



---

Theses and Dissertations

---

2014-03-03

## Managing Autonomy by Hierarchically Managing Information: Autonomy and Information at the Right Time and the Right Place

Rongbin Lin  
Brigham Young University - Provo

Follow this and additional works at: <https://scholarsarchive.byu.edu/etd>



Part of the [Computer Sciences Commons](#)

---

### BYU ScholarsArchive Citation

Lin, Rongbin, "Managing Autonomy by Hierarchically Managing Information: Autonomy and Information at the Right Time and the Right Place" (2014). *Theses and Dissertations*. 4014.  
<https://scholarsarchive.byu.edu/etd/4014>

This Dissertation is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact [scholarsarchive@byu.edu](mailto:scholarsarchive@byu.edu), [ellen\\_amatangelo@byu.edu](mailto:ellen_amatangelo@byu.edu).

Managing Autonomy by Hierarchically Managing Information:  
Autonomy and Information at the Right Time  
and the Right Place

Rongbin Lanny Lin

A dissertation submitted to the faculty of  
Brigham Young University  
in partial fulfillment of the requirements for the degree of  
Doctor of Philosophy

Michael A. Goodrich, Chair  
Bryan S. Morse  
Mark B. Colton  
Kevin Seppi  
David W. Embley

Department of Computer Science  
Brigham Young University  
March 2014

Copyright © 2014 Rongbin Lanny Lin  
All Rights Reserved

## ABSTRACT

Managing Autonomy by Hierarchically Managing Information:  
Autonomy and Information at the Right Time  
and the Right Place

Rongbin Lanny Lin  
Department of Computer Science, BYU  
Doctor of Philosophy

When working with a complex AI or robotics system in a specific application, users often need to incorporate their special domain knowledge into the autonomous system. Such needs call for the ability to *manage autonomy*. However, managing autonomy can be a difficult task because the internal mechanisms and algorithms of the autonomous components may be beyond the users' understanding. We propose an approach where users manage autonomy indirectly by *managing information* provided to the intelligent system hierarchically at three different temporal scales: strategic, between-episodes, and within-episode. Information management tools at multiple temporal scales allow users to influence the autonomous behaviors of the system without the need for tedious direct/manual control. Information fed to the system can be in the forms of areas of focus, representations of task difficulty, and the amount of autonomy desired. We apply this approach to using an Unmanned Aerial Vehicle (UAV) to support Wilderness Search and Rescue (WiSAR). This dissertation presents autonomous algorithms/components and autonomy management tools/interfaces we designed at different temporal scales, and provides evidence that the approach improves the performance of the human-robot team and the experience of the human partner.

Keywords: Unmanned Systems, Path Planning, Navigation, Human-Robot Interaction, Adjustable Autonomy, Sliding Autonomy, Bayesian Modeling

## ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Dr. Michael A. Goodrich for his continuous guidance and support throughout the research, for his encouragement, patience, trust in me, and for making this a thoughtful and rewarding journey. He sets an excellent example of a dedicated scientist. His thoughtful insights, both in scientific research and in living everyday life, have always helped me overcome one after another challenges. I am very grateful to have such a wonderful mentor, colleague, and good friend for life. I want to extend my appreciation to the rest of my dissertation committee: Dr. Bryan Morse, Dr. Mark Colton, Dr. Kevin Seppi, and Dr. David Embley for their support over the years as I grew from a clueless grad school student to a confident researcher.

I want to thank all those (lab mates, colleagues, reviewers, and etc.) who helped me with my research, whether it was a brainstorming idea, collaborative work, or critiques, for helping me grow intellectually. I would also like to thank the BYU Computer Science Department for providing an encouraging and inspiring research environment, so I can turn my passion about Artificial Intelligence and Robotics since childhood into a constructive career.

Last but not least, I would like to thank my wife, Noelle, for her patience/impatience and encouragement/discouragement during the journey, and my children, Adeline and Isaac, for their beautiful smiles, which inspire me to better myself day after day.

# Table of Contents

List of Figures	x
List of Tables	xiv
<b>1 Background, Motivation, and Overview</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.1.1 Problem Motivation . . . . .	1
1.1.2 General Solution Approach . . . . .	2
1.1.3 Application Domain . . . . .	5
1.2 Related Work . . . . .	7
1.2.1 General Research Area . . . . .	8
1.2.2 Supporting Wilderness Search and Rescue with a UAV . . . . .	11
1.3 Thesis Statement . . . . .	14
1.4 Project Description . . . . .	14
1.4.1 Solution Overview . . . . .	15
1.4.2 At the Strategic Scale . . . . .	17
1.4.3 At the Between-Episodes Scale . . . . .	18
1.4.4 At the Within-Episode Scale . . . . .	20
1.5 Dissertation Chapters . . . . .	23
<b>2 Paper: Supporting Wilderness Search and Rescue with Integrated Intelligence: Autonomy and Information at the Right Time and the Right Place</b>	<b>26</b>
2.1 Introduction . . . . .	27

2.2	Related Work . . . . .	28
2.3	UAV Overview . . . . .	30
2.4	Integration Challenges . . . . .	30
2.5	Autonomy Components . . . . .	33
2.5.1	Low-Level Autonomy . . . . .	33
2.5.2	Advanced Autonomy . . . . .	34
2.6	User Interfaces . . . . .	36
2.7	See-ability Metrics . . . . .	39
2.8	Demonstration . . . . .	40
2.9	Conclusions and Future Work . . . . .	42
<b>3</b>	<b>Paper: A Bayesian approach to modeling lost person behaviors based on terrain features in Wilderness Search and Rescue</b>	<b>43</b>
3.1	Introduction . . . . .	44
3.2	Related Work . . . . .	46
3.3	Terrain-Based Bayesian Model . . . . .	48
3.3.1	Model Overview . . . . .	48
3.3.2	Hypothetical Scenario . . . . .	50
3.3.3	Hexagonal Tessellation Discretization . . . . .	51
3.3.4	Model Representation . . . . .	52
3.3.5	Using the Model to Compute the Posterior . . . . .	60
3.3.6	Using the Model to Compute the Predictive Probability Distribution . . . . .	62
3.4	Evaluation of the Model . . . . .	64
3.4.1	Synthetic Data . . . . .	64
3.4.2	Prior vs. Marginal Posterior . . . . .	65
3.4.3	Correlation of Parameters . . . . .	66
3.4.4	Prior Predictive vs. Posterior Predictive . . . . .	67
3.4.5	Bayesian $\chi^2$ Test for Goodness-of-Fit . . . . .	69

3.5	Discussions and Limitations . . . . .	70
3.6	Conclusions and Future Work . . . . .	73
<b>4</b>	<b>Paper: UAV Intelligent Path Planning for Wilderness Search and Rescue</b>	<b>76</b>
4.1	Introduction . . . . .	77
4.2	Problem Formulation . . . . .	78
4.3	Related Work . . . . .	80
4.4	Path Planning Algorithms . . . . .	81
4.4.1	Algorithms without a Set Destination . . . . .	81
4.4.2	Algorithms with a Set Destination . . . . .	85
4.5	Experimental Results and Analysis . . . . .	87
4.5.1	Performance Metrics . . . . .	87
4.5.2	Typical WiSAR Scenarios . . . . .	88
4.5.3	Experimental Results and Analysis . . . . .	89
4.6	Conclusions and Future Work . . . . .	93
<b>5</b>	<b>Paper: Hierarchical Heuristic Search Using A Gaussian Mixture Model for UAV Coverage Planning</b>	<b>95</b>
5.1	Introduction . . . . .	96
5.2	Problem Formulation . . . . .	99
5.2.1	Problem Framework . . . . .	99
5.2.2	Performance Metrics . . . . .	101
5.3	Related Work . . . . .	102
5.4	Path Planning Algorithms . . . . .	105
5.4.1	BA and LHC-GW-CONV algorithms review . . . . .	105
5.4.2	Mode Goodness Ratio . . . . .	109
5.4.3	Top2 Algorithm . . . . .	112
5.4.4	TopN Algorithm . . . . .	113

5.5	Experiment Results and Analysis . . . . .	118
5.5.1	Experiment Set Up . . . . .	118
5.5.2	Experiments Results and Analysis . . . . .	119
5.6	Limitations and Discussion . . . . .	128
5.7	Summary . . . . .	131
<b>6</b>	<b>Paper: Sliding Autonomy for UAV Path Planning: Adding New Dimensions to Autonomy Management</b>	<b>132</b>
6.1	Introduction . . . . .	133
6.2	Autonomy Design Guidelines and New Dimensions . . . . .	136
6.2.1	Autonomy Design Guidelines . . . . .	136
6.2.2	Information Representation Dimension . . . . .	138
6.2.3	Spatial Constraints Dimension . . . . .	140
6.2.4	Temporal Constraints Dimension . . . . .	141
6.3	Related Work . . . . .	143
6.4	Hypotheses . . . . .	145
6.5	User Study Design . . . . .	146
6.5.1	Simulation Environment . . . . .	146
6.5.2	Scenarios . . . . .	149
6.5.3	Secondary Task . . . . .	149
6.5.4	Measures . . . . .	150
6.6	Results and Analysis . . . . .	151
6.6.1	Comparing Across Scenarios . . . . .	151
6.6.2	Comparing Across Planning Methods . . . . .	152
6.6.3	Additional Factors . . . . .	155
6.7	Discussion . . . . .	155
6.7.1	Planning Methods Characteristics . . . . .	155
6.7.2	Reliance on Autonomy . . . . .	159



6.7.3	Why Human-Autonomy Team Performs Better? . . . . .	161
6.7.4	Why Similar Secondary Task Performance in All Three Methods? . . . . .	162
6.8	Conclusions and Future Work . . . . .	163
<b>7</b>	<b>Probability Distribution Map and Task-Difficulty Map Editor Interface</b>	<b>165</b>
7.1	Introduction . . . . .	165
7.2	The DiffEdit Tool . . . . .	166
7.2.1	Editing vs. Starting New . . . . .	167
7.2.2	3D Navigation Controls . . . . .	168
7.2.3	Paint Mode vs. Lasso Select Mode . . . . .	170
7.3	The DistEdit Tool . . . . .	172
7.3.1	Editing vs. Starting New . . . . .	173
7.3.2	3D Navigation Controls . . . . .	173
7.3.3	Paint Mode vs. Lasso Select Mode . . . . .	174
7.3.4	Cross-Platform Support . . . . .	178
<b>8</b>	<b>Conclusions and Future Work</b>	<b>179</b>
8.1	Conclusions . . . . .	180
8.2	Future Work . . . . .	181
8.2.1	At the Strategic Scale . . . . .	181
8.2.2	At the Between-Episodes Scale . . . . .	183
8.2.3	At the Within-Episode Scale . . . . .	184
8.2.4	The Overall Autonomy Management Approach . . . . .	185
	<b>References</b>	<b>188</b>
	<b>Appendix A Complexity Analysis of the UAV Path Planning Problem</b>	<b>200</b>
A.1	Computational Complexity of Dynamic Programming . . . . .	200

A.2 Computational Complexity of Reinforcement Learning (Approximate Dynamic Programming) . . . . .	201
<b>Appendix B Full Experiment Results for Chapter 5</b>	<b>203</b>
<b>Appendix C Hierarchical Decision Making and Hierarchical Coarse-to-Fine Search</b>	<b>205</b>
C.1 Hierarchical Decision Making in Choosing the Appropriate Path Planning Algorithm . . . . .	205
C.2 Hierarchical Coarse-to-Fine Search in Parameter Space . . . . .	206
<b>Appendix D Identifying Modes in a 3D Surface using Local-Hill Climbing with Memory</b>	<b>209</b>
<b>Appendix E Sliding Autonomy User Study Design and Full Results for Chapter 6</b>	<b>212</b>
E.1 User Study Design . . . . .	212
E.1.1 Participants . . . . .	212
E.1.2 Simulation Environment . . . . .	212
E.1.3 Scenarios . . . . .	215
E.1.4 Secondary Task . . . . .	215
E.1.5 Procedure . . . . .	216
E.2 User Study Results . . . . .	216
E.2.1 Comparing Across Scenarios . . . . .	216
E.2.2 Comparing Across Planning Methods . . . . .	217
E.2.3 Additional Factors . . . . .	217
E.2.4 Comparing Against Autonomy Performance Markers . . . . .	218

## List of Figures

1.1	Autonomy integration challenges . . . . .	3
1.2	Components of the overall UAV integrated intelligent system . . . . .	5
1.3	Managing information at three scales . . . . .	6
1.4	An example probability distribution map . . . . .	15
1.5	An example task-difficulty map . . . . .	15
1.6	Autonomous components and autonomy management tools . . . . .	16
1.7	An example probability distribution map generated with DistEdit . . . . .	19
1.8	An example task-difficulty map generated with DiffEdit . . . . .	19
1.9	An example UAV path generated by a path planning algorithm . . . . .	21
1.10	An example scenario of path planning using sliding autonomy . . . . .	22
2.1	A screenshot of the UAV operator interface . . . . .	29
2.2	Example posterior distributions and an algorithm-generated path . . . . .	34
2.3	Example of a video mosaic with an annotation . . . . .	36
3.1	Satellite imagery and terrain data of area by Payson Lake, Utah . . . . .	51
3.2	Hexagonal discretized tessellation showing past historical data . . . . .	52
3.3	Beta Distribution probability density function . . . . .	54
3.4	A graphical illustration of the proposed model . . . . .	59
3.5	Bayesian network with multiple observations . . . . .	60
3.6	Illustration of Monte Carlo method approximating posterior distribution . . . . .	62
3.7	Comparing prior distribution against posterior distribution . . . . .	63

3.8	Comparing prior distribution with marginal posterior distribution . . . . .	66
3.9	Parameter correlation . . . . .	68
3.10	Satellite imagery of Elder Box Peak in Utah . . . . .	73
4.1	Global Warming Effect . . . . .	82
4.2	An example single-point path crossover . . . . .	83
4.3	An example double-point path crossover . . . . .	83
4.4	Examples of mutations in EA-DIR and EA-Path algorithms . . . . .	84
4.5	Examples of mutations in EA-Path_E algorithm . . . . .	84
4.6	Paths found for unimodal, bimodal, and bimodal with overlap distributions .	88
4.7	Algorithm <i>Efficiency</i> comparison for simplified bimodal map . . . . .	91
4.8	Algorithm <i>Efficiency</i> comparison for simplified bimodal with overlap map . .	91
4.9	EA-Path performance for the real and simplified bimodal with overlap map .	92
4.10	More complex multimodal probability distribution map . . . . .	93
5.1	Two approaches to the probability-maximizing path planning problem . . . .	97
5.2	A synthetic WiSAR scenario . . . . .	108
5.3	Paths found when the UAV starts from a subregion with low task-difficulty .	108
5.4	Paths found when the UAV starts from a subregion with high task-difficulty	108
5.5	Illustrations of the Top2 algorithms . . . . .	114
5.6	Top2 algorithm pseudo-code . . . . .	114
5.7	Illustrations of the TopN algorithm . . . . .	116
5.8	TopN algorithm pseudo-code . . . . .	117
5.9	Satellite imagery of the search area for the HikerPaul scenario . . . . .	119
5.10	The HikerPaul scenario . . . . .	120
5.11	Paths generated for HikerPaul scenario with $T = 900$ . . . . .	121
5.12	The Gaussian Mixture identified for the HikerPaul scenario . . . . .	121
5.13	The NewYork53 scenario . . . . .	122

5.14	Paths generated for NewYork53 scenario with $T = 900$ . . . . .	123
5.15	The NewYork108 scenario . . . . .	124
5.16	Paths generated for NewYork108 scenario with $T = 900$ . . . . .	125
5.17	Paths CDPs comparison at $T=900$ with partial detection. . . . .	127
5.18	Synthetic scenario UAV path starting from high task-difficulty subregion . .	130
6.1	Autonomy integration challenges . . . . .	137
6.2	A screen capture of the sliding autonomy tool showing a path segment . . .	139
6.3	Simulation interface and distribution and task-difficulty maps . . . . .	147
6.4	Performance differences for the three path planning methods. . . . .	153
6.5	Box plots of the NASA TLX scores for each scenario. . . . .	155
6.6	Comparing sliding autonomy performance against various markers. . . . .	159
7.1	An example probability distribution map generated with DistEdit . . . . .	166
7.2	An example task-difficulty map generated with DiffEdit . . . . .	166
7.3	Task-difficulty map generated with DiffCreate (real WiSAR scenario) . . . .	167
7.4	Satellite imagery overlay and paintbrush and lasso tools examples . . . . .	169
7.5	Lasso selection tool used to paint an area with task difficulty easy . . . . .	171
7.6	Probability distribution map created with DistCreate (real WiSAR scenario)	172
7.7	Examples of adding/subtracting a Gaussian to a probability distribution . .	174
7.8	Adding a Gaussian to the systematically generated distribution . . . . .	175
7.9	Examples of elliptically-symmetric and asymmetric bivariate distributions . .	176
7.10	An example of overlaying satellite imagery and lasso select . . . . .	178
8.1	Autonomous components and autonomy management tools at each scale . .	181
8.2	A mock up screen for the management tool interface at the strategic scale. .	183
C.1	A synthetic WiSAR scenario . . . . .	206
C.2	Performance of the Top2 algorithm when flight time allocated varies . . . . .	207

D.1	An example 3D surface with 4 modes. . . . .	209
D.2	An example path-type distribution resembling the Great Wall of China. . . . .	211
E.1	Simulation interface and distribution and task-difficulty maps . . . . .	213

## List of Tables

2.1	Autonomy integration challenges . . . . .	27
4.1	Algorithm efficiency comparison for bimodal distribution . . . . .	90
4.2	Algorithm speed comparison for bimodal distribution . . . . .	90
5.1	Algorithms <i>Efficiency<sub>LB</sub></i> and running speed comparison for the HikerPaul scenario . . . . .	121
5.2	Algorithms <i>Efficiency<sub>LB</sub></i> and running speed comparison for the NewYork53 scenario . . . . .	123
5.3	Algorithms <i>Efficiency<sub>LB</sub></i> and running speed comparison for the NewYork108 scenario . . . . .	125
5.4	Algorithms <i>Efficiency<sub>LB</sub></i> comparison for the multi-modal synthetic scenario at $T = 900$ . . . . .	131
6.1	Comparing across planning methods (SE stands for standard error) . . . . .	152
6.2	Percent of participants outperforming autonomy with each method . . . . .	154
6.3	Percent of participants outperforming autonomy performance markers . . . . .	160
B.1	Algorithms speed and <i>Efficiency<sub>LB</sub></i> comparison for the multi-modal synthetic scenario. . . . .	203
B.2	Algorithms speed and <i>Efficiency<sub>LB</sub></i> comparison for the HikerPaul scenario. . . . .	204
B.3	Algorithms speed and <i>Efficiency<sub>LB</sub></i> comparison for the NewYork53 scenario. . . . .	204
B.4	Algorithms speed and <i>Efficiency<sub>LB</sub></i> comparison for the NewYork108 scenario. . . . .	204

E.1	Comparing across scenarios (SE stands for standard error) . . . . .	217
E.2	Comparing across planning methods (SE stands for standard error) . . . . .	217
E.3	ANOVA Analysis Results for Additional Factors . . . . .	218
E.4	Percent of participants outperforming autonomy with each method . . . . .	218
E.5	Percent of participants outperforming autonomy performance markers . . . . .	218



# Chapter 1

## Background, Motivation, and Overview

### 1.1 Introduction

#### 1.1.1 Problem Motivation

Because of rapid advancement in technology, more and more Artificial Intelligence (AI) and robotics systems are appearing in various aspects of people's lives. For example, there are systems that assist humans to schedule limousine services [20], to evaluate and control the damage of oil spills<sup>1</sup>, to support search and rescue missions [17, 71], and to provide treatment to children with autism [96]. Such abundant and rapidly growing applications increase the set of possible interactions between human users and autonomous systems. The humans in such interactions are not likely the designers of the autonomous systems, but these humans must still manage the autonomy.

Although AI and robotics systems have grown to be able to handle increasingly complex tasks in uncertain environments, human assistance and supervision are often needed [3]. Even for so-called fully autonomous systems, human input can potentially improve the system's performance and safety. Human experts can use domain-specific knowledge to assist an AI/robotics system when it deals with changing environments, uncertainty, and case-specific scenarios. Therefore, it is necessary to design tools and interfaces that enable human users working with an AI/robotics system to manage the autonomous behaviors of the system

---

<sup>1</sup><http://spectrum.ieee.org/robotics/industrial-robots/the-gulf-spills-lessons-for-robotics>

efficiently and effectively; such tools can improve task performance and the experience of a human partner in human-autonomy interaction.

However, human users often do not understand how the internal mechanisms of an autonomous system work especially when the system is complicated or when complex algorithms are involved. Instead, humans must rely on their own mental models of the system during operation [80]. Supporting human interaction requires a design approach that lets users understand how autonomous behaviors can be influenced without getting deeply into how autonomy really works. This requirement makes designing for autonomy management especially challenging.

### 1.1.2 General Solution Approach

We propose that autonomy management tools should let users hierarchically manage information provided to an AI/robotics system. Good information management tools should allow users to influence the autonomous behaviors of the system at multiple temporal scales without the need for tedious direct/manual control. This dissertation presents autonomous algorithms/components and autonomy management tools/interfaces designed at different temporal scales and show that this approach improves the performance of the human-autonomy team and the experience of the human-autonomy interaction.

The term “information” here covers a wide range of things including *knowledge of the environment* (including other humans, equipment, and changes in the physical surroundings), *knowledge of the task* at hand (including processes, procedures, rules, past experiences, etc.), and *interactions among various entities* (task, environment, human, and the system). In theory, an AI/robotics system can obtain, process, and analyze information in order to complete the desired tasks. In practice, however, the system often has limited sensing and reasoning capabilities, and there is useful information the system is either not capable of obtaining or not able to understand/process. Often, the human users of such systems have much better “information sensing” capabilities. These capabilities allow humans to

		Attributes of intelligence		
		Capability	Information Management	Performance Evaluation
Organizational Scale	Intelligence of individual tools	Autonomy	Flexibility	Progress toward individual goal
	Intelligence of collaborative agents	Interactivity	Manageability	Team progress (individual or collective goal)
	Intelligence of distributed system	Modularity	Fusion	Collective progress/quality

Figure 1.1: Autonomy integration challenges defined along two dimensions. Horizontal dimension: attributes of intelligence. Vertical dimension: organizational scale.

obtain information from their own resources such as past experiences, domain-specific training, external communications (with team members, external systems, etc.), or even the AI/robotics system itself. The human user is also capable of “digesting” various information and then feeding the “filtered” information to the system in forms the system can understand. In a sense, the human user acts as an “intelligent sensor” for the system. At the same time, by deciding what information to provide to the system, the human user has a way of influencing the system’s autonomous behaviors without the need for tedious manual control.

In an overall integrated intelligent system, autonomy typically exists as component tools with the goal to offload portions of responsibility to autonomous algorithms. Figure 1.1 lists some key elements of the autonomy integration challenges we identified in [71] and Chapter 6 along two dimensions: *attributes of an intelligent system* (capability, information management, performance evaluation) and *organizational scale* (individual versus group). This table provides guidelines on what attributes should be designed into an autonomous component when it is part of a human-autonomy collaboration team and a much larger distributed intelligent system. More detailed description of the guidelines will be presented in Chapter 2 and 6 of the dissertation.

Good autonomy management tools will only let users manage information that allow them to develop a clear causal relationship between information management actions and the

changes of the system’s autonomous behaviors. Examples of such information include what data set to use to train the system and which tasks deserve more attention. Such causal relationships make developing correct mental models of the system easier leading to improved task performance [79].

Information can be managed at different temporal scales. We propose a general hierarchical framework that focuses on the following three temporal scales: **strategic**, **between-episodes**, and **within-episode**<sup>2</sup>.

When an AI/robotics system is given a task, in the long-term span, the system can generate a plan based on data and models from similar problems and follow general trends. Such a plan is normally a strategic one without much details and enforced throughout the entire “campaign”. Therefore, planning happens at the **strategic** scale. Since the same task can be repeatedly performed during the operation (with different environments, constraints, or phases of the operation), case-specific attributes and requirements need to be evaluated, and the initial plan needs to be tailored to the specific case or episode. This step is planning at the **between-episodes** scale with a medium-term span. During execution of the task, the plan is carried out during the short-term span. But as new information becomes available or when the environment changes due to uncertainty, the plan needs to be modified in real time to achieve better task performance. This is planning during a “battle” and happens at the **within-episode** scale. If the user of the system can manage information provided to the system at different temporal scales, he/she can change the system plan and indirectly influence the autonomous behaviors of the system.

To evaluate the usefulness of the proposed autonomy management approach, we apply it to the application domain of using a UAV (Unmanned Aerial Vehicle) to support Wilderness Search and Rescue (WiSAR). In the next section we explain what these temporal scales means in that context.

---

<sup>2</sup>The term “episode” we use is similar to the one Russell and Norvig define in Chapter 2 of [98] when they discussed episodic vs. sequential task environments. Our definition is more relaxed to include cases where actions taken in previous episodes might impact the current episode with respect to task objectives, but each episode is still by itself a separate and self-contained unit.

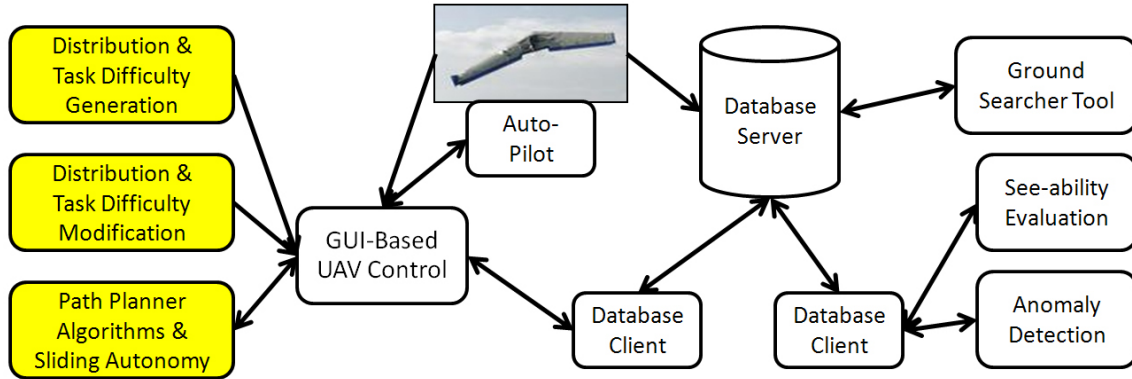


Figure 1.2: Various components of the overall intelligent system (a distributed system) of using a UAV to support Wilderness Search and Rescue. The three highlighted components are related to UAV path planning.

### 1.1.3 Application Domain

A small camera-equipped UAV can quickly and cheaply provide aerial imagery of a wilderness search area, especially hard-to-reach areas [41]. The MAGICC lab, the HCMI lab, and the Computer Vision Lab at BYU have been researching UAV technologies for several years and made great progress in UAV path-planning control, user interface design, and computer vision [71]. Figure 1.2 shows the various components developed by the research group. This dissertation focuses on the three highlighted components that are related to the UAV path planning problem.

Past UAV field trials indicate that real WiSAR searchers like not having to worry about keeping the UAV in the air or setting waypoints manually. Autonomy that offloads or complements some search work is useful, but searchers also need to be able to manage where to send the UAV as new evidence is gathered or hard-to-reach areas are identified. Ideally, searchers need not understand the statistical models or complex algorithms used by the UAV. Rather, searchers should manage autonomy by managing information provided to the UAV system at different temporal scales. We focus on two important representations of information: a *probability distribution map* and a *task-difficulty map* as shown in Figure 1.3.

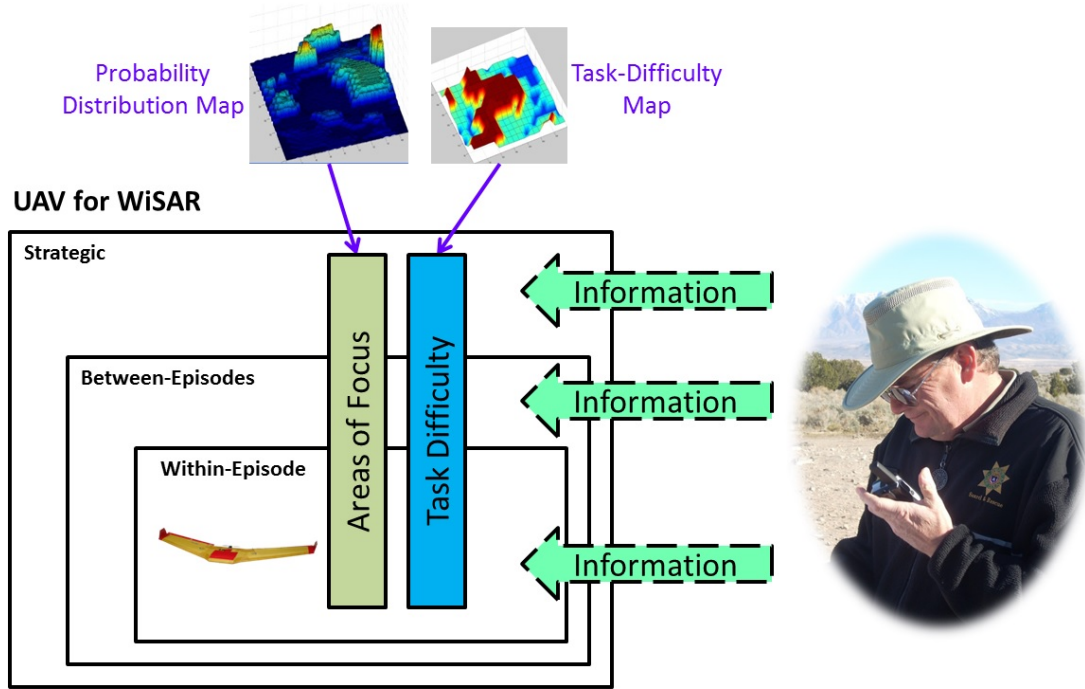


Figure 1.3: WiSAR searchers can manage autonomy by managing a *probability distribution map* and a *task-difficulty map* at different temporal scales of the system.

Following the guidelines in Figure 1.1, we developed autonomous components/algorithms and management tools/interfaces that meet the challenges of an integrated system and also support collaboration between a human agent and an autonomous agent at different temporal scales.

In this context, planning at the **strategic** scale means predicting where are likely places to find the missing person based on past WiSAR scenarios and determining how to allocate resources (ground searchers, K-9 units, volunteers, and manned/unmanned planes) based on task difficulty. At the **between-episodes** scale, between each UAV flight, information collected from previous flights and by other search teams are analyzed and incorporate into the plan to determine how the UAV can be more efficiently used in the next flight. Then at the **within-episode** scale, real-time data collected by the UAV and searcher insights are used to adjust the UAV flight plan while the UAV is still in the air searching.

At the **strategic** scale, we developed the **DistCreate** and **DiffCreate** components. **DistCreate** (Chapter 3) uses a Bayesian model to predict the probability distribution of the

missing person’s likely location, using terrain features of the search area and past human behavior data in the form of GPS track logs. Domain users can affect the model-generated probability distribution by changing prior beliefs parameters and by determining what human behavior data (data for selected regions, season, or missing person characteristics) to feed to the model. **DiffCreate** (Chapter 5) is a tool that automatically generates a task-difficulty map, a representation marking areas with low probability of detection, based on vegetation coverage Landsat data from USGS satellite imagery servers<sup>3</sup>.

At the **between-episodes** scale, we developed two management tools: **DistEdit** (Chapter 7), a tool that lets the user modify the probability distribution to specify areas of focus with simple gestures, and **DiffEdit** (Chapter 7), a tool that lets the user use scribbles to modify the task-difficulty map.

At the **within-episode** scale, we developed multiple path planning algorithms that support partial detection and real-time feedback. We also present a **SlidingAutonomy** interface (Chapter 6) that lets a human manage the amount of UAV autonomy along two dimensions: a temporal constraints dimension deciding how much time is granted to a UAV for each path segment, and a spatial constraints dimension using path segment ending points to impose priorities to the path planning task.

## 1.2 Related Work

In their in-depth survey paper [39], Goodrich and Schultz define the HRI problem as “understanding and shaping the interactions between one or more humans and one or more robots.” They also specified robot-assisted search and rescue as a key area for HRI research. In this section we first present related work in the general research area, then discuss related research more specific to the domains of using UAVs to support Wilderness Search and Rescue.

---

<sup>3</sup><http://www.usgs.gov/pubprod/>

### 1.2.1 General Research Area

When humans and robots work together as a team, balancing responsibilities between human and autonomy becomes a difficult challenge. Drucker defines automation as a “concept of the organization of work [28].” In their 1978 seminal paper [106], Sheridan and Verplank propose the idea of a *level of autonomy* spectrum. At one end of the spectrum is full teleoperation and at the other is full autonomy. In the middle of this spectrum, the robot could suggest actions to humans or make decisions before informing humans. Parasuraman et al. [90] extended this one-dimensional spectrum to four different broad functions: information acquisition, analysis, decision selection, and action implementation.

In [105] Sheridan proposes *supervisory control*, in which a human divides the task into a sequence of subtasks that the robot is capable of performing, and the human then provides guidance when the autonomous system cannot solve a problem on its own. In contrast to the top-down philosophy of supervisory control, a *mixed-initiative* approach advocates the idea of dynamically shifting tasks when necessary [47]. *Collaborative control*, which can be thought of as an instance of mixed-initiative interaction, is a robot-centric model; instead of the human always being in-charge, the robot is treated as a peer and can make requests to humans through dialogs [34]. *Adjustable autonomy* [26] (also referred to as *sliding autonomy* [25] or *adaptive automation* [97]) is another type of mixed-initiative interaction, one that enables the human-autonomy team to dynamically and adaptively allocate functions and tasks among team members. Bradshaw et al. [14] propose two dimensions of Adjustable Autonomy (descriptive and prescriptive) to address the two senses of autonomy (self-sufficiency and self-directedness) and discuss how permissions, obligations, possibilities, and capabilities can be adjusted. Bradshaw et al. [15] also summarized some widespread misconceptions on autonomy and listed seven deadly myths of “autonomous systems.”

Scholtz defines in [113] four roles for a human in human-robot interaction: supervisor, operator, mechanic, peer, and bystander. Goodrich and Schultz suggest two more roles: mentor and information consumer [39]. Humphrey and Adams add another role, abstract



supervisor [54]. With our information management approach, users can act as “intelligent sensors” and manage what information to feed the system at different temporal scales. Therefore, the human is taking on a smart “information sensor” role in HRI.

The idea of using humans as sensors is not new. For example, Kaber et al. advocate using humans as active information processors in complex systems to support situation awareness and effective performance [57]. Bourgault et al. include humans as augmented sensor nodes in a wilderness search task [12]. Other researchers have experimented with management at different resolution. Dias et al. [25] propose enabling interactions at different levels of granularity. However, using information as a control mechanism to manage autonomy at the three distinctive temporal scales we identified is different from previously published approaches.

One component of our proposed solution, the **SlidingAutonomy** interface, falls under the category of *Adjustable Autonomy*. Dorais et al. [27] discuss a framework for human-centered autonomous systems for a manned Mars mission. The system enables users to interact with these systems at an appropriate level of control but minimize the necessity for such interaction. Bradshaw et al. discuss principles and pitfalls of adjustable autonomy and human-centered teamwork, and then present study results on so-called “work practice modeling” and human-agent collaboration in space applications [13]. In [58] Kaber et al. describe an experiment simulating an air traffic control task where manual control was compared to Adaptive Automation (AA). Results suggest that humans perform better with AA applied to sensory and psychomotor information-processing functions than with AA applied to cognitive functions; these results also suggest that AA is superior to completely manual control. Brookshire et al. present preliminary results for applying sliding autonomy to a team of robots performing coordinated assembling work to help the system recover from unexpected errors and to thereby increase system efficiency [16]. Dias et al. identified six key capabilities that are essential for overcoming challenges in enabling sliding autonomy in peer-to-peer human-robot teams [25].

The human is an integral part of the human-autonomy team. When working with autonomy, the human often takes on the supervisor role. Bainbridge points out that automation requires the human operator to take additional management responsibilities [3], and Sartar identified in [102] two automation management policies: *management by consent* and *management by exception*, defining whether the human always retain authority or can the system take initiative.

For complex automation, the human tends to rely on his/her *mental models* (defined by Norman in [86]) to manage the system. Moray [80] provides a good summary of how mental models are used and proposes that mental models “allow operators to think about causal structures and functions in systems which they must control...” Goodrich and Boer present a case study of Adaptive Cruise Control design and explain how an automobile driver can switch among multiple mental models and use different management strategies [37, 38].

Lee and See propose that because people respond to technology socially, trust guides reliance when unanticipated situations make it impractical or impossible to understand automation [66]. Hoffman et al. [50] suggest “active exploration for trusting”(AET) and hope this approach can promote both trust “calibration” and appropriate reliance.

Moray also points out that the operator’s internal model of the environmental and task dynamics can affect how the operator samples information from the environment, and display interfaces should be designed to attract the right amount of attention [78]. In [129] Vicente suggests to follow the ecological approach [94] and design interfaces compatible with the actual constraints of the environment so the operator’s understanding corresponds to the actual behavior of the system. Given these principles of interaction, our information management approach and proposed tools are compatible with these principles because they allow a user to infer causal relationship between user actions and autonomous behavior changes. The user interface designs enable the user to develop mental models of the system that match how the system truly works and thereby improve the human-autonomy interaction experience.

Given the user interface and framework for interaction mentioned above, it is necessary to evaluate the usefulness of the resulting system. Properly evaluating human-robot interaction has always been a challenging problem due to the diversity of team setups, environmental contexts, and tasks involved. Many metrics have been proposed in the literature. Crandall and Goodrich proposed a metric called *neglect time* to measure interaction efficiency [24]. Together with neglect time, Olsen and Goodrich later added *task effectiveness*, *robot attention demand*, *fan out*, and *interaction effort* to the list of metrics [89]. Steinfeld and et al. suggest some common metrics for standardizing task-oriented human-robot interaction [113]. In [88], Olsen presents a set of criteria for evaluating new UI systems. Crandall and Cummings propose in [23] a set of metric classes that can predict how many robots should be in the team and the system effectiveness for single-operator controlling multiple robots. We follow guidelines provided in these papers to validate our proposed solution.

### 1.2.2 Supporting Wilderness Search and Rescue with a UAV

The goal of our research is to support fielded missions in the spirit of Murphy's work [17]. UAV technology has emerged as a promising tool in supporting WiSAR [9, 83]. In [10, 11] Bourgault et al. describe how to use a Bayesian model to create paths for a single UAV or multiple coordinated UAVs to maximize the amount of probability accumulated by the UAV sensors. In [12] they also include scalable collaborative human systems as augmented sensor nodes and created paths for human ground searchers.

The BYU WiSAR research group has developed a variety of technologies to support Wilderness Search and Rescue with small fixed-wing UAVs [5, 41, 42, 71]. The UAV system has many autonomous capabilities. The UAV's autopilot can stabilize the UAV during flight, support waypoint following and auto launch/land modes, and provide gimbaled camera control. Simple flight patterns and safety features are available when combining the autopilot with UAV control interfaces [5, 71]. These basic UAV capabilities have been greatly extended to provide better WiSAR support: A Bayesian model was developed by Lin and Goodrich [70]

that uses terrain features to predict the likely locations of finding a lost person. Then, Generalized Contour Search [41] and intelligent path planning algorithms [69, 85] have been used to automatically generate flight paths for the UAV. A real-time temporally local mosaic technique [81] has been used to “stitch” multiple video frames to provide increased opportunity for detection and increased sense of relative spatial relationships for video analyst. Anomaly detection algorithms [124] are also available that can mark objects with unnatural colors and alert the video analyst. A metric named “see-ability” [82] was also developed to understand search-related video quality and to index geo-tagged video frames.

Many path planning algorithms in the literature address obstacle avoidance while planning a path to reach a destination using A\* [92], LRTA\* [53], D\* [114], Voroni diagrams [5, 8], or probability roadmaps and rapidly-exploring random tree (RRTs) [91]. Hierarchical heuristics approaches were also developed, such as Hierarchical A\* (HA\*) by Holte et al. [51], hierarchical task-based real-time path planning by Naveed et al. [74], and Hierarchical-AO\* (HiAO\*) by Meuleau and Brafman [84]. The algorithms we present solve a different path planning problem by generating paths that make efficient use of the limited travel time and maximizing the probability of finding the missing person. This is similar to the Vehicle Routing Problem [64] and the Orienteering Problem (OP) [36], which is a variation of the Traveling Salesman Problem (TSP) (with names such as Prize-Collecting TSP (PCTSP) [45] or TSP with profits [30]), and is known to be NP-Hard [109]. Vansteenwegen et al. [128] gives a good survey on the topic of OP, and listed various approaches such as exact methods [32, 65, 93], approximate heuristic approaches [19, 77, 93], a genetic algorithm approach [122], and an ant colony optimization approach [67]. These algorithms work well with OP problems that have a small number of nodes (21–100 nodes). However, our path planning problem requires up to 1800 nodes with added challenges of repeated visits and partial detection, which are defined later.

In order to intelligently plan paths for a UAV in a WiSAR context, it is necessary to understand missing person behaviors and generate a probability distribution of likely places to

find the missing person. Many researchers analyzed past WiSAR cases in order to understand missing person behaviors [48, 49, 60, 104, 119]. [120] describes how to use mathematical models to calculate the probability of detection, probability of area and probability of success. The paper also describes an example search mission. Researchers also looked at systematically utilizing GIS (Geographic Information System) information for search and rescue applications [31, 111].

Due to factors such as lighting conditions, dense vegetation, or human observer cognitive workload, even when sensor footprint covers the location of the missing person, probability of detection can be less than 1. In the 1950's, Koopman discussed the uncertainties in the act of detecting hostile submarines with radars and proposed a concept called the *instantaneous probability of detection by one glimpse* [61]. He presented simple search algorithms and demonstrated how search effort should be distributed given a prior probability distribution of the target and a known law of detection when only a limited total amount of search effort (or time) is available [62].

Over the years, search theory has evolved to be able to deal with more complex search problems. Stone [115] presents various search plans with partial detection models using Lagrange multipliers and maximization of Lagrangians in finding stationary target in very basic search problems when no false targets are present. Washburn [130] discusses how to construct optimal search paths for different search problems. The author also developed detection models based on radar/sonar and expanded the fundamentals of search theory to include moving targets. More recent work includes [85] where Niedfeldt et al. present a UAV path planning algorithm that utilizes probability of detection and maximizes the probability of identifying an object using a N-step lookahead method, and [99] where Ryan and Hedrick developed a control formulation for a fixed-wing UAV that minimizes the entropy of an estimate distribution over a receding horizon for searching a moving target over a fixed time horizon. Stone et al. used posterior probability maps and successfully located the wreckage of Air France Flight 447 [116]. Metrics such as Koopman's instantaneous probability of

detection by one glimpse [61], “seeability” proposed by Morse et al. [82], and terrain and vegetation information obtained from USGS [70] can be used to build a task-difficulty map representing probability of detection in different search subregions.

In this dissertation, the UAV technology, human search experts, and associated user interfaces is treated as an intelligent system with the integration of many component autonomous algorithms and user interfaces. Integration at this level requires tremendous effort. Salas and Fiore [100] provide great insights on challenges across people and machines, and across time and space in distributed teams. Sycara and Lewis [118] also asked the questions: 1) can a software agent perform the task? and 2) can the agent’s assistance contribute toward team performance? Tso et al. [127] identified that integrating a UAV into the search task creates at least two roles: a pilot that controls the UAV and a sensor operator that analyzes the sensor outputs. Lessons from other search-related domains [29] show that multiple roles are required and these roles can be supported by autonomy algorithms and user interface technologies. These findings motivate and guide our research in developing UAV technology to support WiSAR operations.

### **1.3 Thesis Statement**

Designing autonomous components and autonomy management tools that let users manage information provided to an intelligent system at different temporal scales allow users to influence the autonomous behaviors of the system without the need for tedious direct/manual control. This approach improves both the human’s experience during the human-autonomy interaction and the performance of the human-autonomy team.

### **1.4 Project Description**

We propose a new autonomy management approach that lets users manage the autonomous behaviors of an AI/robotics system by hierarchically managing information at

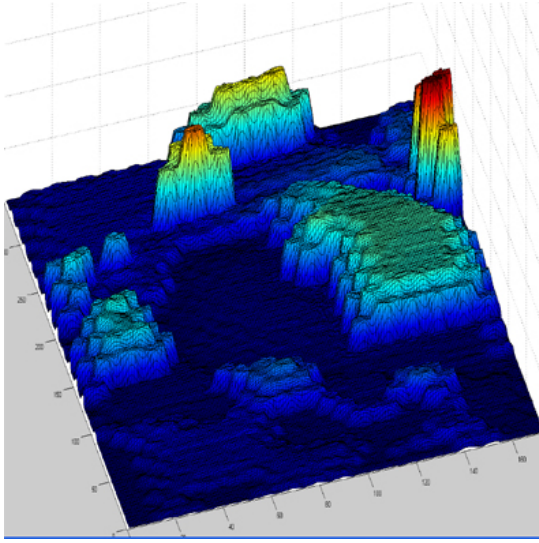


Figure 1.4: An example probability distribution map generated by a Bayesian model.

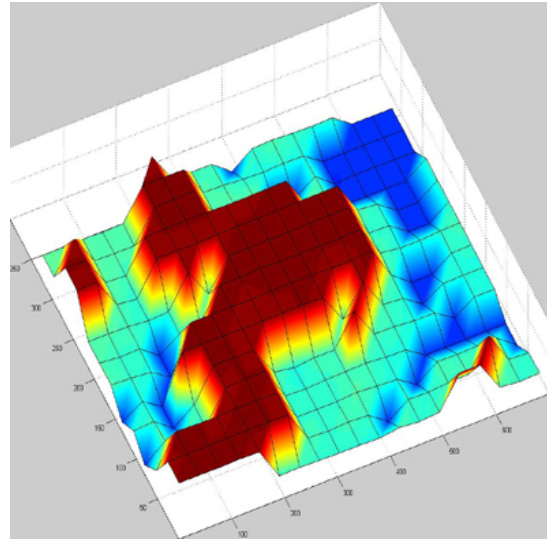


Figure 1.5: An example task-difficulty map generated using vegetation coverage data with three difficulty levels.

different temporal scales. In this section, we describe how we apply the approach to using a UAV to support WiSAR operations, and relate chapters of the dissertation to components of the hierarchical approach. At each temporal scale, we briefly discuss what autonomous components and autonomy management tools we developed, what kind of information the user can manage, and direct readers to the related chapter for more details.

