airspace. Technical Report AHFD-05-05/FAA-05-01, Institude of Aviation, Aviation Human Factors Division, University of Illinois aat Urbana-Champaign, Atlantic City International Airport, NJ, 2005.

[79] Tad McGeer and Juris Vagners. Flying the atlantic - without a pilot. *GPS World*, 10:24–30, 1999.

[80] Tad McGeer and Juris Vagners. Wide-scale use of long-range miniature aerosondes over the world's oceans. In *Proceedings of the AUVSI 26th Annual Symposium, Association for Unmanned Vehicle Systems International*, Baltimore, MD, July 1999.

[81] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, pages 18–73, January 1991.

[82] Ndedi Monekosso and Paolo Remagnino. Robot exploration using the expectation-maximization algorithm. In *Proceedings of 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Kobe, Japan, 2003.

[83] Sujit Nair and Jerry Marsden. Collision avoidance and surveillance measures for multivehicle systems. Technical report, California Institute of Technology, Pasadena, CA, 2008.

[84] Laurence R. Newcome. *Unmanned Aviation: A Brief History of Unmanned Aerial Vehicles*. American Institute of Aeronautics and Astronautics, 2004.

[85] NIMA. National imagery and mapping agency technical repport tr8350.2: Department of defense world geodetic system 1984, its definition and relationships with local geodetic systems, 3rd edition. Technical report, National Imagery and Mapping Agency, 2000.

[86] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations Concepts and Applications of Voronoi Diagrams*. John Wiley and Sons, 1996.

[87] Jarurat Ousingsawat. Quasi-decentralized task assignment for multiple uav coordination. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Keystone, CO, 2006.

[88] Richard Partner. Georanger aeromagnetic uav: Development to commercial survey. *Fugro Explore*, 3:1–4, 2006.

[89] Marios M. Polycarpou, Yanli Yang, and Kevin M. Passino. A cooperative search framework for distributed agents. In *Proceedings of the 2001 IEEE International Symposium on Intelligent Control*, Mexico City, Mexico, 2001.

[90] Anawat Pongpunwattana. *Real-Time Planning for Teams of Autonomous Vehicles in Dynamic Uncertain Environments*. PhD thesis, University of Washington, Seattle, WA, June 2004.

[91] Anawat Pongpunwattana and Rolf T. Rysdyk. Real-time planning for multiple autonomous vehicles in dynamic uncertain environments. *AIAA Journal of Aerospace Computing, Information, and Communication*, pages 580–604, December 2004.

[92] Anawat Pongpunwattana, Rolf T. Rysdyk, Juris Vagners, and David Rathbun. Market-based co-evolution planning for multiple autonomous vehicles. In *Proceedings of the AIAA Unmanned Unlimited Conference*. Autonomous Flight Systems Laboratory, 2003.

[93] Anawat Pongpunwattana, Richard Wise, Rolf T. Rysdyk, and Anthony J. Kang. Multi-vehicle cooperative control flight test. In *Proceedings of the 25th Digital Avionics Systems Conference*, October 2006.

[94] Alison A. Proctor, Suresh K. Kanna, Chris Raabe, Christophersen Henrik B., and Eric N. Johnson. Development of an autonomous aerial reconnaissance system at georgia tech. In *Proceedings of the Association of Unmanned Vehicle Systems International Unmanned Systems Symposium and Exhibition*, 2002.

[95] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[96] Alvin Raj, Amarnag Subramanya, Dieter Fox, and Jeff Bilmes. Rao-blackwellized particle filters for recognizing activities and spatial context from wearable sensors. In *Experimental Robotics: The 10th International Symposium, Springer Tracts in Advanced Robotics*, 2006.

[97] David Rathbun and Brian Capozzi. Evolutionary approaches to path planning through uncertain environments. In *Proceedings of the AIAA Unmanned Unlimited Conference*. AIAA, May 2002.