### 1.4.1 Solution Overview

When using a UAV to support WiSAR operations, there are two important representations of information: a *probability distribution map* and a *task-difficulty map*. The probability distribution map encodes information about the likely location of the missing person and is illustrated in Figure 1.4. In the figure, high values correspond to areas with high probability. The task-difficulty map encodes information about how likely it is for a searcher to detect the missing person if they were in a particular location. Figure 1.5 illustrates a task-difficulty map with high values arising from areas with, for example, dense vegetation or low visibility, indicating that likely detection is low in that area.

	Probability Distribution Map	Task-Difficulty Map
Strategic	<b>DistCreate</b> for map creation	<b>DiffCreate</b> for map creation
Between-Episodes	<b>DistEdit</b> for map update and info management	<b>DiffEdit</b> for map update and info management
Within-Episode	<b>IPPA path planning algorithms</b> that support: <b>real-time feedback</b> and <b>partial detection</b>	
	<b>SlidingAutonomy</b> interface for autonomy management	

Figure 1.6: Autonomous components and autonomy management tools of the dissertation work at each temporal scale/hierarchy.

From a Bayesian perspective, the probability distribution map encodes prior and posterior beliefs, and the task-difficulty map encodes (one minus) the likelihood of detection. Both maps are needed for effective resource allocation and task prioritization. Throughout the operation, domain experts can process information only available to them or not comprehensible by autonomous components and then incorporate such information into the *probability distribution map* and the *task-difficulty map* at different temporal scales (hierarchies), thus managing autonomy by influencing the behavior of the autonomous components through information management.

These two maps can be created systematically using statistical models at the strategic scale by using general trends from reliable sources, acting as the general plan for resource allocation used throughout the entire search operation. They can be easily modified by users at the between-episodes scale to incorporate additional case-specific information obtained from the previous episode of searching. They can then be used to facilitate resource allocation and UAV path planning at the within-episode scale while the UAV is in the current episode of searching.



Figure 1.6 lists the various components of the dissertation work as they relate to these two map representations. We describe them in detail below at each respective temporal scale. All the autonomous components and autonomy management tools were designed following autonomy integration guidelines we defined in [71] (Chapter 2 of the dissertation), which was expanded in Chapter 6 (see Figure 1.1). Specifically, all the autonomous components we designed allow interactivity through the *probability distribution map* and the *task-difficulty map*, which enables the human agent to manage the behavior of autonomy. Additionally, the **SlidingAutonomy** management tool also lets the human interact and manage path planning autonomy through temporal and spatial constraints (Chapter 6 of the dissertation).

#### 1.4.2 At the Strategic Scale

At the strategic scale, a probability distribution map and a task-difficulty map can be created systematically.

We established in [70] (Chapter 2 of the dissertation) a Bayesian model (**DistCreate**) that can generate the probability distribution map systematically using three types of terrain features (topography, vegetation, and elevation) and past human behavior data. Searchers first specify transitional probabilities (Beta distributions) between two terrain features as inputs. Then the model produces the prior/posterior [98] predictive probability distribution(s), which can be used to allocate resources and plan UAV paths. Figure 1.4 shows an example posterior predictive probability distribution map generated by **DistCreate**. The user can influence the model-generated probability distribution by managing two types of information: *model parameters* and *dataset*.

We represent the probability distribution map in discretized form using a hexagonal tessellation of the search region. We use Monte Carlo methods to encode changes in the map, so *model parameters* include the users prior belief of the transition probabilities (probability of a missing person moving from one hexagonal cell to a neighbor). This approach is based

on the assumption that transition probabilities are easily interpreted by search experts, but algorithm parameters are not.

Although the term “dataset” can broadly include many sources of data, here we refer to GPS track logs of past human behavior. The user can choose whether or not to use past human behavior data. If the user chooses not to use past human behavior dataset, the **DistCreate** model will output the prior predictive distribution (prediction based only on the model parameters); otherwise, the model will output the posterior predictive distribution (prediction also based on past observations). The user can also choose what subset of the dataset to use, filtering past human behavior data by categories such as season of the year, region, or missing person profile. The user’s choice will indirectly affect the probability distribution map generated by the Bayesian model.

The **DistCreate** component at this temporal scale can download terrain vegetation coverage Landsat data directly from the USGS satellite image servers in real time when provided with GPS coordinates. Then it generates a task-difficulty map based on the type of vegetation in the specified region using a lookup table. For example, grassland is categorized as sparse vegetation and marked as easy detection area; shrub is categorized as medium vegetation and marked as medium detection area; evergreen forest is categorized as dense vegetation and marked as difficult detection area. Therefore, this component only utilizes vegetation density information when considering the difficulty level in sensor detection. Figure 1.5 shows an example task-difficulty map generated by **DiffCreate**. It is a very simple model, and we include it for completeness. Since it is modular, it can be easily extended or replaced by a more advanced detection model where factors such as “seeability” [82] (lighting condition, viewing angle, etc.) and video observer cognitive workload can be incorporated.

### 1.4.3 At the Between-Episodes Scale

At the between-episodes scale, a searcher might have additional case-specific information (e.g, past experience, knowledge of the search area or weather conditions, or the profile

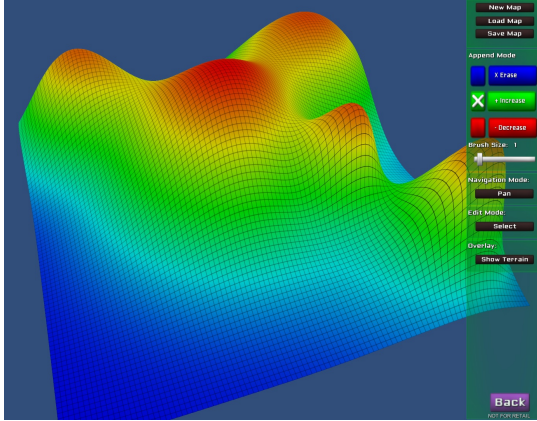


Figure 1.7: An example probability distribution map generated using the DistEdit tool.

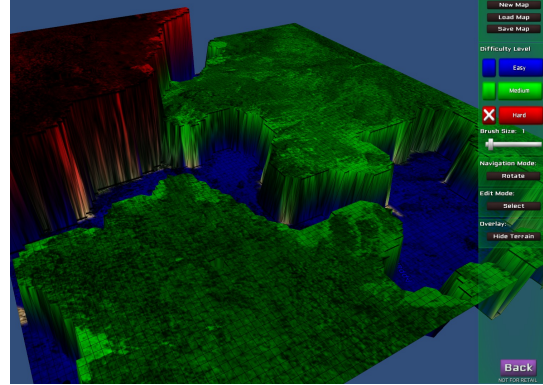


Figure 1.8: An example task-difficulty map generated using the DiffEdit tool with a satellite image of the search region overlaid on top.

of the missing person) and would like to modify the general plan produced at the strategic scale. Moreover, as search progresses, the search plan should change due to newly found evidence (or the lack of it) from either the ground searchers or previous UAV flights. We developed two autonomy management tools at this temporal scale that allow the user to manage two types of information: *areas of focus* and *task difficulty*. Chapter 7 explains how these two tools work in detail.

Searchers can use the **DistEdit** tool to modify a probability distribution map and use the **DiffEdit** tool to modify a task-difficulty map generated at the **strategic** scale. Both tools enable the user to view maps as 3D surfaces where a color map is applied for better distinction (red means high probability area or high task-difficulty level and blue means low vice versa). The user can use mouse and finger gestures to rotate/pan/zoom the respective map and edit the shapes of the maps in 3D to incorporate information that the autonomous components are unable to interpret. The user also has the option to overlay a satellite image of the search area on top of the maps for better alignment.

In **DistEdit** the user can paint Gaussian distributions onto the probability distribution map (in the form of a 3D surface) with a paintbrush tool to specify *areas of focus*. The mouse click (or finger press gesture) position determines the mean of the Gaussian distribution;

brush size determines the standard deviation (with a radius equivalent to three times the standard deviation); and the duration of the click (or finger press gesture) determines the scale (height) of the Gaussian distribution. Using this tool the user can add or subtract Gaussian components to the map to create a mixture of Gaussians. The modified probability distribution can be used later to prioritize tasks and plan UAV paths. By marking an area as a high priority area, the searchers can indirectly manipulate the UAV to search the area before other areas without the need to manually specify waypoints. Figure 1.7 shows an example probability distribution map generated using the **DistEdit** tool.

In **DiffEdit** the user can specify *task difficulty* by using a paintbrush tool to paint on the task-difficulty map with scribbles. The user can also use a lasso tool to specify a region of irregular shape and then mark the region with selected task-difficulty level. By marking areas as difficult, the user can indirectly tell the UAV to make multiple passes over these areas to search more thoroughly. Figure 1.8 shows an example task-difficulty map generated using the **DiffEdit** tool with the satellite image of the search region overlaid on top.

If the user does not like the probability distribution map or task-difficulty map generated at the **strategic** scale, he or she can also use the **DistEdit** and **DiffEdit** tools to create new maps from scratch. Both tools enable the searchers to add additional information (especially the type of information autonomous components cannot interpret) to the intelligent system, relying on UAV path-planning to use the information to search more efficiently.

#### 1.4.4 At the Within-Episode Scale

At the within-episode scale, given a probability distribution map marking likely places to find the missing person and a task-difficulty map indicating sensor detection probability in relationship to the spatial representation of the search area, efficient UAV flight paths need to be created quickly to support WiSAR operations. We have designed multiple path planning algorithms [69, 72] so given a starting point, (optionally) an ending point, and a desired flight duration, the intelligent path planning algorithms (IPPA) can generate flight

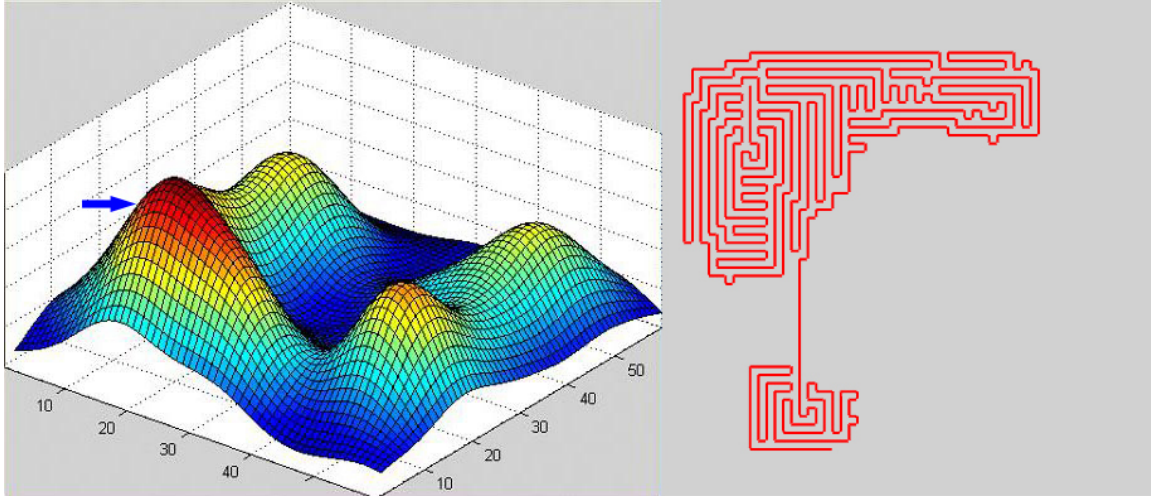


Figure 1.9: Example UAV path generated for a complex multi-modal distribution by a path planning algorithm. (The blue arrow indicates the starting point.)

paths that approximate the optimal path (see Figure 1.9 for an example) to maximize the probability of finding the missing person.

We present in [69] (Chapter 4 of the dissertation) multiple intelligent path planning algorithms using Local-Hill Climbing, Potential Field, lawnmower patterns, and Evolutionary Algorithm techniques. We evaluate the performance of these algorithms against simple and complicated synthetic scenarios with the assumption of 100% detection probability (no task-difficulty map is used). Then in [72] (Chapter 5 of the dissertation) we extended these algorithms to support partial detection by introducing the task-difficulty map, and also present two new (Top2 and TopN) algorithms, which utilize the *Mode Goodness Ratio* heuristic we designed and enable a hierarchical search in the parameter space. We compare the performance of these algorithms against Bourgault's Algorithm [11] and the LHC-GW-CONV (which we present in Chapter 4 and [69]) algorithm using three real WiSAR scenarios, and the Top2 and TopN algorithms outperformed both algorithms. To improve computation time of these algorithms, we implemented hierarchical coarse-to-fine search and hierarchical decision making. Algorithms were also parallelized to take advantage of multi-core processor capabilities. More details of these techniques can be found in Appendix C.

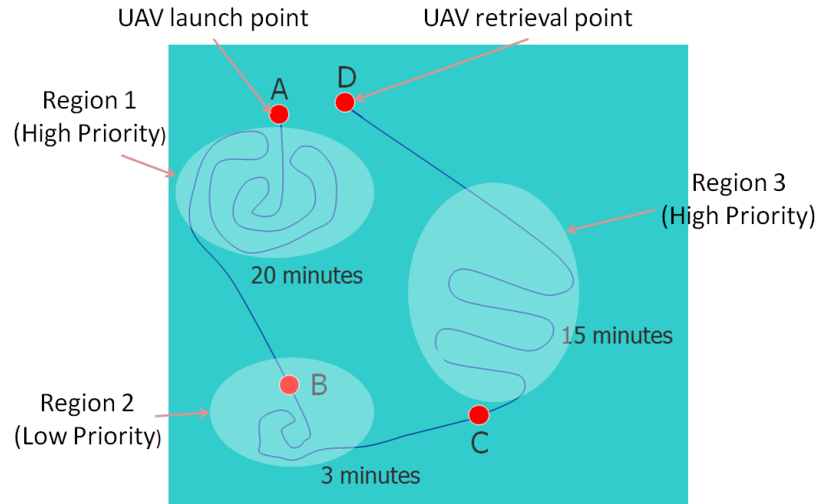


Figure 1.10: An example scenario of path planning using sliding autonomy: The UAV is launched at point A. Because region 1 is a high priority area, the searcher lets the UAV search for 20 minutes before arriving at point B, resulting in a longer flight path. Region 2 has low priority, so the searcher only gives the UAV 3 minutes before sending the UAV to point C, resulting in a short flight path. Region 3 is a high priority area, so the searcher gives the UAV 15 minutes. But the UAV also needs to reach Point D, the UAV retrieval point, at the end of the allocated 15 minutes. A medium length flight path is generated to meet the requirements.

When new evidence is gathered (from UAV aerial imagery or from ground searchers) while the UAV is flying, the search plan might need to be changed in real-time. At this within-episode scale, the information management tools **DistEdit** and **DiffEdit** previously proposed can still be used to update the probability distribution and the task-difficulty maps. This provides flexibility in autonomy management. Additionally, we have designed an autonomy management interface, **SlidingAutonomy**, that enables the user to prioritize search regions and manage the desired amount of the autonomous local search along two new dimensions: changing the desired flight duration (temporal control) and adding constraints (endpoints, spatial control).

The **SlidingAutonomy** tool allows a searcher to specify a starting point and (optionally) an ending point on the terrain satellite image overlay. Then, by moving a slider, the user can control how much flight time is granted, and the IPPA algorithms generate UAV paths within the local region. Beginning from the ending point of the previous flight path

segment, the searcher can plan the next path segment in the next search region. This way, the searcher can specify the order of different search regions and let the algorithm determine what paths the UAV should follow at each region. By setting flight duration (temporal constraints) and adding end points (spatial constraints), the searcher tells the UAV information about search priorities and at the same time, indirectly manages the local path planning based on his/her own judgment of how much the UAV can be trusted to cover a given area well. This autonomy management tool gives the user the flexibility of controlling the amount of autonomy desired without the burden of creating the entire flight path manually through waypoints. Figure 1.10 shows an example path depicting prioritized search regions and local paths. As the user moves the slider in the **SlidingAutonomy** tool, the system provide immediate visual feedback of what local path the system generates. This way, the searcher can easily infer the causal relationship between his/her actions (changes in flight duration) and the autonomous behaviors of the system (what path is generated).

We performed a user study to validate the usefulness of the approach. Experiment results show that the human-autonomy team outperforms human or autonomy working alone, reduces the human's cognitive workload, and improves the human experience in the human-autonomy interaction (Chapter 6 of the dissertation).

## 1.5 Dissertation Chapters

This dissertation consists of five papers, one of which is under review. This section gives a brief description of each chapter.

Chapter 1 gives an overview of our proposed autonomy management approach and provides an overall related literature review with respect to autonomy management approaches and path planning for UAV in the context of Wilderness Search and Rescue. It explains how all the components in our dissertation fit in the hierarchical structure and how they related to chapters of this dissertation.

Chapter 2, which was published in [71], presents autonomy integration guidelines we identified (see Figure 1.1) when integrating UAV autonomy to the WiSAR system. The paper emphasizes on how information management is an important attribute of an intelligent system. It also describes in detail how the UAV path planning problem fits in the overall intelligent system of using UAVs to support Wilderness Search and Rescue.

Chapter 3, which was published in [70], presents a Bayesian model that uses publicly available terrain features data to help model lost-person behaviors. This approach enables domain experts to encode uncertainty in their prior estimations and also makes it possible to incorporate human behavior data collected in the form of posterior distributions. It also enables the searcher to influence the probability distribution map generated by changing prior beliefs in the transitional probabilities between terrain features and by selecting what subset of past human behavior data to feed to the model.

Chapter 4, which was published in [69], explores several path planning algorithms and describe some novel techniques in solving the problem of maximizing sensor (UAV onboard video camera) coverage within a set time to support Wilderness Search and Rescue. The task-difficulty map is not used and we assume 100% detection probability. Performance of these algorithms are compared against typical WiSAR scenarios, and experiment results show that these algorithms yield high quality solutions that approximate the optimal solution.

Chapter 5, which was accepted in [72], proposes a heuristic, *Mode Goodness Ratio*, which uses a Gaussian Mixture Model to prioritize search subregions, and presents two path planning algorithms (Top2 and TopN) that utilize the heuristic and hierarchically search for effective paths through the parameter space at different levels of resolution. Performance of the new algorithms are compared against two published algorithms in simulated searches with three real search and rescue scenarios where both the probability distribution map and the task-difficulty map are used. Results show that the new algorithms outperform existing algorithms significantly when partial detection is considered, and can yield efficient paths that yield payoffs near the optimal.



Chapter 6, which is in preparation to be submitted to *Journal of Human-Robot Interaction*, extends the autonomy integration guidelines we identified in Chapter 2 to include collaborative agents when a human agent works together with the autonomous agent. It proposes two additional dimensions for autonomy management: spatial constraints and temporal constraints, and presents a new flavor of sliding autonomy where the human agent in the human-autonomy team can assign different flight duration to the path planning algorithms to plan path segments and use end points to manage search region priorities. A user study was performed to validate the usefulness of the approach. Experiment results show that this approach enables the human-autonomy team to outperform human or autonomy working alone, reduces the human’s cognitive workload, and improves the human experience in the human-autonomy interaction.

In Chapter 7 we describe the **DistEdit** and **DiffEdit** tools at the **between-episodes** scale in detail, and demonstrate how the *probability distribution map* and the *task-difficulty map* generated at the **strategic** scale can be modified using mouse and finger gestures to incorporate additional information. The two tools can also create maps from scratch.

Chapter 8 concludes findings of the dissertation work and summarizes the contribution of the dissertation. We also describe possible future work to extend the research at the three distinctive temporal scales we proposed for our autonomy management approach.

In Appendix A we present the complexity analysis of the UAV path planning problem and why a heuristic approach is preferred to both dynamic programming and reinforcement learning approaches. Appendix B presents the full experiment results for Chapter 5. Appendix C describes how hierarchical decision making and hierarchical coarse-to-fine search techniques are used in algorithm design. Appendix D explains the algorithm to identify modes on a 3D surface, which is used in our Top2 and TopN algorithms. Appendix E describes the sliding autonomy user study design in detail and presents the full experiment results for Chapter 6.

## Chapter 2

### **Paper: Supporting Wilderness Search and Rescue with Integrated Intelligence: Autonomy and Information at the Right Time and the Right Place<sup>1</sup>**

#### **Abstract**

Current practice in Wilderness Search and Rescue (WiSAR) is analogous to an intelligent system designed to gather and analyze information to find missing persons in remote areas. The system consists of multiple parts — various tools for information management (maps, GPS, etc) distributed across personnel with different skills and responsibilities. Introducing a camera-equipped mini-UAV into this task requires autonomy and information technology that itself is an integrated intelligent system to be used by a sub-team that must be integrated into the overall intelligent system. In this paper, we identify key elements of the integration challenges along two dimensions: (a) *attributes of intelligent system* and (b) *scale*, meaning individual or group. We then present component technology that offload or supplement many responsibilities to autonomous systems, and finally describe how autonomy and information are integrated into user interfaces to better support distributed search across time and space. The integrated system was demoed for Utah County Search and Rescue personnel. A real searcher flew the UAV after minimal training and successfully located the simulated missing person in a wilderness area.

---

<sup>1</sup>Published in Twenty-Fourth AAAI 2010 (Association for the Advancement of Artificial Intelligence) conference. Authors are Lanny Lin, Michael Roscheck, Michael A. Goodrich, and Bryan S. Morse.

	<b>Capability</b>	<b>Information Management</b>	<b>Performance Evaluation</b>
<b>Intelligence of individual tools</b>	Autonomy	Flexibility	Progress toward individual goal
<b>Intelligence of distributed system</b>	Modularity	Fusion (Communication)	Collective progress/quality

Table 2.1: Integration challenges defined along two dimensions. Horizontal dimension: attributes of intelligence. Vertical dimension: scale.

## 2.1 Introduction

Wilderness Search and Rescue (WiSAR) can be thought of as an intelligent system designed to gather and analyze information to find and assist humans who are lost or injured in remote areas such as deserts and mountains. The system consists of multiple parts — various tools for information management (maps, GPS, etc) distributed across personnel who have different skills. Using a camera-equipped mini-Unmanned Aerial Vehicle (UAV) to aid search can provide aerial imagery of a search area with the benefits of quick coverage of large areas, access of hard-to-reach areas, and lower cost than manned aircraft.

Introducing a UAV into the WiSAR system requires autonomy and information technology that itself is an integrated intelligent system to be used by a WiSAR sub-team, and this sub-team and associated technology must be integrated into the overall intelligent system. This integration inevitably creates the need for new roles and responsibilities in order to manage the UAV and the aerial imagery [1, 40]. The task of creating a useful technology for supporting these roles is to make sure that these responsibilities are performed by appropriate people at an appropriate time with a satisfactory level of performance. Doing this requires the creation of algorithms that efficiently offload portions of responsibility to autonomous algorithms, creating an intelligent distributed system that facilitates the coordination and information management among roles. The need for efficiency creates the need to monitor and evaluate the performance of the system as a whole.

In this paper we describe our efforts in developing autonomous algorithms and user interfaces that integrate components of machine and human intelligence with the goal of

making UAV technology useful to real searchers in WiSAR. Thus, this paper is consistent with Drucker’s definition of automation as a “concept of the organization of work [28].” Intelligently organizing work requires that we identify key elements of the integration challenges organized along two dimensions: *attributes of an intelligent system* (capability, information, performance evaluation) and *scale* (individual versus group); see Table 2.1. We then present component algorithms that augment or supplement search responsibilities. Next we describe how autonomy and information are integrated into user interfaces to better support distributed coordination of multiple searcher roles across time and space with respect to the integration challenges we identified.

Validating an integrated system is always difficult. The goal of our research is to develop technology that provides help to real searchers; therefore, we believe a good way to validate our integrated system is to put it through a test in a real-world environment in front of real users. We summarize the experience of a recent field demo for Utah County Search and Rescue team representatives, where a real searcher acted as the UAV operator in a simulated search and rescue mission after minimal training.

## 2.2 Related Work

The goal of our research is to support fielded missions in the spirit of Murphy’s work [17]. UAV technology has emerged as a promising tool in supporting WiSAR [9, 83]. The UAV technology is an intelligent system with the integration of many component autonomous algorithms and user interfaces (related work for these components are referenced in their relative sections). Integration at this level requires tremendous effort. For example, building robots (GRACE and Spartacus) that are capable of attending a conference [75, 107] required the integration of many technologies (e.g., localization/navigation, gesture/face recognition, and speech recognition/generation) and multiple modalities (e.g., mobility, vision, audition, and reasoning).

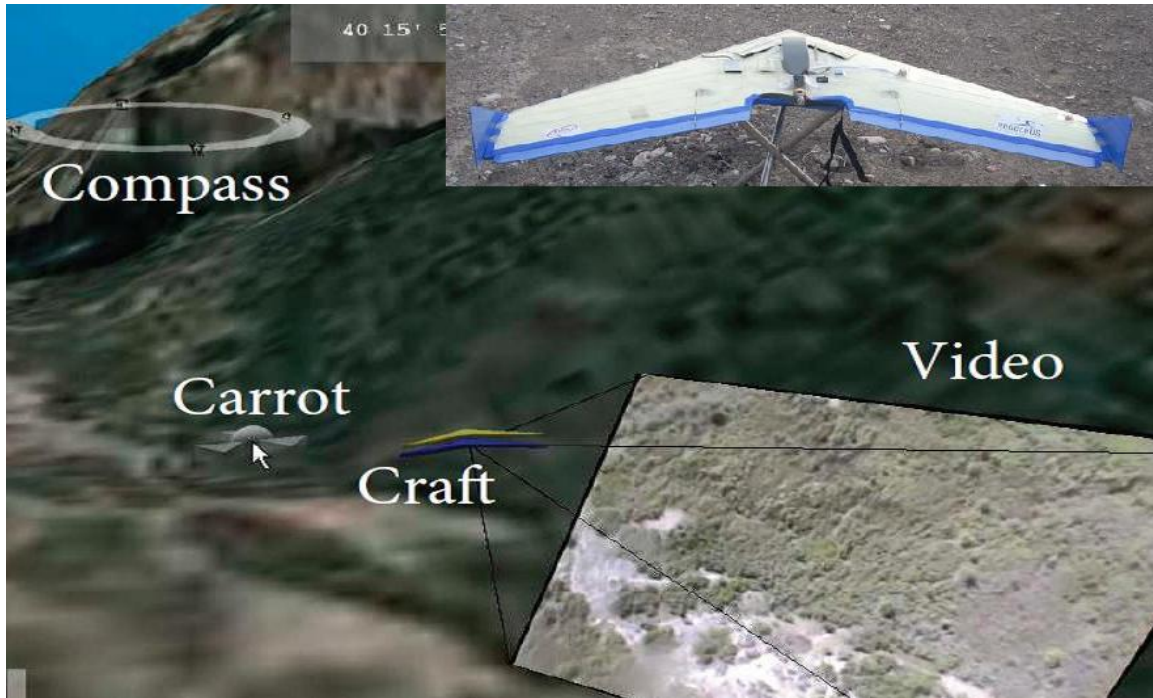


Figure 2.1: A screenshot of the UAV operator interface showing the position/orientation of the UAV, the orientation of the camera, and the projected video. (Top right: The UAV used in our research.)

To integrate the UAV intelligent system into existing WiSAR practices — which we argue is an intelligent system by itself [104] — creates additional challenges. Salas and Fiore [100] provide great insights on challenges across people and machines, and across time and space in distributed teams. Sycara and Lewis [118] also asked the questions: 1) can a software agent perform the task? and 2) can the agent's assistance contribute toward team performance? Tso et al. [127] identified that integrating a UAV into the search task creates at least two roles: a pilot that controls the UAV and a sensor operator that analyzes the sensor outputs, and lessons from other search-related domains [29] show that multiple roles are required and these roles can be supported by autonomy algorithms and user interface technologies. These findings motivate and guide our research in developing UAV technology to support WiSAR operations.

## 2.3 UAV Overview

UAVs used in our research have wingspans of 42-50 inches, weigh approximately 2 lbs, and use lithium battery-powered propellers (see Figure 2.1a). The airframes are designed so each UAV can stay aloft for up to two hours and travel at approximately 12-15 meters per second. The onboard sensors include three-axis rate gyroscopes, three-axis accelerometers, static and differential barometric pressure sensors, a GPS module, and a video camera on a gimbaled mount. An autopilot, designed by the BYU MAGICC lab [5], enables roll and pitch angle stabilization, attitude stabilization, altitude control, and waypoint following. The UAV uses a 900 MHz radio transceiver for data communication and an analog 2.4 GHz transmitter for video downlink. The typical operating height above ground is 60–100 meters so the UAV can avoid trees and slight terrain variations while still provide enough resolution so a human form can be discerned in the video [41].

## 2.4 Integration Challenges

We organize integration challenges along two dimensions: attributes (capability, information management, and performance evaluation) and scale (individual tool vs distributed system), as shown in Table 2.1. We assert that an intelligent system should display several attributes associated with intelligence across multiple scales. Capability pertains to the identification and development of specialized behaviors. Information management focuses on how information is presented, handled, and shared. Performance evaluation deals with monitoring the health of the system or progress toward the intended task goal. In this section we use this taxonomy to describe components of the UAV technology in the context of WiSAR.

The individual tools were designed partly in response to a cognitive task analysis conducted on the WiSAR domain to inform the design of UAV technology [1].

The analysis identified four primary search tactics used in WiSAR: *hasty search*, *constrained search*, *high priority search*, and *exhaustive search*. Also, observations from several user studies [22] show that the best perspective (e.g., chase, north-up) for detecting and localizing a target depends on the type of search and the type of distribution (likely places to find the missing person). These findings suggest that multiple control modes, path planning methods, and perspectives are needed to support various search tactics and scenarios. These are examples of the capability for *individual tools*. Since autonomous algorithms can replace or supplement searcher responsibilities, a wide range of capability is desired.

For the WiSAR system, the cognitive task analysis also identified two key WiSAR subsystems; information acquisition and information analysis. Combining this result with observations from past field trials, we see four roles emerge when a UAV is integrated into the search [41]. *UAV operator*: responsible for guiding the UAV and the gimbaled camera to various locations and monitoring the UAV; *video analyst*: responsible for scanning and analyzing imagery to detect potential clues; *mission manager*: responsible for managing the search and prioritizing efforts based on information obtained; *ground searcher*: (when supporting the UAV) responsible for investigating potential clues found in aerial imagery. Each role consists of a grouping of responsibilities. The task of creating useful technology for supporting these distributed roles is to make sure that these responsibilities are performed by appropriate people at an appropriate time with a satisfactory level of performance. Since people may take on (partial) responsibilities of other roles, the video analyst and the UAV operator might share responsibilities, these behaviors suggest that capabilities of individual systems should be modular to mix and match across roles. **Modularity** is a requirement for an intelligent distributed system – it is the adaptable chunking of responsibility and capability.

**Flexibility**, in information management, is the ability to appropriately match capability to task according to the information available to the operator. The cognitive task analysis indicated that WiSAR search is an iterative process of gathering, analyzing evidence

and planning for gathering more evidence, where probability refinement plays an important part during search. The analysis also identified that searches require considerable human judgment, especially as new evidence is collected. These findings suggest that tools and autonomy need to be flexible so they can be interrupted, temporarily aborted, and possibly resumed later. For example, if an object is spotted in the video, the UAV operator stops the current flight pattern and loiters around the Point Of Interest (POI) to gather more information. Once the UAV operator aborts the loiter mode, the UAV automatically resumes the previous flight pattern to continue to gather information.

For a distributed system, Information **Fusion** is an important element that efficiently combines and presents information from various sources to a user and also shares information among multiple users. For example, the user interface for the UAV operator includes the terrain map, an icon indicating the position and attitude of the UAV, an integrated video feed projected onto the terrain map showing the direction of the gimbaled camera, and various meters showing UAV status (e.g., speed, altitude, battery life); see Figure 2.1. Another example is a video analyst helping to annotate clues in video imagery, and communicating the data to the mission manager who can update the search plan accordingly.

For each individual tool, the ability to evaluate the quality and the progress toward the individual goal can be useful and represents the importance of performance evaluation. A coverage map, for example, improves the UAV operator's situation awareness of how well an area has been covered. Morse et al. [82] defined two see-ability metrics (described in the See-ability section). An *instantaneous see-ability* evaluation helps the video analyst get a sense of the quality of a single frame of video. As for the distributed system, an overall, or group quality evaluation is more appropriate. A mission manager might want to know the *collective see-ability* to understand how well the terrain is seen by all frames of video or combined coverage of the UAV and ground searchers.



In the following three sections we match components of our UAV system to the three attributes (columns) of our intelligent system taxonomy and show how they support various searcher roles at the right time and the right place.

## 2.5 Autonomy Components

This section presents a wide breadth of **autonomy** components currently in place to support searcher roles and responsibilities. They map to the **Capability** column in our taxonomy (Table 2.1). The **modular** design allows mix and match of autonomy components to support the distributed system. Here we use the term “low-level autonomy” to describe components that only involve simple math calculations in contrast to the term “advanced autonomy,” where complex algorithms and interrelationships are required.

### 2.5.1 Low-Level Autonomy

#### **Autopilot:**

Pitch/roll/yaw and altitude stabilization, attitude controls, and waypoint following.

#### **Deploy and retrieve:**

Two auto launch modes (take off to waypoint, spiral take off) and two auto land modes (rally land, deep stall).

#### **Gimbaled camera control:**

A point can be set on the terrain map so the gimbaled camera constantly aims at the point independent of the UAV’s flight path.

#### **Path planning:**

Simple flight patterns include spiral, lawnmowing, and Zamboni.

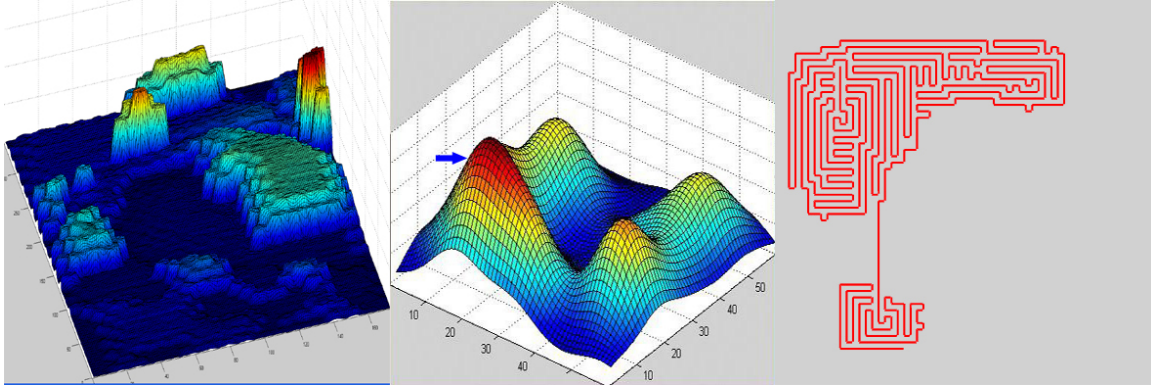


Figure 2.2: Left: a posterior predictive distribution at 200th time step generated using the Bayesian model. Middle: a multimodal distribution used to test path planning (arrow marks starting point). Right: path generated using Intelligent Path Planning.

### Safety:

If no waypoint is set or after reaching the end of set path, the UAV loiters around last waypoint. If the UAV loses communication with the base, it automatically returns to base and loiters.

## 2.5.2 Advanced Autonomy

### Distribution Generation:

A Bayesian model that incorporates past human behavior data and publicly available terrain data (topography, vegetation, and elevation) is used to automatically generate a posterior predictive distribution of the likely places of finding the missing person [68]. The Markov chain Monte Carlo Metropolis-Hastings algorithm generates a temporal distribution showing how the distribution changes over time (Figure 2.2). Given enough time, the distribution converges to a static state. The resulting distribution can be used by the search manager to prioritize search resources and develop search plans. It can also be used by the UAV operator for automatic path planning to maximize accumulated probability for a fixed flight duration.

## **Path planning:**

Two advanced path planning algorithms are described here: *Generalized Contour Search* [41] and *Intelligent Path Planning* [69]. The ability to control the gimbaled camera to aim at a target point while orbiting enables a *Generalized Contour Search* path planning algorithm. A queue of target points that follow the contours of the distribution of the missing person's likely locations can be created from which the algorithm interpolates (bicubic interpolation) and resamples at uniform distances. Lawnmower and spiral paths naturally emerge from the algorithm for uniform and Gaussian distributions respectively, and they are the optimal paths. It is also possible to use the algorithm to follow the contours of steep terrain by aiming the camera out the side of the UAV. The second path planning algorithm aims to maximize the accumulated probability for the path generated given a distribution, a starting point (optionally an ending point), and desired flight time. The camera footprint traverses a grid of probability nodes (enabled by the gimbaled camera) while the UAV approximates the path generated. Near optimal flight paths are generated using an evolutionary approach, where seed paths are generated using various hill-climbing and potential fields algorithms. Simulation results show the algorithm works well with a variety of probability distributions, including complicated multi-modal distributions (Figure 2.2). These advanced algorithms enrich the autonomy tool set for the UAV operator and can potentially be useful for the *high priority search* and *exhaustive search* techniques when systematic coverage is desired.

## **Video mosaicing:**

The term *mosaic* means to “stitch” together multiple frames of video of a static scene from a moving camera [121]. A real-time *temporally local mosaic* technique [81] was developed using Harris corner detector to identify feature points and then using RANSAC [33] to estimate the Euclidean transformation between each pair of frames. User studies using simulations and experience from field trials show that small mosaics of only the last few



Figure 2.3: Example of a video mosaic with an annotation (indicated by a red circle).

seconds of video is sufficient to provide both increased opportunity for detection and increased sense of relative spatial relationships. Figure 2.3 shows an example of the local mosaic view where the same object is only visible in a few frames in original video but is visible for nearly one hundred frames using the technique.

### **Anomaly detection (under development):**

A color anomaly detection algorithm is currently under development that adapts hyperspectral anomaly detection methods to find man-made objects in wilderness scenes. This algorithm adds another autonomy capability to the tool set and can recommend points of interest in the video imagery to the video analyst, potentially reducing mental workload. We mention this component in this paper for completeness.

## **2.6 User Interfaces**

In this section we describe the user interfaces developed to support various searcher roles with a focus on explaining how we integrate autonomy components (including control modes) and human intelligence. Interface techniques provides control **flexibility** with current

information state, and information sharing and **fusion** improves the efficiency of the overall distributed system. They map to the **Information Management** column in our taxonomy (Table 2.1).

The UAV software consists of several components. **Phairwell** is the *augmented virtuality* interface used to “fly” the UAV (see Figure 2.1). The **Wonder Server** consists of central management software for capturing, storing, and retrieving video, UAV telemetry, annotations, and other related information. Finally, the **Wonder Client** is the GUI used by the video analyst and mission manager and provides video mosaic and annotation capabilities. Video and telemetry data are streamed from the Wonder Server.

**Phairwell for UAV Operator:** The UAV operator’s main responsibilities include assigning the UAV a specific task, ensuring that it is properly performing that task while monitoring the UAV’s “health and safety,” monitoring the live video, and interacting with the UAV when needed (e.g., once the video analyst spots a suspicious object, a change in the plan is made, or when the UAV needs attention).

Phairwell supports four flight modes while searching: *manual*, *carrot and camera*, *flight plan*, and *loiter now*. These modes represent autonomous behaviors that help the UAV operator efficiently assist the video analyst. *Manual* mode commands the UAV to match a course and altitude set using the arrow keys. *Carrot and camera* allows the operator to direct the UAV and camera with the mouse. *Flight plan* mode commands the UAV to fly waypoints that the operator selects manually or that are generated automatically by one or more search patterns. The *loiter now* mode interrupts the UAV’s current behavior so the operator or UAV team can briefly ignore the UAV.

While directing the UAV is important, the primary goal is to manipulate the camera efficiently to provide the video analyst with the needed video. The speed of the UAV coupled with slow user response makes it impossible for the operator to manually track specific ground objects, a commonly required task. Instead, the UAV operator can select a terrain location in Phairwell and have the UAV fix the camera on the location. The UAV autonomy can

maintain this lock, even when the UAV is turning rapidly or knocked about by gusts of wind. This allows the UAV operator to easily adjust the camera according to the needs of the video analyst.

Although the UAV's autonomy allows it to fly a predefined flight plan, the UAV operator must often interrupt the autonomy and then resume it later. Phairwell allows the UAV operator to effortlessly change control modes, perform some task, and resume the previous control mode. This capability has been used routinely when the UAV is flying a search pattern and the video analyst sees something and wants the UAV to go back and take a closer look. This specific autonomy is described more fully in the next section.

**Wonder Client for Video Analyst:** The Wonder Client interface serves as the video analyst's primary tool. They have the flexibility to select between either the live video or mosaiced views. The interface also provides tools to modify the brightness, contrast, and other image properties of the live video or mosaic, which often need to be adjusted to make objects more recognizable.

The video analyst also uses the Wonder Client to annotate the video. Annotations mark objects in the video with a timestamp and user notes so that they can be found quickly in the future. An example can be seen in Figure 2.3. When an annotation is placed on the video mosaic, it is tied to the geo-referenced coordinates of the underlying terrain. Therefore, annotations marked on previous video frames are automatically displayed on future frames, ensuring that the video analyst does not repeatedly identify the same object while also providing an efficient means of visually tracking the object.

The video analyst can also indicate any geo-referenced location in the video as a POI that they want to immediately return to and investigate. The system then automatically communicates this information to Phairwell, giving the UAV operator the option of letting the autonomy redirect the UAV to investigate.

**Wonder Client for Mission Manager:** The mission manager is responsible for assessing what has been searched and how well. This information is then used to plan further search

efforts. While assessing ground searcher coverage is a common practice, UAV-assisted search adds a new and challenging aspect to this task. A coverage map showing the quality of the search is generated from the collective see-ability estimates to provide the mission manager with a complete view of the terrain the UAV video covered.

The Wonder Client gives the mission manager access to all the video analyst's POIs and annotations. The mission manager can then review the POIs and classify them as worth investigating or not. Those that are worth investigating are prioritized and placed in a pending search queue. The mission manager then assigns the UAV-team or ground searchers to investigate these points. Once the POI is located, the findings are reported back to the mission manager for assessment. However, investigations executed by the UAV-team will lead to this whole process being repeated.

**For Ground Searchers (under development):** Successful search requires that ground searchers quickly and thoroughly search their assigned area. We have begun development of a system that utilizes the concept of see-ability to support ground searchers in these efforts. A portable GPS device will be used to display a see-ability map, providing a visual representation of the thoroughness and quality of their search based on what they should have seen.

Communication between ground searchers and the UAV-team has proven limited and difficult. This same portable device will be used to bridge this communication gap. For example, when a ground searcher is assigned to investigate a POI, instead of radioing the GPS coordinates, the device will automatically receive the coordinates, overlay the UAV aerial imagery with the annotations, and provide the searcher with directions to the location.

## 2.7 See-ability Metrics

The "see-ability" metric [82] was developed to address the challenge of understanding the search-related quality given by UAV video. This involves two different measures: *instantaneous see-ability* measures the quality of a single video frame while *collective see-*

*ability* measures the overall quality provided by all video frames. They map directly to the **Performance Evaluation** column in our taxonomy (Table 2.1).

The instantaneous see-ability computation uses the semi-accurate camera's location and pose information, terrain models, satellite imagery and computer vision techniques to geo-register each frame of video. The geo-registered frame is used to estimate the resolution with which each point in the video is seen. This metric could provide information about the quality of the video coverage for the video analyst. A user study showed that there was a moderately strong correlation between the instantaneous see-ability estimates and measured detection rates [82]. Collective see-ability is determined by the number of times each point has been seen, from what distance, and from which and how many different angles. This is done by combining all of the instantaneous see-ability estimates available for a single point on the terrain. This metric provides the mission manager with information about the overall quality of the entire search.

## 2.8 Demonstration

We believe a good way to validate our system is to demonstrate its usability in front of real searchers in a real-world environment. In the past several years, many field trials were operated by students pretending to be searchers. A demo to real searchers focuses more on the intended intelligence of the system. That led to a field demo on November 21, 2009 for representatives of the Utah County Search and Rescue team in a remote wilderness area in Elberta, Utah.

Three searchers participated in the demo. One searcher, R, acted as the UAV operator and flew the UAV in a simulated search and rescue mission while the other two searchers observed the mission and inquired about the capabilities of the system, the system structure, and the operation protocols. Professors and students of BYU volunteered as video analysts and ground searchers. R had received 30 minutes UAV operator training and also practiced in a simulated environment for a few hours. The mission objective was to locate a simulated



missing person (a dummy placed in the wilderness) as quickly as possible in a team effort (UAV operator, video analyst, and ground searchers) utilizing the UAV technology. The responsibilities of the mission manager were split between the UAV operator and the video analyst. The missing person was successfully identified in the mosaic view, and the GPS location was radioed to the ground searchers, who successfully located the missing person. The entire mission completed in under 35 minutes.

The anomaly detection autonomy component and the GUI for ground searchers were not fully implemented and, therefore, were not included in the demo. Other than the distribution generation, intelligent path planning, and see-ability metric components (implemented and validated but not fully integrated), all other technologies described in this paper were available and functional.

We conducted an in-depth interview with R several weeks after the demo. Here we share only a portion of his feedback due to space limitations. R thinks the UAV operator interface is “very easy to pick up” and 30 minutes of training was plenty. His reason for practicing in the simulated environment was to explore and avoid silly mistakes. A few new features were available at the demo, but he was able to learn them quickly. He liked the video feed inside the UAV operator GUI because it helped him align the map with the video. One interesting incident was that he was able to identify the simulated missing person before the video analyst, probably the result of his trained eye. He also suggested that including ruler type tools in Phairwell could help him get a better perspective of the map. Feedback from his fellow searchers included comments such as “That was cool!” and “This could work!”

Another key benefit of the demo is that it raises interest from the WiSAR community on technologies that can potentially assist WiSAR operations and opens the door for more direct collaboration between the WiSAR community and academic researchers in the near future.

## 2.9 Conclusions and Future Work

To make UAV technology useful for WiSAR requires the integration of an intelligent UAV system into the existing intelligent WiSAR system. The autonomy components of the UAV technology also need to be integrated to support both individual searcher roles and the distributed system as a whole. We analyze and identify key elements of the integration challenges along two dimensions: attributes of intelligent system and scale. Component technologies are presented and matched to responsibilities of different searcher roles. Then we describe how components of autonomy are integrated into the user interfaces to support the integration of human intelligence for each search role in order to address the integration challenges we identified. Finally we validate the usefulness of the integrated system via a demonstration to Utah County Search and Rescue team representatives. A real searcher acted as the UAV operator and successfully located the simulated missing person using the intelligent UAV system through a team effort. Positive feedback from real searchers about the demonstration give us high hopes that research efforts in designing the UAV intelligent system can really help real WiSAR operations in the near future.

Immediate future work includes implementing and integrating system components identified in this paper but not included in the demo. Research is also planned for providing more flexibility for the existing tool set (e.g., interactive distribution modification and sliding autonomy for intelligent path planning). Long term goals focus on better integration of ground search situation awareness to improve system situation awareness and overall planning.

## Chapter 3

### Paper: A Bayesian approach to modeling lost person behaviors based on terrain features in Wilderness Search and Rescue<sup>1</sup>

#### Abstract

In Wilderness Search and Rescue (WiSAR), the Incident Commander (IC) creates a probability distribution map of the likely location of the missing person. This map is important because it guides the IC in allocating search resources and coordinating efforts, but it often depends almost exclusively on prior experience, subjective judgment, and a missing-person profile. We propose a Bayesian model that uses publicly available terrain features data to help model lost-person behaviors. This approach enables domain experts to encode uncertainty in their prior estimations and also makes it possible to incorporate human behavior data collected in the form of posterior distributions. These distributions are used to build a first-order Markov transition matrix for generating a temporal, posterior predictive probability distribution map. The map can then be augmented as desired by search and rescue workers to incorporate additional information. Using a Bayesian  $\chi^2$  test for goodness-of-fit, we show that the model fits a synthetic dataset well. This model also serves as a foundation for a larger framework that allows for easy expansion to incorporate additional factors, such as season and weather conditions, that affect the lost-person's behaviors.

---

<sup>1</sup>Published in CMOT 2010 (Computational and Mathematical Organization Theory) journal. Authors are Lanny Lin, and Michael A. Goodrich.

### 3.1 Introduction

In the priority search phase<sup>2</sup> of Wilderness Search and Rescue (WiSAR), a probability distribution map for the likely place to find the missing person is created using terrain features, a profile of the missing person, weather conditions and subjective judgment of expert searchers (see [60]). The Incident Commander (IC) uses this map to allocate resources, to direct the search, and to coordinate rescue workers. Search and rescue resources are typically limited, meaning only a small portion of the area can be covered in the first few hours of the search. However, [104] and [120] show that as time progresses, the survivability of the missing person decreases and the effective search radius increases by approximately 3km/hour. Therefore, areas with high probabilities are searched first in hope of finding the missing person quickly. The probability distribution map created by the IC can also be used by manned or unmanned aerial vehicles for path planning purposes, thus facilitating effective aerial search. The quality of the probability distribution map is critical to the WiSAR operations and can mean the difference between life and death for the missing person.

We propose a Bayesian approach in modeling lost-person behaviors to generate such a probability distribution map automatically. The search and rescue workers can then augment this base map to incorporate their own beliefs to generate the final probability distribution map. We argue that using the Bayesian approach to automatically generate the map can be beneficial in the following ways:

- 1) The Bayesian approach easily allows the inclusion of prior data (in the form of subjective judgment of the SAR volunteers), the profile (travel direction and dispersion characteristics) of the missing person, etc.

- 2) This approach allows the search and rescue workers to naturally incorporate their uncertainty by specifying a mean and a variance, which we will then incorporate into a Beta distribution to facilitate a robust Bayesian model.

---

<sup>2</sup>Four qualitatively different types of search strategies are used in WiSAR: hasty search, constraining search, priority search, and exhaustive search. See [41] for more details.

3) This approach allows the incorporation of actual human behavior data collected to generate posterior beliefs.

4) The map generated using the Bayesian model means that the search and rescue workers do not have to build the probability distribution map from scratch and it reduces the chance that the search and rescue workers might overlook a certain area that should have been allocated higher probability.

5) The probability distribution map can be dynamically updated as time progresses. Assuming a first-order Markov process, the Bayesian model can easily incorporate the time element and thereby allow the search and rescue workers to observe how the proposed probability distribution map changes over time, especially as information is collected. Such capability can be useful if the search and rescue operation takes an extended period of time.

Many factors affect how the probability distribution map might turn out. Examples include the season of the year, the weather conditions, the profile of the missing person (age, gender, professions, intention, etc., which translate into direction, distance, and dispersion of travel), and the terrain features of the area. The Bayesian model proposed here mainly focuses on the terrain features, specifically, the topography type, vegetation coverage, and local slope. However, the model is designed so that it can be easily extended to take other factors into consideration.

The proposed Bayesian model has the following components: The search area is first discretized into a honeycomb pattern hexagonal tessellation where each cell represents a state with topography type, vegetation type, and elevation information, which are treated independently. Local slope can then be calculated using elevation differences between the current cell and its neighbors. Expert opinions in human behaviors, experience in past search and rescue incidents, and past statistical data are incorporated to specify the terrain feature transition probability from one topography type (or vegetation or slope, respectively) to another in the form of a mean and a variance. Using samples generated from such priors, a state transition matrix is built to specify the transition probabilities from each state to

all other states, which can be used to generate the prior predictive probability distribution map for any given number of time steps. Data in the form of GPS track logs<sup>3</sup> are then incorporated into the model as observations so posterior beliefs can be calculated. Using the posterior beliefs, a new state transition matrix is built and used to generate the posterior predictive probability distribution map for any given number of time steps.

The rest of paper is organized as follows: In Section 2 we discuss related work. We describe the proposed model in detail in Section 3 and analyze the experiment results in Section 4. In Section 5 we evaluate the model using the Bayesian  $\chi^2$  test for goodness-of-fit by [55]. Section 6 presents conclusions and future work.

## 3.2 Related Work

Many search and rescue researchers have worked on analyzing historical search and rescue cases and have tried to understand and explain missing person behaviors. [104] re-told accounts of various rescue situations by the authors and others to describe wilderness search and rescue techniques and lost-person behaviors. [49] discusses a number of reorientation strategies such as random traveling, direction traveling, route sampling, direction sampling, backtracking, using folk wisdom, and staying put. [120] describes how to use mathematical models to calculate the probability of detection, probability of area and probability of success. He also describes an example search mission. In [119], he also presents a series of case studies. [48] tabulate crow's-flight distance traveled and dispersion of travel by different categories of wilderness users using data from between 1987 and 1996 for 162 lost-person incidents near Peter Lougheed Provincial Park in Alberta, Canada. [60] described the International Search & Rescue Incident Database (ISRID), which contains 50,692 SAR incidents at the time the book was written. Chapter 8 of the book presents important statistical content of lost person

---

<sup>3</sup>A GPS (Global Positioning System) tracking unit can log the position of the device at regular intervals with time stamps. The sequence of these position points make up a GPS track log.

behavior by subject categories. Conclusions drawn from these publications are good resources for specifying priors with our proposed model.

As more geographical information is available to the public via the Internet, researchers have begun looking at systematically utilizing such information for search and rescue applications. [31] discusses the application of GIS (Geographic Information System) to manage the search for a missing autistic youth in the Dolly Sods Wilderness area of West Virginia. GIS provides a platform to integrate data from various sources, allowing the search to be segmented into probability regions based on statistical analysis and a behavioral profile of the missing subject. [111] present a case study about a plane crash near Kutahya, Turkey and demonstrate how probability distribution maps can be generated that shrink the incident area and enable the search team to reach the area in an optimal way. Both papers show great examples of how to use GIS information to build probability distribution maps that can be used to facilitate search and rescue operations. However, they do not allow the experts to specify their uncertainty and also do not incorporate existing human behavior data into the model in a meaningful way.

Once the probability distribution map is generated, computer algorithms can take advantage of it to perform path planning for Unmanned Aerial Vehicles (UAV). [41] present field reports on how UAV technology can be integrated into existing WiSAR teams. In [11] and a series of related papers, Bourgault et al. describe how to use a Bayesian model to create paths for one or multiple coordinated UAVs to maximize the amount of probability accumulated by the UAV sensors. [12] also include scalable collaborative human systems in the loop and generated paths for human operators. [69] present a series of path-planning algorithms for a UAV used in WiSAR operations, which yield high-quality solutions that approximate an optimal path using a given probability distribution map.

Bayesian modeling has been used in many aspects of human behavior modeling. [123] proposes a Bayesian model for human concept learning that gives precise fits to human behavior data. [43] present a Bayesian-based approach to extract a human player's strategic behavior

and movement patterns in interactive computer games. [87] describes a Bayesian computer vision system for modeling and recognizing human interactions in a visual surveillance task. [101] describe a method that uses correspondence between a model of human choice and the choices made by the Markov chain Monte Carlo (MCMC) algorithm.

### 3.3 Terrain-Based Bayesian Model

In this section we describe the proposed Bayesian model in detail. First we present an overview of the model and explain how Bayes' Theorem is used to update beliefs.

#### 3.3.1 Model Overview

The Bayesian model has two distinctive parts. The first part uses previously collected human behavior data (observations of how people actually traveled in various terrains) to update prior beliefs on how the lost person would transition between different terrain features. The update is achieved using the Gibbs Sampling and Metropolis-Hastings flavor of the MCMC class of algorithms shown in [35]. The second part then uses posterior beliefs (updated beliefs from first part) to construct a state transition matrix based on the terrain features of the search area. Using a generative approach and assuming a first-order Markov process, we can predict how the lost person might have traveled from the point last seen as time progresses.

In the first part of the model, we ask domain experts to specify the probability that the lost person would travel from one terrain feature to another. The probability is a continuous distribution and we only ask for transitional probabilities within the same category of terrain features. Therefore, specifying the probability of traveling from topography feature "plain" to topography feature "hill" would make sense while from topography feature "plain" to vegetation density feature "sparse" would not.

We use  $\theta_i$  to denote each probability distribution specified by domain experts, meaning each  $\theta_i$  is a prior belief and a parameter in the model.  $\underline{\theta}$  represents a vector containing all the



parameters of the model. We also use  $\underline{F}$  to denote the set of terrain features for the search area. Following standard Bayesian notations, we use  $\pi(\underline{\theta})$  to denote the joint prior belief of the entire parameter space, and  $f(z|\underline{\theta}, \underline{F})$  to denote the likelihood that the lost person traveled toward various directions given the terrain feature based transitional probabilities and the known terrain features. If we already have multiple observations of what directions a lost person traveled given various terrain features, by applying Bayes' Theorem, we can write the posterior beliefs of our parameters:

$$\pi(\underline{\theta}|\underline{z}) = \frac{f(\underline{z}|\underline{\theta})\pi(\underline{\theta})}{\int_0^1 f(\underline{z}|\underline{\theta})\pi(\underline{\theta})d\underline{\theta}} \quad (3.1)$$

where  $\underline{z}$  is a vector containing multiple observations. Here we can drop  $\underline{F}$  from our equation because  $\underline{F}$  is known.

In the second part of the model, since we already have the posterior beliefs (in continuous form) of how the lost person would transition from one terrain feature to another, we can sample from the posterior beliefs and then construct a state transition matrix based on the terrain features of the search area. Each state is a cell in a hexagonal tessellation, and each row of the state transition matrix consists of the transitional probabilities of traveling from one cell to every cell in the tessellation (including itself). Therefore, the state transition matrix is an  $n \times n$  matrix where  $n$  is the total number of cells in the tessellation, or the total number of possible states.

At the time when the lost person was last seen (indicated by the point last seen), the probability of the lost person being in the cell containing the point last seen is 1. At the next time step, with the help of the state transition matrix, we can compute the probability of the lost person being in each cell of the tessellation. Using the same method, we can predict the probability distribution of where the lost person will be at any time interval  $t$ , where  $t$  is the number of time steps since the time the lost person was last seen. This

probability distribution is called the posterior predictive probability distribution, and this is the probability distribution map we really care about in Wilderness Search and Rescue.

Note that in each time interval, we re-sample from the posterior beliefs (from the first part of the model) because they are continuous probability distributions. It is also worth mentioning that if we build the state transition matrix by sampling from prior beliefs then we are not taking advantage of the observed human behavior data. The corresponding probability distribution of where the lost person will be at time interval  $t$  is called the prior predictive probability distribution.

### 3.3.2 Hypothetical Scenario

To illustrate how the model works, it is helpful to use an exercise scenario as an example. Figure 3.1 shows the satellite imagery of an area by Payson Lake in the Uinta National Forest, Utah, obtained through Google Earth by specifying longitude and latitude between (39°55'56.67" N, 111°38'27.82" W) and (39°55'45.58" N, 111°38'05.68" W). The lake is in the northwest corner, and there is a campground in the southwest region. The three small plots in Figure 3.1 are generated using real terrain feature data downloaded from the USGS web site<sup>4</sup> using the exact longitude and latitude range. The topography dataset was discretized into three types: lake, plain, and hill. The vegetation dataset was also discretized into three types: sparse, medium and dense. Let us imagine a 14-year-old scout is reported missing. He was last seen in the forest on the hill (marked by the white arrow in Figure 3.1) 3 hours and 20 minutes ago, where he took off on his own after a quarrel with his fellow scouts. Now let us assume we have some track log data from past scouts who also became lost in the area but happened to be carrying a GPS unit. The objective is to build a probability distribution map for the area by combining our knowledge of the terrain features with domain experts' estimations of how the child might travel given the terrain features and historical human behavior data.

---

<sup>4</sup><http://seamless.usgs.gov/index.php>

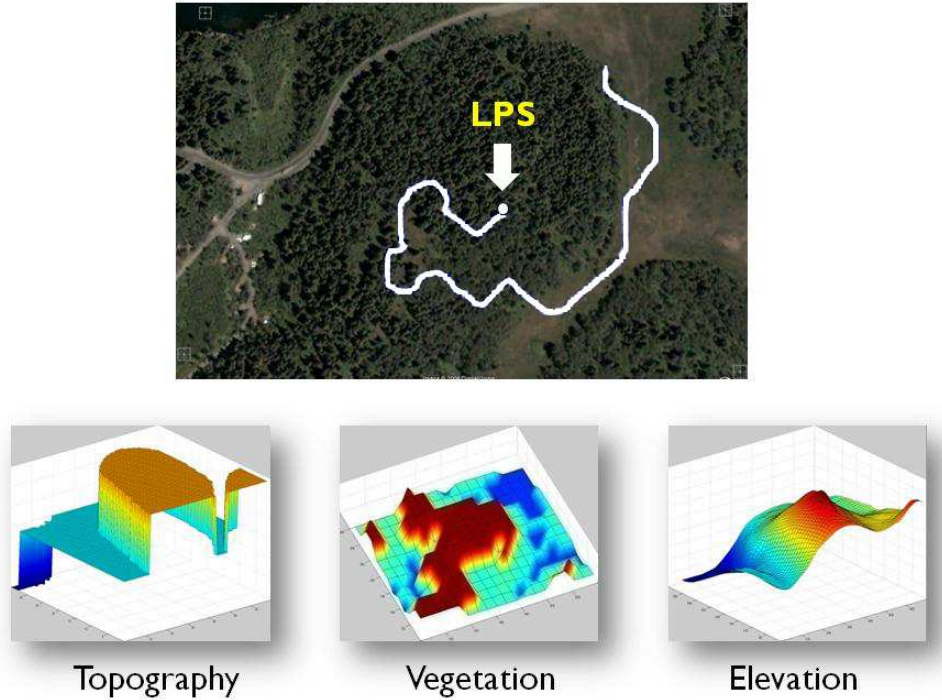


Figure 3.1: Satellite imagery of area by Payson Lake, Utah together with relative topography, vegetation, and elevation data downloaded from USGS web site. The topography and vegetation plots are discretized into (lake, plain, hill), and (sparse, medium, dense) respectively.

### 3.3.3 Hexagonal Tessellation Discretization

The first step in the model is to discretize the area into a hexagonal tessellation as shown in Figure 3.2. The reason we use a hex tessellation is because the hex tessellation ensures the distance from the center of one cell to the center of any neighboring cell is always the same. The width of each hex cell is 24 meters. We picked this number because we believe such a granularity allows us to have enough detailed information about the terrain features without going into excessive details to burden the amount of computation. Future work should systematically explore how changing this granularity affects the tradeoffs between computational complexity and precision. In a real WiSAR scenario, the width can also be determined by the level of detail available for the terrain feature data at hand.

The resulting tessellation is a  $16 \times 38$  tessellation with 608 distinct states. Using the terrain feature data we have, each state is really a 3-tuple of (topography type, vegetation type, elevation). When we transition from one state to another neighboring state (including

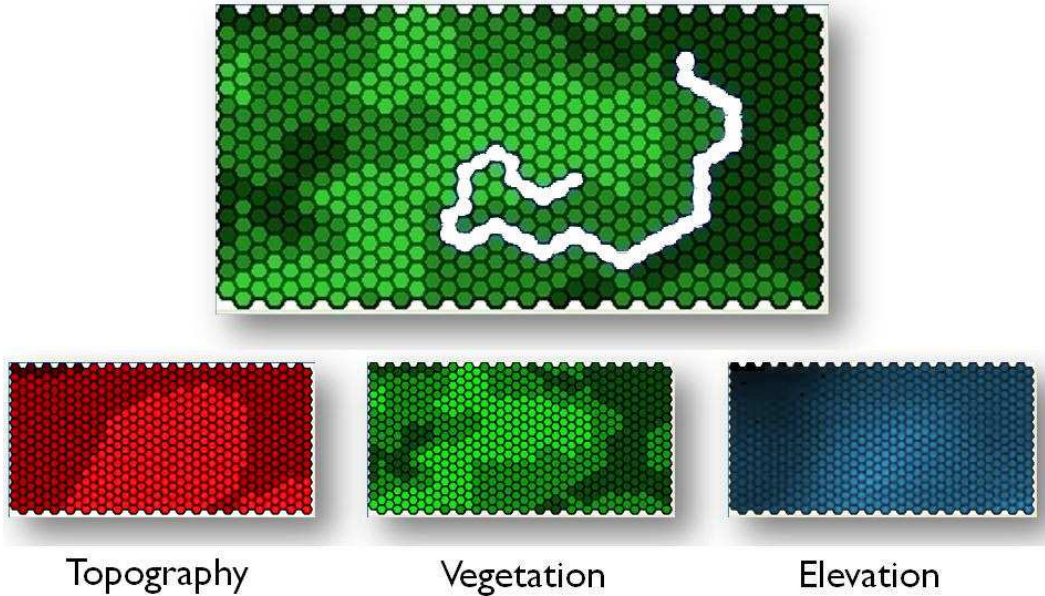


Figure 3.2: Hexagonal discretized tessellation showing past historical data (the path marked by the white cells in the main image) together with topography, vegetation, and elevation information in hexagonal tessellation.

remaining in the same state), we can identify whether the topography type and vegetation type change. By calculating the elevation difference between the two states, we can find out whether the local slope is going uphill, downhill, or neither. Here we decide whether there is a local slope by calculating the angle of the elevation difference. If the difference is more than 20 degrees, we mark it as a local slope. We subjectively picked the threshold of 20 because we want to emphasize the extra effort of going uphill (as might be representative of a typical missing person such as a 14-year-old scout). Future work should systematically evaluate the impact of this threshold on usefulness of the probability distribution function.

### 3.3.4 Model Representation

In this sub-section, we define the Bayesian model in terms of the prior, the state transition, and the likelihood, and discuss each component in detail.

## The Prior

With the knowledge of past WiSAR incidents and expert opinion on human behavior, we can ask domain experts to specify their prior beliefs on how the missing person would behave with respect to different terrain features (e.g., a transition from a medium vegetation type to a dense vegetation type). For example, since we have three different topography types, we need a  $3 \times 3$  transition matrix as shown in Equation (3.2), representing the probability of transitioning from one topography type to another. The rows and columns are both indexed by the different topography types, and it is possible to remain in the same topography, yielding

$$\begin{bmatrix} T'_{00} & T'_{01} & T'_{02} \\ T'_{10} & T'_{11} & T'_{12} \\ T'_{20} & T'_{21} & T'_{22} \end{bmatrix} \quad (3.2)$$

For example,  $T'_{00}$  is the transitional probability for remaining in the lake type topography, and  $T'_{00}$  is a number between 0 and 1 inclusive. The definition of the transition matrix requires that the values in each row of the matrix sum up to 1.

However, we would also like to enable the domain experts to incorporate uncertainty in their beliefs. Therefore, for each topography transition, instead of a number, a continuous Beta distribution is used, and a probability value, such as  $T'_{00}$ , can be generated by sampling from the Beta distribution. We use a Beta distribution because the domain of a Beta distribution's probability density function is  $x \in [0; 1]$ , which matches the parameter space of probability values. The curves of the Beta distribution also have the shape we desire because the mean and the mode of the curve can shift between 0 and 1 with various variances depending on what parameters we pick (illustrated in Figure 3.3). To specify uncertainty, for each topography transition probability, we ask the domain experts to provide a mean and a variance because these parameters are much easier to understand for non-statisticians compared to the  $\alpha$  and  $\beta$  parameters for the Beta distribution. Then we solve for the  $\alpha$  and  $\beta$  parameters automatically.

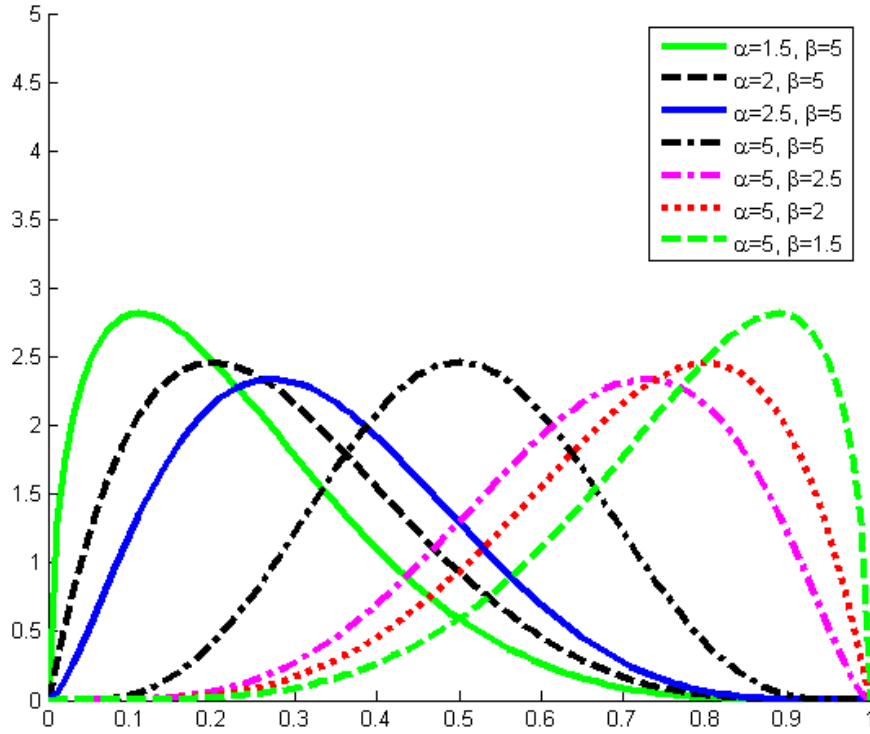


Figure 3.3: Beta Distribution probability density function

We have to be careful here to make sure the transition matrix, shown in Equation (3.2), is still valid. For example, the domain experts can specify means for Beta distributions relating to transitioning from topography type “plain” to all three topography types as 0.5, 0.3, and 0.2. These numbers sum up to 1. Unfortunately when we sample from the three Beta distributions, the values we get could possibly be something like 0.5, 0.35, and 0.22, which do not sum up to 1. That means these numbers are not true probability values, and we have to normalize them so they become true probability values. Therefore, we use  $T_{ij}$  to denote the value we generate from the Beta distribution corresponding to transitioning from topography feature  $i$  to topography feature  $j$ , and use  $T'_{ij}$  to denote the true probability value (after normalization) for the transition. The probability distributions of  $T_{ij}$  are the domain experts’ prior beliefs with respect to topography terrain features, and there are 9 of them.

Similarly, since we have three different vegetation density types, we also need a  $3 \times 3$  transition matrix to represent the probability of transitioning from one vegetation density

type to another. This adds 9 vegetation density related priors to the model. With respect to local slopes, since there are only three possible transitions (uphill, no slope, and downhill), there are only 3 more priors to specify.

Therefore, our model has a total of 21 priors (9 related to topography, 9 related to vegetation density, and 3 related to local slope). For simplicity, we denote the joint distribution of all the priors as  $\pi(\underline{\theta})$ , where

$$\underline{\theta} = T_{00}, T_{01}, \dots, T_{22}, V_{00}, V_{01}, \dots, V_{22}, S_0, S_1, S_2 \quad (3.3)$$

and each prior follows a Beta distribution with known  $\alpha$  and  $\beta$  values (solved using the mean and variance values provided by domain experts). Thus

$$T_{ij} \sim Beta(\alpha_{T_{ij}}, \beta_{T_{ij}}) \quad (3.4)$$

$$V_{ij} \sim Beta(\alpha_{V_{ij}}, \beta_{V_{ij}}) \quad (3.5)$$

$$S_i \sim Beta(\alpha_{S_i}, \beta_{S_i}) \quad (3.6)$$

where  $T_{ij}$  represents the probability of transitioning from topography type  $i$  to  $j$  (possibly  $i = j$ ) where  $i = 0, 1, 2$  and  $j = 0, 1, 2$ . Similarly,  $V_{ij}$  represents the probability of transitioning from vegetation type  $i$  to  $j$ , and  $S_i$  represents the probability of following a certain local slope type  $i$ .

## State Transition

In an earlier part of the paper we described how the search area is discretized into a hexagonal tessellation. Each cell becomes a state. Let  $X$  represent a state, then  $X$  can be defined as a vector containing information about the hexagonal cell:

$$X = [\text{topography, vegetation density, elevation, index of tessellation}]$$

With our model, we assume the state transition follows a first-order Markov process, meaning that the next state the lost-person (LP) will be in is only dependent on the current state the LP is in.

$$P(X_t|X_0, X_1, \dots, X_{t-1}) = P(X_t|X_{t-1}) \quad (3.7)$$

This is a strong assumption and it might not hold. For example, the amount of time traveled following the same direction (e.g., 20 minutes) could affect whether the LP wants to turn around and backtrack; similarly, the intended destination might affect which path the LP chooses while looking for the way. However, we argue that because the LP is in a disoriented state (although the LP might think otherwise) in the wilderness, the direction the LP follows could very well not be the direction the LP thinks he/she is following. Therefore, this assumption should not prevent the model from having useful predictive power. However, we also plan to extend the model in future work that will take into consideration the intended destination and incorporate that information into the representation of the current state.

When we compute  $P(X_t|X_{t-1})$ , it is necessary to combine the topography transition probability with vegetation density and local slope so we can borrow strength from each of the terrain features. We denote  $T(Y|X)$  as the probability of transitioning from the topography of state  $X$  to the topography of state  $Y$ , and  $V(Y|X)$  as the probability of transitioning from the vegetation density type of state  $X$  to the vegetation density type of state  $Y$ . Using the elevation difference between state  $X$  and state  $Y$ , we can identify the local slope of going from state  $X$  to state  $Y$ . We denote  $S(Y|X)$  as the probability of transitioning from state  $X$  to state  $Y$  only based on local slope information.  $T(X|Y)$ ,  $V(X|Y)$ , and  $S(X|Y)$  are all true probability values, and they correspond to the relevant entries in the terrain features transition matrices such as Equation (3.2). Assuming the three terrain features are independent of each other we can combine the three probabilities by taking the product of



the three,

$$P(X_t|X_{t-1}) \propto T(X_t|X_{t-1})V(X_t|X_{t-1})S(X_t|X_{t-1}), \quad (3.8)$$

where  $P(X_t|X_{t-1})$  is the entry in the row indexed by  $X_t$  and the column indexed by  $X_{t-1}$  in the state transition matrix describing the probability of transitioning from any state to any other state (including transitioning into the same state). Here  $P(X_t|X_{t-1})$  is a true probability value.

Because a person can only travel from one hexagonal cell to its neighboring cells (or remain in the original cell), in each row of the state transition matrix, the transitional probabilities for all  $X_t \notin N(X_{t-1})$  will be 0, where  $N(X_{t-1})$  is the set of neighboring states of state  $X_{t-1}$  (including  $X_{t-1}$ ). That means the sum of  $P(X_t|X_{t-1})$  for all  $X_t \in N(X_{t-1})$  is 1 (elements in each row of the state transition matrix should sum to 1).

If we look at each hex cell closely, we can see that from each cell a person can travel to one of the six neighboring cells or remain in the same cell.

$$\text{Let } \underline{\phi}' = P(X_t|X_{t-1}) \text{ where } X_t \in N(X_{t-1}) \quad (3.9)$$

$$\text{Let } \underline{\phi} = T(X_t|X_{t-1})V(X_t|X_{t-1})S(X_t|X_{t-1}) \text{ where } X_t \in N(X_{t-1}) \quad (3.10)$$

We know  $\underline{\phi}$  and  $\underline{\phi}'$  each have 7 elements, and  $\sum_{i=1}^7 \phi'_i = 1$ , where

$$\phi'_i = \frac{\phi_i}{\sum_{j=1}^7 \phi_j} \quad (3.11)$$

Equation (3.11) normalizes the products of terrain feature transition probabilities to compute the  $P(X_t|X_{t-1})$  entries for all neighbors of state  $X_{t-1}$ .

## The Likelihood

Because from each cell a person can travel to one of the six neighboring cells or remain in the same cell, the likelihood of one observation (how the lost person traveled from one cell to one of the neighboring cells), denoted as  $f(z|\underline{\theta})$ , follows a categorical distribution with 7 dimensions. Relating to the previous section,  $z$  can be defined as

$$z = (X_t, X_{t-1}), \text{ where } X_t \in N(X_{t-1}). \quad (3.12)$$

In other words,  $z$  is really a vector in the form of a 7-tuple, but to avoid notation confusion, we will only use  $\underline{z}$  to denote multiple observations in a later section when we discuss posteriors. If we use  $z_i$  to represent each element in vector  $z$ , then  $z_i$  is constrained by

$$z_i \in \{0, 1\} \text{ and } \sum_{i=1}^7 z_i = 1, \quad (3.13)$$

meaning exactly one element in the 7-tuple is 1 and the others are all 0s. For example, an observation can be of the form of (0,0,0,0,1,0,0), meaning the person traveled to the fifth neighboring cell. Thus, our observation given all the prior beliefs is governed by

$$z|\underline{\theta} \sim CAT(\underline{\phi}'), \text{ where} \quad (3.14)$$

$$\underline{\phi}' = \phi'_1, \phi'_2, \dots, \phi'_7, \text{ and} \quad (3.15)$$

$$f(z|\underline{\theta}) = \prod_{i=1}^7 \phi_i'^{z_i}, \text{ where} \quad (3.16)$$

$$\underline{\theta} = T_{00}, T_{01}, \dots, T_{22}, V_{00}, V_{01}, \dots, V_{22}, S_0, S_1, S_2. \quad (3.17)$$

Note that in order to compute the likelihood for  $z$  using Equation (3.16), we need to identify  $X_{t-1}$ , the state the person is in, and all neighboring states  $X_t \in N(X_{t-1})$ . We also need to sample from our priors  $\pi(\underline{\theta})$  in order to construct terrain features transition matrices, which are then used in Equations (3.9) – (3.11) to compute  $\underline{\phi}'$ .

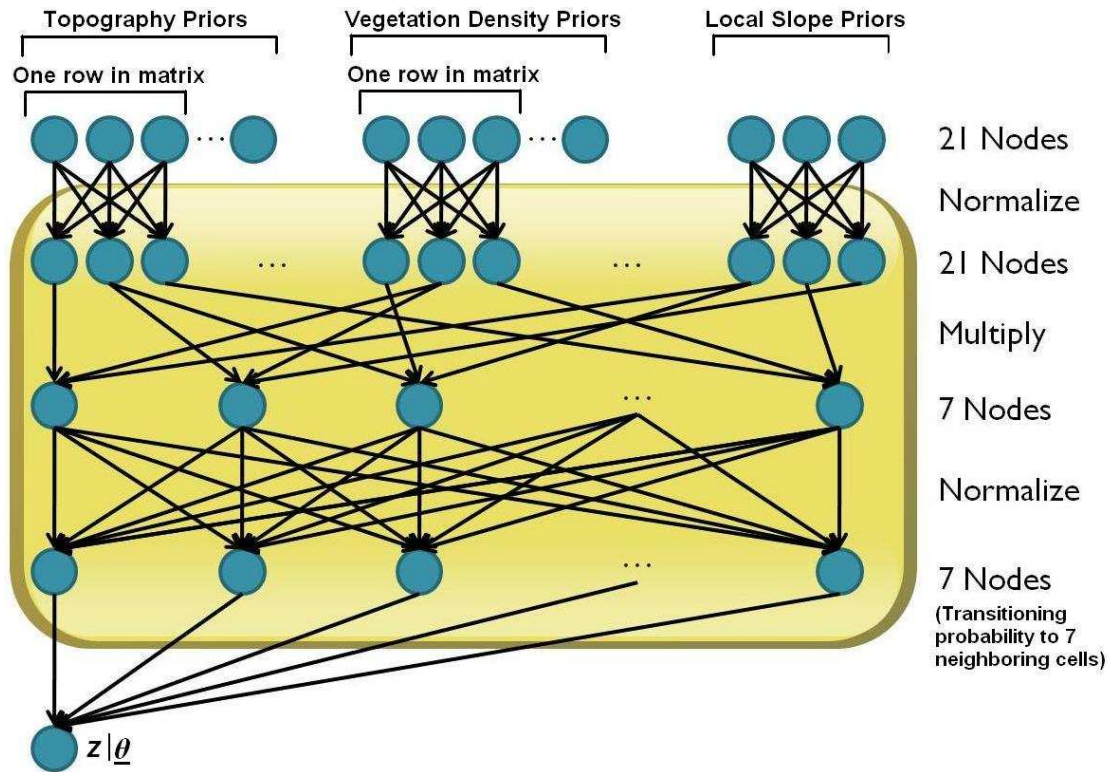


Figure 3.4: A graphical illustration of the proposed model. Top row: probability distribution for each prior belief. Fourth row: probability of transitioning into a neighboring cell in the hex tessellation. Bottom row: an observation indicating possible travel directions for the lost person.

Figure 3.4 illustrates the process of computing the likelihood graphically. The top row shows the 21 priors we sample from to generate  $\underline{\theta}$  (9 for topography, 9 for vegetation density, and 3 for local slope). After normalization, we obtain all the entries for the terrain features transition matrices as shown in the second row (9 priors from the  $3 \times 3$  topography transition matrix:  $T'_{00}, T'_{01}, \dots, T'_{22}$ , 9 from the  $3 \times 3$  vegetation density transition matrix:  $V'_{00}, V'_{01}, \dots, V'_{22}$ , and 3 local slope probabilities:  $S_0, S_1, S_2$ ). Depending on the  $X_{t-1}$  and  $X_t$  states associated with  $z$ , relevant  $T(X_t|X_{t-1})$ ,  $V(X_t|X_{t-1})$ , and  $S(X_t|X_{t-1})$  probability values are identified and multiplied to compute  $\underline{\phi}$  (third row). The elements of the  $\underline{\phi}$  vector are further normalized to produce  $\underline{\phi}'$  (fourth row), which are the probabilities of the lost person traveling from state  $X_{t-1}$  to all the neighboring states.

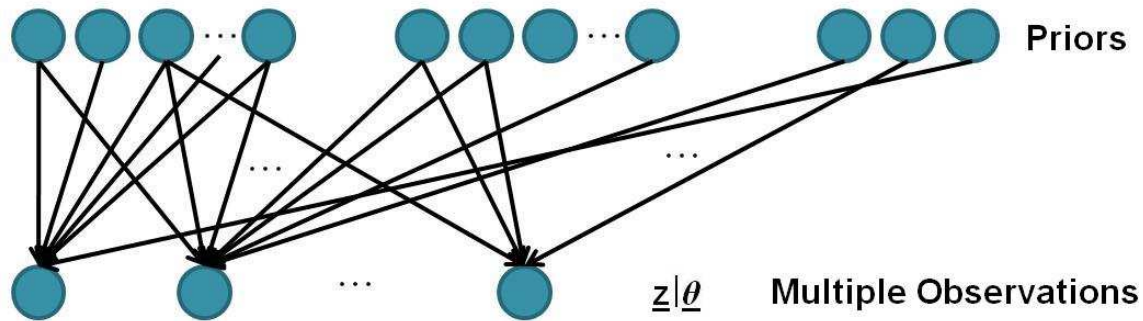


Figure 3.5: Bayesian network with multiple observations. Top row: probability distribution for each prior belief. Bottom row: multiple observations from previously collected human behavior data in the form of segments of GPS track logs together with terrain features associated with the track logs.

Once samples are generated from the priors  $\pi(\underline{\theta})$ , we can deterministically compute the values for the middle layers — they are simply delta functions. Therefore, when we build the Bayesian network to compute the posteriors, we collapse all the middle layers and only keep the top and bottom layers.

### 3.3.5 Using the Model to Compute the Posterior

A great benefit of using a Bayesian model is that we can incorporate existing observations to update prior beliefs. The updated beliefs are the posterior beliefs.

Existing observations in the model are in the form of sections of GPS track logs (also discretized to a hexagonal tessellation) together with the terrain features associated with the track logs. By combining existing human behavior data with prior beliefs, we can reduce the domain experts' uncertainty.

To incorporate multiple observations, we simply add multiple  $z|\underline{\theta}$  nodes in the bottom row of the Bayesian network (illustrated in Figure 3.5). The network is dynamically built with appropriate parent nodes identified and linked to the observation nodes dynamically.

Because of the complexity of the model, it is impossible to solve for the posterior distribution  $\pi(\underline{\theta}|\underline{z})$  in closed form. That is why we used an MCMC approximation algorithm as the generation tool. Specifically, we used a random walk flavor of MCMC that uses the Gibbs

Sampling algorithm, shown in [35], on the outside loop, with Metropolis-Hastings algorithm, shown in [35], inside each iteration of the Gibbs Sampling. Gibbs sampling is an algorithm for generating samples from a joint probability distribution of multiple random variables when the conditional distribution of each variable is known. It generates samples from the distribution of each variable in turn, conditioned on the current values of other variables. The Metropolis-Hastings algorithm generates a first-order Markov chain in each state and uses a proposal density, which depends on the current state, to generate a new proposed sample. This value is accepted if a value drawn from a uniform distribution between 0 and 1 meets certain requirements. Otherwise, the current value is retained. In our implementation, we used a Gaussian function as the proposal density.

In our implementation, we used 500 iterations for burn (throwaways) and kept 10,000 samples for each parent node. In each iteration, the Gibbs Sampling algorithm tries to sample from the distribution of each parent node in turn, conditioned on the current values of other parent nodes. However, Gibbs Sampling relies on the Metropolis-Hastings algorithm to really generate samples from the posterior distribution by using a proposal density function (a Gaussian distribution in our case). These samples approximate the posterior distribution for each of our 21 priors.

Figure 3.6 illustrates how the Monte Carlo method approximates the posterior distribution of one parameter (a parent node). In each iteration, the Metropolis-Hastings algorithm probabilistically generates a sample for the node based on the complete conditional constructed by Gibbs sampling (points inside smaller graphs in the upper portion where each point represents a probability value generated from a Beta distribution). If we combine all these samples into one graph and bin the points into small clusters (bigger graph in lower portion where the y axis is the count), we can connect the top of the bins and draw a curve. This curve is an approximation of the posterior distribution of the node, and as the number of samples approaches infinity, the curve matches the actual posterior distribution.

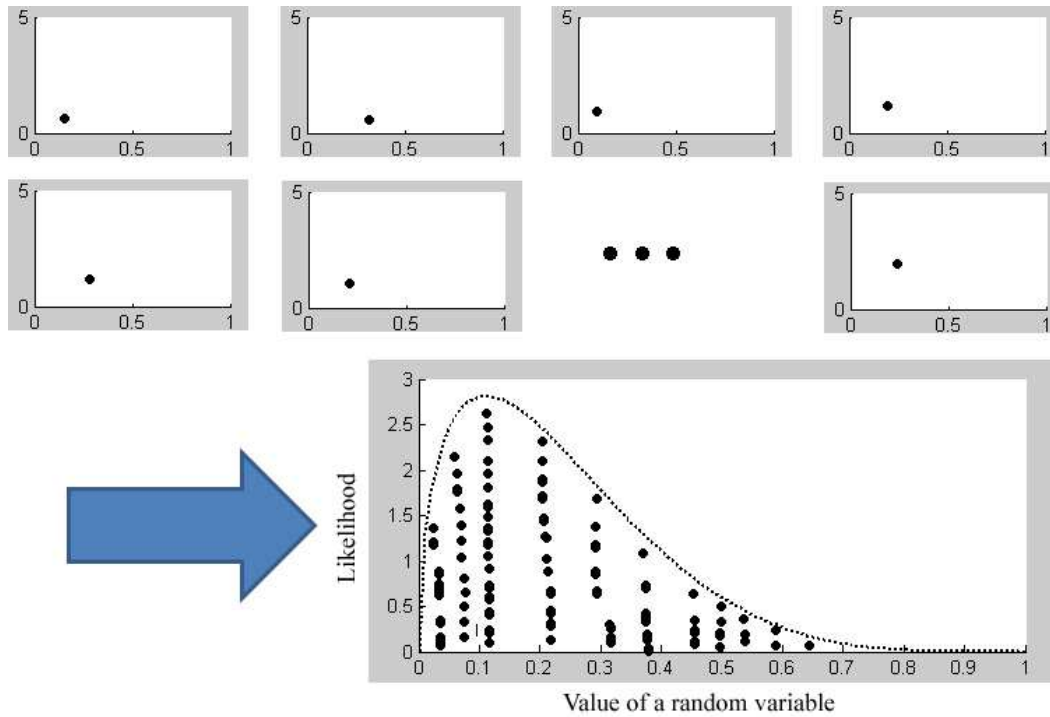


Figure 3.6: Graphical illustration of how the Monte Carlo method approximates the posterior distribution of one parameter. Upper: multiple iterations of sampling. Lower: samples clustered to approximate the real distribution.

In our experiments, the MCMC algorithm completed in 220 seconds on a Dual-core AMD 3800+ PC with 3GB of memory.

### 3.3.6 Using the Model to Compute the Predictive Probability Distribution

Using the model described above, once we have the priors specified, we can build our  $608 \times 608$  state transition matrix. In our implementation, we sample once from each Beta distribution for each time step. Starting from the lost person's point last seen, we can generate the prior predictive probability distribution by multiplying the state transition matrix in each time interval. This method allows the search and rescuers to see how the predictive probability distribution changes as time progresses.

[104] and [120] show that in WiSAR scenarios, as time progresses, the effective search radius increases by approximately 3km/hour, which is equivalent to 50m/minute. Because the age of the lost-person affects the speed the person travels, we can adjust the size of the

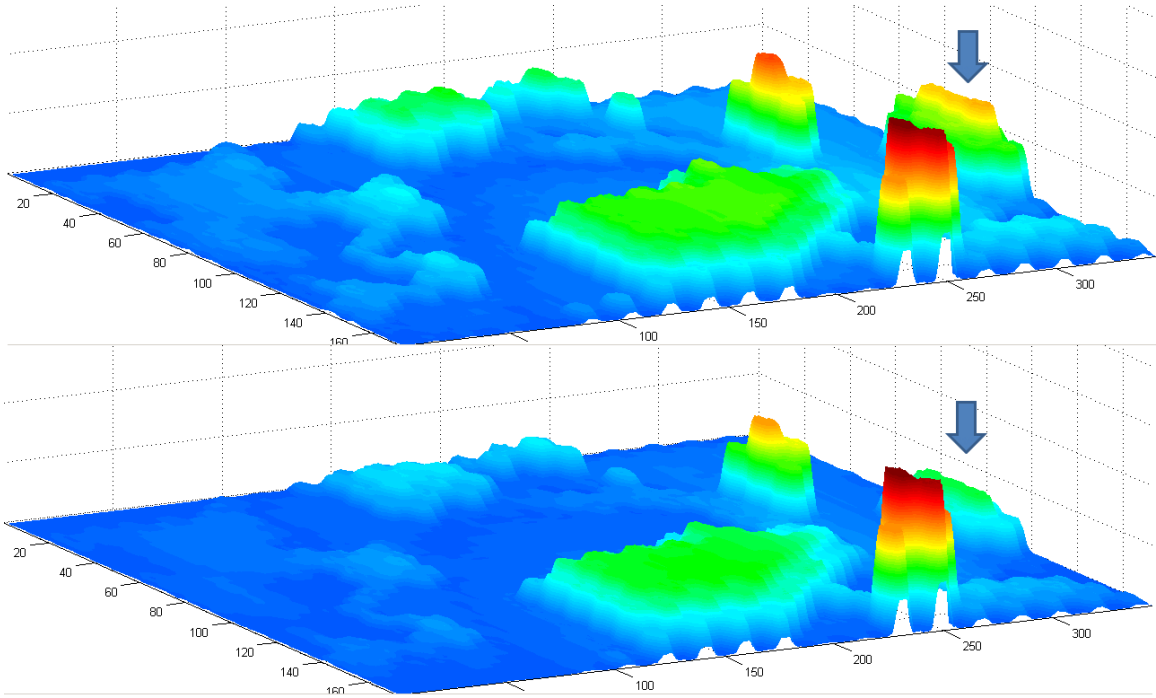


Figure 3.7: Comparing prior predictive distribution (upper row) against posterior predictive distribution (lower row)

time interval accordingly. With our lost scout scenario, because children generally travel slower than adults, we assume the lost scout travels at roughly 24m/minute; therefore, we define each time step as 1 minute. When we multiply the state transition matrix (sampled once from the prior distributions at each time step) 200 times (3 hours and 20 minutes = 200 minutes), we have the prior predictive probability distribution map as shown in the upper row of Figure 3.7.

Once we combine previously collected human behavior data and approximate the joint posterior distribution of all the parameters, we can sample from the posterior beliefs instead of the prior beliefs. Following the same state transition matrix multiplication, we can also generate the posterior predictive probability distribution. The lower row of Figure 3.7 shows this distribution. In the Wilderness Search and Rescue case, the posterior predictive probability distribution is the 2D probability distribution map we are seeking.