[98] R.T. Rockafellar. *Network Flows and Monotropic Optimization*. Athena Scientific, Belmont, Mass, 1st edition, 1998.

[99] R.T. Rockafellar. Fundamentals of optimization. Technical report, University of Washington, Seattle, WA, 2006.

[100] Juan Carlos Rubio, Juris Vagners, and Rolf T. Rysdyk. Adaptive path planning for autonomous uav oceanic search missions. In *Proceedings of the 1st AIAA Intelligent Systems Technical Conference*, 2004.

[101] Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. Pearson Education, Inc., Upper Saddle River, NJ, 2nd edition, 2003.

[102] Rolf T. Rysdyk. Unmanned aerial vehicle path following for target observation in wind. *Journal of Guidance, Control, and Dynamics*, pages 1092–1100, September 2006.

[103] Rolf T. Rysdyk, Christopher W. Lum, and Juris Vagners. Autonomous orbit coordination for two unmanned aerial vehicles. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, San Francisco, CA, August 2005.

[104] Ketan Savla, Carl Nehme, Tom Temple, and Emilio Frazzoli. Efficient cooperative strategies between uavs and humans in a dynamic environment. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Honolulu, HI, 2008.

[105] Ketan Savla, Tom Temple, and Emilio Frazzoli. Human-in-the-loop vehicle routing policies for dynamic environments. In *Proceedings of the Conference on Decision and Control*, Cancun, Mexico, 2008.

[106] Robert E Schapire. A brief introduction to boosting. In *IJCAI*, 1999.

[107] Eric Sholes. Evolution of a uav autonomy classification taxonomy. In *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, 2007.

[108] Bruce D. Smith, Michael J. Cain, Allan K. Clark, David W. Moore, Jason R. Faith, and Patricia L. Hill. Helicopter electromagnetic and magnetic survey data and maps, northern bexar county, texas. Technical report, U.S. Geological Survey, Reston, VA, 2005.

[109] Brian L. Stevens and Frank L. Lewis. *Aircraft Control and Simulation*. John Wiley and Sons, Hoboken, NJ, 2nd edition, 2003.

[110] Zheng Sun and John H. Reif. On finding approximate optimal paths in weighted regions. *Journal of Algorithms*, pages 1–32, January 2006.

[111] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *International Journal of Robotics Research*, 2000.

[112] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[113] Anthony P. Tvaryanas, Bill T. Thompson, and Stefan H. Constable. U.s. military unmanned aerial vehicle mishaps: Assessment of the role of human factors using hfacs. Technical Report HSW-PEBR-TR-2005-0001, 311th Performance Enhancement Directorate, United States Air Force, 2005.

[114] Anthony P. Tvaryanas, Bill T. Thompson, and Stefan H. Constable. U.s. military unmanned aerial vehicle (uav) experience: Evidence-based human systems integration lessons learned. Technical Report RTO-MP-HFM-124, 311th Performance Enhancement Directorate, United States Air Force, 2005.

[115] Unmanned Dynamics, Hood River, OR. *AeroSim Aeronautical Simulation Blockset User's Guide Version 1.2*.

[116] Thaddeus Vincenty. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review*, 176:88–93, 1975.

[117] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, 2001.

[118] Linh Vu and Kristi A. Morgansen. Modeling and analysis of dynamic decision making in sequential two-choice tasks. In *Proceedings of the Conference on Decision and Control*, Cancun, Mexico, 2008.

[119] Xudong Wang and Vassilis L. Syrmos. Interacting multiple particle filters for fault diagnosis of non-linear stochastic systems. In *Proceedings of the 2008 American Control Conference*, Seattle, WA, June 2008.

[120] Richard Wise and Rolf T. Rysdyk. Uav coordination for autonomous target tracking. In *Proceedings of the 2006 AIAA Guidance Navigation and Control Conference*, Keystone, CO, 2006.

[121] Ian H. Witten and Eibe Frank. *Data Mining.* Morgan Kaufmann Publishers, San Francisco, CA, 2nd edition, 2005.

[122] El-Mane Wong and Tomonari Bourgault, Frederic Furukawa. Multi-vehicle bayesian search for multiple lost targets. In *Proceedings of the 2005 IEEE Internationalo Conference on Robotics and Automation*, Barcelona, Spain, April 2005.

Appendix A

PUBLICATION LIST

- Rolf T. Rysdyk, Christopher W. Lum, and Juris Vagners. Autonomous Orbit Coordination for Two Unmanned Aerial Vehicles. In *Proceedings of the 2005 AIAA Guidance, Navigation, and Control Conference*, San Francisco, CA, August 2005.

- Christopher W. Lum, Rolf T. Rysdyk, and Anawat Pongpunwattana. Autonomous Airborne Geomagnetic Surveying and Target Identification. In *Proceedings of the 2005 Infotech@Aerospace Conference*, Arlington, VA, September 2005.

- Christopher W. Lum, Rolf T. Rysdyk, and Anawat Pongpunwattana. Occupancy Based Map Searching Using Heterogeneous Teams of Autonomous Vehicles. In *Proceedings of the 2006 AIAA Guidance, Navigation, and Control Conference*, Keystone, CO, August 2006.

- Christopher W. Lum and Rolf T. Rysdyk. Time Constrained Randomized Path Planning Using Spatial Networks. In *Proceedings of the 2008 American Control Conference*, Seattle, WA, June 2008.

- Christopher W. Lum, Matthew L. Rowland, and Rolf T. Rysdyk. Human-in-the-Loop Distributed Simulation and Validation of Strategic Autonomous Algorithms. In *Proceedings of the 2008 Aerodynamic Measurement Technology and Ground Testing Conference*, Seattle, WA, June 2008. Outstanding Paper Award.

- Christopher W. Lum and Rolf T. Rysdyk. Feature Extraction of Low Dimensional Sensor Returns for Autonomous Target Identification. In *Proceedings of the 2008 AIAA Guidance, Navigation, and Control Conference*, Honolulu, HI, August 2008.

- Christopher W. Lum and Juris Vagners. A Modular Algorithm for Exhaustive Map Searching Using Occupancy Based Maps. Accepted to the *2009 Infotech@Aerospace Conference*, Seattle, WA, April 2009.

Appendix B

MIN PATH/MAX TENSION ALGORITHM

The Min Path/Max Tension Algorithm is best described by Rockafellar [98]. Given a directed graph with a starting node set N^+ and an ending node set N^-, assigned span intervals $d^+(j)$ and $d^-(j)$, and initial potential u_o which is feasible with respect to the span intervals, it may be used to find a path through the graph which minimizes the cost function given by Eq. 5.55. The flow diagram for the algorithm is shown in Figure B.1.