## 3.4 Evaluation of the Model

### 3.4.1 Synthetic Data

Pretending to be domain experts, we specified all the prior distributions by setting the means and the variances. The matrices below show the prior distributions we set for the vegetation type transition matrix. The first matrix shows the means and the second matrix shows the variances.

$$\begin{bmatrix} \mu_{V_{00}} = 0.6 & \mu_{V_{01}} = 0.25 & \mu_{V_{02}} = 0.15 \\ \mu_{V_{10}} = 0.5 & \mu_{V_{11}} = 0.3 & \mu_{V_{12}} = 0.2 \\ \mu_{V_{20}} = 0.4 & \mu_{V_{21}} = 0.4 & \mu_{V_{22}} = 0.2 \end{bmatrix} \quad (3.18)$$

$$\begin{bmatrix} \sigma_{V_{00}}^2 = 0.14^2 & \sigma_{V_{01}}^2 = 0.15^2 & \sigma_{V_{02}}^2 = 0.1^2 \\ \sigma_{V_{10}}^2 = 0.15^2 & \sigma_{V_{11}}^2 = 0.15^2 & \sigma_{V_{12}}^2 = 0.15^2 \\ \sigma_{V_{20}}^2 = 0.15^2 & \sigma_{V_{21}}^2 = 0.15^2 & \sigma_{V_{22}}^2 = 0.15^2 \end{bmatrix} \quad (3.19)$$

We set these values following common sense. For example, we believe a lost scout is more likely to remain in sparse vegetation type ( $\mu_{V_{00}} = 0.6$ ) and unlikely to transition from a sparse vegetation type to a dense vegetation type ( $\mu_{V_{02}} = 0.15$ ). We also believe a lost scout is more likely to transition from a dense vegetation type to a medium or sparse vegetation type and from a medium vegetation type to a sparse vegetation type ( $\mu_{V_{21}} = 0.4$ ,  $\mu_{V_{20}} = 0.4$ ,  $\mu_{V_{10}} = 0.5$ ). However, for most of these Beta distributions, we are not certain about our estimation, which is why we specified large variances for most of the parameters. For example,  $\sigma_{V_{10}}^2 = 0.15^2$  means we believe the probability to transition from vegetation type medium to sparse could be as low as 0.05 and as high as 0.95. In real WiSAR scenarios, the priors should come from past statistical analysis of lost-person behaviors, such as [48] and [119], combined with domain experts' opinions.

Each observed data point is a transition from one cell to another neighboring cell (including remaining in the same cell) in previously collected GPS track logs. The track logs



do not even have to be in the same search area of the current incident. What we really care about is how terrain features affect a person’s behavior in the wilderness. If the track logs contain this kind of information, we can use it to update our prior beliefs. These posterior beliefs can then be used in a generative approach to predict how the lost person might travel from the point last seen as time progresses. For the lost scout scenario, our observed data is partly shown in Figure 3.2 as the path of white cells. We use the word “partly” because during the travel, the person sometimes stayed in the same state during the 1-minute time interval. To test the robustness of the model, we intentionally designed the data set so that the person remained in the same vegetation type most of the time. We also repeated the same path three times in our synthetic dataset to simulate three different past GPS track logs. By repeating these we are basically adding more strength to the data and we expect the data to have a much stronger effect on the posterior distributions for the parameters. Each path consists of 45 transitions; therefore, our dataset has 135 data points.

### 3.4.2 Prior vs. Marginal Posterior

Using the posterior samples, we can compare the marginal prior distribution with the marginal posterior distribution for each of our parameters. Figure 3.8 shows the comparison for some of the parameters. The dotted lines represent the prior distributions and the solid lines represent the posterior distributions.

The plot in the upper left is for parameter  $T_{02}$ , the terrain feature transition probability from lake to hill. Since we do not have any data point in our dataset that transitioned from lake to hill, here we see the posterior is almost identical to the prior. The plot in the upper right is for parameter  $T_{12}$ , the terrain feature transition probability from plain to hill. In our dataset, a good segment of the path basically followed the contour line but stayed in the plain states. This characteristic of the dataset explains why the posterior distribution is much narrower and had a much lower mean—because the data did not show many changes from plain to hills, the posterior probability of this transition is much lower with less variance.

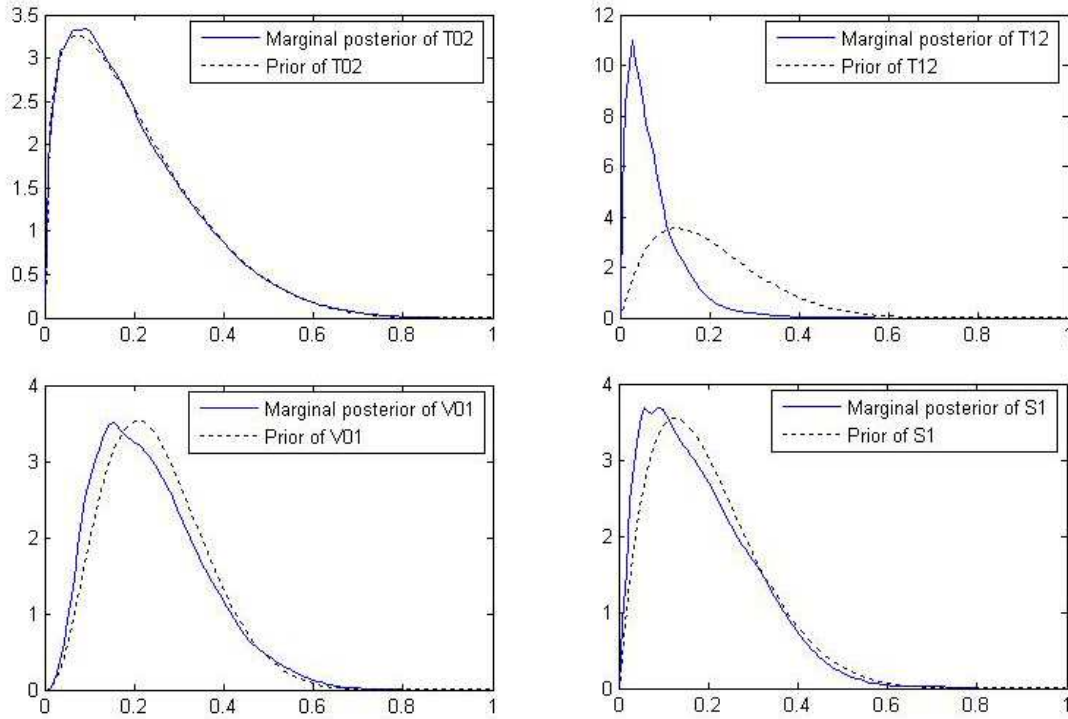


Figure 3.8: Comparing prior distribution with marginal posterior distribution. Upper Left:  $T_{02}$  Upper Right:  $T_{12}$  Lower Left:  $V_{01}$  Lower Right:  $S_1$ .

The plot in the lower left is for parameter  $V_{01}$ , the terrain feature transition probability from vegetation type sparse to medium. The plot in the lower right is for parameter  $S_1$ , the terrain feature transition probability from no slope to no slope. Both of these posteriors are only slightly different from the priors.

### 3.4.3 Correlation of Parameters

When we ask the domain experts to specify the priors, we assume the parameters are independent. Because we have 21 parameters, the joint posterior distribution in the parameter space is really a distribution with 21 dimensions, which is impossible to plot. Instead, we use a correlation image to show whether there exist correlations between pairs of parameters.

Figure 3.9 shows a graphical representation of the correlation between each pair of parameters. A grey value, such as cell(1,21) in the lower left corner, indicates that there is

no correlation between the two parameters. A white cell, such as cell(1,1) in the upper left corner, means the two parameters are fully positively correlated, and a black cell means the two parameters are fully negatively correlated. Here we see that the vegetation parameters  $V_{20}$  (dense to sparse),  $V_{21}$  (dense to medium), and  $V_{22}$  (dense to dense) showed positive correlation. Other positive correlations also mostly appear between neighboring parameters. There is also a clear positive correlation between  $V_{22}$  (Vegetation: dense to dense) and  $S_0$  (uphill). These positive correlations are marked by the big circle in Figure 3.9.

The results of the correlation analysis indicate that there is likely a correlation among different terrain features. Interestingly, from this figure we can see that parameter  $V_{12}$  (Vegetation: medium to dense) and  $T_{11}$  (Topography: plain to plain) are clearly, negatively correlated (marked by the upper small circle in Figure 3.9). Parameter  $V_{22}$  (Vegetation: dense to dense) and  $T_{11}$  (Topography: plain to plain) are also clearly, negatively correlated (marked by the lower small circle in Figure 3.9). A closer look at the terrain features of the area shows that dense vegetation is mostly located on the hill topography type and medium vegetation is mostly located on the plain topography type. This explains why we see such a negative correlation. This emergence of correlations that are compatible with terrain features suggests that the process of combining prior information with observed track logs is useful. However, when we let the domain experts specify the prior distributions, it is much more intuitive for them to assume independence instead of specifying conditional probabilities (to specify how the parameters are correlated), and we rely on data to identify the dependence relationship.

#### 3.4.4 Prior Predictive vs. Posterior Predictive

In this section we compare the prior predictive probability distribution and the posterior predictive probability distribution. The prior predictive is generated by sampling from the prior beliefs specified in the first part of the model. The posterior predictive is generated by sampling (using MCMC) from the posterior beliefs generated from the first part of the model. If no previous human behavior data is available, then the prior predictive

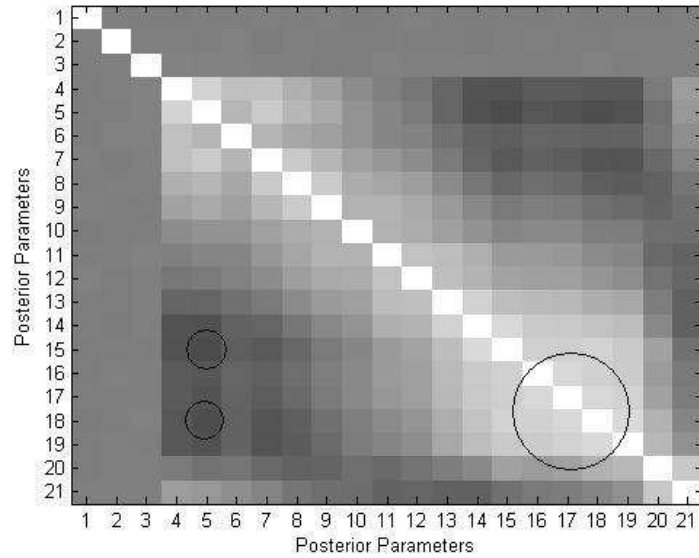


Figure 3.9: Parameter correlation: a grey value of 128, such as (1,21) in the lower left corner, represents no correlation. A white cell, such as (1,1) in the upper left corner, represents a correlation of 1. A black cell represents a correlation of -1. Parameters are in the following order:  $T_{00}, T_{01}, \dots, T_{22}, V_{00}, V_{01}, \dots, V_{22}, S_0, S_1, S_2$ . The big circle marks a positive correlation between parameters, and the small circles mark a negative correlation between parameters.

can still be used to show likely places to find the missing person; otherwise, the posterior predictive should be used because combining existing human behavior data enables the model to reduce uncertainty in the posterior beliefs.

Both probability distributions use a generative approach to predict how the lost person might travel from the point last seen as time progresses. The 2D probability distribution map generated is the final product of the model and can be used by Incident Commanders in WiSAR operations.

The lower row of Figure 3.7 shows the posterior predictive distribution created using the samples generated for all the parameters through MCMC. After 200 time steps (equivalent to 3 hours and 20 minutes), we can see that near the right side of the map, clearly much less probability mass is allocated compared with the prior predictive distribution (indicated by arrows). The center of the northern region also has lower probability compared with the prior predictive distribution, but the difference is not dramatic. Therefore, with our lost scout

scenario, this posterior predictive probability distribution map suggests that we should send search and rescue workers to the regions marked by the two highest peaks first to maximize the likelihood of finding the missing scout.

### 3.4.5 Bayesian $\chi^2$ Test for Goodness-of-Fit

We used the Bayesian  $\chi^2$  test for goodness-of-fit proposed by [55] to evaluate the quality of posterior beliefs in the proposed model. This test is closely related to the classical  $\chi^2$  goodness-of-fit statistic, but different in many aspects. The classical  $\chi^2$  goodness-of-fit test computes a single  $p$ -value. The Bayesian version, however, computes the goodness-of-fit at each iteration of the MCMC, conditional on the current set of model parameter values sampled from the posterior distribution of all the model parameters. The posterior distribution of the resulting  $p$ -values converges to a  $\chi^2$  distribution with  $k - 1$  degrees of freedom as the number of iterations approaches infinity. [55] defined the Bayesian  $\chi^2$  test for goodness-of-fit using the following equation:

$$R^B(\vec{\theta}) = \sum_{k=1}^K \left[ \frac{(m_k(\vec{\theta}) - np_k)}{\sqrt{np_k}} \right]^2 \quad (3.20)$$

where  $\vec{\theta}$  is a set of model parameters sampled from the posterior distribution in a single iteration,  $m_k(\vec{\theta})$  represents the number of observations that fell into the  $k$ th bin,  $n$  is the total number of observations, and  $p_k$  is the probability assigned by the null model to this interval. Values of  $p_k$  are held fixed while the bin counts  $m_k(\vec{\theta})$  are considered as random quantities.

Because our likelihood function is a categorical distribution with 7 parameters (a discrete distribution) we used 7 bins, so  $k = 7$ . It is worth mentioning that  $p_k$  is different for each data point in our case. For each set of model parameters, we calculate the probability values (for each neighbor of the cell, into which the data point fell) for the 7 bins for each observed data point, and then sum up all the probability values for each bin across all data

points. By dividing the sum for all bins, we normalize the probability value, and the result is the probability for that bin.

With  $k-1 = 6$  degrees of freedom, we computed the  $\chi^2$  distribution and then computed the quantiles (the  $p$ -values) for each of the 10,000  $R^B(\vec{\theta})$  values. The results show that only 6 out of 10,000  $p$ -values are smaller than 0.05, the statistical significance value we selected. The Bayesian  $\chi^2$  test of goodness-of-fit suggests that our model fits the synthetic dataset well.

### 3.5 Discussions and Limitations

First we summarize some of the assumptions made throughout the development of the model and our rationale behind them. We assume that the state transition follows a first-order Markov process. Our argument is that the lost person is likely in a disoriented state, therefore, the assumption should not be a big problem (see section 3.3.4 for more details). Another assumption is that the three terrain features are independent. Although correlation analysis shows possible dependence between terrain features, we believe it is more intuitive for domain experts to assume independence instead of specifying conditional probabilities, and we rely on data to identify the dependence relationship (see section 3.4.3 for more details). We also assume that the Markov process is stationary with homogeneous time steps (see section 3.3.6 and section 3.4.4). However, we argue that the flexibility of specifying finer time intervals and the possibility to stay in the same state can “simulate” a non-stationary process with various time durations, thus alleviating the restriction.

After analyzing 162 lost-person incidents near Peter Lougheed Provincial Park in Alberta, Canada, [48] come to the conclusion that there is a close correlation between a lost-person’s intended destination and the angle of dispersion (calculated from the lost-person’s point last seen and the point the person was eventually found). This finding suggests that it might be a good idea to incorporate the missing person’s intended destination into our existing model.

One limitation of this paper is that we are using synthetic data for our evaluation. To address this, we recently collected all the GPS track logs within the US that were uploaded to the popular web GPS track log repository, everytrail.com. We were able to identify 329 GPS track logs that contained the word “geocache” (or “geocaching”). Because most of the geocache “treasures” are hidden in the wilderness off of a designated trail, we believe that GPS track logs created by geocachers are likely to contain behavior data indicating how a human might react to different terrain features. After closer examination of these track logs, we see a clear trend that the locations of the geocache “treasures” play an important role in the person’s behavior in the wilderness in addition to terrain features. If we want to use this kind of GPS track log data as existing human behavior data, our model has to take into consideration the intended destination.

Another trend we observed from these GPS track logs is that a majority of the geocachers first followed some trails to get closer to the “hidden treasure”. When the trail starts to clearly lead the person further away from the geocache, or when the person decides to take a shortcut somewhere along the trail, the geocacher then abandons the trail and creates a new path.

During the summer of 2009, a student in our research lab went for a geocache hunt near Box Elder Peak in Utah. After successfully finding the “hidden treasure”, he decided to not return the same way he came from, but to try some alternative route. Soon he found himself lost and struggled for several hours to reorient himself. Eventually he stumbled onto a hiking trail, a different one from what he took before completing the geocache mission, then he followed the trail and found his way back. Figure 3.10 shows the GPS track log displayed in Google Earth for the period when he was off-trail and lost in the wilderness. The point we want to make here is that after running into another unknown hiking trail, he immediately decided to stay on the trail. This kind of behavior can only be predicted by models that handle trail-following, and our present model clearly lacks this capability.

We plan to extend our model to support intended destination and trail-following. Doing so would allow us to take advantage of abundant GPS track logs and incorporate such human behavior data into the posterior distribution. Once we have the newer model, we can also take advantage of the existing Bayes factor analysis methods, such as Akaike Information Criterion (AIC) proposed by [2], Bayesian Information Criterion (BIC) presented in [103], and Deviance Information Criterion (DIC) described in [112], to perform extended validation.

In the present model, for every time step, only one sample is generated from each Beta distribution. A possible improvement is to use the idea of particles. At each time step, the model would generate, for instance, 100 examples from each Beta distribution, and then compute 100 sets of categorical distributions (each has 7 discrete values representing probabilities of transitioning to 7 directions), which can then be averaged to produce a final categorical distribution with better quality and a better representation of the experts' uncertainty. Because the added computation is outside of the MCMC algorithm and the matrix multiplications, the added execution time should be minimal.

Another important question we should ask is: How well would an experienced IC trust the probability distribution map generated using the proposed model? We strongly believe that the predictive probability distribution generated using the proposed model should only be used as a base onto which domain expertise can be further projected. The objective of the model is to provide a tool that reduces the IC's workload and supports the IC's operation, but not to replace the IC's responsibilities. With abundant experience, training, and the ability to incorporate much richer information (e.g., lost person profile, weather), the IC is responsible for validating the probability distribution suggested by the model and also for coming up with the final distribution map. To really make the tool useful for ICs in WiSAR operations, two additional elements are necessary: 1) a human factors analysis (user study) of the users' trust of the algorithms and automation in the WiSAR domain, and 2) an interface component/tool that enables the IC to easily modify the probability distribution generated by the model. The ability for the IC to modify the probability distribution both





Figure 3.10: Satellite imagery of Elder Box Peak in Utah. The GPS track log shows the path taken by a hiker in summer 2009 when he went off the hiking trail and became lost for several hours before stumbling into another hiking trail.

at the beginning and during the search can potentially improve how (much) users trust the system and make the proposed model more acceptable to users. Future work should develop an interface tool that enables a user to modify a probability distribution map. We believe results from such research can help improve the usability and usefulness of the proposed model.

### 3.6 Conclusions and Future Work

In WiSAR operations, the Incident Commander typically has limited resources and relies on a probability distribution map to allocate resources, to direct the search, and to coordinate rescue workers. Because as time progresses, the survivability of the missing person decreases and the effective search radius increases by approximately 3km/hour, it is critical to find the missing person quickly. That is why areas with high probabilities are searched first, and the quality of the probability distribution map can have a great impact on the

search and rescue operations. We proposed a Bayesian approach to help generate such a probability distribution map by modeling lost-person behaviors based on three terrain features: topography, vegetation, and local slope. Our objectives are to ease the generation of probability distribution maps for the search and rescuers and to improve the quality of these maps.

Our proposed model uses publicly available geographic information and enables domain experts to specify uncertainty in their prior beliefs of how the missing person will transition from one terrain feature to another. Using the Bayesian model, past human behavior data in wilderness can be incorporated into the model to generate posterior beliefs. Following a first-order Markov process, the posterior beliefs can be used to build a temporal state transition matrix that allows the generation of the posterior predictive probability distribution map for any given time interval. We evaluated our model using the Bayesian  $\chi^2$  test of goodness-of-fit from [55] because it allows the evaluation of multiple  $p$ -values for samples generated from the posterior parameter space. Results from the test suggest that our model fits the synthetic dataset well. The proposed Bayesian approach is promising, but we also acknowledge that the present model is limited to the proposed terrain features and could benefit from incorporating additional factors such as intended destination and trail-following.

In future experiments, we plan to let search and rescue experts specify terrain-based transitional probabilities so the prior predictive probability distribution can be generated using our model. Then we also let the experts directly specify a probability distribution on the regional map (with and without the restriction of only considering how terrain features would affect the lost-person's behavior). It would be very interesting to compare the resulting distributions and analyze the causes of any differences. However, because there is no "ground truth" with respect to the "correct" probability distribution, such comparisons will not be used as a form of validation. Instead, such information can be used to enrich prior beliefs in our model.

Future work should also explore how the generated probability distribution map can be used as a base by the search and rescue workers to reduce workload and also reduce the chance that the search and rescue workers might overlook certain areas that should have been allocated higher probabilities. It is also worth investigating what effect different spatial resolution (granularity) when sampling GPS track logs might have on the quality of the predicted probability distributions. The temporal model enables the search and rescue workers to view the dynamic changes of the probability distribution map over time. It will be beneficial to investigate further how search and rescuers can take advantage of this kind of information to improve search efficiency.

Most importantly, the proposed terrain feature-based Bayesian model is only the foundation of a larger framework. Future work should include incorporating more factors that affect lost-person behaviors into the network. Such factors include but are not limited to direction of travel, missing person profile, panicking factor, weather conditions and season of the year. The framework should allow incorporating observed data, such as a piece of clothing or candy wrapper, into the model as the search and rescue operation progresses. Our ultimate goal is to provide tools that will improve the efficiency and effectiveness of each search and rescue operation so the search and rescue workers can locate the missing persons in the minimum amount of time required, so lives can be saved.

## Chapter 4

### Paper: UAV Intelligent Path Planning for Wilderness Search and Rescue<sup>1</sup>

#### Abstract

In the priority search phase<sup>2</sup> of Wilderness Search and Rescue, a probability distribution map is created. Areas with higher probabilities are searched first in order to find the missing person in the shortest expected time. When using a UAV to support search, the onboard video camera should cover as much of the important areas as possible within a set time. We explore several algorithms (with and without set destination) and describe some novel techniques in solving this problem and compare their performances against typical WiSAR scenarios. This problem is NP-hard, but our algorithms yield high quality solutions that approximate the optimal solution, making efficient use of the limited UAV flying time.

---

<sup>1</sup>Published in IROS 2009 (IEEE/RSJ International Conference on Intelligent Robots and Systems) conference. Authors are Lanny Lin, and Michael A. Goodrich.

<sup>2</sup>Four qualitatively different types of search strategies are used in WiSAR: hasty search, constraining search, priority search, and exhaustive search. See [41] for more details.

## 4.1 Introduction

The use of mini-UAVs (Unmanned Aerial Vehicles) in Wilderness Search and Rescue (WiSAR) has gained interest for researchers and experienced advancement in recent years due to its low cost, portability, and potential field use [41]. The UAV onboard video camera provides visual support, enables search and rescue workers to systematically survey large areas of importance in real time [41, 92], and increases the workers' awareness of the environment.

For WiSAR, as time progresses, the survivability of the missing person decreases and the effective search radius increases by approximately 3km/hour [104, 120]. Therefore, search efficiency can dramatically affect the outcome of the search and rescue. In the prioritized search phase, the incident commander creates a probability distribution map for finding the missing person based upon terrain features, profile of the missing person, weather conditions, and subjective judgment of expert searchers. Such maps can also be created systematically by utilizing geographical information available to the public via the Internet [31, 68, 111]. UAVs have limited flying time, and in most cases, it is not long enough for the onboard video camera to cover the entire search area. For these reasons, the important question is this: given a probability distribution map, a starting point, an ending point (optional), and specified flying time, what is the best path that enables the UAV onboard video camera to “cover” as much of the probability distribution as possible?

Characteristics such as possibly repeated visits and probability cumulation make this a more challenging problem than standard Orienteering Problem (OP) and coverage problem. Contributions of this paper include novel path planning techniques (“global warming effect”, path crossover/mutation), additional specified-destination constraint while accumulating probability, a solid validation of the algorithms' performance, and applying algorithms to a practical, real-world application. Experimental results from this paper are conducted in simulation and not on-board a real UAV.

## 4.2 Problem Formulation

We model this problem as a discretized combinatorial optimization problem with respect to probability accumulated in the 2D space for UAVs that use gimbaled cameras. Using Koopman’s search metric of the instantaneous probability of detection by one glimpse [63], we assume the observer has a 100% target detection rate. This means that as the UAV camera footprint moves along the probability distribution map, it collects (“zeros out”) all the probability along the way and accumulates the probability. A good analogy would be thinking of the UAV as a vacuum cleaner sucking up probabilities with 100% efficiency.

In WiSAR operations, a UAV maintains an altitude of approximately 60m above ground and travels at roughly 12–13m/s [41]. With this height, the onboard camera footprint size comes to about 32m×24m. The batteries on the UAV can keep it airborne for approximately 1–2 hours depending on weather conditions. We assume that the UAV will always maintain the same height of 60m above ground (through Height-Above-Ground automation) and travel at the constant speed of 12m/s, and use 24m×24m as the effective camera footprint size. Given these parameters, a 60×60 probability grid, where each probability node is 24m×24m, represents an area of 2.0736km<sup>2</sup> that will take the UAV 2 hours to cover entirely. In our path planning, we restrict the direction a UAV can travel to only North, South, West and East (making only 90 degree turns), and it takes the UAV 2 seconds (1 time step) to travel from one node to its direct 4-connected neighbor. In real flights, a UAV can approximate a 90 degree turn (covering 3 nodes) in 4 seconds, so this model is close to UAV’s capabilities. Also during roll or yaw, the gimbaled camera can rotate to remain aiming straight down, enabling the 90 degree turn of the camera footprint.

Using  $i$  for the row number and  $j$  for the column number, each probability node (cell in grid) can be written as  $N_{ij}$  where  $0 \leq i, j < 60$ . The value of each  $N_{ij}$  is the total volume of

probability within the grid cell and thus

$$\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} N_{ij} = 1, \quad (4.1)$$

where  $n=60$ . Let  $T$  be the total number of time steps allowed for the UAV (specified flying time). Let  $P$  be the set of all possible paths for the UAV on the probability grid for  $T$  time steps. Each path,  $p_k \in P$ , can be represented by a sequence of probability nodes  $\{N_0, N_1, N_2, \dots, N_T\}$  consisting of  $T+1$  nodes. If the UAV is allowed to visit a node more than once, then the same node can be in a different part of the sequence.

If we use a binary variable  $x_{ij}$  to represent whether  $N_{ij} \in p_k$ ,  $x_{ij}$  becomes a function of path  $p_k$ :

$$x_{ij}(p_k) = \begin{cases} 1, & N_{ij} \in p_k \\ 0, & \text{otherwise} \end{cases} \quad (4.2)$$

The number of unique nodes visited is less than or equal to the length of the path:

$$\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_{i,j}(p_k) \leq T + 1, \quad (4.3)$$

and the total probability accumulated,  $PC_{p_k}$ , if the UAV follows path  $p_k$  is

$$PC_{p_k} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_{ij}(p_k) N_{ij}. \quad (4.4)$$

The optimal path  $p^* \in P$  is defined such that  $\forall p_k \in P, PC_{p^*} \geq PC_{p_k}$ , and our goal is to find or approximate the path  $p^*$ , which produces the maximum cumulative probabilities within reasonable computation time.

### 4.3 Related Work

Many algorithms have been used for UAV path planning such as Voronoi Diagram with Eppstein's  $k$ -best paths algorithm [5], A\* [92], LRTA\* [53], and Probability Roadmaps [91]. These papers focus on obstacle avoidance and sensing multiple targets.

For path planning in searching for a target, some researchers propose to use a probabilistic model and try to maximize accumulated probability along the path. In [46], Hansen et al. propose three search strategies: greedy, contour, and composite search, using a probability grid. In a series of papers (e.g. [11, 12]), Bourgault et al. describe a Bayesian framework for trajectory planning to maximize the chances of finding the target given restricted time using one or multiple UAVs and human systems. However, the solution uses a very simple 1-step lookahead approach which generates paths far from optimal and difficult to improve upon. Both papers do not consider the possible set destination constraint and also lack solid validation of the path efficiency.

If we disallow visiting the same node more than once, this problem falls within a variation of the Traveling Salesman Problem (TSP) called the Orienteering Problem (OP) [93] or the Prize-Collecting Traveling Salesman Problem (PCTSP) [45], both of which are NP-Hard [109]. Many exact solving methods for the OP have been developed ([32, 65, 93]). These exact methods can find optimal solutions to small OP problems, but for large-scale OP problems, approximation heuristic approaches are preferred. Mittenthal and Noon [77] present a heuristic approach that inserts or deletes a city from the subset-tour. Tasgetiren and Smith propose a Genetic Algorithm in [122] that encodes tours using a sequence of points and uses a penalty function to help search infeasible regions. Liang and Smith present an Ant Colony Optimization approach that uses an unusual sequenced local search and a distance-based penalty function in [67]. These algorithms work well with OP problems of small number of nodes (21–100 nodes) but can be slow with large number of nodes. They also don't allow repeated visits.



## 4.4 Path Planning Algorithms

Because none of the path-planning algorithms we discussed above work well under our model of the problem, we developed a set of algorithms based on the following ideas: Local Hill Climbing (LHC), Convolution, and Evolutionary Algorithms (EA). We also verify the paths generated to ensure the UAV is not flying backward or going outside of the allowed search area.

### 4.4.1 Algorithms without a Set Destination

In situations where the operator does not have a preference for where the path should end, the following algorithms were built and evaluated.

#### Complete-coverage Algorithm (CC)

The algorithm plans flight paths by following a lawnmower pattern. It first identifies the smallest  $m \times n$  bounding rectangle that contains all the non-zero probability nodes. If the starting location is inside the pattern, the algorithm simply generates a path following the pattern. Otherwise, it first plans a shortest path to the edge of the bounding rectangle. When allowed flight time is large enough, this algorithm is guaranteed to collect all the probabilities.

#### Local Hill Climbing Algorithms (LHC)

This is a greedy algorithm that always follows the direction with the highest value. A direct implementation of LHC does not work well with a multi-modal probability distribution map because the path generated stays with one mode until it has covered it completely before moving on to another. To address this problem, we use a global warming metaphor where the “ocean surface” represents all the zero-valued nodes and the “islands” represent the probability modes; see Fig. 4.1. We subtract a constant  $C$  from all nodes but keep all node values non-negative, where  $C = \max(N_{ij})/l$ , and  $l$  defines how fine grained the search

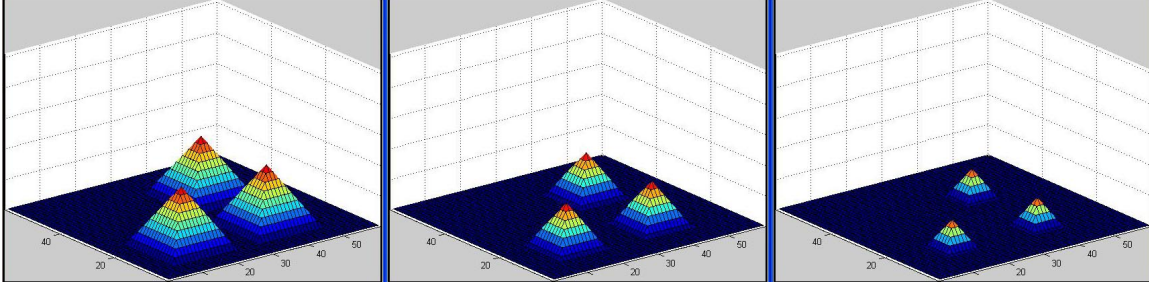


Figure 4.1: Global Warming Effect

should be:

$$N'_{ij} \leftarrow \begin{cases} N_{ij} - C, & N_{ij} > C \\ 0, & \text{otherwise} \end{cases} \quad (4.5)$$

When the ocean surface rises  $C$  each time, the volume of islands above water decreases, and if the ocean surface rises  $l$  times, all islands will be below water. In our experiments we set  $l=40$  and use the LHC algorithm to generate 40 paths: one before the ocean surface rises and one for each time the ocean surface rises (before water covers everything). We then recompute the probability accumulated for these 40 paths using the original probability grid and return the best path. This global warming technique allows the LHC algorithm to break out of one mode before completely covering that mode and move toward another. In case of a tie as to where to go next, we use two methods as the tie-breaker: LHC-GW-CONV uses a convolution kernel (with small, medium and large sizes) to determine which neighbor is more promising, and LHC-GW-PF uses Potential Fields (PF) with various discounting factors to determine where to go next.

## Evolutionary Algorithms

We developed two Evolutionary Algorithms: EA-Dir and EA-Path. Both use the probability accumulated for each path as the fitness function and employ the proportional selection method [76]. The difference between the two algorithms lies in the path representation during crossover.

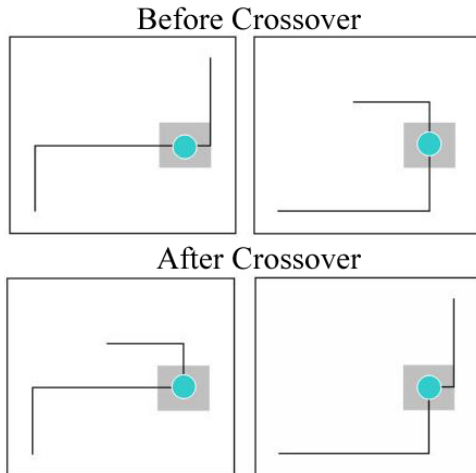


Figure 4.2: An example of single-point path crossover (Upper row: the parents. Lower row: the children)

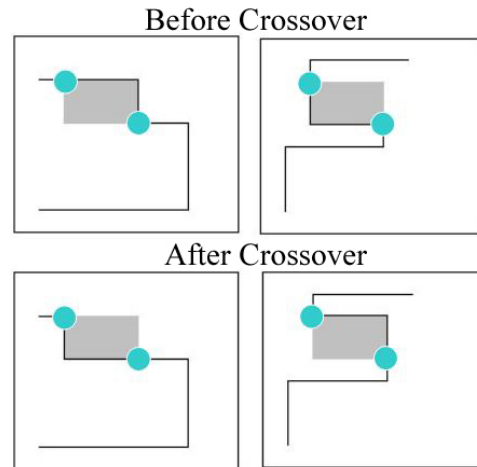


Figure 4.3: An example of double-point path crossover (Upper row: the parents. Lower row: the children)

With the EA-Dir algorithm, a path is encoded as a string of directions consisting of North, East, South, and West in the crossover phase (e.g. “NNWEE...”). Because the paths generated using single-point crossover [76] have a very high probability of being invalid (flying out of the map), we only use double-point crossover [76] and restrict the mid-section to a fixed 5-direction string.

With the EA-Path algorithm, a path is encoded as a sequence of node positions. If the two parent paths share only one common node, then single-point crossover is used; if they share two common nodes in the same order, then double-point crossover is used; otherwise, the two parent paths are discarded and the process starts over. For the single-point crossover method the two parent paths are crossed at the common node; see Fig. 4.2. For double-point crossover method, the first common node and the second common node in the parent paths mark the middle sections to be swapped; see Fig. 4.3. Both techniques could result in one longer path and one shorter path. The longer path is truncated back to the original path length and the shorter path is extended by performing crossover again and then truncating.

Two types of mutation methods [76] are used for flight path evolution; see Fig. 4.4. They follow a greedy approach with the hope that small positive changes to the path will

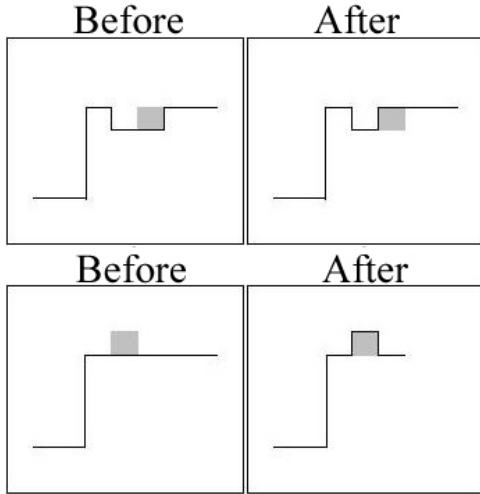


Figure 4.4: Examples of mutations in EA-DIR and EA-Path algorithms. (Upper row: method 1. Lower row: method 2)

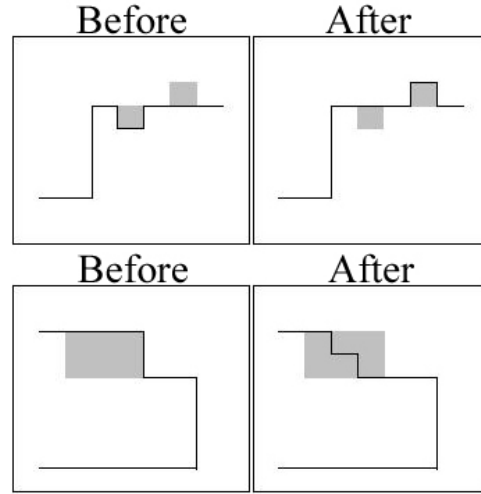


Figure 4.5: Examples of mutations in EA-Path\_E algorithm. (Upper row: method 2. Lower row: method 3)

lead to larger positive changes to the path. First we randomly select a node in the flight path and see if the next two nodes along the path would form an L shape with this node or a straight line (these are the only two possibilities). In the first case, method 1 (“flip”) is used and the algorithm replaces the middle node with the node that mirrors the middle node if we connect the first node and the third node with a line. This is like flipping a section of the path. In the second case, method 2 (“pull”) is used and the algorithm inserts two nodes into the path on one side of the line next to the first and the second nodes. This effectively extends the path by two nodes, so we simply truncate the last two nodes from the path. This is like pulling a string from the middle when the beginning end of the string is fixed. Which side to select for insertion depends on whether the new path is a valid path. If both sides allow valid paths, then the algorithm prefers inserting nodes that are not already in the path. Random selection is the last tie-breaker. If all four nodes on either side of the line are already included in the path, then a new mutation point is randomly selected and the same procedure repeats.

We use an initial population of 100 paths including various paths generated using other algorithms and 95 randomly generated paths. LHC-GW-PF is not used because it is too slow. Other parameters include replacement rate at 30% and mutation rate at 50%. The best three paths are always kept in each iteration. The algorithm runs for at least 500 iterations and stops if either the best path does not improve after 200 iterations or if the algorithm has completed 1000 iterations.

#### **4.4.2 Algorithms with a Set Destination**

In WiSAR, an operator might prefer the path to end at a specific destination node to support UAV retrieval, persistent visualization of a specific region at a specific time, or planning multiple path segments that make up a longer path. The following algorithms are modified versions from the previous section to handle the additional requirement. We simply add “\_E” to the algorithm names to distinguish them.

##### **Complete-coverage Algorithm (CC\_E)**

This algorithm is identical to the CC algorithm up to the time when the remaining flight time is just enough to fly the UAV to the end node, then it flies toward the end node using the LHC-GW-CONV\_E algorithm (discussed shortly).

##### **Local Hill Climbing Algorithms**

The LHC-GW-CONV\_E and LHC-GW-PF\_E algorithms have an additional constraint where nodes that prevent the path from reaching the end node within the remaining time will not be selected.

##### **Evolutionary Algorithm**

The direction representation of a path does not work with a set destination, so the EA-Path\_E algorithm also uses a sequence of node positions to encode the path. Here we

increased mutation rate to 90% to force more exploration of the state space. The initial population of 100 paths includes various paths generated using other algorithms as seeds (both from start node to end node and reversed) and 90 randomly generated paths.

The EA-Path\_E algorithm uses both single-point and double-point crossover. The difference is that when the child path is too long, the algorithm truncates the path to the original path length, then backtracks the path until the distance between the end of the child path and the desired end node matches the remaining time. The LHC-GW-CONV\_E algorithm is then used to complete the path with the desired end node. If the child path is too short, the LHC-GW-CONV\_E algorithm is used to complete the path.

The EA-Path\_E algorithm uses three types of mutation methods. First, we randomly select a node in the path and see if the next two nodes along the path would form an L shape with this node or a straight line. In the first case, method 1 (“flip”) is used (identical to the one used in the EA-Path algorithm); see Fig. 4.4. If the nodes form a straight line, then method 2 (“pull”) or 3 (“shake”) is selected with equal probabilities; see Fig. 4.5.

Mutation method 2 (“pull”) is a modified version from the EA-Path algorithm. This method does not truncate two nodes at the end of the path; instead, it deletes two nodes in the middle of the path. This is like pulling a string from the middle when both ends of the string are fixed.

Mutation method 3 (“shake”) works by first marking a small mid-section in the path (to keep it short, we set it to 6 nodes). We first randomly select a node in the path, then traverse the path and find the fifth node down the path. If the path between these two nodes is not a straight line, the method replaces the mid-section with random flying while maintaining the same length for the mid-section. This is similar to shaking a chain where the beginning and ending points remain fixed but the middle section shifts.

## 4.5 Experimental Results and Analysis

### 4.5.1 Performance Metrics

We use *Efficiency*, *Efficiency<sub>LB</sub>* and Running Time as metrics to measure the performance of the algorithms, where *Efficiency* is calculated if we know what's the best possible and *Efficiency<sub>LB</sub>* is used as an estimation when we have no way of calculating the best possible. Sorting all the probability nodes by their values in descending order would generate a list  $\{N_1, N_2, N_3, \dots, N_{3600}\}$ . For the best possible path  $p^*$ , the probability accumulated  $PC_{p^*}$  is constrained by a theoretical upper bound  $B$ :

$$PC_{p^*} \leq \sum_{n=1}^{T+1-d} N_n = B, \quad (4.6)$$

where  $d$  is the distance from the start node to the closest non-zero valued node. Then for any path  $p_k$ , we define *Efficiency* and *Efficiency<sub>LB</sub>* as the following:

$$Efficiency = \frac{PC_{p_k}}{PC_{p^*}} \quad (4.7)$$

$$Efficiency_{LB} = \frac{PC_{p_k}}{B} \quad (4.8)$$

$PC_{p_k}$  can be calculated using (4.4). *Efficiency* can be calculated when  $PC_{p^*}$  is known and *Efficiency<sub>LB</sub>* can be calculated anytime. Clearly,  $Efficiency_{LB} \leq Efficiency$ .

For example, a path with 95% *Efficiency* means the amount of probability accumulated following this path is 95% of the maximum possible. A path with 85% *Efficiency<sub>LB</sub>* means the probability accumulated is 85% of the maximum amount possible if the UAV can teleport from node to node, and the true *Efficiency* could be much higher.

All experiments are run on a Dual-core AMD 3800+ PC with 1GB of memory. For each algorithm, running time is recorded so we can compare algorithm speed.

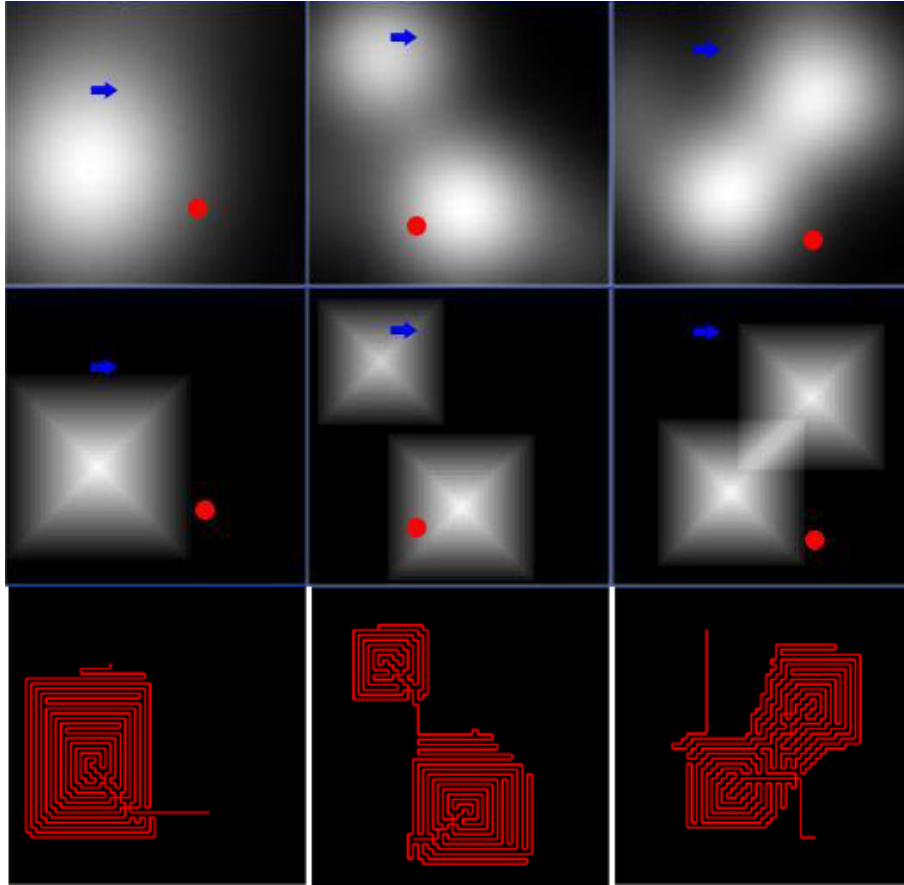


Figure 4.6: Top row: 2D representations of unimodal, bimodal, and bimodal with overlap probability distribution maps. Middle row: Simplified versions of the three types of maps. Bottom row: Best paths found for each map.

#### 4.5.2 Typical WiSAR Scenarios

In our experiments, we focus on probability distribution maps of three abstract but representative WiSAR scenarios: unimodal, bimodal, and bimodal with overlap. The top row of Fig. 4.6 shows the 2D representations where each pixel is a probability node; the lighter the pixel, the higher the probability value. The middle row shows three simplified versions of the distributions, which can be used to manually identify the best path possible for each map and compute  $PC_p^*$ . Then we can measure the true *Efficiency* of paths generated. The blue arrows on the maps mark the starting node (possible location for a WiSAR command center) and the red dots mark the ending node (intentionally selected at a different region from the starting nodes). The bottom row shows the best paths generated for the real maps at  $T=900$ .



### 4.5.3 Experimental Results and Analysis

For each distribution type (real and simplified maps) we ran each algorithm (with or without set destination) using  $T=120, 300,$  and  $900$  (4, 10, and 30 minutes). Because of random factors, we ran each experiment 10 times and calculated mean and standard deviation of the results. Due to space limitation, only a subset of the experimental results are presented (e.g. Table 4.1, 4.2 and Figure 4.7–4.9).