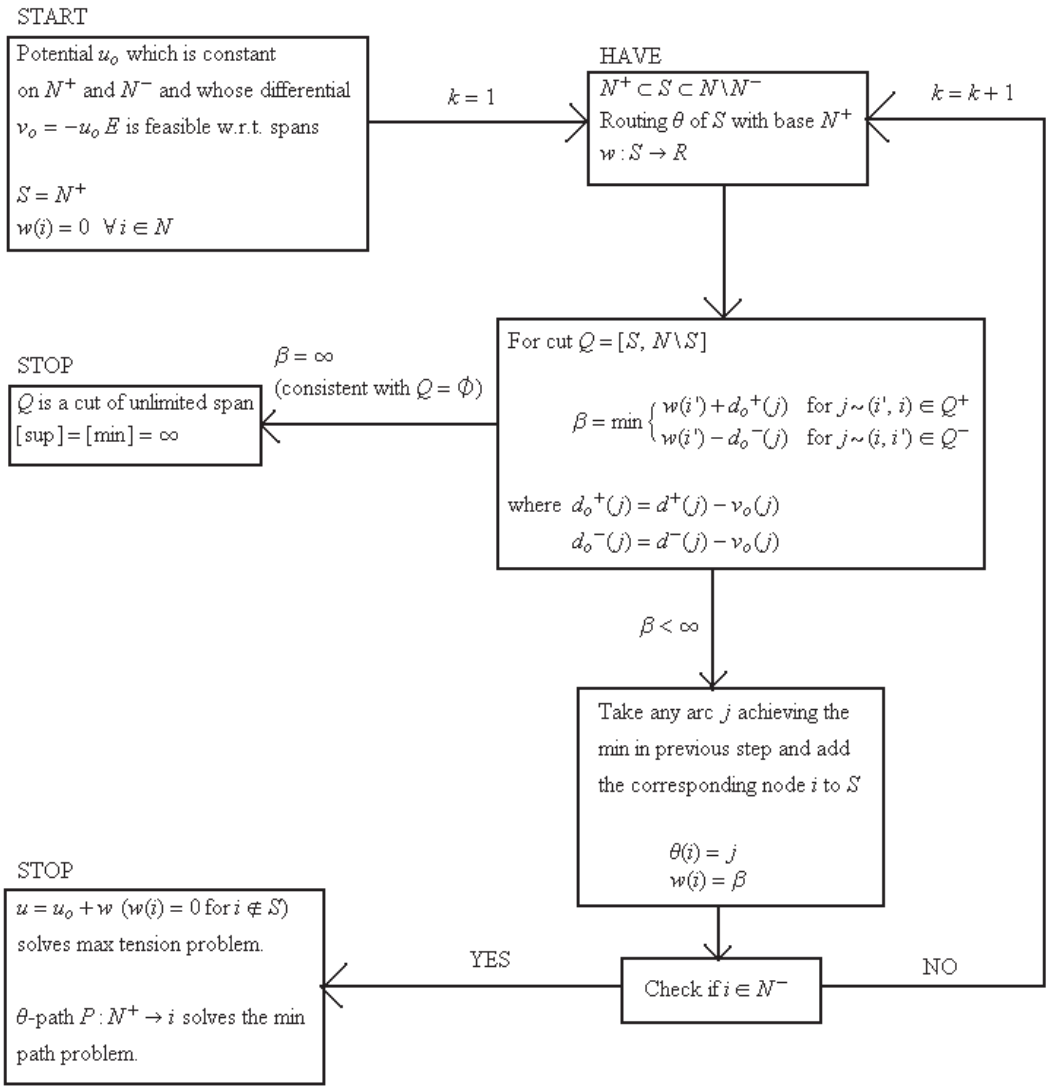


Figure B.1: Flow diagram of the Min Path Algorithm.

Appendix C

RESEARCH CIVIL AIRCRAFT MODEL

A high-fidelity, nonlinear, 6 degree of freedom model of a medium sized twin engine transport jet is used as a verification model. This is based off the Research Civil Aircraft Model [64]. This is modeled as a 12 state system and implemented in simulation using the Matlab/Simulink environment [109]. The general block diagram of the simulation is shown Figure C.1.

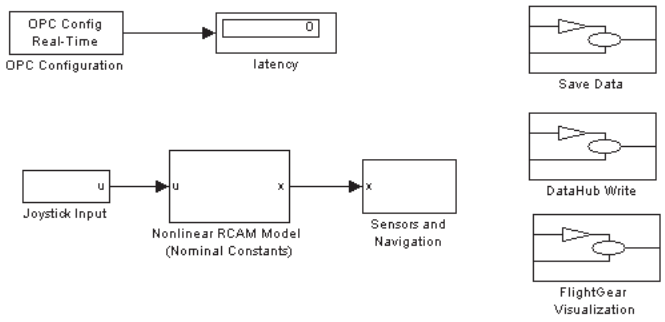


Figure C.1: Simulink block diagram of aircraft simulation.

The 'OPC Configuration' block is a block from the OPC blockset which forces the Simulink model to execute in real time. This is crucial when interfacing with human operators who must be trained on real-time simulations. It is also useful for measuring latency and performance. Relevant data from the model is then written to the OPC DataHub server as shown in Figure C.2.

The 'Joystick Input' block is a custom subsystem which handles the inputs from the joystick. The actual interface to the joystick is achieved via a block from the Aerosim blockset.

The 'Sensors and Navigation' block is shown in Figure C.3. This block is mostly con-

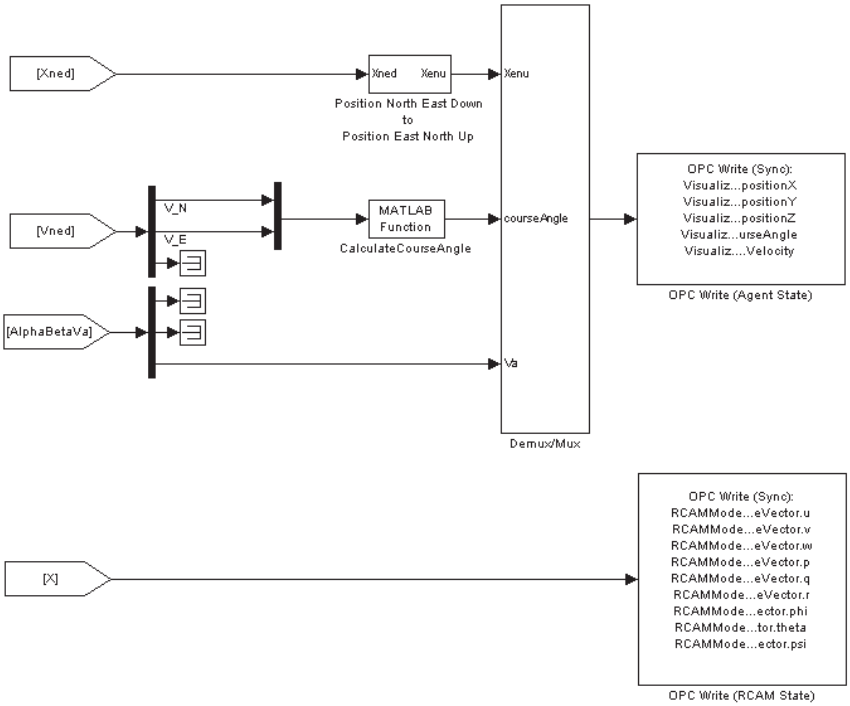


Figure C.2: Expanded view of the 'DataHub Write' block.

cerned with implementing navigation equations and calculating the inertial and geodetic position of the aircraft [64]. This information is crucial for visualization of the state of the aircraft via FlightGear as shown in Figure C.4.

In Figure C.4, the 'FlightGear 0.9.8 Interface' is provided by the Aerosim blockset to send relevant states to the FlightGear simulator in order to visualize the state of the aircraft.