For all the experiments we performed, algorithm running time exhibited the same trend: from the fastest to the slowest we have LHC-GW-CONV(\_E), EA(\_E) and LHC-GW-PF(\_E). For example, with the simplified unimodal map, the LHC-GW-PF algorithm ran for 9.419, 41.952 and 164.383 seconds for  $T=120, 300$  and  $900$  respectively. Because the EA(\_E) algorithms use the path generated from other algorithms as seeds in the initial population, they are generally slower. However, most of the running time is spent generating the initial population and the evolutionary part of these algorithms only takes a fraction of a second. LHC-GW-PF(\_E) algorithms are always the slowest, and that is why we do not include them as seeds in the EA algorithms. For the group of algorithms with set destination, we perform path planning both from the starting node to the ending node and also from the ending node to the starting node (then reverse the path), and then select the better one; we include both runs when we record the algorithm running time. Therefore, the “\_E” algorithms always take more time to complete compared to the version before modification.

For the simplified unimodal map, the LHC-GW-CONV(\_E) algorithms are the clear winners in each respective group if we consider both the *Efficiency* and the running time. For the group of algorithms without set destination, all algorithms gave above 99.5% *Efficiency*. The LHC-GW-CONV algorithm is always the fastest (e.g. 6.483 seconds for  $T=900$ ) and achieved 100% *Efficiency* in all cases. The EA-Dir and EA-Path algorithms also achieved 100% *Efficiency*, but at a much slower speed (e.g. 62.236 seconds for  $T=900$  with EA-Path). For the group of algorithms with set destination, the LHC-GW-CONV\_E algorithm is also the fastest (e.g. 14.173 seconds for  $T=900$ ) and achieved 99.955% or higher *Efficiency* in all

(%)	Simplified ( <i>Efficiency</i> )			Real ( <i>Efficiency<sub>LB</sub></i> )		
$T$	120	300	900	120	300	900
LHC-GW-CONV	88.89	96.80	98.35	81.64	93.97	97.75
LHC-GW-PF	96.63	96.70	96.07	90.28	92.43	96.67
EA-Dir	98.59	97.31	98.80	90.62	94.96	97.96
EA-Path	98.66	98.09	99.07	91.18	95.71	98.02

Table 4.1: Algorithm efficiency comparison for bimodal distribution

(seconds)	Simplified			Real		
$T$	120	300	900	120	300	900
LHC-GW-CONV	0.90	2.26	7.35	0.52	1.16	5.66
LHC-GW-PF	9.44	29.11	131.35	2.61	8.64	92.38
EA-Dir	9.36	15.56	41.71	10.97	16.69	35.11
EA-Path	10.63	22.89	66.31	12.61	21.20	53.73

Table 4.2: Algorithm speed comparison for bimodal distribution

cases. Although the EA-Path\_E algorithm achieved slightly better *Efficiency* (less than 0.1% improvements), it did so at the cost of more running time (e.g. 78.334 seconds for  $T=900$ ).

For the simplified bimodal map, the LHC-GW-CONV(\_E) algorithms did not always perform well because it does not handle the space between the two modes very well, especially for very short flight time. Fig. 4.7 shows the *Efficiency* comparison of the group of algorithms without set destination. The LHC-GW-PF(\_E) algorithms still achieved 96% and above *Efficiencies*, but they are also the slowest. The EA(\_E) algorithms are more attractive in this case because they achieved the best *Efficiencies* (98.095%+ for EA and 97.857%+ for EA\_E) very quickly.

For the simplified bimodal with overlap map, the EA(\_E) algorithms achieved the best *Efficiencies* (98.302%+ for EA and 98.653%+ for EA\_E), but the LHC-GW-CONV(\_E) algorithms were able to achieve equivalent or slightly lower *Efficiencies* (97.391%+ for LHC-GW-CONV and 98.429%+ for LHC-GW-CONV\_E) with much less time (8.283 seconds and 16.296 seconds for  $T=900$  respectively). Fig. 4.8 shows the *Efficiency* comparison of the group of algorithms with set destination.

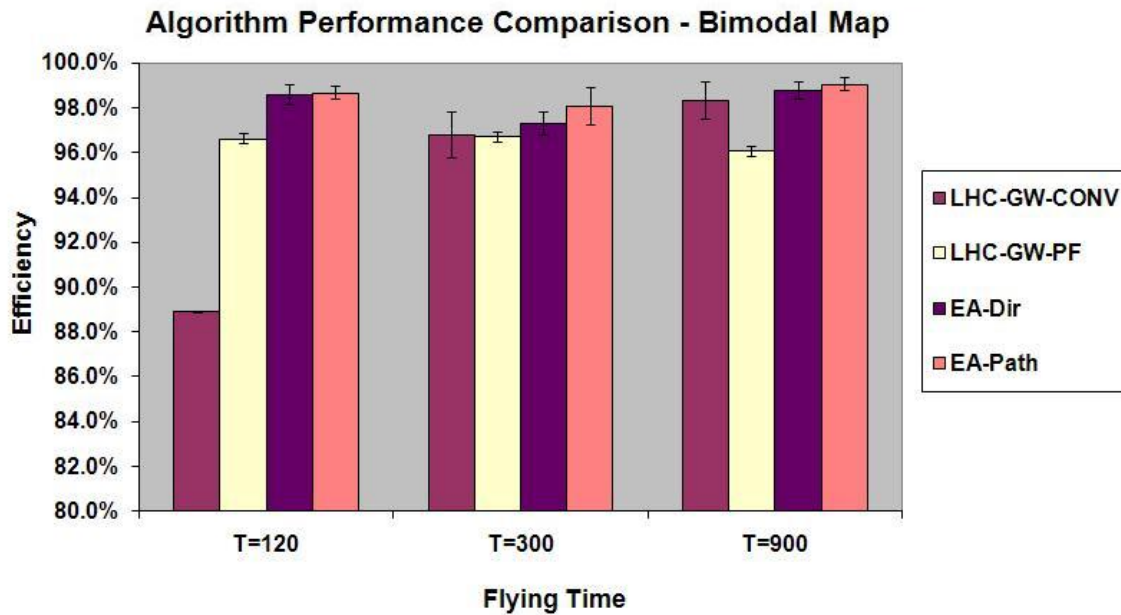


Figure 4.7: *Efficiency* comparison for group of algorithms without set destination for simplified bimodal map

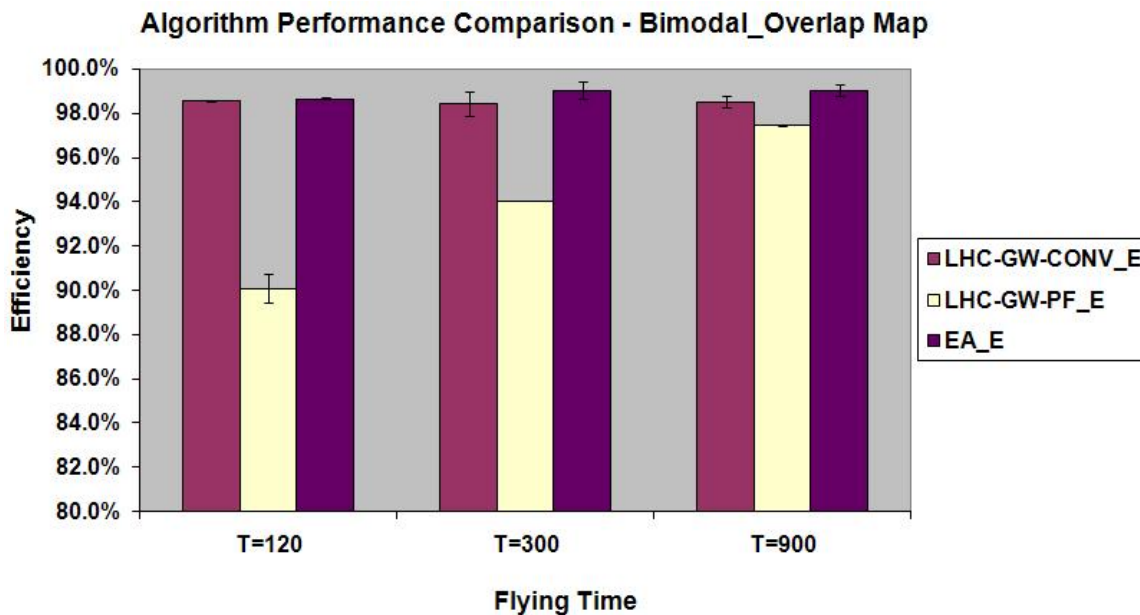


Figure 4.8: *Efficiency* comparison for group of algorithms with set destination for simplified bimodal with overlap map

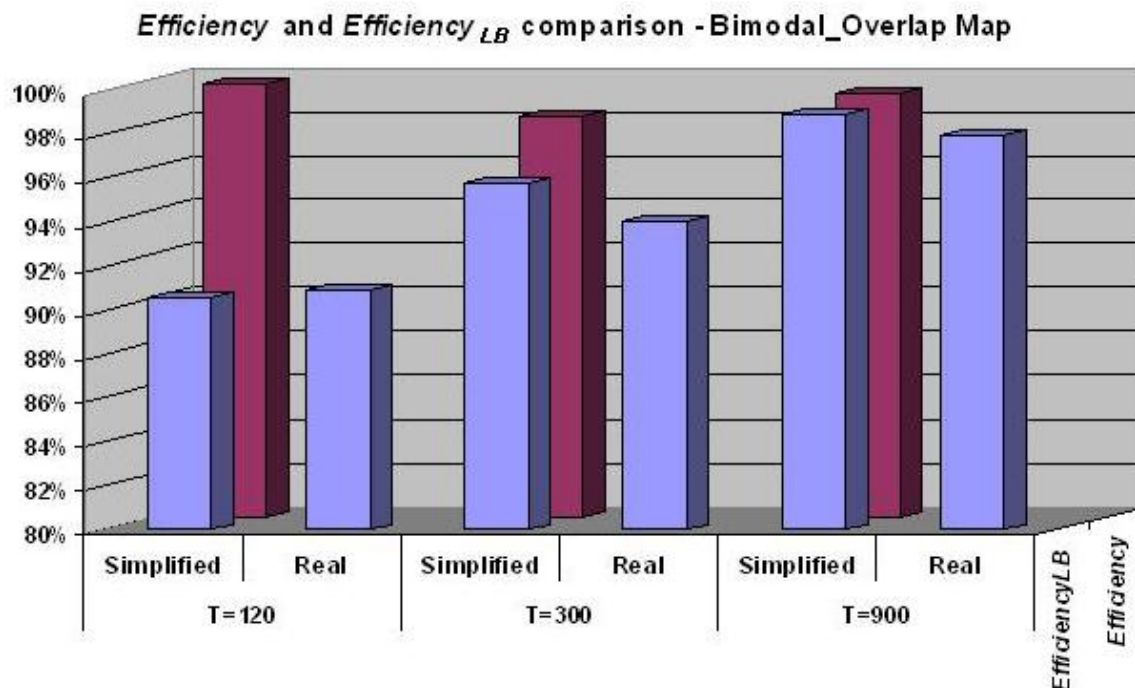


Figure 4.9: EA-Path performance for the real and simplified bimodal with overlap map

For each of the three real distribution maps (unimodal, bimodal, and bimodal with overlap), since  $PC_{p^*}$  is unknown, we can only calculate  $Efficiency_{LB}$ . We observed that the  $Efficiency_{LB}$  for each real map is very close to the  $Efficiency_{LB}$  for each of the counterpart simplified maps, and we hypothesize that the  $Efficiency$  for each real map should also be close to the  $Efficiency$  for each of the counterpart simplified maps. Fig. 4.9 shows an example of the EA-Path algorithm performance for the real and simplified bimodal with overlap map. The columns in the front row are  $Efficiency_{LB}$  values and the columns in the back row are  $Efficiency$  values. Based on this graph, we estimate that the  $Efficiency$  values for the real map here are above 97% for all  $T$  values.

To further evaluate our algorithms, we tested our algorithms on a more complex multimodal distribution map generated by mixing multiple Gaussian distributions with various standard deviations; see Fig. 4.10. The LHC-GW-CONV algorithm achieved 97.206%  $Efficiency_{LB}$  in 5.516 seconds and the EA-Path algorithm achieved 97.609%  $Efficiency_{LB}$  in 63.984 seconds. Note here that the  $Efficiency$  percentiles can only be better.

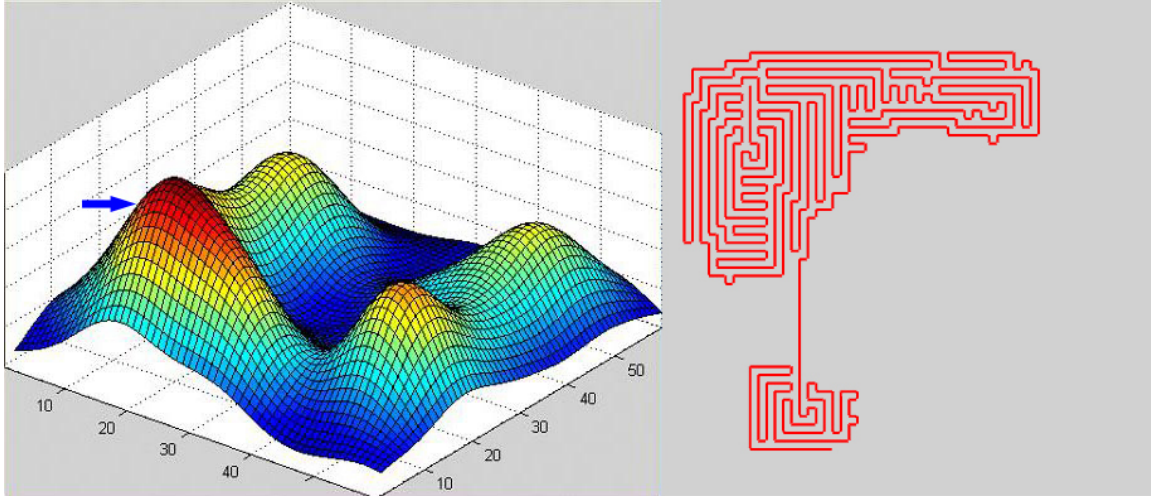


Figure 4.10: More complex multimodal probability distribution map

In every experiment, the EA(\_E) algorithms always achieved the best *Efficiency* and *Efficiency<sub>LB</sub>*. Therefore, if the operator has some time for computation, they seem to be attractive candidates. If the operator needs a path generated quickly, the LHC-GW-CONV(\_E) algorithms can be used. Although the LHC-GW-PF(\_E) algorithms do not work as well with these three distribution maps, initial tests on other distribution types such as sparse map and small-multimodal map suggest that they could perform better than other algorithms.

## 4.6 Conclusions and Future Work

We model the UAV path planning problem in WiSAR as a discretized combinatorial optimization problem and design two groups of algorithms for path planning with or without a set destination using algorithms based on Local Hill Climbing, and Evolutionary Algorithms using novel techniques such as “global warming effect” and path crossover/mutation. We evaluate the performances of these algorithms on six (3 simplified, 3 “real”) representations of typical WiSAR probability distribution maps, unimodal, bimodal, and bimodal with overlap, with various flight times and use the simplified maps to validate true efficiencies in real maps. Experimental results show that our algorithms can generate good paths with high *Efficiency*

or estimated *Efficiency* that approximate the optimal solution within reasonable computation time. Specifically, the LHC-GW-CONV(\_E) algorithms should be used for unimodal maps, and if a few minutes computation time is available, because the EA(\_E) algorithms always keep the best path found from seed algorithms, they can always find a path with the highest *Efficiency* compared with other algorithms experimented.

Experimenting with more types of distribution maps, designing a more advanced global warming search model, allowing 8-connected path planning, and dealing with dynamic distribution maps that change over time are all natural extensions for future work. Specifically, the set of algorithms with set destinations enables us to further investigate how the path planning task can be segmented so human operators can plan more strategically while the algorithms plan tactically, and what interface can make this an intuitive, smooth, and effective task for the UAV operator in WiSAR operations.

## Chapter 5

### Paper: Hierarchical Heuristic Search Using A Gaussian Mixture Model for UAV Coverage Planning<sup>1</sup>

#### Abstract

During UAV search missions, efficient use of UAV flight time requires flight paths that maximize the probability of finding the desired subject. The probability of detecting the desired subject based on UAV sensor information can vary in different search areas due to environment elements like varying vegetation density or lighting conditions, making it likely that the UAV will only be partially able to detect the subject. This adds another dimension of complexity to the already difficult (NP-hard) problem of finding an optimal search path. We present a new class of algorithms that account for partial detection in the form of a task-difficulty map and produce paths that approximate the payoff of optimal solutions. The algorithms use the *Mode Goodness Ratio* heuristic, which uses a Gaussian Mixture Model to prioritize search subregions. The algorithms search for effective paths through the parameter space at different levels of resolution. We compare the performance of the new algorithms against two published algorithms (Bourgault's algorithm and LHC-GW-CONV algorithm) in simulated searches with three real search and rescue scenarios, and show that the new algorithms outperform existing algorithms significantly and can yield efficient paths that yield payoffs near the optimal.

---

<sup>1</sup>Submitted to and accepted by SMC-B (IEEE Transactions On Systems, Man And Cybernetics Part B, Cybernetics) journal. Authors are Lanny Lin, Michael A. Goodrich and Spencer Clark

## 5.1 Introduction

Mini-UAVs (Unmanned Aerial Vehicles) are becoming useful tools in many reconnaissance, remote-sensing, surveillance, and search operations thanks to advances in UAV technologies. They can help firefighters map forest fires, help news crews provide coverage, help police monitor crowds, and help wilderness search and rescue workers locate a missing person. In these applications, the UAV uses its on-board cameras to provide useful visual information in support of the specific operation.

This paper focuses on using mini-UAVs to support Wilderness Search and Rescue (WiSAR). The aerial view from a UAV enables WiSAR workers to survey large areas of importance in real time [41]. Search efficiency is very important in WiSAR because, as time progresses, the survivability of the missing person decreases and the effective search radius increases by approximately 3km/hour [120]. Therefore, a good flight path should rapidly maximize the probability of finding the missing person to make efficient use of the limited flying time.

Each UAV path accumulates information over time as the UAV's sensors scan the ground. As illustrated in Fig.5.1, various paths do so in different ways depending on how information is distributed in the environment. The goal is to maximize the total probability of detection. There are two quality metrics for the probability-maximizing path planning problem [62, 115, 130]. First, find the path that maximizes the Cumulated Detection Probability (CDP) after a specific flight time (blue vertical dotted line). Out of the three example paths in Fig. 5.1 path 3 becomes the winner. Second, find the path that achieves a desired CDP in the shortest amount of time (red horizontal dotted line). Path 1 would become the winner out of the three, instead. We model the problem following the first approach.

When using a UAV's on-board camera to assist WiSAR operations, factors such as dense vegetation, lighting conditions, shadows, or distance between the camera and the ground can lower the quality of the UAV aerial view and decrease the probability of detection [82].



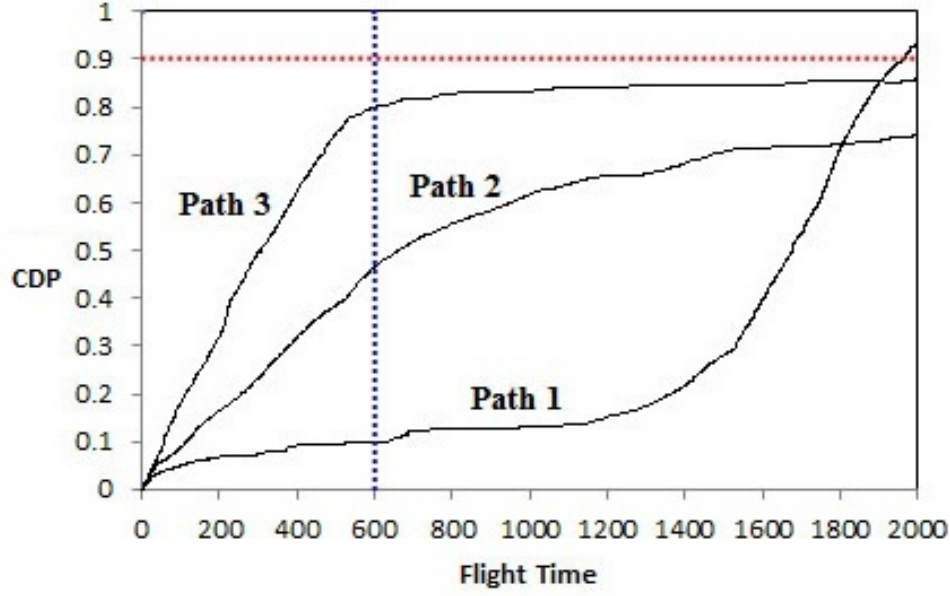


Figure 5.1: Two approaches to the probability-maximizing path planning problem. With three paths generated by various algorithms, the first approach prefers the path maximizing the Cumulated Detection Probability (CDP) given a specific flight time (Path 3 is the winner) and the second approach prefers the path achieving a specified CDP in the shortest amount of time (Path 1 is the winner)

This can be attributed to both sensor and human limitations (such as limited attention span and cognitive workload). We propose to represent *partial detection* in the form of a task-difficulty map, where a more difficult subregion on the map has lower probability of detection. Using a task-difficulty map enables us to integrate geo-referenced and spatial-related sensor constraints into the problem formulation, which supplements traditional sensor modeling methods (e.g., [11]) and potentially improves search performance in real world search scenarios. Because detection difficulties vary in different search subregions, flying patterns such as lawnmower and Zamboni don't guarantee optimal coverage. Integrating the task-difficulty map into path planning adds another dimension of complexity to the already difficult problem and causes the performance of existing greedy-type algorithms (Bourgault's Algorithm [11] and LHC-GW-CONV [69]) to suffer.

We model the path planning problem as a discrete combinatorial optimization problem, and propose a new heuristic, the *Mode Goodness Ratio*. This heuristic uses a Gaussian Mixture

Model to identify and prioritize search subregions. We then present two new algorithms (*Top2* and *TopN*) that utilize the heuristic in hierarchical path planning by forcing the UAV to visit high priority subregions. The hierarchical structure enables the algorithms (a) to cluster probability volumes and (b) to prioritize search subregions at different levels of resolution. It also makes it easy to parallelize the two new algorithms and improve computation speed. We compare the performance of the new algorithms against two published algorithms (Bourgault's Algorithm [11] and LHC-GW-CONV algorithm [69]) in simulated searches based on three real search and rescue scenarios. Results show that the new algorithms outperform existing algorithms significantly and can yield efficient paths that approximate the payoff of the optimal path.

The contributions of the paper are (a) the introduction of Gaussian Mixture Model (GMM) to compute the Mode Goodness Ratio heuristic, which can be used to prioritize search subregions in a hierarchical planner, (b) two new path planning algorithms that utilize the Mode Goodness Ratio heuristic to improve path-planning performance, and (c) the use of a spatial representation (task-difficulty map) in modeling sensor detection probability with terrain and vegetation information and incorporating that into UAV path planning.

Section 5.2 defines the problem and the metrics used to evaluate algorithm performance. Section 5.3 discusses related literature. Section 5.4 first reviews two existing algorithms and then demonstrates the weakness of these algorithms with a synthetic scenario. This section then presents the *Mode Goodness Ratio* heuristic and the two new algorithms (*Top2* and *TopN*). Section 5.5 compares algorithm performance with three real search and rescue scenarios. Section 5.6 discusses the limitations of the approach, and Section 5.7 presents the summary.

## 5.2 Problem Formulation

### 5.2.1 Problem Framework

Typical UAVs (fixed-wing or rotorcraft) are highly mobile and variable, but we will assume a set of useful constraints on their capabilities: they have a gimbaled camera, can maintain a constant height above ground and can travel at constant speed. A gimbaled camera enables the camera to aim straight down even when the UAV is performing roll or yaw maneuvers. We assume that the UAV's speed is much higher than the speed of the missing person and treat the missing person as stationary. At every UAV flight time step, we treat the camera footprint of the search area as a glimpse. This way we can discretize the search area, and model the UAV path planning problem as a discrete combinatorial optimization problem with respect to probability accumulated and define it following the framework described in [126].

The search space is represented as a finite, connected graph  $G = (V, E)$ .  $V$  denotes the set  $\{v_1, \dots, v_n\}$  of vertices of  $G$ , and  $E$  denotes the set of edges. Each edge in  $E$  can be viewed as an unordered pair of vertices  $\{v_i, v_j\}$ . The missing person is located at one of the vertices of  $G$ . A given probability distribution map for the missing person is discretized to match graph  $G$ , with  $p_i$  being the probability that the missing person is located at vertex  $v_i$ . It is obvious that

$$\sum_{i=1}^n p_i = 1. \quad (5.1)$$

The UAV search is conducted in discrete time. During each time step, the UAV camera footprint can cover one vertex. For a desired flight with  $T$  time steps, let  $S$  denote the set  $\{0, 1, 2, \dots, T\}$ . The UAV's motion is constrained by the structure of the graph  $G$ . Let  $\Psi$  be the set of functions  $\psi : S \mapsto V$  with the property that for any two consecutive integers  $t$  and  $t + 1$  in  $S$ , either  $\psi(t) = \psi(t + 1)$  or  $\{\psi(t), \psi(t + 1)\} \in E$ . Here  $\Psi$  represents all possible paths, and under path  $\psi$ , vertex  $\psi(t)$  is searched during step  $t$ . The conditions on the set  $\Psi$

guarantee that at each time step the UAV camera footprint will either remain at the current vertex (only possible for a rotorcraft) or move to a neighboring vertex.

Even when the UAV camera footprint covers the vertex occupied by the missing person, it is not certain that a detection will occur. The probability of detection is described by a glimpse probability function  $g$ , which is defined by a given task-difficulty map. The task-difficulty map is a spatial representation of sensor detection probability and defines areas where it is difficult to detect the missing person (with lower probability of detection), with  $d_i$  being the task-difficulty level at vertex  $v_i$ . Let  $d_{max}$  be the maximum task-difficulty level in the given map. If at time step  $t$  the UAV camera footprint covers vertex  $v$ , then let  $g(v, t)$  be the probability that a detection will occur, given that the missing person is at vertex  $v$ . We model this as

$$g(v_i, t) = 1 - \frac{d_i}{d_{max} + 1}. \quad (5.2)$$

so that more difficult tasks (higher  $d_i$  values) have lower glimpse detection values,  $g$ .

Let  $P_T(\psi)$  represent the Cumulative Detection Probability (CDP) for path  $\psi \in \Psi$  with  $T$  time steps. For each (cell, time) pair  $(i, t)$  with  $1 \leq i \leq n$  and  $0 \leq t \leq T$ , we define the probability of failure  $f(i, t, \psi)$  by

$$f(i, t, \psi) = \begin{cases} 1 - g(v_i, t) & \text{if } \psi(t) = v_i \\ 1 & \text{otherwise.} \end{cases} \quad (5.3)$$

Let  $D_j$  represent a detection on the  $j$ th observation so  $\bar{D}_j$  is a detection failure. Then the probability of failing to detect the the missing person after  $N$  observations of vertex  $v_i$  given the missing person is at vertex  $v_i$  is the joint probability  $P(\bar{D}_1, \bar{D}_2, \dots, \bar{D}_N | v_i)$ . Assuming each observation is conditionally independent of each other (typical in the WiSAR literature), we can rewrite the joint probability as

$$P(\bar{D}_{1:N} | v_i) = \prod_{j=1}^N P(\bar{D}_j | v_i), \quad (5.4)$$

and the probability of detecting the missing person after  $N$  observations is

$$P(D_{1:N}|v_i) = 1 - P(\bar{D}_{1:N}|v_i), \quad (5.5)$$

which is equivalent to

$$P(D_{1:N}|v_i) = 1 - \prod_{t=0}^T f(i, t, \psi), \quad (5.6)$$

where  $N$  is how many times  $v_i$  shows in path  $\psi$ . Then  $P_T(\psi)$  can be computed by

$$P_T(\psi) = \sum_{i=1}^n p_i \left( 1 - \prod_{t=0}^T f(i, t, \psi) \right), \quad (5.7)$$

where  $p_i$  is the probability that the missing person is located at vertex  $v_i$ . Define  $\exists \psi^* \in \Psi$  such that for any alternate path  $\psi' \in \Psi$ ,  $P_T(\psi^*) \geq P_T(\psi')$ . Our goal is to find the optimal path  $\psi^*$  that produces the maximum CDP (path 3 in Fig.5.1 at  $T = 600$  if there are only three possible paths) or find an efficient path  $\psi'$  that produces payoff approximating the payoff of the optimal path within reasonable computation time.

When the path needs to end at an operator-specified vertex (for easy UAV retrieval or to join with other path segments), we simply add the constraint to the problem formulation (only add edges to a path that does not violate the constraint so the UAV has enough time to reach the end point). Also for fixed-wing UAVs, additional motion constraints (such as not allowing the UAV to fly backward) are also introduced as velocity constraints affecting edges  $\{\psi(t), \psi(t_1)\}$ , effectively creating a directed graph. Both proposed path planning algorithms satisfy these constraints.

### 5.2.2 Performance Metrics

We will use two measures of algorithm performance: the quality of the path and the run-time of the algorithm. Run-time is self-explanatory, but we need a measure of path quality. Ideally the efficiency of an algorithm should be computed with the following equation:

$$Efficiency(\psi') = \frac{P_T(\psi')}{P_T(\psi^*)}, \quad (5.8)$$

where  $\psi^*$  is the optimal path. However, because we do not know what  $\psi^*$  is, we bound the efficiency. Let  $\psi_{teleport}$  be defined as the path constructed as follows: (a) deduct the time needed to move from the start vertex to the nearest vertex with non-zero  $p_i$  (plus time for doing the same with the end vertex if specified), and (b) at each step the UAV teleports to the vertex that allows the UAV to collect the highest amount of probability after considering the task-difficulty at that vertex. Then, all the probability collected during the teleport flight is summed, giving  $Efficiency_{LB_i}$  for path  $i$ :

$$Efficiency_{LB_i} = \frac{P_T(\psi_i)}{P_T(\psi_{teleport})} \quad (5.9)$$

Since  $P_T(\psi^*) \leq P_T(\psi_{teleport})$ ,  $Efficiency$  can be no worse than  $Efficiency_{LB}$ , so the latter sets a lower bound for the true efficiency. Note that the majority of the teleport path  $\psi_{teleport}$  is made up of disjointed points because the UAV would be “jumping” (teleporting) from vertex to vertex, always landing on the vertex that promises highest amount of probability collectible.

### 5.3 Related Work

Many path planning algorithms in the literature address obstacle avoidance while planning a path to reach a destination using A\* [92], D\* [114], Voroni diagrams [8], or probability roadmaps and rapidly-exploring random tree (RRTs) [91]. Hierarchical heuristics approaches were also developed, such as Hierarchical A\* (HA\*) by Holte et al. [51], hierarchical task-based real-time path planning by Naveed et al. [74], and Hierarchical-AO\* (HiAO\*) by Meuleau and Brafman [84]. The algorithms we present solve a different path planning problem by generating paths that make efficient use of the limited travel time and maximizing the probability of finding the missing person. This is similar to the Vehicle Routing Problem [64]

and the Orienteering Problem (OP), which is a variation of the Traveling Salesman Problem (TSP) and is known to be NP-Hard [109]. However, our path planning problem is even more difficult with added challenges of repeated visits and partial detection. Tasgetiren and Smith propose a Genetic Algorithm in [122] to solve OP. Liang and Smith present an Ant Colony Optimization approach that uses an unusual sequenced local search and a distance-based penalty function for path planning [67]. These algorithms work well with OP problems with few nodes (21–100) but can be slow with many nodes. Unfortunately, they do not allow repeated visits and do not support partial detection. Although classic dynamic programming [108] method can solve TSP, because TSP is NP-hard, it cannot be solved in polynomial time, unless  $P=NP$ . The method suffers the “curse of dimensionality” and does not scale well with complex problems. Reinforcement learning (approximate dynamic programming) methods [7, 117] have four main sub-elements: a policy, a reward function (immediate payoff), a value function (long-term payoff), and optionally, a model of the environment. Because in our path planning problem, a node can be visited multiple times, and because our Bayesian approach allows for partial collection of information, the score/prize collected for each visit is different. The reward function and the value function both become path dependent, the state space becomes exponentially large. We seek a real-time solution that scales well when search area and flight duration expand, therefore we prefer a heuristic approach.

In the 1950’s, Koopman discussed the uncertainties in the act of detecting hostile submarines with radars and proposed a concept called the instantaneous probability of detection by one glimpse [61]. He presented simple search algorithms and demonstrated how search effort should be distributed given a prior probability distribution of the target and known law of detection when only a limited total amount of search effort (or time) is available [62]. Stone [115] presents various search plans with partial detection models using Lagrange multipliers and maximization of Lagrangians in finding stationary target in very basic search problems when no false targets are present. Washburn [130] discusses how to

construct optimal search paths for different search problems. The author also developed detection models based on radar/sonar and expanded the fundamentals of search theory to include moving targets. In [11] Bourgault et al. describe a Bayesian framework for UAV trajectory planning to maximize the chances of finding the target given restricted time. Partial detection was modeled based on a downward-looking millimeter wave radar, and a one-step lookahead method was used for path planning using posterior distributions obtained from Bayes filter [125] updates. More recent work includes [85] where Niedfeldt et al. present a UAV path planning algorithm that utilizes probability of detection and maximizes the probability of identifying an object using a N-step lookahead method, and [99] where Ryan and Hedrick developed a control formulation for a fixed-wing UAV that minimizes the entropy of an estimate distribution over a receding horizon for searching a moving target over a fixed time horizon. N-step lookahead and receding horizon methods are greedy-type algorithms that run into scalability bounds and generate sub-optimal paths in situations when a complicated detection model is used, such as a task-difficulty map.

Koester compiled statistics from large set of past WiSAR incidents [60]. These statistics can be used to construct probability distribution maps. Ferguson describes how GIS can be used to segment search areas into probability subregions [31]. Goodrich et al. [41] describe how a probability distribution of likely places to find the missing person can be useful for UAV path planning. Lin and Goodrich [70] propose a Bayesian model to create such a distribution based on terrain features and past human behavior data. The model has been evaluated using real search and rescue scenarios at George Mason University's MapScore web portal [18] and performed well compared to other statistical models. Stone et al. used posterior probability maps and successfully located the wreckage of Air France Flight 447 [116]. Metrics such as Koopman's instantaneous probability of detection by one glimpse [61], "seeability" proposed by Morse et al. [82], and terrain and vegetation information obtained from USGS [70] can be used to build a task-difficulty map representing probability of detection in different search subregions.



The *Mode Goodness Ratio* heuristic is used to evaluate the “peakedness” of a bivariate Gaussian. The traditional way to evaluate the peakedness of a distribution uses kurtosis [4]. Mardia [73] extends the concept to multivariate distributions. Because multivariate kurtosis is difficult to compute and may show inconsistency in the meaning the peakedness of a distribution, Khurshid et al. [59] extend Horn’s measure of peakedness [52] into a measure for bivariate normal distributions. The heuristic we propose is an even simpler method to measure the peakedness and is well-adapted to support hierarchical search algorithms.

## 5.4 Path Planning Algorithms

In this section we review two existing path planning algorithms, Bourgault’s Algorithm [11] (referred to as BA from here on) and LHC-GW-CONV [69], and demonstrate the weakness of these algorithms with a synthetic scenario. Next we formally define the Mode Goodness Ratio heuristic. Then we present the Top2 & TopN algorithms.

### 5.4.1 BA and LHC-GW-CONV algorithms review

The BA algorithm [11] is a Bayesian approach to the UAV path planning problem. Given a prior probability distribution of the missing person, it uses the Bayes filter as described in [125] to compute the posterior probability distribution at every time step. The probability of detection follows an active model of a downward looking millimeter wave radar where signal power is determined by factors such as emitted power, antennae footprint, and sensor distance to the target. Distributions are discretized into a grid for calculation and a (greedy) one-step lookahead method is used to determine which cell the UAV should fly to next (the grid cell with the highest posterior probability).

Our formulation in Section 5.2 can be viewed as a Bayes filter with the following assumptions:  $p_i$  is the prior probability,  $g(v_i, t)$  is the detection likelihood, and we assume a stationary object of interest. Instead of using a greedy approach, we look ahead much further down the path. In order to address the increased computational complexity, we use a

heuristic (introduced in the next section) and rely on a hierarchical approach to improve the search for efficient paths. Because the speed of our algorithms is very fast, our algorithms can turn into a “greedy” algorithm with extended horizon when dealing with moving object or changing environment.

The sensor model in BA uses target distance and signal strength, which implicitly considers the spatial information of the environment. We use a task-difficulty map to take advantage of explicit prior knowledge of the environment and how it affects the detection probability spatially. For a fair comparison, we used the same downward-looking camera visual sensor model when we implemented the BA algorithm, and we note that our algorithm can use detection models similar to the one used in [11].

The LHC-GW-CONV algorithm [69] is a combinatorial optimization approach to the UAV path planning problem. It discretizes the given probability distribution of the missing person and the task-difficulty map into a grid and uses a Local Hill-Climbing algorithm to select the next cell to fly to (the grid cell with the highest one glimpse detection probability). Spatial averaging is performed by convolving the combined probability distribution and the task-difficulty map using box filters. This serves as the tie-breaker, enabling the algorithm to look beyond local neighbors in order to plan paths toward broader areas with high probability. Even with spatial smoothing a typical problem of LHC is that it favors local maxima, resulting in the UAV getting stuck in a local probability hill for too long before it can move to another probability hill. To overcome the problem, a “Global Warming” technique is used<sup>2</sup>. After each “ocean rise”, a new path is created, and the best path is returned as the final path found. Equation 5.10 shows how the probability  $p_i$  that the missing person is at vertex  $v_i$

---

<sup>2</sup>The name “Global Warming” comes from the metaphor where the “ocean surface” represents all the grid cells with zero probability and the “islands” represent probability hills with non-zero grid cells; as the “ocean” rises the volume of probability hills above the water decreases.

changes when the “ocean” rises.

$$p'_i \leftarrow \begin{cases} p_i - nC, & \text{if } p_i > nC \\ 0, & \text{otherwise} \end{cases} \quad (5.10)$$

where  $C$  is a constant height of each “ocean rise” and  $n$  is how many times the “ocean” will rise.

Both BA and LHC-GW-CONV are greedy algorithms. The advantages of greedy algorithms include low computational cost and flexibility in quickly adapting to changes (e.g., a changing environment or a moving target). A major drawback is that such algorithms tend to get stuck in local maxima. We can demonstrate this using the synthetic scenario in Fig.5.2, which shows a multi-modal distribution of the missing person location and a simple task-difficulty map with three difficulty levels.

For a UAV path where  $T = 900$ , if the UAV starts from a subregion with low task-difficulty (upper left corner), the BA algorithm achieved 65.99%  $Efficiency_{LB}$  and the LHC-GW-CONV algorithm achieved 96.28% (averaged for 10 runs); Fig.5.3 shows the paths generated by the two algorithms. The BA algorithm’s performance is okay but not great, while the LHC-GW-CONV algorithm performed really well (actually slightly better than the performance of the Top2 and TopN algorithms, which we will discuss in detail in section 5.6). But if the UAV starts from a subregion with high task-difficulty (lower right corner), both algorithms perform poorly (much worse than the performance of the Top2 and TopN algorithms), with BA scoring 41.91% and LHC-GW-CONV scoring 53.71% (averaged for 10 runs) in  $Efficiency_{LB}$ ; Fig.5.4 shows the paths generated by the two algorithms. This is because both greedy algorithms fail to move the UAV quickly out of the local probability hill. The Top2 and TopN algorithms we propose address this problem by forcing the UAV to visit other search subregions and also allocate more flight time to subregions where the UAV can be more efficient. In order to identify better subregions, we propose the Mode Goodness Ratio heuristic.

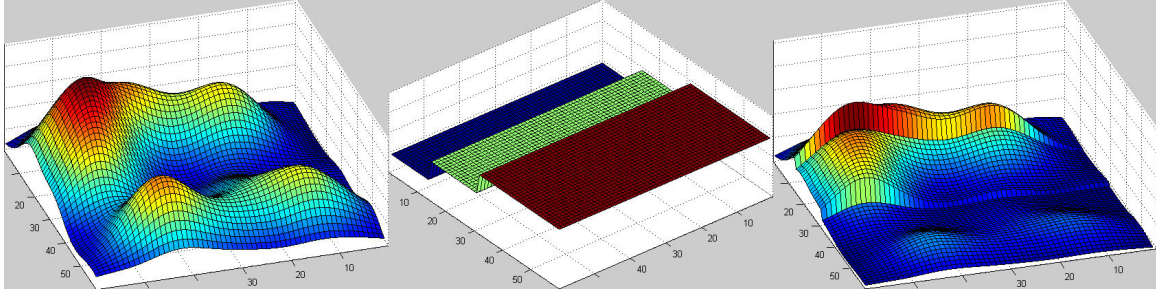


Figure 5.2: A synthetic WiSAR scenario. Left: Multi-modal probability distribution. Middle: A simple task-difficulty map. Right: Probability collectible on first visit (combining probability distribution and task-difficulty map).

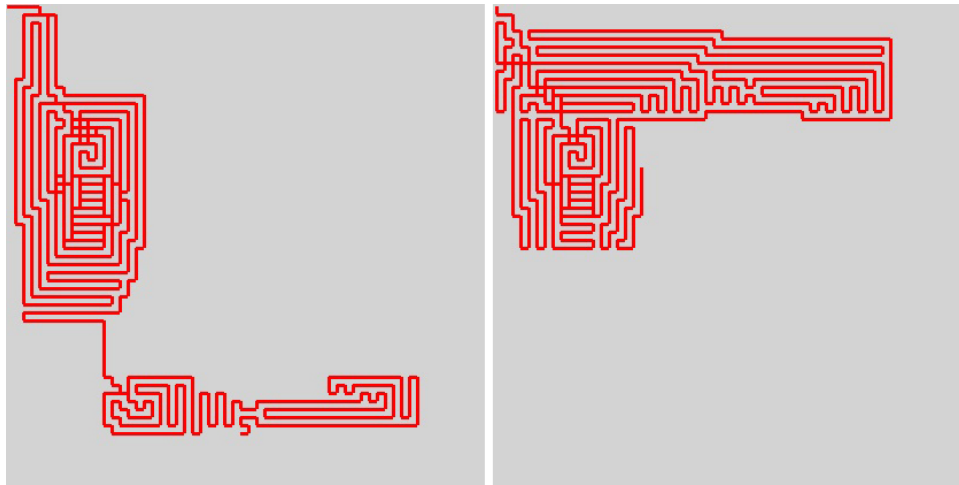


Figure 5.3: Paths found at  $T = 900$  when the UAV starts from a subregion with low task-difficulty (upper left corner). Left: Path created by BA. Right: Path created by LHC-GW-CONV.

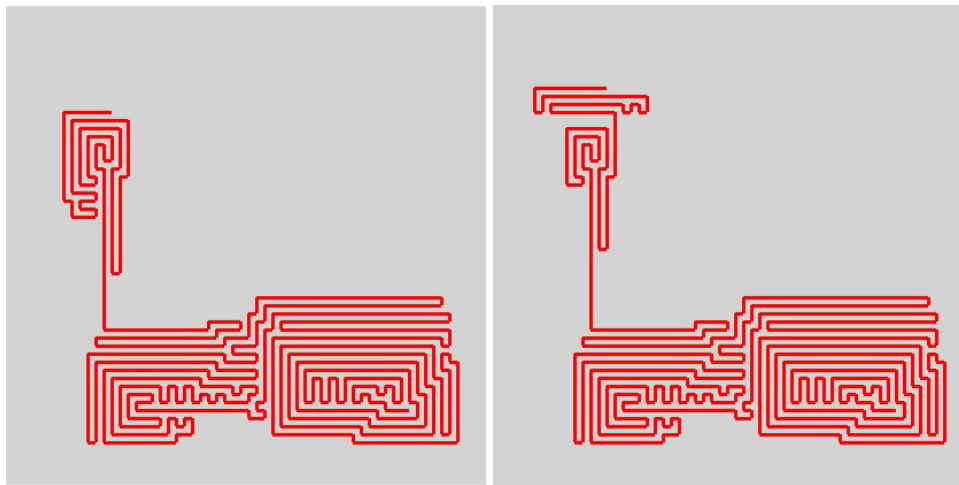


Figure 5.4: Paths found at  $T = 900$  when the UAV starts from a subregion with high task-difficulty (lower right corner). Left: Path created by BA. Right: Path created by LHC-GW-CONV.

### 5.4.2 Mode Goodness Ratio

The Mode Goodness Ratio heuristic prioritizes search subregions, where each subregion represents a cluster of probability volume that can be “collected” by the UAV sensor. Compute the heuristic as follows: First, combine the probability distribution map and the task-difficulty map to construct a new grid/surface  $G'$ . The value of each cell in  $G'$  represents how much probability can be collected the first time the cell is visited (e.g., the right part of Fig.5.2). Second, use a Gaussian Mixture Model (GMM) to partition  $G'$  into high quality clusters/subregions. We subjectively set the maximum number of subregions to 5 to reduce computational complexity.

A GMM is a probabilistic model for finding sub-populations within an overall population and is often used for data clustering. We choose the GMM method for two reasons: 1) We can take advantage of the resulting Gaussian parameters and coefficients to estimate the peakedness of the probability hills. 2) A GMM is a parametric method, so we can define subregions by cluster probability volume hierarchically and search through the parameter space.

It is important to point out that when a task-difficulty map (especially a complicated one) is applied, the resulting grid/surface  $G'$  is unlikely to resemble a mixture of Gaussians and we only use GMM to approximate the probability hills.

We used the Accord.MachineLearning library in the Accord.NET framework<sup>3</sup> to estimate GMM parameters. We generate data points to approximate  $G'$  (create a 2D histogram of  $G'$  and generate number of points proportional to each bin count) and then feed these points to the Accord library, which first uses the K-Means algorithm to generate  $k$  initial clusters, and then uses the Expectation Maximization (EM) algorithm to iteratively fit data to a mixture of Gaussians. Gupta and Chen provide detailed description on how to use EM to learn a GMM model in [44]. The results are a set of ( $k$ ) scaled Bivariate Gaussian distributions with their means, covariance matrices, and the coefficient (scale) for

---

<sup>3</sup><http://code.google.com/p/accord/>

each Gaussian. For completeness, Equation 5.11 shows the density function for a multivariate Gaussian distribution.