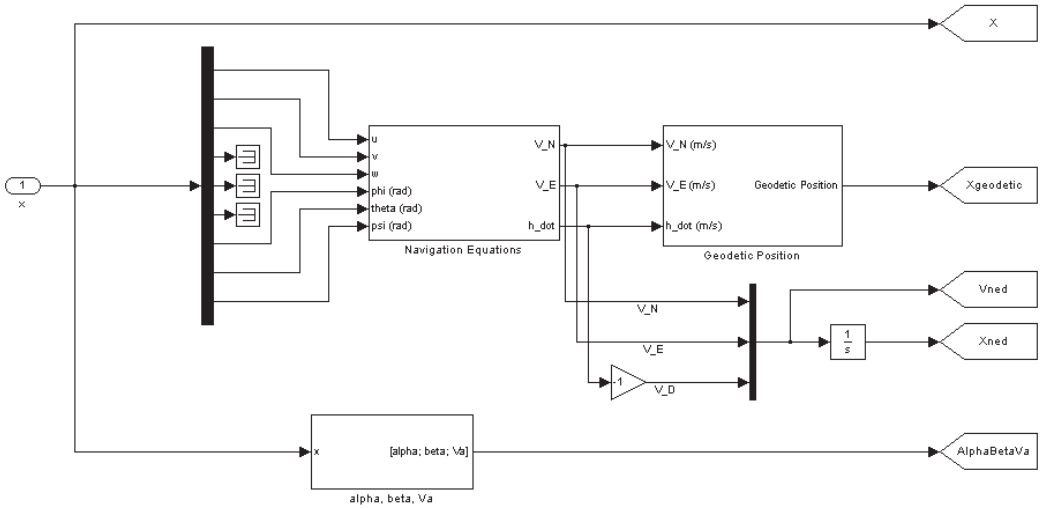


Figure C.3: Expanded view of the 'Sensors and Navigation' block.

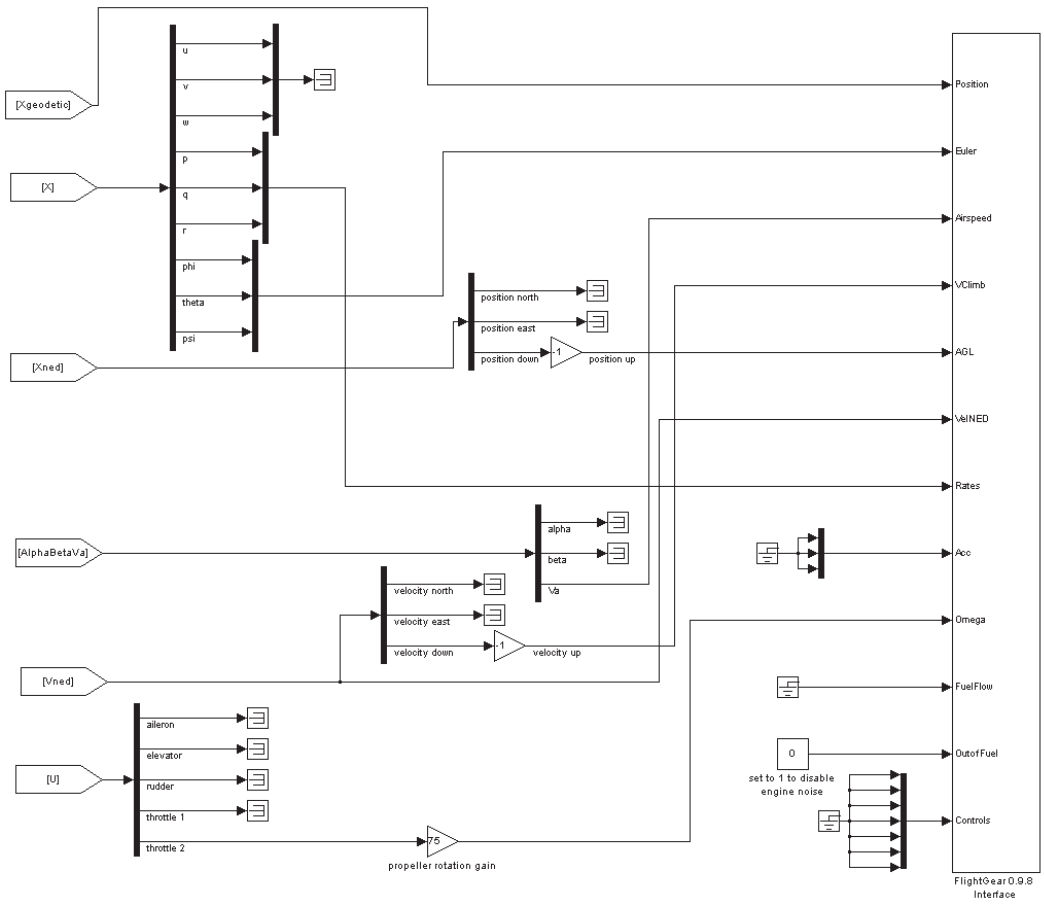


Figure C.4: Expanded view of the 'FlightGear Visualization' block.

Appendix D

VORONOI DIAGRAMS

The set $P = \{p_1, p_2, ..., p_n\}$ with $2 \leq n < \infty$ is often referred to as the generator set of the Voronoi diagram. Given the generator set, the Voronoi polygon or Voronoi partition associated with the generator point p_i is given by

$$V(p_i) = \{\overline{z} \mid ||\overline{z} - p_i|| \leq ||\overline{z} - p_j|| \text{ for } j \neq i, j \in I_n\} \tag{D.1}$$

The Voronoi diagram associated with this generator set is referred to as

$$\overline{V} = \{V(p_1), V(p_2), ..., V(p_n)\} \tag{D.2}$$

Since the Voronoi polygons are defined as closed sets (due to the $\leq$ instead of a $<$ in Eq. D.1), the Voronoi edge between two Voronoi polygons $V(p_i)$ and $V(p_j)$ can be written as

$$e(p_i, p_j) = V(p_i) \bigcap V(p_j) \tag{D.3}$$

It is worth noting that the edge $e(p_i, p_j)$ may consist of a single point or be empty. If it is neither empty nor a single point, the two the two polygons are considered to be adjacent to each other.

Similar to the Voronoi edge, the bisector between two Voronoi generators p_i and p_j is defined as

$$b(p_i, p_j) = \{\overline{z} \mid ||\overline{z} - p_i|| = ||\overline{z} - p_j||\} \tag{D.4}$$

Note that the Voronoi edge is a subset of the bisector ($e(p_i, p_j) \subseteq b(p_i, p_j)$) and the bisectors can be used to define a half space given by

$$H(p_i, p_j) = \{\overline{z} \mid ||\overline{z} - p_i|| \leq ||\overline{z} - p_j||\} \tag{D.5}$$

The half space $H(p_i, p_j)$ is often referred to as the dominance region of p_i over p_j. Using these dominance regions, a Voronoi polygon can alternatively be defined as

$$V(p_i) = \bigcap_{i \in I_n \setminus \{i\}} H(p_i, p_j) \tag{D.6}$$

Eq. D.6 effectively defines the Voronoi polygon $V(p_i)$ as an intersection of half spaces. Given that each half space is convex and the intersection of convex sets is still convex, each Voronoi polygon is convex by definition.

S is the space in which the generator set P is defined. Notice that the Voronoi diagram $\overline{V}$ spans the entire space S in the sense that

$$\bigcup_{i \in I_n} V(p_i) = S \tag{D.7}$$

In this sense, almost every point $\overline{z} \in S$ is assigned to at most two Voronoi partitions. The majority are assigned to single polygon and those points $\overline{z} \in e(p_i, p_j)$ are assigned to both polygons $V(p_i)$ and $V(p_j)$ (since it is equally close to both polygons).

This work considered the case where $S \subseteq \Re^2$ and $p_i \in \Re^2$. In this case, the resulting Voronoi diagrams are referred to as planar Voronoi diagrams.

An example of such a planar Voronoi diagram is shown in Figure D.1. In this figure, the generator points P are shown as black dots. The edges between the Voronoi polygons are shown as the solid black lines. The Voronoi polygon $V(p_1)$ is shown as the shaded grey region. It should be noted that many of these edges extend to infinity.

Voronoi diagrams and their properties are extensively detailed in [86].