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}((\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}))}, \quad (5.11)$$

Next we identify all modes in the grid/surface  $G'$  (using simple local hill-climbing with verification for plateaus and ridges), and match the mean of each Gaussian to the closest mode centroid (in case the mode has a flat peak) and then use that centroid,  $C_i$ , to represent the subregion. Note that the number of modes in  $G'$  can be more than the number of modes in the probability distribution after a task-difficulty map is applied. If there are fewer than 5 modes in  $G'$ , we reduce  $k$  accordingly to reduce computation.

We evaluate three factors when computing Mode Goodness,  $MG_i$ , for subregion  $i$ : distance ratio  $D_i$ , probability volume  $V_i$ , and subregion area  $A_i$ .

The first factor, the *distance ratio*  $D_i$ , is defined as:

$$D_i = \log \left( \frac{T}{\alpha_i + 1} \right), \quad (5.12)$$

where  $T$  is the total UAV flight time (in time steps) and  $\alpha_i$  is the L1 norm distance from the start location of the path to the centroid of the subregion,  $C_i$ . If an end location is specified for the path, then that distance is also added:

$$\alpha_i = \begin{cases} \|\text{Start} - C_i\|_1, & \text{no End} \\ \|\text{Start} - C_i\|_1 + \|\text{End} - C_i\|_1, & \text{otherwise} \end{cases} \quad (5.13)$$

We add 1 to the denominator in Equation 5.12 to make sure it will never be 0, and use the log scale to reduce wide-ranging quantities to a smaller range.

The idea behind the distance ratio is that a subregion is less attractive when it takes a large percentage of the total flight time to reach the center of the subregion because the

trip to get there might not be very efficient. Therefore, higher  $D_i$  values indicate closer subregions.

The second factor, the *probability volume*  $V_i$ , is defined as:

$$V_i = V_{3\sigma_{x'_i}3\sigma_{y'_i}} w_i, \quad (5.14)$$

where  $V_{3\sigma_{x'_i}3\sigma_{y'_i}}$  is a constant (roughly 99.46%) representing the volume of probability under a standard bivariate Gaussian surface within 3 standard deviations, and  $w_i$  is the weight of each Gaussian component  $G_i$ , which is the coefficient of the Gaussian in the mixture as shown below with the property of  $\sum_{i=1}^k w_i = 1$ .

$$p(\mathbf{x}) = \sum_{i=1}^k w_i G_i \quad (5.15)$$

The idea behind the probability volume is that a subregion is more attractive when the volume of probability within the subregion is high, meaning visiting the subregion has the potential of collecting a large amount of probability. Therefore, higher  $V_i$  values indicate subregions with more probability.

After rotating the axes of the bivariate Gaussian to align with the eigenvectors of the covariance matrix  $\Sigma_i$ , the area under the surface within 3 standard deviations in both axes can be estimated using a rectangle with width  $3\sigma_{y'_i}$  and height  $3\sigma_{x'_i}$  where  $\sigma_{x'_i}$  and  $\sigma_{y'_i}$  are the square roots of the eigenvalues of the Gaussian's covariance matrix. The *area* of the rectangle  $A_i$  is the third factor in the heuristic. A larger  $A_i$  means it takes more time steps for a UAV to cover the area. Therefore, the lower  $A_i$  is, the better the subregion.

$$A_i = (3\sigma_{x'_i})(3\sigma_{y'_i}) = 9\sigma_{x'_i}\sigma_{y'_i}. \quad (5.16)$$

When we divide  $V_i$  by  $A_i$ , we are basically estimating the peakedness of the Gaussian. Then assuming the peakedness is independent of the distance ratio  $D_i$ , we can multiply them

together to compute the Mode Goodness of the subregion  $i$ :

$$MG_i = D_i V_i A_i^{-1}. \quad (5.17)$$

Since all we really care about is the priority order of the search subregions, we can simplify computation by computing the Mode Goodness Ratio,  $MGR_i$ , for subregion  $i$  with respect to subregion 1 as the following:

$$MGR_i = \frac{MG_i}{MG_1} = \frac{D_i V_i A_i^{-1}}{D_1 V_1 A_1^{-1}} \quad (5.18)$$

$$= \frac{D_i V_{3\Sigma} S_i (9\sigma_{x'_i} \sigma_{y'_i})^{-1}}{D_1 V_{3\Sigma} S_1 (9\sigma_{x'_1} \sigma_{y'_1})^{-1}} \quad (5.19)$$

$$= \frac{D_i S_i (\sigma_{x'_i} \sigma_{y'_i})^{-1}}{D_1 S_1 (\sigma_{x'_1} \sigma_{y'_1})^{-1}} \quad (5.20)$$

Naturally,  $MGR_1$ , the Mood Goodness Ratio for subregion 1 with respect to subregion 1 will always be 1 and  $MGR_i$  for other subregions can be less or greater than 1. By sorting the Mode Goodness Ratios of all the subregions, we have a way of prioritizing them according to their mode goodness.

### 5.4.3 Top2 Algorithm

The Top2 algorithm is designed to generate paths that force the UAV to visit the top 2 subregions in the search area. This way the heuristic-based path planner can escape from a probability hill where task-difficulty is high and probability of detection is low. First the Mode Goodness Ratio heuristic is used to identify the top 2 search subregions (represented by centroids). Then, local hill climbing is used to create the shortest path segment from the start location to the nearest centroid. If an end location is specified in the path planning request, another path segment is created similarly from the end location to the other centroid.

The algorithm then identifies a point (vertex) equidistant from the two centroids (the green square) and launches two path planning tasks to plan path segments from each



centroid to that point using local hill climbing. By allocating different percentages of the remaining flight time to these two path planning tasks, the Top2 algorithm can effectively search within a new dimension of time allocation. The subregion with more flight time allocated ends up with a longer path segment. Note that it is possible for the path to cover other subregions (other than the top 2) when a lot of flight time is allocated. Fig.5.5 shows three time allocation examples.

A coarse-to-fine search is performed starting from a low resolution (large chunks of flight time transferred from one path planning task to the other) and gradually increasing the resolution (smaller chunks) until the best path is found. Then the path segments are joined together to form a full flight path. Fig.5.6 shows the pseudo-code for the Top2 algorithm.

Because we can specify how many Gaussians to fit during the GMM step, we can actually cluster the probability hills hierarchically, and this structure enables us to search through different hierarchy layers with different  $k$  values (e.g., top 2 out of 5, top 2 out of 4, etc.). These path planning tasks at different layers can each run the Top2 algorithm in parallel, taking advantage of the computing power of a multi-processor system; the path with the best performance is returned as the final result.

#### 5.4.4 TopN Algorithm

The TopN algorithm forces the UAV to visit  $N$  subregions ( $5 \geq N > 1$ ). The algorithm first selects the top  $N$  search subregions using the Mode Goodness Ratio heuristic. Then, similar to the Top2 algorithm, it plans the two shortest path segments connecting the start and end locations of the path with the nearest centroids (mode A and D respectively). Next, the algorithm starts multiple path segments from the  $N$  centroids as shown in Fig.5.7 ( $N = 4$  in this example), one from the centroid nearest to the start (segment 1 from mode A), one from the centroid nearest to the end (segment 2 from mode D), and two segments for each other centroid (segment 3–6 in mode B and C). Segment 3 and 4 are connected at the center of mode B and segment 5 and 6 are connected at the center of mode C. The four segments

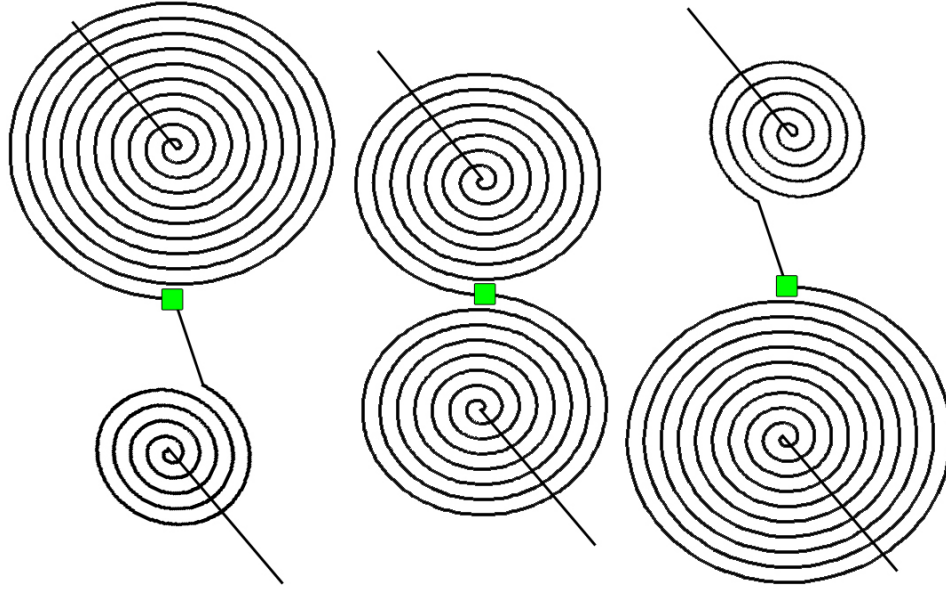


Figure 5.5: Illustrations of the Top2 algorithms where the top is subregion 1 and the bottom is subregion 2. Left: More flight time allocated to subregion 1. Middle: Equal flight time allocated to both subregion 1 and 2. Right: More flight time allocated to subregion 2.

```

Path Top2(Point start, int T, ArrayList centroids, ProbabilityMap map, int tChunk) {
1. Find closest centroid to start c1 and time needed t1
2. Plan straight path from start to c1 and store in path1
3. Find point center equidistant from c1 and c2
   map.VacuumProbability(path1);
   t2min = L1dist(c1, center);
   t3min = L1dist(c2, center);
   double efficiency = 0;
   int t2 = T-t1-t3min;
   int t3 = T-t1-t2;
   while (t2 ≥ t2min) {
       t2 -= tChunk;
       t3 += tChunk;
       (e2, path2) = LHC(c1, center, t2);
       (e3, path3) = LHC(c2, center, t3);
       if (e2 + e3 > efficiency) {
           efficiency = e2 + e3;
           pathRest = JoinPaths(path2, path3)
       }
   }
   return JoinPaths(path1, pathRest);
}

```

Figure 5.6: Pseudo-code for the Top2 Algorithm when no end point is specified at one layer of the hierarchy (e.g., top 2 Gaussians out of 5) and one coarse-to-fine level defined by tChunk.

spiral outward from the center. This technique allows the UAV to fly to the desired centroid in a “spiral in” fashion and then leave the centroid in a “spiral out” fashion without any overlaps, thus heuristically minimizing unnecessary revisits and still providing a good coverage of the probability hill. Six path segments perform local hill climbing at the same time and at each one-step lookahead, only the path segment with the maximum gain gets to add the neighboring vertex to the path. This process continues until the remaining flight time is just enough to connect all six segments in the shortest way possible. In the last step, path segments are connected into one continuous path using local hill climbing. In the example shown, segment 3 and 1 join to connect mode A and B; similarly segment 4 and 5 connect mode B and C and segment 6 and 2 connect mode C and D. Note that by planning two path segments from the center of the same Gaussian mode, this allows the UAV to spiral in to the center of the mode and then spiral out without crossing paths and revisit nodes, approximating a Fermat’s spiral (a special type of Archimedean Spiral), and improve the search efficiency (especially for an area with relatively uniform detection probability). Fig.5.8 shows the pseudo-code for the TopN algorithm.

Similar to the Top2 algorithm, the algorithm can specify how many Gaussians to fit during the GMM step and, in addition, search through different  $N$  values (e.g., 4 out of 5, 3 out of 5, 2 out of 5, etc.). The TopN algorithm for each hierarchy layer is run in parallel and returns the path with the best performance as the final result.

Although the Top2 algorithm might appear similar to a special case of the TopN algorithm where  $N = 2$ , it is not. First, the Top2 algorithm would force a path to go through the vertex (the green square in Fig.5.5) equidistant from the two centroids; The TopN algorithm does not have this constraint. Secondly, although both algorithms would plan two path segments and join them together to form the final path, Top2 algorithm actually generates multiple final paths (by allocating different portion of flight time to the two path segments) at the current hierarchy and then searches for the one with the best turnout. The TopN algorithm, however, only generates one final path at the current hierarchy. At each

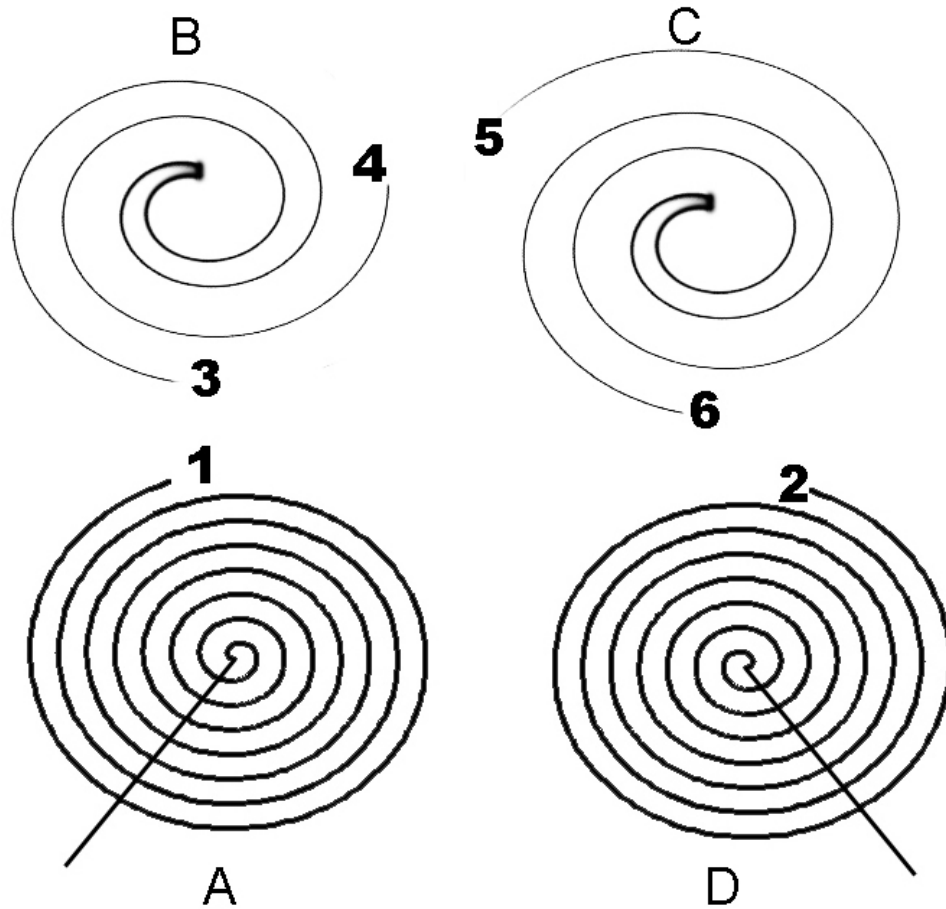


Figure 5.7: Illustrations of the TopN algorithms with top 4 subregions ( $k = 5$  and  $N = 4$ ).

time step, only the segment with the maximum gain in the next move grows (deducting a time step from the remaining flight time), until the remaining flight time is just enough to connect the two path segments. And with only one path generated, there's no need to search further at the current hierarchy.

Although simple, the Top2 and TopN algorithms become powerful when combined with the MGR heuristic and a hierarchical structure.

```

Path TopN(Point start, Point end, int T, ArrayList centroids, ProbabilityMap map) {
1. Find closest centroid to start c1 and time needed t1
2. Remove c1 from ArrayList centroids
3. Plan straight path from start to c1 and store in path1
   map.VacuumProbability(path1);
4. Find closest centroid to end cN and time needed tN
5. Remove cN from ArrayList centroids
6. Plan straight path from end to cN and store in path2N
   map.VacuumProbability(path2N);
   int TLeft = T - t1 - tN;
   Path path2 = new Path();
   Path path2.add(BestNeighbor(c1));
   Path path2NMinus1 = new Path();
   Path path2NMinus1.add(BestNeighbor(cN));
   ArrayList segments = new ArrayList();
   ArrayList segments.add(path2);
   ArrayList segments.add(path2NMinus1);
   foreach (Point c in centroids) {
       Path p1 = new path();
       p1.add(c);
       Path p2 = new path();
       p2.add(BestNeighbor(c));
       segments.add(p1);
       segments.add(p2);
   }
   while (EnoughTimeToJoinAllSegments(TLeft)) {
       Path path = SegmentWithBestNeighbor(segments);
       Point p = BestNeighbor(p.lastPoint());
       path.add(p);
       map.VacuumProbability(p);
       TLeft--;
   }
   return JoinPaths(path1, path2N, segments);
}

```

Figure 5.8: Pseudo-code for the TopN Algorithm with end point specified at one layer of the hierarchy (e.g., top 4 Gaussians out of 5).

## 5.5 Experiment Results and Analysis

### 5.5.1 Experiment Set Up

We selected three real WiSAR scenarios to test the performance of the proposed algorithms for ecological validity. All three scenarios were obtained from George Mason University, and all came from the International Search and Rescue Incident Database (ISRID) [60]. In each scenario, the missing person's Last Known Position (LKP) is at the center of a 2.4km×2.4km search area, therefore, we always start the UAV path from the center of the map. The probability distribution map of the missing person for each scenario is generated using the Bayesian model presented in [70]. These probability distribution maps have been evaluated at George Mason University's MapScore web portal [18] and performed better than most other models evaluated<sup>4</sup>. The task-difficulty map for each scenario is built using vegetation density data downloaded from the USGS web site and categorized into three difficulty levels (sparse, medium, and dense). Although this method only considers the vegetation density, it gives us a reasonable task-difficulty map and serves well for the purpose of demonstrating algorithm performances<sup>5</sup>. The probability distribution maps and the task-difficulty maps are discretized into 100×100 grids.

For each scenario, we compare the performance of the BA, LHC-GW-CONV, Top2, and TopN algorithms in  $Efficiency_{LB}$  and running time for three flight durations ( $T = 300, 600, 900$ , equivalent to 10, 20, and 30 minutes). Because we re-implemented the BA algorithm in MATLAB and the rest algorithms in C#, for a fair comparison we omit the running time for the BA algorithm. We also present the performance of the Top2 and TopN algorithms for just one hierarchy layer to demonstrate that the two algorithms can achieve much better  $Efficiency_{LB}$  in comparable running time with even arbitrary parameters ( $k = 5$  Gaussians and  $N = 3$  for top 3 subregions). In all the experiments we did not specify the ending

---

<sup>4</sup>Scoring 0.8184, 0.9858, and 0.9892 on a [-1,1] scale where the higher the score the better. <http://sarbayes.org/projects/>

<sup>5</sup>In real wilderness search and rescue operations, these maps would be further improved by domain experts before they are used for path planning.



Figure 5.9: Satellite imagery of the search area for the HikerPaul scenario (near the Grayson Highlands State Park in Virginia) showing the vegetation density. The Last Known Position (LKP) of the missing person is at the center of the image.

location for the UAV because the BA algorithm does not support this feature. All the other algorithms, however, do support this feature.

Experiments were performed in simulated searches and not on-board real UAVs. All paths generated in the experiments were for a hexacopter although the algorithms also work for fixed-wing UAVs. All experiments were run on a Intel 4-core i7-2600 PC with 16GB of memory. For each scenario we ran 10 experiments and recorded the mean and standard deviation of  $Efficiency_{LB}$  and running time. Due to space limitations, only a subset of the experiment results are presented.

### 5.5.2 Experiments Results and Analysis

In the first WiSAR scenario (HikerPaul), an elderly couple was reported missing near the Grayson Highlands State Park in Virginia (Fig.5.9 shows a satellite imagery of the search area for the scenario). In the second WiSAR scenario (NewYork53), a 46 year old male camper was reported missing near Adirondack Park in upper state New York. In the

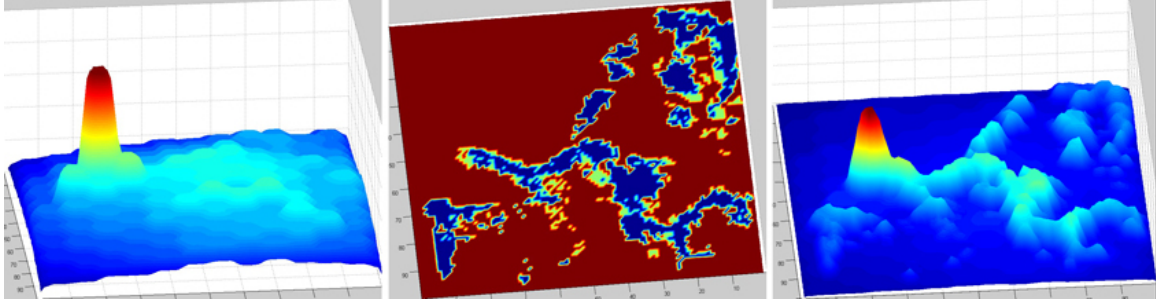


Figure 5.10: The HikerPaul scenario. Left: Probability distribution map. Middle: Task-difficulty map. Right: Surface after combining the probability distribution map and the task-difficulty map.

third WiSAR scenario (NewYork108), two teenage female hikers were reported missing near West Chesterfield in Massachusetts. For each scenario the Last Known Position (LKP) of the missing person is in the center of the search region. Fig.5.10, 5.13 and 5.15 show the probability distribution map (left) and the task-difficulty map (middle) for these scenarios. The right part of the figure shows the resulting surface for each scenario when we combine the probability distribution map and the task-difficulty map, which is the amount of probability the UAV can collect on its first visit to each vertex (or grid cell). The task-difficulty maps indicate that large areas of these search regions were covered with dense vegetation, which makes detecting the missing person more difficult. There are also small subregions with sparse vegetation (higher probability of detection). Fig.5.11, 5.14 and 5.16 show the paths generated by the BA, LHC-GW-CONV, Top2, and TopN algorithms for each scenario, respectively. The teleport paths for these scenario are not shown because they are mostly made up of disjointed points. Note that the paths sometimes revisit vertices that have already been visited (path segments cross with previous segments), but the combined surfaces we show in Fig.5.10, 5.13 and 5.15 (right) only represent the amount of probability the UAV can collect on its first visit. Each surface is updated after each vertex visit to reflect the amount of probability collectible on the next visit.

For the HikerPaul scenario, Fig.5.11 shows that both the BA and LHC-GW-CONV greedy-type algorithms generated paths that centered around the starting point and could



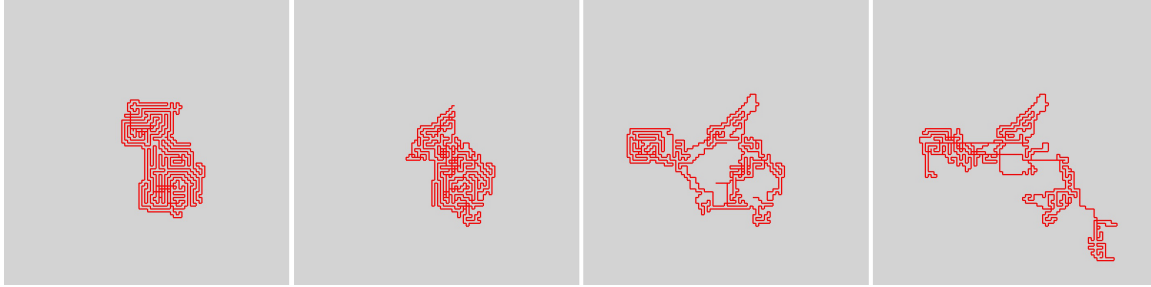


Figure 5.11: Paths generated for HikerPaul scenario with  $T = 900$  a) BA b) LHC-GW-CONV c) Top2 d) TopN

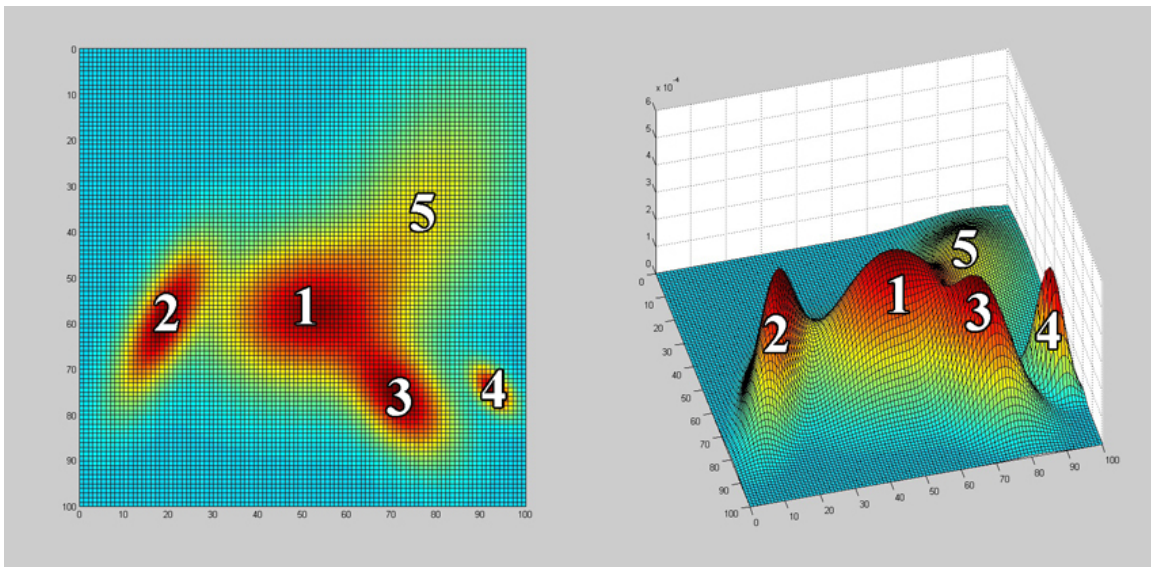


Figure 5.12: The Gaussian Mixture identified for the HikerPaul scenario with  $T = 900$  and  $k = 5$ . The numbers show the ranking of the Gaussians using Mode Goodness Ratio. Left: Gaussians in 2D. Right: Gaussians in 3D.

Table 5.1: Algorithms  $Efficiency_{LB}$  and running speed comparison for the HikerPaul scenario. All numbers shown are averages of 10 runs. All  $Efficiency_{LB}$  standard deviations are below 0.1.

$T$	$Efficiency_{LB}$ (%)			Speed (seconds)		
	300	600	900	300	600	900
BA	56.95	60.07	57.11	-	-	-
LHC-GW-CONV	60.18	56.76	55.18	0.30	0.47	0.98
Top2 (1 layer)	66.68	65.21	66.08	<b>0.24</b>	0.30	0.41
TopN (1 layer)	76.19	71.02	68.26	0.25	<b>0.24</b>	<b>0.22</b>
Top2 (Hierarchy)	78.67	73.81	72.75	0.73	0.84	1.19
TopN (Hierarchy)	<b>81.43</b>	<b>75.48</b>	<b>74.13</b>	1.52	1.73	1.68

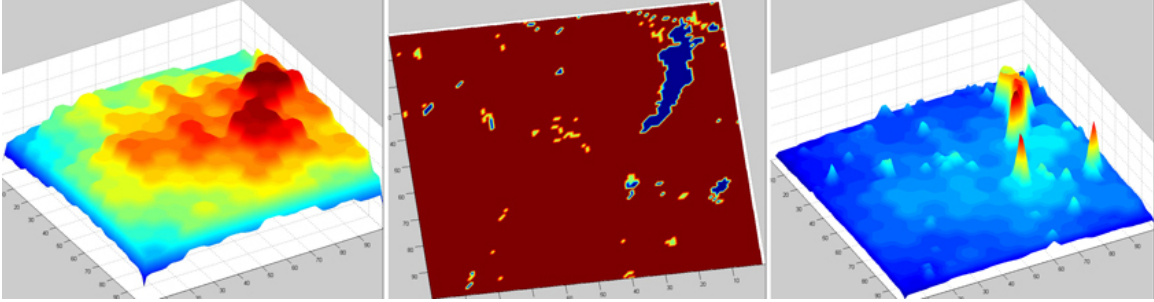


Figure 5.13: The NewYork53 scenario. Left: Probability distribution map. Middle: Task-difficulty map. Right: Surface after combining the probability distribution map and the task-difficulty map.

not break away from the probability hills near the center of the search area. The Top2 algorithm, on the other hand, directed the UAV to cover the tall probability hill on the left side of the search area, and the TopN algorithm additionally directed the UAV to cover subregions in the lower right of the search area where more probability can be accumulated. Fig.5.12 demonstrates how a GMM can be used to prioritize search subregions and shows the 5 Gaussians identified when we performed the Gaussian fitting for the HikerPaul scenario with  $T = 900$  and  $k = 5$ . The Gaussians are ranked using the Mode Goodness Ratio heuristic values (1.39, 1.01, 1, 0.87, and 0.46 respectively). Table 5.1 shows the performance of the four algorithms and also the Top2 and TopN algorithms with specific parameters (Number of Gaussians to fit:  $k = 5$  and top  $N$  subregions for TopN algorithm:  $N = 3$ ). The Top2 and TopN algorithms clearly outperform the BA and LHC-GW-CONV algorithms (whether using arbitrary parameters or search through the hierarchy) with significantly better *Efficiency<sub>LB</sub>*. Searching through the hierarchy generated more efficient paths than only working with one layer of the hierarchy. The TopN algorithm also achieved slightly better *Efficiency<sub>LB</sub>* than the Top2 algorithm. When using arbitrary parameters (only generating a path for one layer of the hierarchy), both the Top2 and TopN algorithms are faster than the LHC-GW-CONV algorithm. When searching through the hierarchy, the Top2 and TopN algorithm did take a little bit longer, but still completed within 2 seconds.

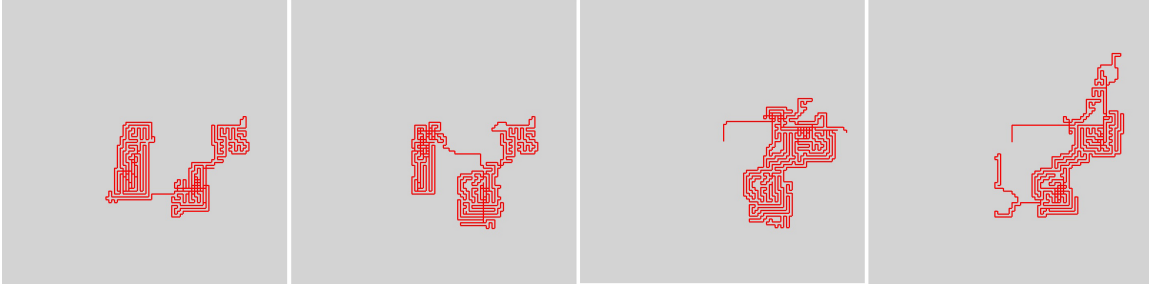


Figure 5.14: Paths generated for NewYork53 scenario with  $T = 900$  a) BA b) LHC-GW-CONV c) Top2 d) TopN

Table 5.2: Algorithms  $Efficiency_{LB}$  and running speed comparison for the NewYork53 scenario. All numbers shown are averages of 10 runs. All  $Efficiency_{LB}$  standard deviations are below 0.07.

$T$	$Efficiency_{LB}$ (%)			Speed (seconds)		
	300	600	900	300	600	900
BA	39.95	54.27	65.08	-	-	-
LHC-GW-CONV	38.47	56.91	67.38	<b>0.01</b>	<b>0.02</b>	<b>0.02</b>
Top2 (1 layer)	54.42	66.61	72.79	0.75	0.92	0.81
TopN (1 layer)	59.15	68.78	74.54	0.70	0.77	0.69
Top2 (Hierarchy)	57.18	69.29	74.44	1.87	2.06	1.92
TopN (Hierarchy)	<b>65.39</b>	<b>71.47</b>	<b>77.36</b>	5.01	5.76	5.32

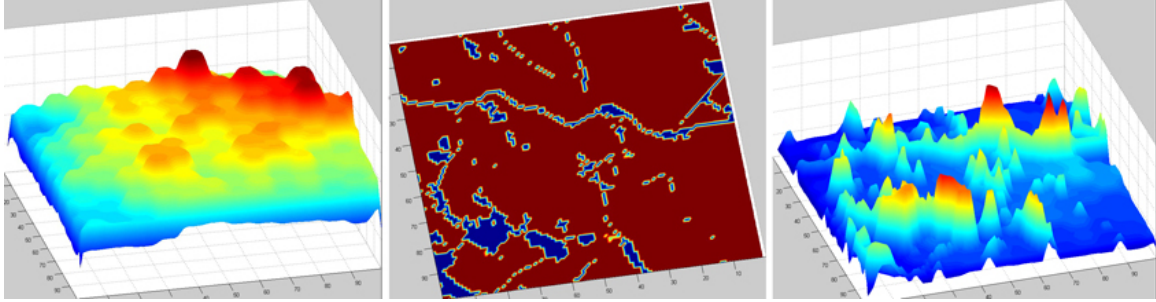


Figure 5.15: The NewYork108 scenario. Left: Probability distribution map. Middle: Task-difficulty map. Right: Surface after combining the probability distribution map and the task-difficulty map.

For the NewYork53 scenario, Fig.5.14 shows that both the BA and LHC-GW-CONV greedy-type algorithms generated paths that spent a good amount of time right at the center of the search area around the starting point before sending the UAV to two other subregions on the right. The Top2 and TopN algorithms, by contrast, did not waste any time at the center subregion and immediately directed the UAV to cover the two subregions on the right side of the search area. The TopN algorithm also directed the UAV to cover a subregion in the upper right part of the search area. Table 5.2 shows the performance of the four algorithms and also the Top2 and TopN algorithms with specific parameters ( $k = 5$  and  $N = 3$ ). The results show the same trend as with the first scenario where the Top2 and TopN algorithms outperform the BA and LHC-GW-CONV algorithms significantly in  $Efficiency_{LB}$ . Even with arbitrary parameters, the Top2 and TopN algorithms generated much more efficient paths (e.g., 59.15% for TopN with one layer vs. BA with 39.95%). The TopN algorithm also outperformed Top2 algorithm in  $Efficiency_{LB}$ . When looking at the algorithm completion time, LHC-GW-CONV algorithm is the clear winner in this scenario. When arbitrary parameters are used, the Top2 and TopN algorithms both completed within 1 second, but when searching through the hierarchy, both algorithms took much longer (about 2 seconds for Top2 and 6 seconds for TopN) to complete.

For the NewYork108 scenario, Fig.5.16 shows that both the BA and LHC-GW-CONV greedy-type algorithms generated paths that spent a good amount of time at the center of

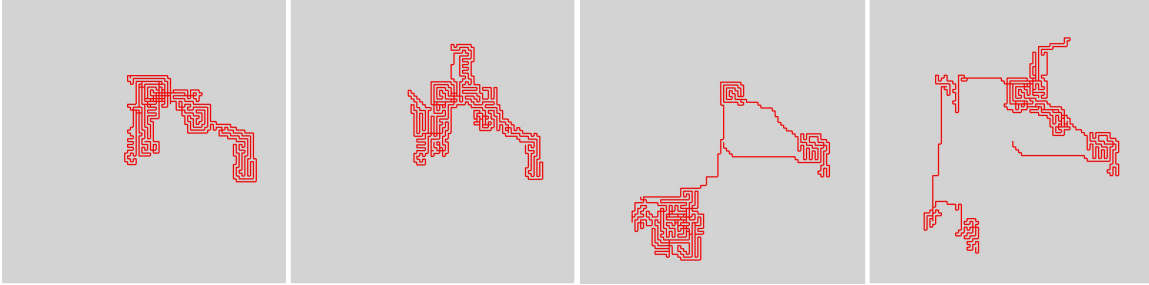


Figure 5.16: Paths generated for NewYork108 scenario with  $T = 900$  a) BA b) LHC-GW-CONV c) Top2 d) TopN

Table 5.3: Algorithms  $Efficiency_{LB}$  and running speed comparison for the NewYork108 scenario. All numbers shown are averages of 10 runs. All  $Efficiency_{LB}$  standard deviations are below 0.07.

$T$	$Efficiency_{LB}$ (%)			Speed (seconds)		
	300	600	900	300	600	900
BA	39.92	45.34	49.39	-	-	-
LHC-GW-CONV	41.38	52.88	52.61	<b>0.01</b>	<b>0.01</b>	<b>0.02</b>
Top2 (1 layer)	58.37	54.18	57.33	0.98	0.90	1.44
TopN (1 layer)	54.03	53.91	57.91	0.92	0.83	0.97
Top2 (Hierarchy)	<b>60.73</b>	55.91	57.94	2.42	2.52	2.50
TopN (Hierarchy)	59.60	<b>60.26</b>	<b>60.99</b>	6.81	6.59	7.42

the search area around the starting point before moving on to the upper right subregion of the search area to cover the probability ridge. The Top2 and TopN algorithms, however, did not waste any time at the center subregion and immediately directed the UAV to cover the probability ridge at the upper right subregion of the search area. Both of them also sent the UAV to another subregion at the lower left part of the search area where a good amount of probability can be collected. Table 5.3 shows the performance of the four algorithms and also the Top2 and TopN algorithms with specific parameters ( $k = 5$  and  $N = 3$ ). The results show the same trend as with the previous two scenarios where the Top2 and TopN algorithms outperform the BA and LHC-GW-CONV algorithms significantly in  $Efficiency_{LB}$ . Even with arbitrary parameters, the Top2 and TopN algorithms generated more efficient paths (e.g., 58.37% for Top2 with one layer vs. BA with 39.92%). In this scenario, the Top2 algorithm performed slightly better than the TopN algorithm in  $Efficiency_{LB}$  at  $T = 300$  (60.73% for Top2 and 59.60% for TopN), but the TopN algorithm performed much better than the Top2 algorithm for the other two cases. When looking at the algorithm completion time, LHC-GW-CONV algorithm is still the clear winner in this scenario. When arbitrary parameters are used, the Top2 and TopN algorithms both completed in about 1 second, but when searching through the hierarchy, both algorithms took much longer (about 2.5 seconds for Top2 and 7 seconds for TopN) to complete.

Note that the performance metric  $Efficiency_{LB}$  is computed using Equation 5.9, which assumes that the UAV can teleport within the search area. Because the amount of probability accumulated following this teleporting path can be much better than the optimal path, the true search efficiency is likely much better than the value of the  $Efficiency_{LB}$ . Fig.5.17 shows the comparison of the algorithms performance with respect to CDP collected over time for the NewYork53 scenario when  $T = 900$ . The dotted red line represents CDP accumulated over time if the UAV could teleport from vertex to vertex. Therefore this line represents the theoretical CDP upperbound. If we know the optimal path and can plot the performance, that line would most likely be somewhere below the teleport path line. The TopN algorithm

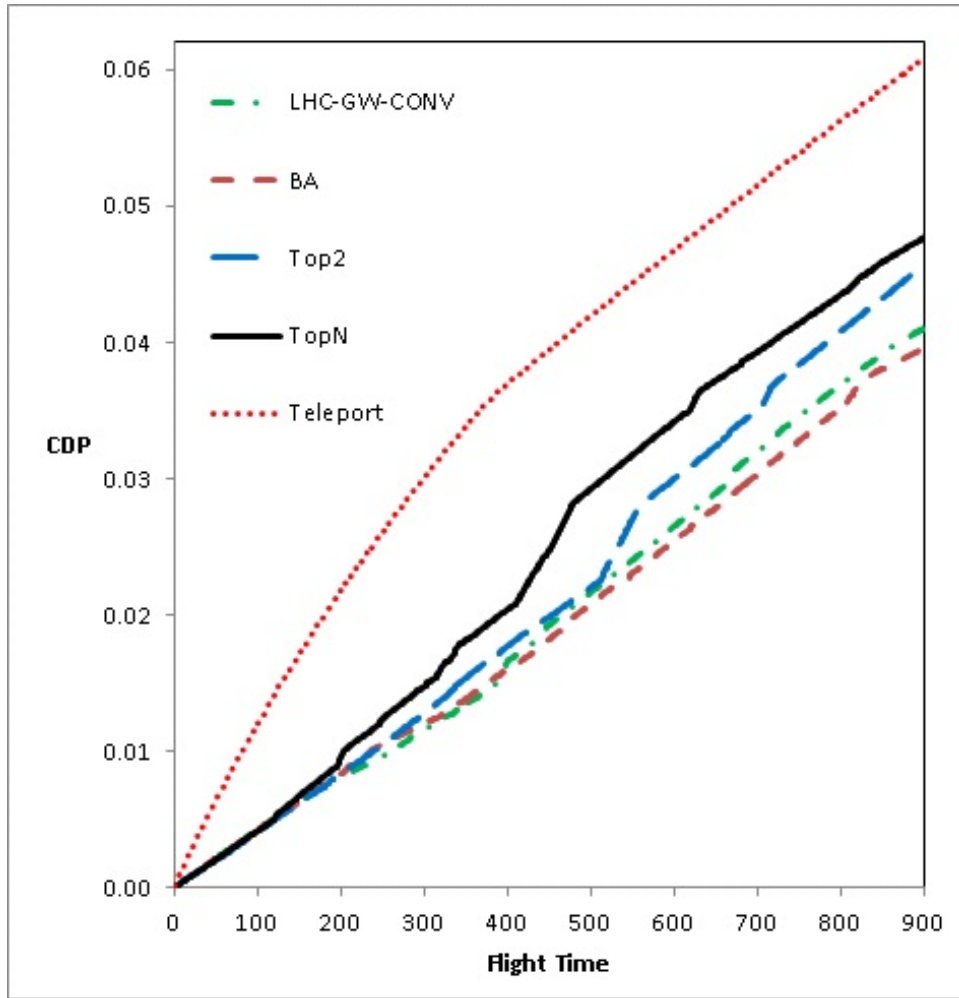


Figure 5.17: Paths CDPs comparison at  $T=900$  with partial detection.

(black solid line) performed the best (highest CDP value at time step 900), and the Top2 algorithm (blue dash line) ranked second. Both the Top2 and TopN algorithms outperformed the BA and LHC-GW-CONV algorithms (the bottom two dashed lines) significantly.

Across experiments, results show that by taking advantage of the Mode Goodness Ratio heuristic, the Top2 and TopN algorithms (even when arbitrary parameters,  $k = 5$  Gaussians to fit and  $N = 3$  for top 3 subregions, are used) always generated more efficient paths than those generated by the BA and LHC-GW-CONV algorithms. When hierarchical search is performed, the improvement from Top2 and TopN algorithms is significant. In most cases, the TopN algorithm outperformed the Top2 algorithm. However, when hierarchical search is performed, the Top2 and TopN algorithms did take a little longer to complete.

## 5.6 Limitations and Discussion

In our problem formulation, we treat the missing person as stationary because the speed of the missing person in wilderness is relatively low when compared with the speed the UAV travels in. False detection is not an issue because the UAV can simply follow the path generated to continuously collect detection probability while human operators verify the accuracy of the detection. For other application domains where the target might be moving or the probability distribution might be changing during search, by setting  $T$  to a small value, we can easily adapt the Top2 and TopN algorithms to handle these situations. The two algorithms effectively turn into greedy (a  $T$ -step look ahead approach compared to the one-step look ahead method in [11]) algorithms with flexible time horizons and scalability. We leave the evaluation of the two algorithms in such scenarios to future work.

In Equation 5.4, we assume that each observation at vertex  $v_i$  is conditionally independent of each other. This assumption certainly has its limitation. If the environment features remain the same (e.g., lighting conditions, vegetation density) and the sensor platform (e.g., camera) has stable performance, then a high probability of no detection on the first visit might indicate high probability of no detection on future visits. However, in practical applications, a sensor operator's ability to recognize the missing person from video footprint is affected by many factors such as his fatigue level and his cognitive workload [41], especially when the sensor operator might also be in charge of flying the UAV. In this case, the operator's chance performance can be regarded as independent trials (as in successive coin tosses).

The detection model used in our experiments is a simple decay model only parameterized by a difficulty factor. In SAR (Search and Rescue) literature, the parameters of the decay factor could be affected by environment features and sensor properties (e.g., distance to radar, signal strength [11]). Also we only consider vegetation density when we constructed the task-difficulty maps. Because we use a camera sensor and keep the UAV flying at the same height above ground, we believe this model is sufficient to show the algorithms' capability in handling partial detection, and we intentionally kept the sensor model and environment



model simple for demonstration purposes. However, a more complicated sensor model and environment model can easily be applied to the system.

Although GMM is a statistically mature method for clustering, it has several limitations. First, convergence is not guaranteed for the iterative EM algorithm used to estimate the Gaussian mixture. In our implementation, we re-run GMM multiple times if convergence is not achieved to overcome the problem. Second, how many Gaussians should we fit? There is the possibility that the Gaussians might not fit the data very well. We arbitrarily set the maximum Gaussians to 5 to reduce computational complexity. Experiment results show that we were still able to generate good paths. Since the Mode Goodness Ratio is only a heuristic, as long as it provides useful information to our search most of the time, it serves its purpose.

Another limitation is that the algorithms do not handle tough terrains where the UAV might not be able to climb fast enough to fly over the terrain. Future work should explore how to modify the algorithms to consider such constraints and actual flight dynamics.

When defining the goodness of a subregion,  $MG_i$ , we considered three factors: distance ratio, probability volume, and subregion area. The last two factors, when combined, give us a sense of the peakedness of a probability hill. Then we multiply the peakedness with the distance ratio in order to compare the Mode Goodness of subregions. Here we assume the two measures are independent of each other, which is a limitation of the heuristic. It also creates a trade off problem. For example, when subregion A's distance ratio is half of that of subregion B but A's peakedness is twice in size compared to B. A and B would still have identical  $MG_i$  values. Should they be? We leave this to future work.

Going back to the synthetic scenario we presented in section 5.4.1, Table 5.4 shows the performance of the BA, LHC-GW-CONV, Top2, and TopN algorithms in two different scenarios: starting from a subregion with low task-difficulty (upper left) and starting from a subregion with high task-difficulty (lower right). When starting from a high task-difficulty area, the BA and LHC-GW-CONV algorithms tend to get stuck in a local probability hill, while the Top2 and TopN algorithms force the UAV to visit other subregions, therefore

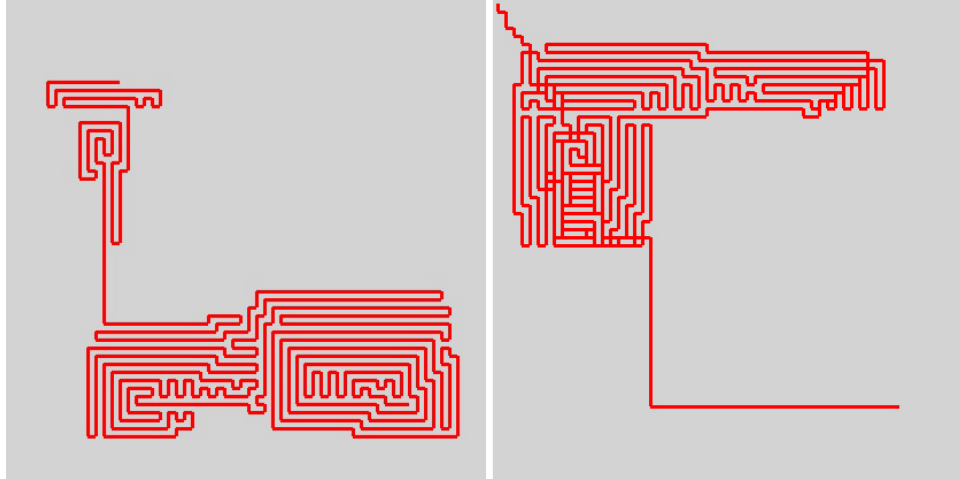


Figure 5.18: Paths found for the synthetic scenario at  $T = 900$  when the UAV starts from a subregion with high task-difficulty (lower right corner). Left: Path created by LHC-GW-CONV without specifying end point. Right: Path created by LHC-GW-CONV with specified end point at upper left corner.

achieving better paths with significant improvement. One interesting observation we noticed is that if an ending position is specified for the desired UAV path and the ending point is in a subregion with low task-difficulty, the LHC-GW-CONV algorithm also forces the UAV to visit other subregions, and by doing so, improve the efficiency of the path. Fig.5.18 shows an example where the path on the right achieved 93.60% in  $Efficiency_{LB}$  (computed within 0.01 second), which is slightly better than the Top2 algorithm but not as good as the TopN algorithm. Another thing to note with this scenario is that the LHC-GW-CONV algorithm actually did slightly better than the Top2 and TopN algorithms when the UAV starts from a low task-difficulty area. We have noticed from various experiments that after combining the probability distribution map and the task-difficulty map, if the resulting surface is not a complicated one (meaning it only has a few distinctive probability hills), the LHC-GW-CONV algorithm generally performs well. For more complicated surfaces (such as the three real WiSAR scenarios we tested the algorithms with), the Top2 and TopN algorithms are more reliable in generating good UAV paths.

Table 5.4: Algorithms  $Efficiency_{LB}$  comparison for the multi-modal synthetic scenario at  $T = 900$

(%)	BA	LHC-GW-CONV	Top2	TopN
Start from upper left	65.99	<b>96.28</b>	94.96	95.82
Start from lower right	41.91	53.71	93.46	<b>95.29</b>

In the current implementation, we used a grid representation for the probability distribution map, task-difficulty map, and the path generated. However, the algorithms also support other tessellation methods such as a hexagonal tessellation.

## 5.7 Summary

We proposed a new heuristic, the Mode Goodness Ratio, which uses Gaussian Mixture Model to prioritize search subregions, and presented two new algorithms that utilize the heuristic in hierarchical path planning. The hierarchical structure enables searching for better paths through the parameter space at different scales and enables us to parallelize the two algorithms for better performance. The probability of detecting the desired subject based on UAV sensor information can vary in different search areas due to factors such as varying vegetation density or lighting conditions. We represented this type of partial detection in the form of a task-difficulty map, a spatial representation of sensor detection probability, and incorporate it into UAV path planning. We compared the performance of the new algorithms against two published algorithms BA and LHC-GW-CONV in simulated searches with three real search and rescue scenarios. Experiment results showed that by using the Mode Goodness Ratio heuristic, the two new algorithms Top2 and TopN consistently outperform the BA and LHC-GW-CONV algorithms, yielding efficient paths that produce payoff approximating the payoff of the optimal path.

## Chapter 6

### **Paper: Sliding Autonomy for UAV Path Planning: Adding New Dimensions to Autonomy Management<sup>1</sup>**

#### **Abstract**

Increased use of autonomy also increases human-autonomy interaction and the need for humans to manage autonomy. We propose a new variation of the concept of sliding autonomy that is useful for planning problems over a spatial region. In this sliding autonomy approach, the user can influence the behavior of the autonomous system via three categories of input: information, spatial constraints, and temporal constraints. We present a set of user interface designs to implement sliding autonomy for UAV (Unmanned Aerial Vehicle) path planning to support Wilderness Search and Rescue (WiSAR). Interactivities along these new dimensions allow the user to allocate degrees of authority and flexibility to the robot's algorithms. We analyze how this approach fits in the integration challenge guidelines we identified in our prior work and evaluate the usefulness of the approach against manual and simple pattern path planning methods with a user study. Results show that the sliding autonomy approach performs significantly better than the other two methods without increasing the users' mental workload, and the performance of the human-autonomy team outperforms either human or autonomy working alone. We also discuss some interesting observations from the user study.

---

<sup>1</sup>To be submitted to JHRI (Journal of Human-Robot Interaction) journal. Authors are Lanny Lin, Michael A. Goodrich, and Spencer Clark.

## 6.1 Introduction

With the rapid advancement in technology, people are seeing increased use of autonomy to augment human abilities and support human decision-making in many application domains (e.g., [17, 20, 71, 96]). At the same time, increased use of autonomy also means increased human-autonomy interaction and increased need for humans to manage autonomy [3]. Even for so-called fully autonomous systems, human input can potentially improve the system's performance and safety [15]. The humans in such interactions manage autonomy because "only people are held responsible for consequences (that is, only people can act as problem holders) and only people decide on how authority is delegated to automata" [132].

When humans manage autonomous systems, their managerial responsibilities often include monitoring the safety of the autonomous system, supervising autonomy to achieve acceptable performance, and making sure autonomy is working toward the collective goal of the overall system. In many emerging domains, the human operators are domain experts who can use domain-specific knowledge to assist the autonomous system when it deals with changing environments, uncertainty, and case-specific scenarios. Therefore, it is necessary to design tools and interfaces that enable human users to manage the autonomous behaviors of the system efficiently and effectively; such tools can improve task performance and the experience of the human operator in human-autonomy interaction. Wilderness Search and Rescue (WiSAR) is one such domain that could benefit from autonomy management tools when a mini-UAV (Unmanned Aerial Vehicle) is used in search.

Camera-equipped mini-UAVs can be useful tools in WiSAR operations by providing aerial imagery of a search area with the benefits of quick coverage of large areas, access to hard-to-reach areas, and lower cost than manned aircraft [41, 83]. In fact Canadian mounties claim that they have successfully saved a person with a police drone in a recent rescue mission<sup>2</sup>. UAV path planning is an important task because a good flight path can increase

---

<sup>2</sup><http://www.theverge.com/2013/5/10/4318770/canada-draganflyer-drone-claims-first-life-saved-search-rescue>

the probability of finding a missing person by making efficient use of the limited flying time. Various algorithms have been developed to support UAV path planning autonomy (e.g., [9, 69, 72]), but the question remains how best to incorporate searcher expertise in such a way that the UAV path planning is as efficient and effective as possible. The key constraint that we impose on this question is that we want to do this without requiring the searcher to understand how the autonomy works “behind the scene.”

We propose a new autonomy management approach where the user can influence the behavior of an autonomous system along three new dimensions: 1) **Information Representation**: information used by the robot is presented to the human in a human-readable form, and the human directly modifies this information to effect change in robot behavior; 2) **Spatial Constraints**: a human can add constraints or priorities to different spatial regions, thereby affecting how the robot plans and performs its task; and 3) **Temporal Constraints**: A human can impose time limits for a subtask or impose ordering constraints on a subtask. We refer to this approach as *Sliding Autonomy* because, properly designed, it can allocate degrees of authority and flexibility to the robot’s algorithms by adding or removing constraints, or by shaping input information. Indeed, we will explicitly use a **slider** as one GUI tool for managing UAV path planning.

As the human modifies information, adds priorities, or changes constraints, the sliding autonomy tool shows immediately how those changes influence the UAV’s plan. This instant feedback provides the searcher the ability to perform “what-if” analysis and see the causal effect between his/her action and changes in autonomous behavior. This allows an interactive approach where autonomous algorithms perform tasks that they are good at and humans do tasks that they are good at, but in a collaborative and interactive way that avoids the pitfalls of simple task allocation [15, 105]. Properly done, the human-robot team should perform better than a human or robot working alone.

Many approaches to autonomy management already exist and are called many different things, such as *supervisory control* [105], *mixed-initiative* [47], *collaborative control* [34],

*adjustable autonomy* [26, 27] (also referred to as *sliding autonomy* [25] or *adaptive automation* [57, 97]). The approach we propose falls under the category of *adjustable autonomy*. The three dimensions we identified are in addition to dimensions of *adjustable autonomy* identified by Bradshaw et al. so we design tools and algorithms that operate in a particular place in Bradshaw's taxonomy [14].

In our previous work [71] we identified key elements of autonomy integration challenges along two dimensions: *attributes of an intelligent system* (capability, information management, performance evaluation) and *organizational scale* (individual versus group), which can serve as guidelines in designing autonomous components and autonomy management tools. In this paper we extend the guidelines to include attributes needed when a human and autonomy work collaboratively and analyze how our proposed sliding autonomy approach fits in the guidelines. By applying sliding autonomy to the UAV path planning task, we argue that this approach:

- enables the domain expert user to incorporate information only available to or understandable by the expert;
- is easy to understand without knowing how autonomy works behind the scene;
- lets the human do what the human is good at (planning strategically and balancing performance tradeoffs) and autonomy do what autonomy is good at (planning tactically), resulting in better performance than human or autonomy working alone;
- enables the user to align the task goal (find the path that maximizes probability collected along the path) with the system goal (finding the missing person quickly) when the user has more information or more up-to-date information than autonomy; and
- improves human's experience during the human-autonomy interaction.

To evaluate the usefulness of the proposed approach, we performed a user study and compared the sliding autonomy method against two other planning methods (manual and simple pattern path planning) in two WiSAR scenarios (a synthetic scenario and a

real scenario). We measured each user's performance with each method and also the user's performance on a secondary task (answer questions in a group chat window). Experiment results show that the sliding autonomy method performed significantly better than the manual or simple pattern planning methods with no increased mental workload. The human-autonomy team also performed better than the human or autonomy working alone.

In Section 6.2 we explain how the proposed approach fits into the extended autonomy design guidelines and describe how a user can manage autonomy along each of the three new dimensions in the context of UAV path planning. Section 6.3 covers related work in literature. Section 6.4 lists our hypotheses followed by user study design in Section 6.5. Then we present experiment results in Section 6.6 and discuss our observations in Section 6.7. In Section 6.8 we conclude the paper and list possible future work.

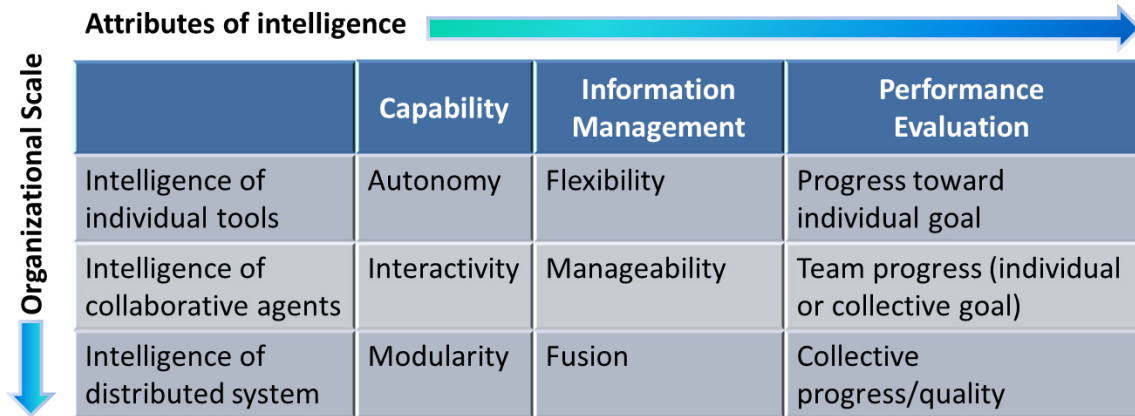
## 6.2 Autonomy Design Guidelines and New Dimensions

### 6.2.1 Autonomy Design Guidelines

In our previous work [71] we organized the challenges of autonomy and management tool design along two dimensions: *attributes of an intelligent system* (capability, information management, performance evaluation) and *organizational scale* (individual versus group), which can serve as guidelines in designing autonomous components and autonomy management tools. In our definition we treat a system of human(s) and algorithms working together as an intelligent system. In this paper we extend this table by adding a row in the middle describing what attributes are needed when multiple agents work collaboratively (see Figure 6.1). A human-autonomy team working on the same task falls within this category.

As an individual tool, each autonomous component needs to be able to perform a task (**Autonomy**); the operator can match the component's capability to a specific task according to the information available to the operator, which requires that autonomy can be





Organizational Scale	Attributes of intelligence		
	Capability	Information Management	Performance Evaluation
Intelligence of individual tools	Autonomy	Flexibility	Progress toward individual goal
Intelligence of collaborative agents	Interactivity	Manageability	Team progress (individual or collective goal)
Intelligence of distributed system	Modularity	Fusion	Collective progress/quality

Figure 6.1: Autonomy integration challenges defined along two dimensions. Horizontal dimension: attributes of intelligence. Vertical dimension: organizational scale.

interrupted, paused, aborted, and resumed (**Flexibility**); the performance is evaluated by how well the component accomplishes the task goal in the absence of human input.

When a human-autonomy team works on the same task collaboratively, the autonomous component needs to provide interfaces so the human can interactively influence the autonomous behavior (**Interactivity**); the human should be able to manage how autonomy works in order to jointly find a solution by utilizing information only available to the human and/or feed information to autonomy in a representation that the autonomy can understand (**Manageability**); and when performance is evaluated, the human operator can judge whether the individual goal aligns with the collective goal of the system.

As part of a larger distributed system, each component and collaborative subsystem needs to be modular (**Modularity**), so they can be mixed and matched to support different user roles; information from various sources need to be combined and presented to one or multiple users (**Fusion**); and performance of the complete human-machine system needs to be evaluated as a whole.

This paper focuses on the middle row of the guidelines: intelligence of collaborative agents (human-autonomy team). The three dimensions we propose are ways path planning autonomy can be managed, and our path planner interface is designed to accept human input along the three dimensions to provide interactivity. The human can also incorporate

information from various sources and influence the behavior of path planning autonomy by allocating degrees of authority and flexibility, making sure the task goal aligns with the ultimate goal of finding the missing person quickly.

### 6.2.2 Information Representation Dimension

The right information representation is task specific. Many path planning algorithms use a *probability distribution map* that encodes the likely location of a target; for WiSAR, instead of a target we are interested in the probable location of the missing person, so the path planning algorithm should be able to account for this distribution. Adopting a Bayesian perspective, this distribution represents the prior probability of the location of the missing person.

Completing the Bayesian approach requires a likelihood that encodes the probability of seeing the missing person given that the person is located in a particular location. We represent the likelihood using what we call a *task-difficulty map*, which is a spatial representation showing (one minus) the sensor detection probability in different parts of the search region. For example, the prior probability of the missing person being near the last known position (LKP) is normally high (high prior probability); and the probability of detecting the missing person in a dense vegetation area using an airborne camera is normally low (high task difficulty). The objective of path planning is to find a path that maximize the cumulative posterior detection probability of the missing person given a fixed flying time.

Since WiSAR experts use maps and probabilities in much of their work, we argue that presenting information about prior probabilities and likelihoods in a map form to the human allows the human to influence path planning autonomy by “shaping” these two maps. Prior work has demonstrated that these two maps can be systematically generated based on terrain features and vegetation data [70, 72]. However, the searcher likely wants to include his/her domain expertise (past experience, knowledge of the search region, etc.) and additional information (maybe new evidence found during the search) in the planning. These types of

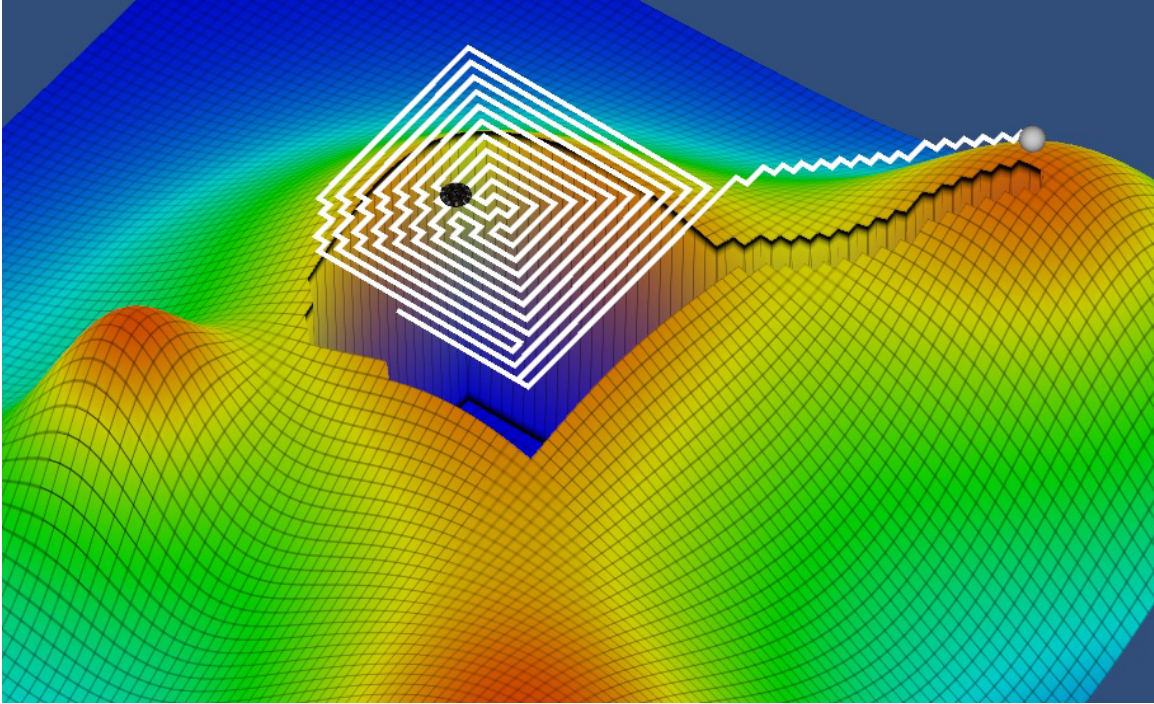


Figure 6.2: A screen capture of the sliding autonomy tool showing a 20-minute path segment. The 3D surface shows the probability distribution map. The UAV icon in the middle indicates the start point of the path segment and the sphere on the right indicates the end point.

information are not directly understandable by autonomous algorithms, but the searcher can incorporate them into the probability distribution map and the task-difficulty map using map editing tools<sup>3</sup> and thereby influence the behavior of path planning autonomy.

Marking an area with high probability, the searcher indirectly tells the UAV to treat the area with high priority; marking an area with high task-difficulty, the UAV might make multiple passes over the area to search more thoroughly. The 3D surface in Figure 6.2 shows an example probability distribution map where hills (red) indicate high probability and the flat area (blue) indicates low probability.

We hypothesize that by allowing a human to directly manage this type of information, the human can quickly figure out how his or her actions will affect the behavior of autonomy, even though he/she has no idea about how autonomy works behind the scene. We have designed a user interface that allows a human to do this and have conducted subjective

<sup>3</sup>Such as these tools at <http://tech.lannyland.com/demos.html>.

evaluations of the interface, but statistical validation of this hypothesis using a careful user study is left to future work.

### 6.2.3 Spatial Constraints Dimension

The searcher should also be able to influence the behavior of path planning autonomy by setting/changing spatial constraints. Spatial constraints can be in the forms of start/end points of the path segment or task-specific zones.

Setting an end point in an area is a way for the searcher to indirectly tell path planning autonomy that the area should have higher priority than other areas. For example, if a piece of clothing is found by the ground team, the searcher can force the path planning autonomy to go visit that area by setting an end point there. Because the UAV must allocate part of the fix-length flight time to reach this specified area, some areas that had good payoffs before this constraint is set can become relatively costly and, therefore, no longer attractive to path planning autonomy. Importantly, since we have assumed a fixed flight duration, setting an endpoint not only directly causes the UAV to focus search effort around that location but also indirectly causes the UAV to avoid other areas because the budget does not allow them to be searched well. In Figure 2.1, the UAV icon in the middle indicates the start point of the path segment and the sphere on the right side indicates the desired end point.

In the GUI, the end point can be dragged around the search region and path planning autonomy suggests different paths accordingly. This capability enables the user to adjust how much freedom is granted to autonomy. When the end point is close to the start point, autonomy has greater authority and flexibility in creating paths. If the end point is far from the start point, authority and flexibility for autonomy is reduced because a major part of path planning is simply moving the UAV toward the end point with the shortest path.

A task-specific zone can be a *no-fly zone*, a *coverage zone*, or a *sampling zone*. A no-fly zone is a pretty straight-forward way to restrict the UAV from visiting certain areas [21, 56]. The decision might be for safety reasons or part of the searcher's strategic planning depending

on resource allocation. A coverage zone requires the UAV to fully cover the area [69]; a sampling zone only asks the UAV to collect a few sensor samples from the zone, so the visit can be very brief [21]. A task-specific zone can be dragged around and the searcher can also change the shape and/or size of the zone to influence the path generated by autonomy.

The interactive ability to move the end point around (or changing the shape of the no-fly zone) and see immediately how the change would affect the UAV path recommended by path planning autonomy gives the user the power to perform “what-if” analysis. It also allows the user to see the causal effect between his/her action and changes in autonomous behavior.

Spatial constraints are easy to understand, so the searcher knows how these constraints will affect the behavior of path planning autonomy. By managing autonomy along this dimension, the searcher has another way to incorporate additional information to the path planning task, improve task performance, and align the task goal with the overall goal of finding the missing person quickly.

In our user study we fixed the start point of the path to the center of the map because that was the last known position of the missing person. The searcher can set the end point for the current path segment anywhere on the map, and this end point automatically becomes the start point for the next path segment. We disabled the ability to move an end point once a path segment is planned to reduce computation, but we let users reset an entire plan, effectually allowing them to try different combinations of starting and ending path segments. Task-specific, sampling, and explicit no-fly zones were evaluated in a separate user study [21].

#### 6.2.4 Temporal Constraints Dimension

In the UAV path planning problem, temporal constraints include a *time limit* for a subtask (path segment), *subtask ordering*, and *valid time window*.

With the time limit constraint, the searcher can decide how much flight time to allocate to a path segment out of the total flight time. This enables the searcher to break the

path planning task into multiple subtasks and then plan each path segment separately. In our interface design we let the searcher control time allocation to autonomy using a slider, and as the searcher moves the slider, the path planning autonomy shows how the suggested path segment changes respectively. Similar to the spatial constraints, this instant feedback enables “what-if” analysis and provides instant feedback on the causal effect between searcher action and changes in autonomous behavior.

For example, for a 60-minute total flight with an end point set to the probability hill on the right (Figure 6.2), the searcher can move the slider to set time limits and see immediately what path segment the autonomy would suggest. The path segment shown is when 20 minutes are allocated out of a total flight time of 60 minutes. If the searcher is happy with the suggestion, he or she approves the path segment. The UAV moves to the end point in the path planner and “vacuums up” the probability along the path (how much can be vacuumed up is determined by the task-difficulty map). Then the searcher works with the autonomy to plan the path for the remaining 40 minutes. The two (or more) path segments are joined to form the final path.

A subtask ordering constraint adds temporal dependency to the subtasks (e.g., the sampling task must be flown before the coverage task). This type of constraint lets the searcher directly specify priorities in different search areas.

A valid time window constraint specifies a time interval during which a subtask must be completed. This is less restrictive than giving a time limit constraint because the specified task can be accomplished at anytime during the window, and more restrictive than an ordering constraint because the task must be accomplished before a deadline and after a start time.

By managing autonomy along the temporal constraint dimension, the searcher can break the path planning task into subtasks and incorporate additional information into the path planning task. The user study described in this paper includes the time limit constraint. Subtask ordering and valid time window constraints are evaluated in a separate user study [21]).

In the example shown in Figure 6.2, the combination of the information representation, end point constraint (spatial), and time limit constraint (temporal) allows the searcher to cover the middle area pretty well and then move to the search area on the right to search there. If no end point constraint is set, autonomy might decide to search the probability hill on the left with a low time limit or the hill at the bottom with a high time limit. If more flight time is allocated with the set end point, the autonomy might start searching the area on the right. Less time allocation reduces the authority and flexibility of autonomy, and forces autonomy to focus more on the local area; more time allocation increase authority and flexibility, so autonomy has more freedom on deciding what areas to cover. Instant feedback on path changes when the search moves the endpoint or varies the time limit lets the searcher interactively review multiple options and select the path segment that fits best with his/her strategic planning. This design enables human to plan more strategically (prioritizing areas in the entire search region) while autonomy works more tactically (covering the current search area well), using strengths of each when they work collaboratively. Ideally such a human-autonomy team should work better than either human or autonomy working alone.

### 6.3 Related Work

Many approaches on how human-autonomy team can work together have been proposed in the literature. Drucker defines automation as a “concept of the organization of work [28].” Goodrich and Schultz define the HRI problem as “understanding and shaping the interactions between one or more humans and one or more robots” [39]. They also specified robot-assisted search and rescue as a key area for HRI research. In their 1978 seminal paper, Sheridan and Verplank propose the idea of a *level of autonomy* spectrum, with full teleoperation at one end and full autonomy at the other [106]. In the middle, the robot could suggest actions to humans or make decisions before informing humans. Parasuraman et al. extended this one-dimensional spectrum to four different broad functions: information acquisition, analysis, decision selection, and action implementation [90]. Sheridan proposes *supervisory control*,

in which a human divides the task into a sequence of subtasks that the robot is capable of performing, and the human then provides guidance when the autonomous system cannot solve a problem on its own [105]. In contrast to the top-down philosophy of supervisory control, a *mixed-initiative* approach advocates the idea of dynamically shifting tasks when necessary [47]. *Collaborative control*, which can be thought of as an instance of mixed-initiative interaction, is a robot-centric model; instead of the human always being in-charge, the robot is treated as a peer and can make requests to humans through dialogs [34]. *Adjustable autonomy* [26] (also referred to as *sliding autonomy* [25] or *adaptive automation* [97]) is another type of mixed-initiative interaction, one that enables the human-automation team to dynamically and adaptively allocate functions and tasks among team members.

Many implementations of different flavors of adjustable autonomy exist. Dorais et al. discuss a framework for human-centered autonomous systems for a manned Mars mission [27]. The system enables users to interact with these systems at an appropriate level of control but minimize the necessity for such interaction. Bradshaw et al. discuss principles and pitfalls of adjustable autonomy and human-centered teamwork, and then present study results on so-called “work practice modeling” and human-agent collaboration in space applications [13]. Kaber et al. describe an experiment simulating an air traffic control task where manual control was compared to Adaptive Automation (AA) [58]. Results suggest that humans perform better with AA applied to sensory and psychomotor information-processing functions than with AA applied to cognitive functions; these results also suggest that AA is superior to completely manual control. Brookshire et al. present preliminary results for applying sliding autonomy to a team of robots performing coordinated assembling work to help the system recover from unexpected errors and to thereby increase system efficiency [16]. Dias et al. identified six key capabilities that are essential for overcoming challenges in enabling sliding autonomy in peer-to-peer human-robot teams [25]. Bradshaw et al. propose two dimensions of Adjustable Autonomy (descriptive and prescriptive) to address the two senses of autonomy (self-sufficiency and self-directedness) and discuss how permissions, obligations, possibilities,



and capabilities can be adjusted [14]. Bradshaw et al. also summarized some widespread misconceptions on autonomy and listed seven deadly myths of “autonomous systems” [15].

The human is an integral part of the human-autonomy team. When working with autonomy, the human often takes on the supervisor role. Bainbridge points out that automation requires the human operator to take additional management responsibilities [3], and Sartar identified two automation management policies: *management by consent* and *management by exception*, defining whether the human always retain authority or can the system take initiative [102]. For complex automation, the human tends to rely on his/her *mental models* [86] to manage the system.

Searchers working together with a UAV is an example of a human-autonomy team. UAV technology has emerged as a promising tool in supporting WiSAR [9, 83]. The goal of our research is to support fielded missions in the spirit of Murphy’s work [17]. Many path planning algorithms in the literature address obstacle avoidance while planning a path to reach a destination using A\* [92], LRTA\* [53], D\* [114], Voroni diagrams [5, 8], or probability roadmaps and rapidly-exploring random tree (RRTs) [91]. Bourgault et al. [10, 11] describe how to use a Bayesian model to create paths for a single UAV or multiple coordinated UAVs to maximize the amount of probability accumulated by the UAV sensors. The algorithms we used in this paper are algorithms designed from our previous work [69, 72] using techniques such as global warming technique, convolution, Gaussian mixture models, and Evolutionary Algorithm.

## 6.4 Hypotheses

We performed a user study to evaluate the usefulness of the sliding autonomy approach. More specifically we verify the following hypotheses:

H1: The sliding autonomy method performs better than either a manual path planning method and a semi-autonomous path planning method that uses standard search patterns to cover an area.

H2: The sliding autonomy method performs better than autonomy working alone.

H3: The sliding autonomy method does not increase the mental workload of the operator when compared against the manual and pattern methods.

## 6.5 User Study Design

We performed a  $2 \times 3$  within-subject design with 2 scenarios (easy vs. difficult) and 3 planning methods (manual, pattern, and sliding autonomy). All participants completed all 6 exercises. The order of the scenarios and planning methods was counterbalanced to reduce learning effect. We recruited a total of 26 college students (14 males and 12 females) between the age of 19 and 30 (average 22.89).

After the demographic survey, each participant completed four 5-minute long non-skippable training sessions (one for each planning method with no task-difficulty map, and one for the manual method with a task-difficulty map) and then completed the 6 exercises. Each participant had up to 5 minutes for each exercise. Once the participant was happy with the path generated, he/she could finish the exercise early. We chose this design because we do not want the user to put all effort into completing the secondary task once he/she considers the primary task completed, which would skew the measurements on secondary task performance. At the end of each exercise, the participant completed a partial NASA TLX survey. Then at the very end of the user study, the participant filled out a survey describing his/her subjective preference with the three planning methods.

### 6.5.1 Simulation Environment

The user study was conducted in a 3D simulation environment (see Figure 6.3) where both the probability distribution map and the task-difficulty map were displayed as 3D surfaces with a color map (red means high and blue means low). The user could switch between the two maps at any time and rotate/pan/zoom a map at will. The UAV was a hexacopter capable of flying in all directions or hovering in the same spot. The UAV start

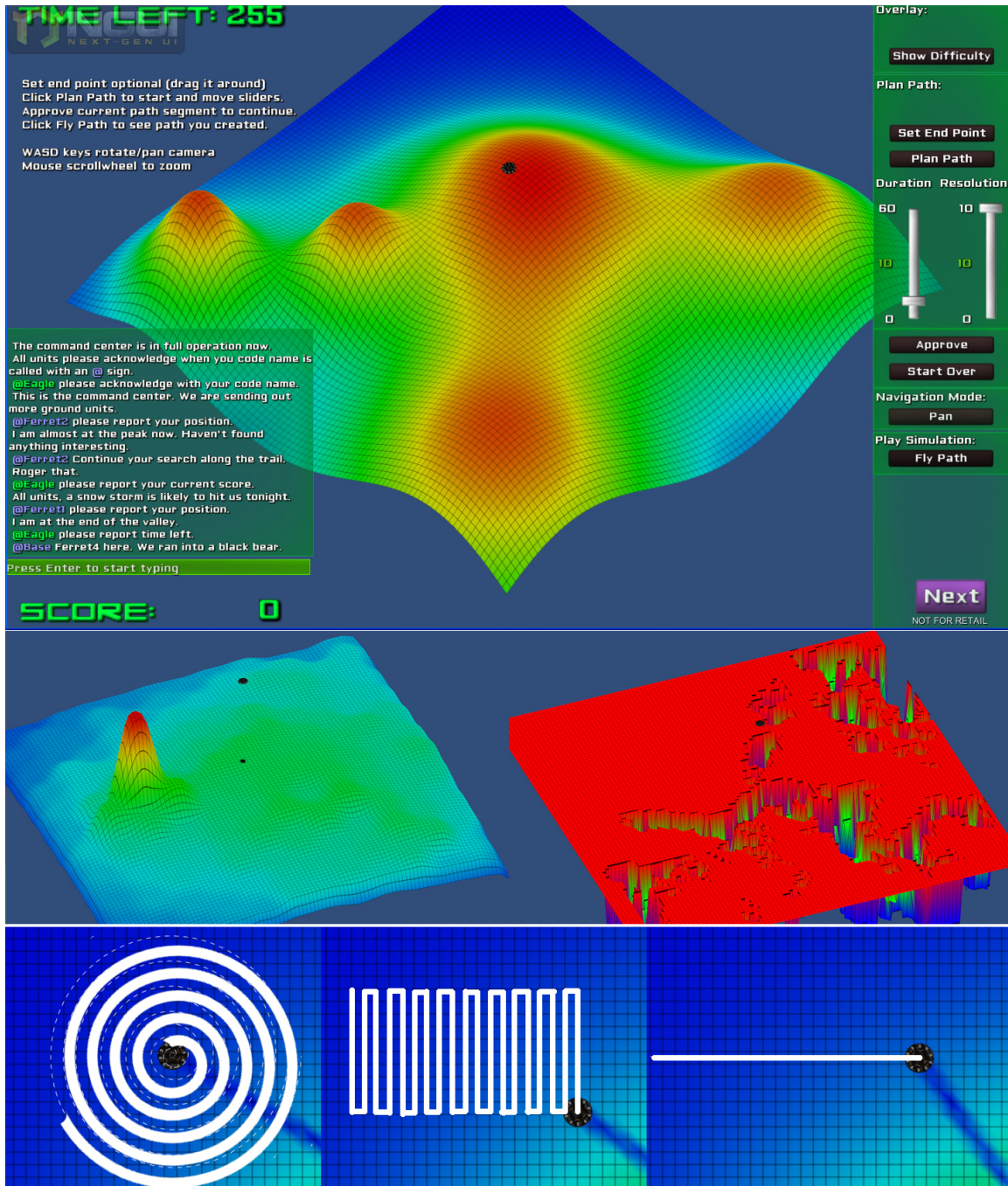


Figure 6.3: Top: User study simulation interface with the sliding autonomy method showing the probability distribution map for scenario 1. Middle left: Probability distribution map for scenario 2. Middle Right: Task-difficulty map for scenario 2. Bottom: The three patterns available to user with the pattern planning method, spiral, lawnmower, and line.

point is set at the center of the search region because that was the last point known (LPK) for the missing person.

With the **manual** planning method, the user flew the UAV with the arrow keys in a sped up fashion (i.e., enabling the user to cover ground faster than the UAV could cover it in real flight). The user could switch between two flying modes (turn and strafe) and four camera views (global, behind, bird's eye, and free form). The user could also pause/resume the flight for the secondary task or better planning.

With the **pattern** planning method, the user chose from spiral, lawnmower, and line patterns (see Figure 6.3 bottom) and joined these patterns to form the final path. The end point of the previous path segment (LPK if at the very beginning) automatically became the start point of the current selected pattern. As the user moved the cursor around, the size of the pattern changed with the cursor position marking the end point of the pattern (The start/end points pair determined the radius of the spiral pattern, the diagonal of the rectangle for the lawnmower pattern, and the start/end points for the line pattern). Once the user was happy with the location, shape, and size of the pattern, he/she could approve the pattern with a left click. The user could also undo the last path segment (pattern) planned. This planning method was “semi-autonomous” because the patterns were generated automatically without manually setting waypoints.

With the **sliding autonomy** method (see Figure 6.3 top), the user could set an end point (optional), and then drag the left slider to change the amount of time allocated to autonomy. The path suggested by the autonomy changed as the slider moved. The slider's max value always reflected the remaining flight time (in minute). If the user were happy with the current path segment, he/she could approve it, the UAV then moved to the end of the path segment, and the process repeated until a path has been planned that accounts for all of the available flight duration. The path planning algorithm used was the LHC-GW-CONV algorithm [69, 72], because it is the fastest algorithm out of all the algorithms we designed

and produces satisfactory sub-optimal performance when compared to other state-of-the-art algorithms for the problem.

With all three planning methods, the user could choose to start over at any time during the exercise, and could restart as many times as exercise time allowed. We recorded the best path out of all tries.

### 6.5.2 Scenarios

The user study contained two WiSAR scenarios, a synthetic case (see Figure 6.3 top) with no task-difficulty map (assuming uniform detection probability), and a real WiSAR scenario (see Figure 6.3 middle) with a task-difficulty map, in which an elderly couple was reported missing near the Grayson Highlands State Park in Virginia [60]<sup>4</sup>.

Scenario 2 is clearly more complex than scenario 1 because the user also had to consider the different detection probability defined by the task-difficulty map. We refer to scenario 2 as the high information scenario and scenario 1 as the low information scenario. These two scenarios exhibited significantly different amounts of workload in a pilot study and gave us confidence that the results scale to different types of scenarios.

### 6.5.3 Secondary Task

In each exercise, each participant also performed a secondary task. This provided a second measure of mental workload. In a group chat window (see Figure 6.3 top) when the user's code name appeared, the user had to type answers to simple questions. Roughly every 3 seconds a message was sent to the chat window, and every 5th message asked the user a simple question (4 per minute). For the same scenario and the same planning method, all users received the same set of chat messages.

---

<sup>4</sup>The probability distribution map used for this scenario (Figure 6.3 middle left) was generated using a Bayesian model [70]. The map has been evaluated at George Mason University's MapScore web portal [18] and performed better than most other models evaluated, scoring 0.8184 on a [-1,1] scale where the higher the score the better. <http://sarbayes.org/projects/>. The task-difficulty map (Figure 6.3 middle right) was generated using vegetation density data downloaded from the USGS web site and categorized into three difficulty levels (sparse, medium, and dense, with detection probability of 100%, 66.67%, and 33.33% respectively).

We chose to use a group chat window as the secondary task because this is typical in WiSAR operations. We also designed the chat messages to simulate a real WiSAR search and improve ecological validity. The user was asked to acknowledge connection and report path planning status periodically.

#### 6.5.4 Measures

We used the following five measurements for the primary path planning task:

- **Percent score:** In each exercise, an exercise score was computed by summing the amount of probability collected by the UAV if it followed the path planned. The user's best score for each exercise (out of multiple tries) was normalized by dividing the best score from all users for the same scenario to compute the percent score. This way we could compare planning methods across scenarios.
- **Time spent:** How much time was spent with each exercise.
- **Try count:** How many times the user tried in each exercise. Note that because the manual planning method takes much longer to plan a path than the other two methods by design, this measurement is used mainly to compare between the pattern and sliding autonomy planning methods.
- **Mouse clicks per try:** How many times the user left-clicked the mouse within a try. Again, this measurement is used to compare pattern and sliding autonomy planning methods because the manual planning method does not require a lot of mouse clicks by design.
- **NASA-TLX raw score:** The sum of user subjective evaluation of cognitive workload in six dimensions normalized to a 100-point scale.

The following two measurements were used for the secondary task:

- **Percent of questions missed:** What percentage of questions directed to the user were missed before the user completed the exercise. Here we did not measure the

percent of questions answered correctly because all the questions are very simple and all users answered the questions correctly.

- **Chat latency:** The number of seconds between the time a question was presented to the user and the time when the user answered the question.

## 6.6 Results and Analysis

We analyzed the user study data with a mixed measures analysis of variance (ANOVA) and report results in this section.

### 6.6.1 Comparing Across Scenarios

Mouse clicks per try for the two scenarios are significantly different ( $F[1, 25] = 28.65, p < .0001$ ) indicating scenario 2 required participants to be more active than in scenario 1. This result supports observations in the pilot study that scenario 2 imposed higher workload on participants than scenario 1. Evaluating logs of user activity indicates that participants created more path segments (for pattern and sliding autonomy planning methods) in scenario 2 than scenario 1.

NASA TLX scores are also significantly different ( $F[1, 25] = 31.35, p < .0001$ ) between the two scenarios. The average score difference is 9.98 (out of a total of 100 points), almost a full “pip” on the TLX survey, indicating that on average each participant felt his/her cognitive workload was much higher in the high information scenario.

The percent of questions missed is almost identical between scenarios (54.88% and 54.90%), and the chat latency is also very close (10.39 and 11.17 seconds). This shows that participants on average performed about the same with the secondary task across scenarios. No statistically significant differences were found across scenarios for percent score, time spent, and try count.

Table 6.1: Comparing across planning methods (SE stands for standard error)

	M	P	SA	SE	Significance
% Score	59.40	72.75	94.60	1.39	<b><math>F[2, 50] = 223.03, p &lt; .0001</math></b>
Time spent	243.35	240.02	228.37	12.06	$F[2, 50] = 1.16, p = .32$
Try count	1.75	3.56	3.31	0.43	$F[2, 50] = 9.47, p = .0003$
Clicks/try	13.01	35.64	25.58	2.90	<b><math>F[2, 50] = 19.47, p &lt; .0001</math></b>
NASA TLX	61.51	49.18	48.86	2.81	<b><math>F[2, 50] = 14.15, p &lt; .0001</math></b>
% Q. missed	52.94	56.69	55.04	5.17	$F[2, 50] = 1.26, p = .29$
Chat latency	10.39	11.17	10.92	0.65	$F[2, 50] = 0.46, p = .63$

### 6.6.2 Comparing Across Planning Methods

For each scenario, three path planning methods were used (manual, pattern, and sliding autonomy). Table 6.1 lists comparison among these three methods.

Percent score differences are statistically significant ( $F[2, 50] = 223.03, p < .0001$ ) with sliding autonomy (94.60%) performing better than pattern (72.75%) and manual (59.40%). As shown in Figure 6.4, this trend is also clear in both scenario 1 and 2 individually. Therefore, user study results support our first hypothesis: sliding autonomy method performs better than either the manual method or the pattern method. This holds for both high and low information scenarios, suggesting some robustness of the result across a range of scenarios.

Statistically significant differences ( $F[2, 50] = 19.47, p < .0001$ ) were also found in mouse clicks per try (starting over means having another try). The manual method uses arrow keys to fly the UAV around and only uses mouse clicks when switching camera modes or stop the timer in order to perform the secondary task. By design, this method does not use a lot of mouse clicks. Pattern and sliding autonomy methods both use mouse clicks for the actual path planning task, and the pattern method clearly generated more mouse clicks per try (35.64) than the sliding autonomy method (25.58). Two factors might have contributed to this difference: First, the pattern method allowed a participant to "undo" a path segment (in addition to reset and start over) whereas sliding autonomy did not allow this. Second, sliding autonomy allowed a participant to drag a slider, which produced different suggested



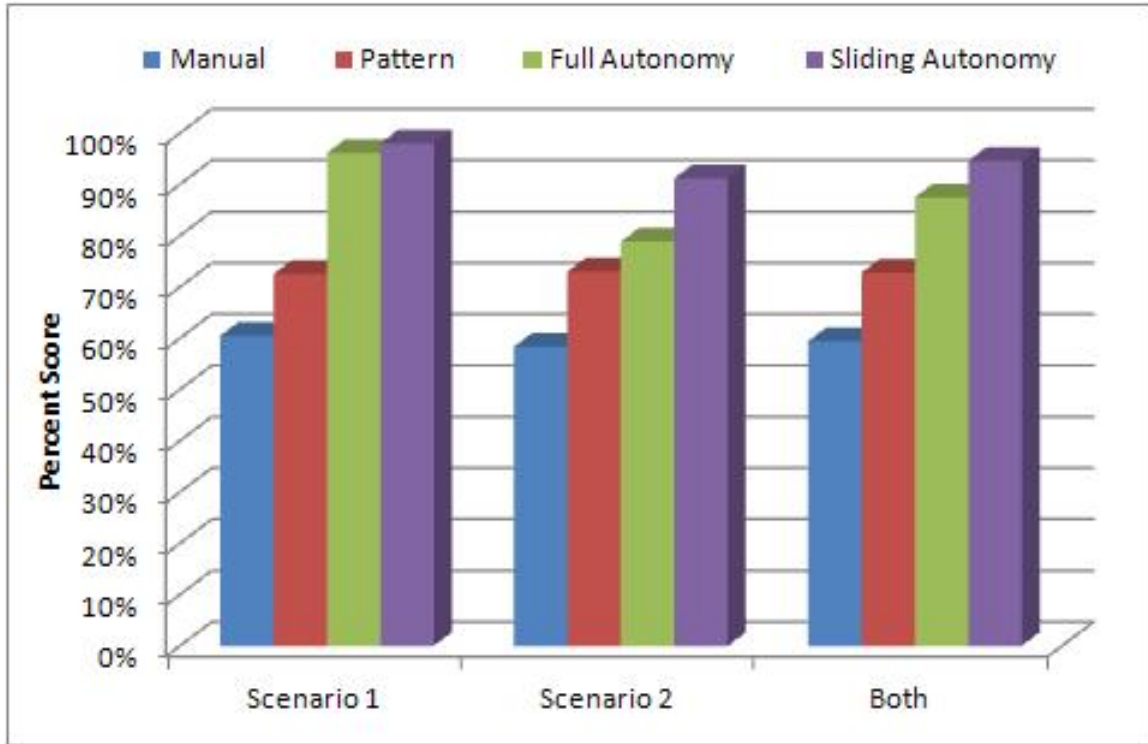


Figure 6.4: Performance differences for the three path planning methods.

paths; this accomplishes the same type of “what if” interaction as “undo” in the pattern method, but required many fewer mouse clicks.

It is informative to compare these interactive planning methods with a fully autonomous path. This is useful because, due to the computational complexity of the planning problem, only suboptimal solutions can be generated by the planning algorithms. Completely autonomous path planning (without human input) produces paths with a score of 96.13% for scenario 1 and 78.33% for scenario 2. It is instructive to compare these values to those produced by the different planning methods in the different scenarios (see Figure 6.4). This places the performance of full autonomy ahead of manual and pattern planning methods in both scenarios, but behind sliding autonomy in both scenarios. This indicates that the sliding autonomy approach outperforms both manual, pattern, and full autonomy approaches to the problem.