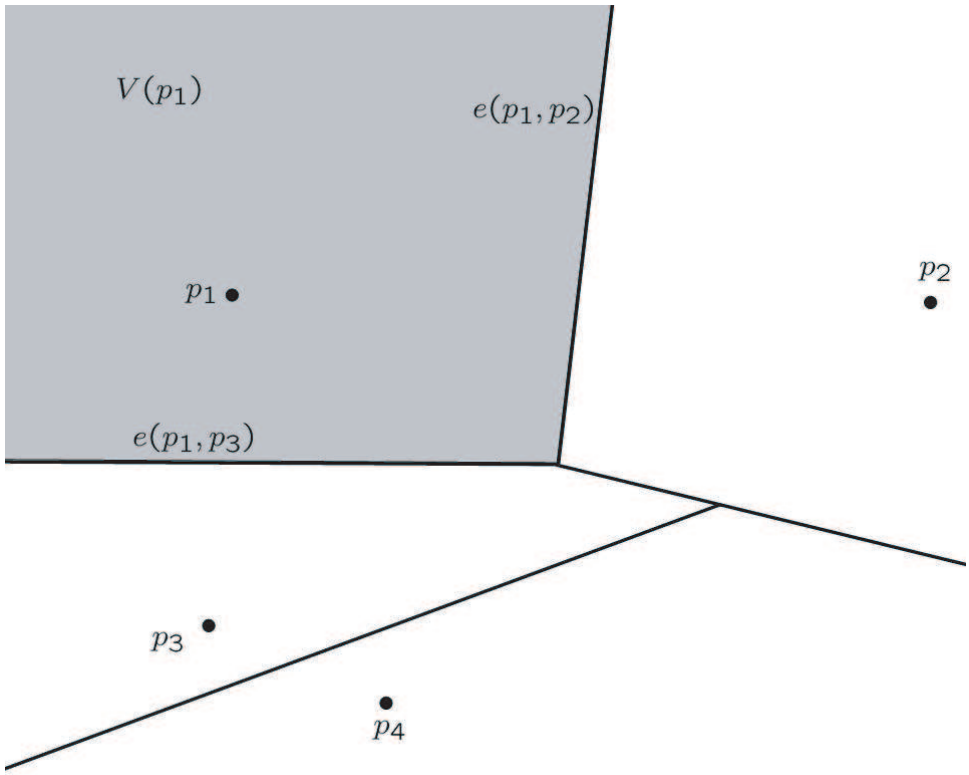


Figure D.1: Example Voronoi Diagram with $n = 4$.

VITA

Christopher Lum was born in Bellevue, Washington in 1980. He entered the University of Washington immediately following graduation from Mercer Island High School in 1999. As an undergraduate, he earned aerospace experience working at the Kirsten Wind Tunnel on campus and eventually began studies in the Department of Aeronautics and Astronautics in 2001. During this time, he was inducted into the Sigma Gamma Tau Aerospace Honor Society and awarded the George E. Solomon Academic Award and the University of Washington Aeronautics and Astronautics Alumni Scholarship. He graduated in 2003 with a Bachelor of Science in Aeronautical and Astronautical Engineering.

Christopher began work on a graduate degree in the same department in 2003 with a focus on controls and dynamical systems. During his tenure as a Masters student, he held various teaching assistant positions in undergraduate and graduate level controls courses and was nominated for the University of Washington Outstanding Teaching Assistant Award. In addition, he worked as an engineering intern at the Insitu Group, working on embedded systems for the ScanEagle UAV. During this time he was awarded the Andris Vagners Latvian Memorial Fellowship. He obtained a Master of Science in Aeronautics and Astronautics in 2005.

Following his Masters, in 2005, Christopher began his PhD program at the University of Washington under the supervision of Dr. Rolf Rysdyk and Dr. Juris Vagners. He continued and expanded his research of autonomous systems and unmanned aerial vehicles. He also furthered his teaching experience by instructing two graduate classes on aircraft flight dynamics and simulation. During this time he was awarded the Osberg Family Trust and the NASA Space Grant Consortium Graduate Fellowships. He obtained a Doctor of Philosophy in Aeronautics and Astronautics in 2009.

He currently lives on Bainbridge Island, Washington with his lovely wife, Alison.