As shown in Table 6.2, for scenario 1, no participants were able to outperform full autonomy using manual or pattern approaches, but 23 of 26 participants (88.46%) were able

Table 6.2: Percent of participants outperforming autonomy with each method

	Manual	Pattern	Sliding Autonomy
Scenario 1 (Low)	0%	0%	<b>88.46%</b>
Scenario 2 (High)	0%	19.23%	<b>92.31%</b>

to outperform full autonomy using sliding autonomy. For scenario 2, no participants were able to outperform full autonomy using manual control, but 5 of 26 participants (19.23%) and 24 of 26 participants (92.31%) were able to outperform full autonomy using pattern and sliding autonomy, respectively. Thus, results of the study support the second hypothesis: Sliding autonomy methods perform better than a fully autonomous approach given state-of-the-art planning algorithms for this problem.

The full autonomy we refer to here is the specific path planning algorithm we used in the user study (LHC-GW-CONV). In Section 6.7.2, we discuss how the sliding autonomy approach compares to other path planning algorithms.

NASA TLX raw scores show significant differences ( $F[2, 50] = 14.15, p < .0001$ ) among the three methods, with the manual method showing the highest cognitive mental workload (61.51), a full “pip” more than the other two methods on the TLX survey. The average score difference between the pattern method and the sliding autonomy method is not significantly different. Figure 6.5 shows the box plots of the NASA TLX scores for each scenario.

For all three planning methods, participants performed about the same on the secondary task, as shown by percent of questions missed and chat latency in Table 6.1. Combining this with percent score and NASA TLX we can conclude that sliding autonomy performed best without increasing participants’ mental workload, which support our third hypothesis: Sliding autonomy method does not increase the mental workload of the operator when compared against manual and pattern methods.

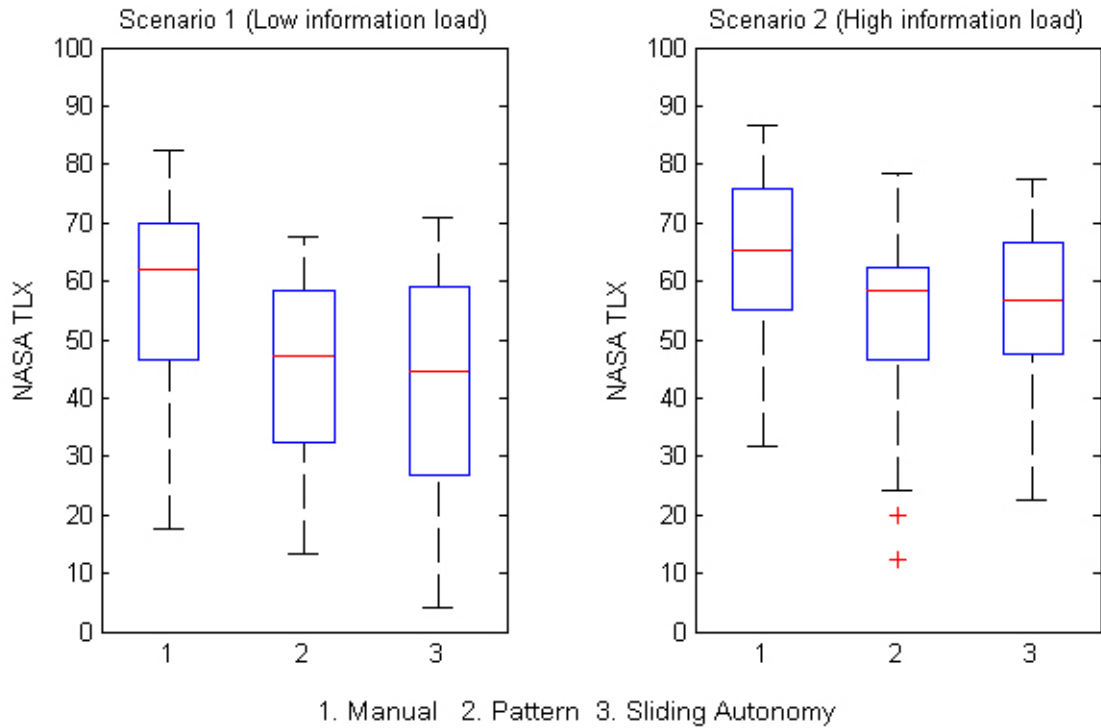


Figure 6.5: Box plots of the NASA TLX scores for each scenario.

### 6.6.3 Additional Factors

We also performed ANOVA analysis on some additional factors that might create differences: gender, experience in video games, order of the scenarios, and whether participants used full autonomy with the sliding autonomy method. No significant differences were found for these factors overall, across scenarios, or across methods. There is also no significant correlation (-0.23) between percent of questions missed in the secondary task and the NASA TLX raw scores.

## 6.7 Discussion

### 6.7.1 Planning Methods Characteristics

*Manual Method*

With the manual method, the user uses arrow keys to move the UAV around to create a path, so by design the method is very intuitive, flexible, and requires more physical interactions (keyboard, not mouse clicks). But in order to plan a 60-minute path faster than real-time (participants had to accomplish this in 2 minutes), although we designed the experiment using input from a pilot study, ensuring that each participant could complete at least two attempts during the 5 minutes allocated, many participants reported that the arrow keys were too “sensitive” and recommended slowing down the UAV.

Because of the time pressure, when errors are made, in practice it is too costly to start over (we did not provide an undo function). Although it is possible to pause the simulation to allow for participants to plan, participants reported that they did not feel that they had the luxury to do so. Naturally, when this continuous process is interrupted by the secondary task where the user has to pause planning and answer questions in the group chat window, user frustration is high.

More physical work, higher frustration, and lower performance score are the main factors contributing to a much higher NASA TLX score for the manual method. During training, participants actually had one extra session with the manual method, but this method still performed the worst.

### *Pattern Method*

With the pattern method, the user joins a mixture of three patterns (spiral, lawnmower, and line) together to form the final path. This is more of an episodic process, so it is very easy to pause in the middle of the planning and shift attention to the secondary task. There is also less time pressure because the user can quickly plan for the remaining time with just one big spiral (or lawnmower) in one click. Therefore, the user has plenty of time for many tries with different strategies.

In the post user study survey, many participants commented that with the spiral and lawnmower pattern it is really easy to run out of time. They suggested adding the ability to

allocate time to the patterns similar to the sliding autonomy method. This means that with this method, a user enjoys the systematic coverage of an area but has a hard time estimating how much time it takes the UAV to cover the area following the pattern. Several participants also suggested adding more patterns to the method.

The pattern method is the only method that allows a participant to “undo” a plan. This ability impacted the number of mouse clicks per try and participants’ preference over the three planning methods. Another interesting observation is that participants seemed to be overly optimistic about their performance using the pattern method. For example, although sliding autonomy created better paths than pattern in all scenarios for all participants, 46.15% of participants (as measured in the NASA TLX with the performance dimension) and 26.92% (as reported in the post study survey) reported that the pattern method created best paths.

### *Sliding Autonomy Method*

Similar to the pattern method, the sliding autonomy method is also episodic. Therefore, stopping in the middle of the planning to answer questions for the secondary task was easy. Since it only takes a few clicks to let autonomy plan path for the remaining time, there is not much time pressure and the user can have many tries.

Because the user does not know how autonomy works behind the scene, many participants were surprised by the path recommended by autonomy, and feel that autonomy did not do what they wanted it to do. For example, when a user sets the end point in region A, autonomy might plan a path that spends most of time in a seemingly unrelated region B and only goes toward region A at the end of the path, because such a path is more efficient (scores higher). In such cases, the slider becomes the only tool that lets the user “force” autonomy to do what the user wants, and path planning turns into a fight between the human and autonomy. However, the instant feedback (displaying path and the predicted “vacuuming effect”) does help the user figure out why autonomy would suggest something different, and some participants were glad that autonomy suggested better paths they had not considered.

Most participants were generally happy with the path segment recommended by autonomy covering a local region, even when the region is in an irregular shape (not a circle or rectangle). Many participants also expressed that they did not have enough control over the path generation and recommended adding the ability to include constraints such as “middle points” where the path segment has to go through these middle points. In fact, such “middle points” can already be achieved with the current method by setting multiple endpoints, effectually creating a multi-segment approach. Several participants complained that this method does not have the undo function. With both the pattern and sliding autonomy methods, many participants expressed the desire to be able to modify the path after it is generated.

### *User Preferences*

In scenario 2 where a task-difficulty map was used, most participants switched between map views. The probability map used is similar to a unimodal distribution (see Figure 6.3 middle left). For the first part of the planning, they viewed the probability distribution map and “covered” the high mode. They then switched the view to the task-difficulty map for the remaining time, only occasionally switching back to the probability distribution map view. This pattern of behavior was seen in each of the three planning methods. Some participants suggested showing both maps side by side or have a way to combine the two maps into one. These ideas are worth exploring in future user studies.

In the post user study survey, the majority of the participants think manual is the easiest to learn (53.85%), pattern is the easiest to use (57.69%), and sliding autonomy performed the best (65.38%). However, most participants preferred the pattern method (69.23%) out of all three. We believe the inability to undo and operator-induced oscillation when moving the slider had negative impacts on participants’ preference over the sliding autonomy method. This is relevant for the design of future sliding autonomy systems, suggesting that some combination of pattern-based planning and sliding autonomy, augmented

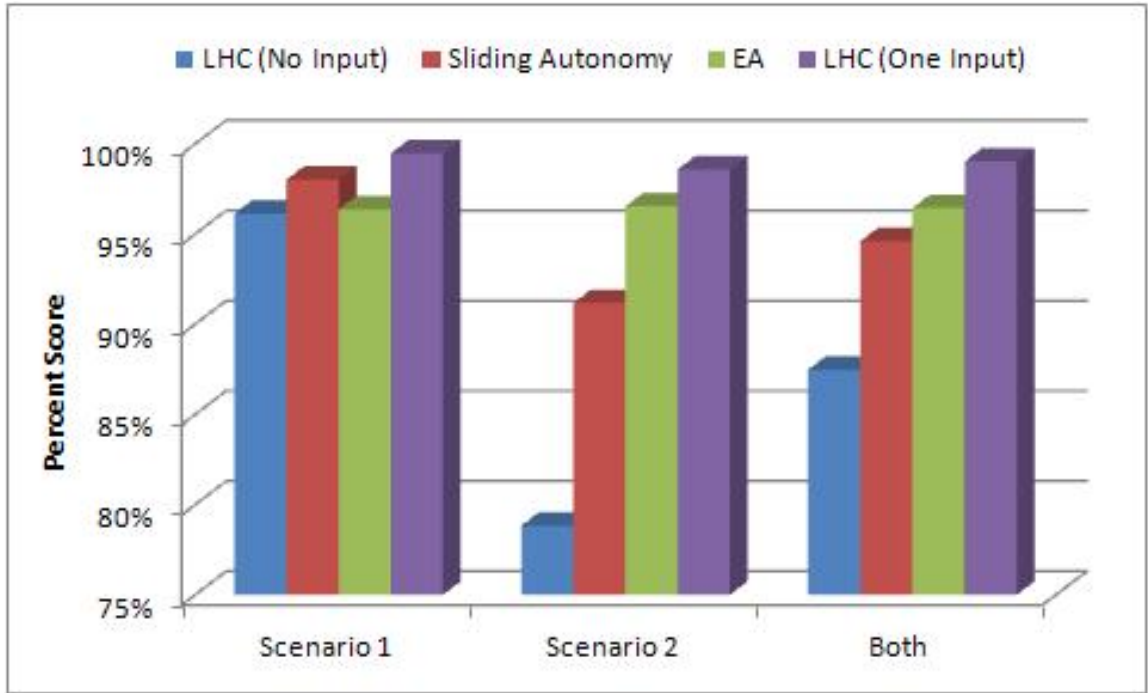


Figure 6.6: Comparing sliding autonomy performance against various markers.

with the ability to undo decisions and flexibly alter or constrain paths, will produce a high-performing GUI with high user acceptance.

### 6.7.2 Reliance on Autonomy

We have claimed that a human interacting with an autonomous algorithm via sliding autonomy outperforms full autonomy, but this claim naturally depends on the quality of the autonomous algorithm. The algorithm we used was selected from a comparison of various algorithms in our prior work [69, 72] because it worked in real-time and produced high quality paths, but there exist other algorithms that produce higher quality paths if we allow more time for path-planning. It is useful to compare performance of the sliding autonomy algorithm with these other algorithms.

As a basis for comparison, we consider an evolutionary algorithm (EA) that takes the output of several real-time planning algorithms, including the one we used in the user study, as seeds for the evolutionary process. Thus, the EA approach takes high quality solutions and

Table 6.3: Percent of participants outperforming autonomy performance markers

	Autonomy	EA	Autonomy+1
Scenario 1 (Low)	<b>88.46%</b>	<b>88.46%</b>	7.69%
Scenario 2 (High)	<b>92.31%</b>	26.92%	15.38%

then adds further optimizations. As shown in Figure 6.6, when such optimization is applied to scenario 1 and scenario 2, the optimization produces a path that is only slight worse than sliding autonomy for scenario 1 and a path that is much better than sliding autonomy in scenario 2. This suggests that better path planning might be more important than interactive path planning.

Because we are arguing that human plus autonomy is better than either alone, we explored how the number of human input can affect the output of the sliding autonomy approach. Results indicate that the sliding autonomy algorithm we used in the user study can generate high quality paths with only one point of human input (specifying an end point). We call this approach the *autonomy+1* approach.

Using the best score out of 3 tries (roughly equal to the average number of tries in the user study), we computed the percent score for this autonomy+1 human input approach: 99.47% for scenario 1 and 98.58% for scenario 2. Using the EA and Autonomy+1 scores as additional markers, we plotted participants average performance in each scenario against these markers. Figure 6.6 shows the result.

First, sliding autonomy (human-autonomy team) outperformed nominal autonomy in both scenarios. Sliding autonomy also outperformed EA in scenario 1 (low information). In scenario 2, the performance of sliding autonomy is not very far from EA (5.36%), and the difference is even smaller (1.85%) when averaged over both scenarios. However, the most interesting observation is that autonomy+1 actually outperformed all others in both scenarios (99.47% for scenario 1 and 98.58% for scenario 2). Although a few participants did score higher than autonomy+1, the difference is less than 1.5%. Table 6.3 lists what percentage of participants outperformed full autonomy, EA, and autonomy+1.



What Figure 6.6 suggests is that with the sliding autonomy method, it does not need a lot of human input to perform really well. Instead of spending the effort creating many path segments and setting many end points, it may be more effective to search in the right region by setting just a few constraints. However, 88.68% of the participants gave more than 1 input when they used sliding autonomy (81.13% for 2 inputs and 69.81% for 3 inputs). In the post user study survey, only 46.15% of the participants acknowledged trying full autonomy with the sliding autonomy method, meaning that these participants did not specify any endpoints and simply relied on the autonomous path-planner to do all the planning. When using the sliding autonomy method, a good strategy is actually to start with full autonomy (as the worst scenario) and then see how additional human input can improve the path, but this leads to questions of over- and under-reliance on autonomy [15].

### 6.7.3 Why Human-Autonomy Team Performs Better?

User study results show that the human-autonomy team outperformed both human or autonomy working alone. But how were they able to achieve this? We hypothesize that this is because the sliding autonomy approach enabled the human to focus on what the human is good at and autonomy to focus on what autonomy is good at. Bradshaw et al. point out [15]: “Humans, though fallible, are functionally rich in reasoning strategies and their powers of observation, learning, and sensitivity to context.” Our observation suggests that a human may be better equipped than autonomy to think strategically and to recognize bad path segments.

The sliding autonomy method lets the user plan at a higher abstract level by specifying priorities in search sub-regions and how well each sub-region should be covered. Autonomy, on the other hand can generate a path that covers a sub-region (or some nearby sub-regions) precisely and quickly, and can handle all kinds of irregular sub-region shapes. Therefore, the sliding autonomy method combines the strengths of both human and autonomy.

A human user is also very good at recognizing bad moves in solutions suggested by autonomy. The sliding autonomy approach enables the human user to select from a bunch of suggested paths. Selecting a path segment with fewer bad moves will probably increase the chance of a good final path.

#### **6.7.4 Why Similar Secondary Task Performance in All Three Methods?**

The pattern and sliding autonomy methods are episodic, suggesting that it is easier for the user to pause planning and shift attention to the secondary task of answering questions in the group chat window. However, user study data show that there is no significant differences in secondary task performance across all three path planning methods.

The manual method requires a lot of continuous keyboard interaction (great physical demand and temporal demand) to move the UAV around. However, it does not actually require much mental demand and effort because the planning process is more sporadic and spontaneous. If a mistake is made, because there is no way to correct it, the user quickly stops worrying about it and moves on. The low mental demand and effort make monitoring the group chat window an easy task, even though switching back and forth between primary task and secondary task is very frustrating.

With the pattern and sliding autonomy methods, path planning is more like piecing together a puzzle. The user is deeply drawn into problem solving, constantly comparing tradeoffs, which actually requires more mental involvement. With the sliding autonomy method, the user is interacting with complicated algorithms, so while planning a path, the user is also trying to build a mental model of how autonomy works. As a result, the user actually paid less attention to the secondary task. Fighting with autonomy when human and autonomy had disagreements also drew user attention away from the group chat window. But when the group chat window catches the user's attention, he/she can perform the secondary task leisurely.

David Woods and Eric Hollnagel describe the law of stretched systems [132]: "every system is stretched to operate at its capacity; as soon as there is some improvement, for example in the form of new technology, it will be exploited to achieve a new intensity and tempo of activity." With the pattern and sliding autonomy methods, users had more tries and evaluated more options and tradeoffs. With the sliding autonomy method, users played more with spatial and temporal constraints, and evaluated more paths suggested by path planning autonomy, which resulted in better quality paths at the cost of no performance increase in the secondary task.

## 6.8 Conclusions and Future Work

In this paper we propose a new autonomy management approach, sliding autonomy, which lets the user influence the behavior of the autonomous system along three new dimensions: information representation, spatial constraints, and temporal constraints. We extend the autonomy design guidelines in our prior work by adding a new row for intelligence of collaborative agents (human-autonomy team), and explain how the three new dimensions fit into the guidelines when we apply the proposed approach to the task of UAV (Unmanned Aerial Vehicle) path planning to support Wilderness Search and Rescue (WiSAR). We present interface designs that let the user allocate degrees of authority and flexibility to the robot's algorithms through interactivities along these new dimensions. Experiment results from a user study support our hypotheses and show that the sliding autonomy method performs significantly better than either the manual or pattern path planning method without increasing the user's mental workload, the human has a better interaction experience, and human-autonomy team outperforms either human or autonomy working alone.

Over- and under-reliance on autonomy are related to issues of trust. Both Lee and See [66] and Bradshaw et al. [15] suggest that trust in automation should be "calibrated". We believe that the sliding autonomy approach we propose can be useful in this aspect, because as the user is moving the slider, he/she is calibrating his/her reliance on path planning

autonomy. In our user study, all participants only had 30 minutes of training before using the sliding autonomy method. We speculate that as the user gets more familiar with the sliding autonomy approach in the long run, he/she would be able to calibrate reliance better. Validating this hypothesis is a natural extension of the present work.

When more complex and/or outdated probability distribution maps and task-difficulty maps are used, or when the user has the ability to modify information representation in the sliding autonomy interface, it would be interesting to see how these affects the human-autonomy interaction and the performance of the human-autonomy team. We leave these for future work.

## Chapter 7

### Probability Distribution Map and Task-Difficulty Map Editor Interface

#### 7.1 Introduction

At the **between-episodes** scale, a searcher might have additional case-specific information (e.g, past experience, knowledge of the search area or weather conditions, or the profile of the missing person) and would like to modify the general plan produced at the strategic scale. Moreover, as search progresses, the search plan should change due to newly found evidence (or the lack of it) from either the ground searchers or previous UAV flights. We developed two autonomy management tools at this scale that allow the user to manage two types of information: the *probability distribution map* and the *task-difficulty map*.

Searchers can use the **DistEdit** tool to modify a *probability distribution map* and use the **DiffEdit** tool to modify a *task-difficulty map* generated at the **strategic** scale. Both tools enable the user to view maps as 3D surfaces where a color map is applied for better distinction (red means high probability area or high task-difficulty level and blue means low). The user can use mouse and finger gestures to rotate/pan/zoom the respective map and edit the shapes of the maps in 3D to incorporate information that the autonomous components are unable to interpret. The user also has the option to overlay a satellite image of the search area on top of the maps for better alignment and precision.

If the user is dissatisfied with the *probability distribution map* or *task-difficulty map* systematically generated at the **strategic** scale, using the **DistEdit** and **DiffEdit** tools he

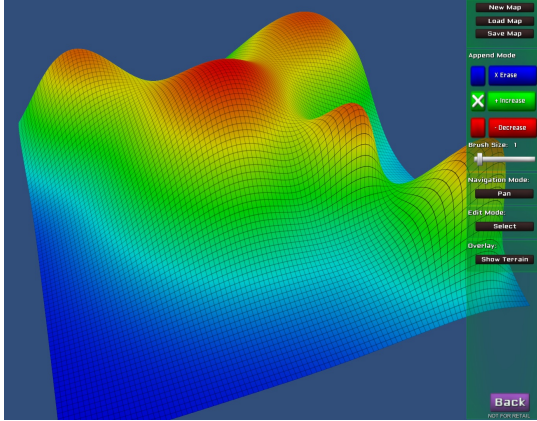


Figure 7.1: An example *probability distribution map* generated using the **DistEdit** tool.

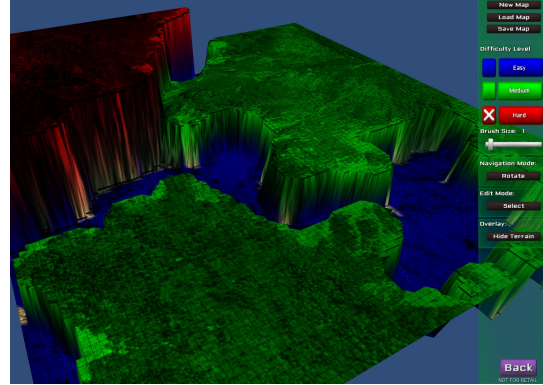


Figure 7.2: An example *task-difficulty map* generated using the **DiffEdit** tool with a satellite image of the search region overlaid on top.

or she can also create a *probability distribution map* and/or a *task-difficulty map* from scratch. Figure 7.1 and 7.2 show screen captures of these two tools and also example maps generated using the two tools.

Both tools enable the searchers to incorporate additional information only available to or understandable by the user into the two information representations – in the form the autonomous components of the UAV system can understand – and then interactively use the UAV path-planner to use the additional information produce highly efficient paths. We designed both tools to support common touch-screen finger gestures. The user has the option to perform all tasks using only finger gestures, only keyboard/mouse controls, or a hybrid of the two.

## 7.2 The DiffEdit Tool

The **DiffEdit** tool enables the user to create or modify a *task-difficulty map* by marking areas with different levels of difficulty. When using a UAV to support WiSAR, task difficulty is related to sensor detection probability. A difficult area on the map represents a place where the likelihood of detecting the missing person is low (maybe due to terrain features, vegetation density, or lighting conditions). By marking an area with high task

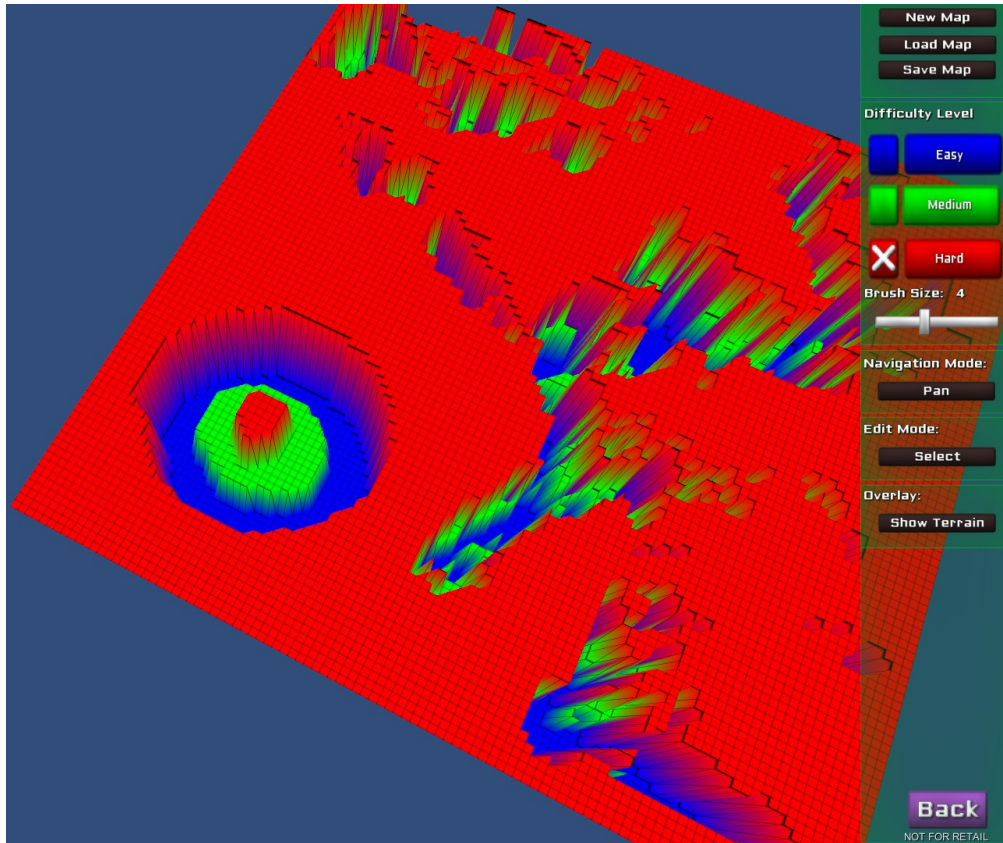


Figure 7.3: An example *task-difficulty map* systematically generated at the **strategic** scale for a real WiSAR scenario with modification made on the lower left corner using the **DiffEdit** tool.

difficulty, the user can indirectly tell the UAV to make multiple passes in the area, or avoid the area and set high priority to areas marked with low task difficulty. When combined with a prior probability, encoded as a *probability distribution map*, an area with medium probability and low task difficulty may be more attractive than an area with high probability but high task difficulty.

### 7.2.1 Editing vs. Starting New

The tool is modular for easy integration into the overall intelligent system. A *task-difficulty map* is stored as an external file, and can be loaded (imported) into the tool. A modified map can be saved (exported) to the hard drive. The group of buttons on the upper right corner of Figure 7.3 are for this purpose.

The user can click the **New Map** button to start a map from scratch, if the user is dissatisfied with the systematically generated map from the **strategic** scale. The **Load Map** button lets the user load an existing map. This map could be from the **strategic** scale, or could be from a UAV flight episode in the **between-episodes** scale, so information collected from the previous UAV flight could be incorporated into the map.

In Figure 7.3, the right part of the map is systematically generated from vegetation density information at the **strategic** scale where the red plateau areas are areas with high vegetation density. The canyon shapes indicate areas with sparse vegetation density. The circular hole on the lower left corner of the map shows user-made modifications to the systematically generated map using the **DiffEdit** tool. Then the user can click the **Save Map** button to store the map as an file externally.

The tool also lets the user overlay satellite imagery of the search area on top of the *task-difficulty map* for better reference and precision (Figure 7.4 shows an example). What image to use can be configured in tool settings. Then the user can click the overlay button **Show Terrain (Hide Terrain)** to toggle back and forth.

### 7.2.2 3D Navigation Controls

The **DiffEdit** tool is a true 3D environment where the *task-difficulty map* is shown as a 3D surface. Task difficulty is represented by both height and color (blue is easy, green is medium, and red is hard). The ability to rotate the surface in 3D and zoom in/out allows the user to get a good grip of task difficulty in different areas of the search region.

The user can click the navigation mode toggle button to switch between **Rotate** mode and **Pan** mode. In the **Rotate** mode, the user can use the arrow keys and the WASD keys to rotate the map both vertically and horizontally in 3D. In the **Pan** mode, the arrow keys and the WASD keys can pan the map left-right and up-down. Finally, the mouse scroll wheel can be used to zoom the map in or out.



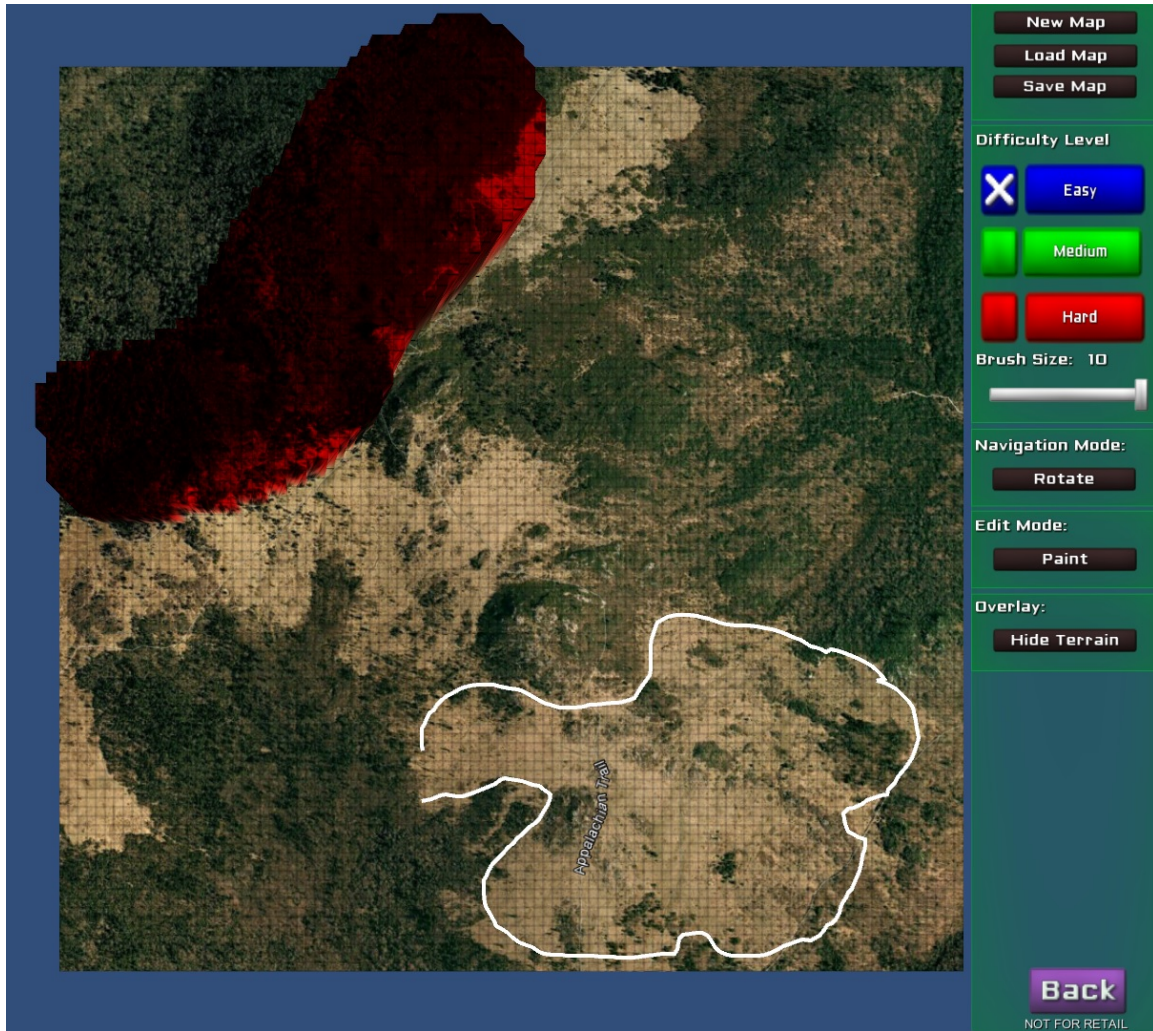


Figure 7.4: A satellite imagery of the search area loaded into the **DiffEdit** tool as an overlay, an example high task-difficulty area on the top left corner made using the paintbrush tool, and an example selection made using the lasso selection tool.

With a touch-screen device, buttons can be pressed with simple finger touches. A common two-finger rotate gesture will rotate the map; moving two fingers toward the same direction will pan the map to that direction; and the two-finger pinch gesture lets the user zoom the map in or out.

Using the rotate/pan/zoom function, the user has total control of the 3D environment. With the satellite imagery of the search area overlaid on top, the user can easily mark task difficulty at the desired resolution and precision.

### 7.2.3 Paint Mode vs. Lasso Select Mode

The **DiffEdit** tool lets the user edit a *task-difficulty map* using two modes: the *paint* mode and the lasso *select* mode. The user can select which mode to use by clicking the toggle button **Paint (Select)**. First the user needs to select a desired task difficulty level by clicking one of the difficulty level buttons. The tool supports three task difficulty levels: easy, medium, and hard. Each task difficulty level is represented on the 3D *task-difficulty map* by color and height. The easy level (color blue and low height) could represent an area with no vegetation coverage or sparse type vegetation (e.g., grass). The medium level (color green and medium height) could mean the area is covered by plants such as short shrubs. The hard level (color red and high height) might be used to mark areas with dense forest (e.g., evergreen type plants). Although the tool only supports three task difficulty levels presently, it can be easily extended to support more difficulty levels through, for example, a slider bar.

In the **paint** mode, the cursor becomes the brush and is shown as a circular shadow projected onto the 3D surface from above, with its color matching the selected task-difficulty level. The user can move the brush size slider in the control panel (see the right side of Figure 7.4) to select a desired brush size between 1 and 10. The size of the brush is indicated on the map by the radius of the circular shadow. Then the user can mark task difficulty on the *task-difficulty map* by painting different areas using the paintbrush. If a satellite imagery is overlaid on top of the *task-difficulty map*, the zoom function described in the previous section can be combined with different brush size selection to achieve the level of precision desired by the user. Figure 7.4 shows an example where an area on the satellite imagery with dense vegetation (upper left part of the map) is marked with high task difficulty (red) using the paintbrush tool.

In the **select** mode, the user can drag a freehand selection around the desired area. The tool will automatically connect the starting point and the end point of the line to form a closed selection, and the selected area is automatically marked with the selected task difficulty

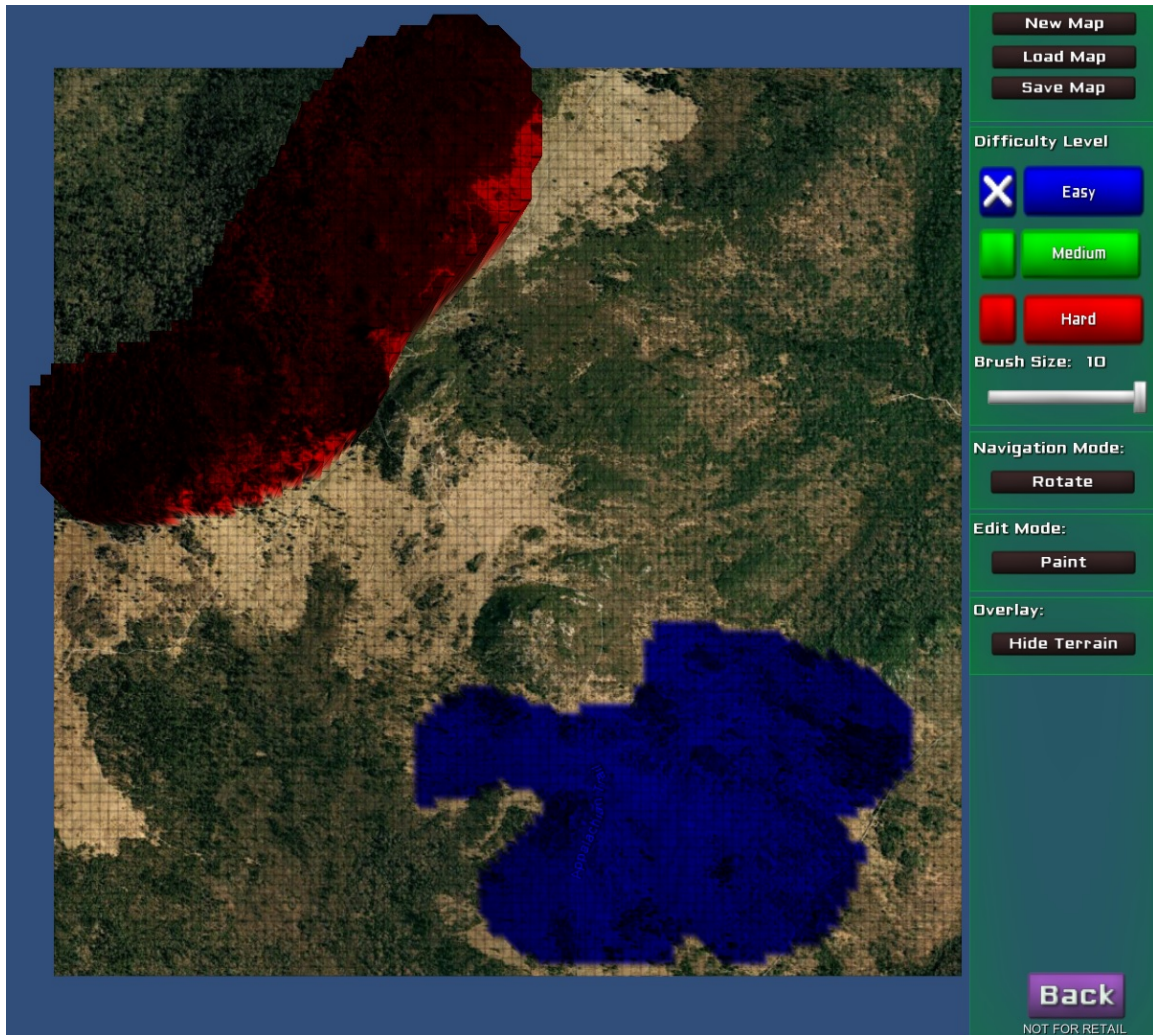


Figure 7.5: The area selected in Figure 7.4 is automatically marked with the selected task difficulty level easy.

level. The white line in Figure 7.4 shows an area selected by a user, and Figure 7.5 shows how the selected area is automatically marked with the selected easy task difficulty.

Similarly, using a touch-screen device, the user can use a finger to paint on top the *task-difficulty map* in the **paint** mode. The user can also use a finger to draw freehand selection on the map to select an area in the **select** mode.

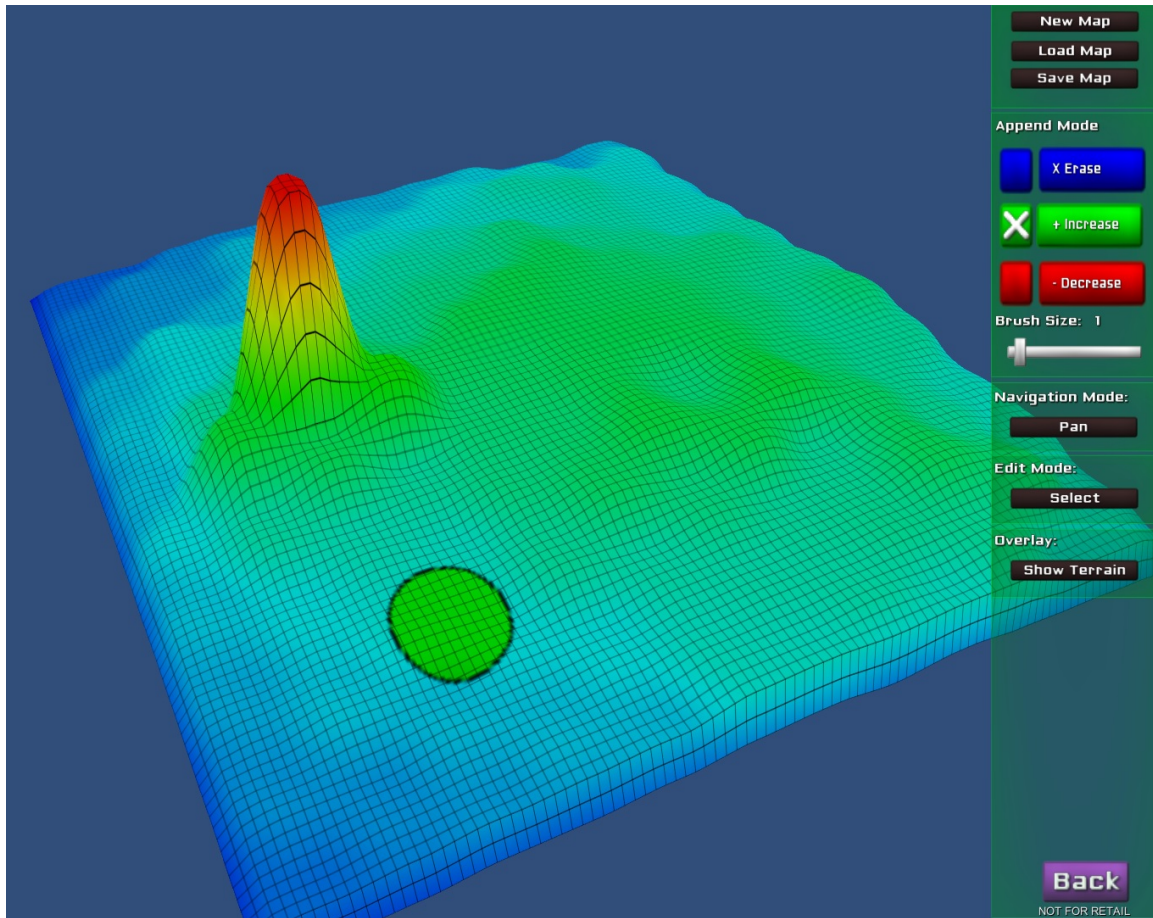


Figure 7.6: An example *probability distribution map* systematically generated using the **DistCreate** tool at the **strategic** scale for a real WiSAR scenario.

### 7.3 The DistEdit Tool

The **DistEdit** tool enables the user to create or modify a *probability distribution map* by adding or subtracting Gaussian distributions. This way the user can generate a mixture of Gaussians to represent the desired probability distribution, which is common in real WiSAR operations. When using a UAV to support Wilderness Search and Rescue (WiSAR), a *probability distribution map* shows the place where the missing person is likely to be found. The modified probability distribution can be used later by the path planner to prioritize tasks and plan UAV paths. By marking an area as a high priority area, the searchers can indirectly manipulate the UAV to search the area before other areas without the need to manually specify waypoints.

### 7.3.1 Editing vs. Starting New

Similar to the description of the **DiffEdit** tool used to modify the *task-difficulty map*, the **DistEdit** tool is modular and the *probability distribution map* is stored as an external file. The user can load a *probability distribution map* systematically generated at the **strategic** scale using the **DistCreate** tool and improve it, or start from scratch if the user is dissatisfied with the automatically generated map. This map can then be updated after each UAV flight episode as more information is collected in the previous flight. Areas already covered by the UAV can be marked with lower probability; areas where possible evidence has been found by the UAV or ground searchers can be marked with high probability.

The group of buttons are identical to the **DiffEdit** tool (**New Map**, **Load Map**, and **Save Map**) and the user also can similarly overlay satellite imagery on top of the *probability distribution map*. Figure 7.6 shows an example probability distribution systematically generated by the **DistCreate** component at the **strategic** scale. This map was generated using the HikerPaul WiSAR scenario [72] from the International Search & Rescue Incident Database (ISRID) [60]. The probability hill on the left side of the map indicates an area where the probability of finding the missing person is very high. The *probability distribution map* is encoded with a color map where red indicates high probability areas and blue indicates low probability areas. Figure 7.10 shows a *probability distribution map* with satellite imagery overlaid on top of the map.

### 7.3.2 3D Navigation Controls

The 3D navigation controls in the *DistEdit* tool is identical to the *DiffEdit* tool described in the previous section. Arrow keys and the WASD keys are used to rotate the map vertically and horizontally in the **Rotate** mode and pan the map left-right and up-down in the **Pan** mode, and finger gestures on a touch screen device perform identical functions as the **DistEdit** tool. Mouse scroll wheel can be used to zoom the map in or out.

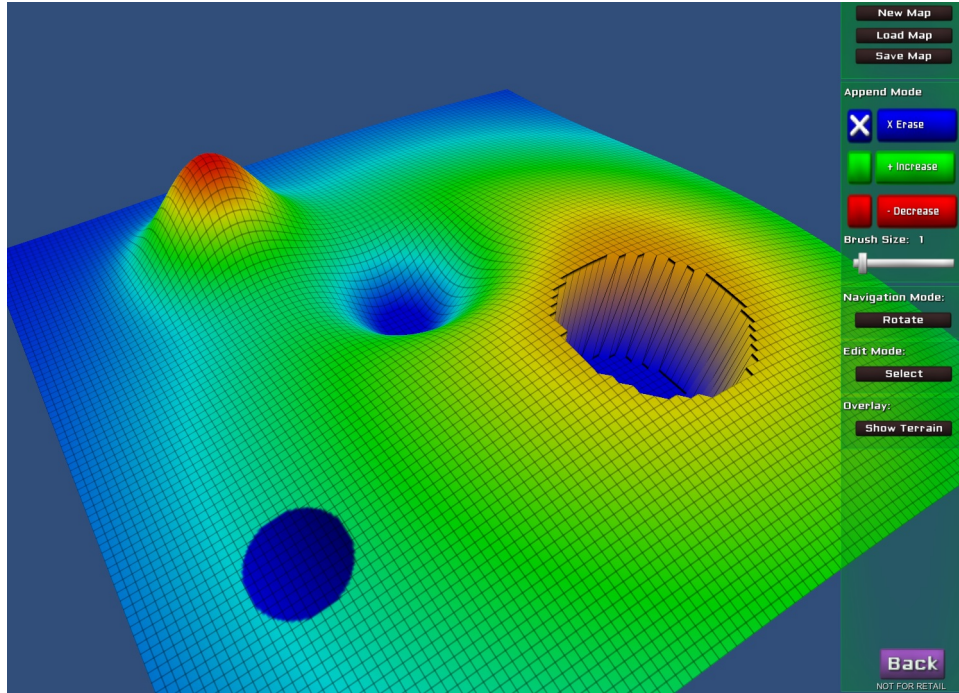


Figure 7.7: An example *probability distribution map* demonstrating how a Gaussian can be added to or subtracted from the map and how probability in an area can be completely erased.

### 7.3.3 Paint Mode vs. Lasso Select Mode

The **DistEdit** tool lets the user edit a *probability distribution map* using two modes: the *paint* mode and the lasso *select* mode. The user can select which mode to use by clicking the toggle button **Paint (Select)**.

In the **paint** mode, the user can choose to add a Gaussian to (or subtract a Gaussian from) the current *probability distribution map* or erase the probability in an area. The user first clicks one of the three color coded buttons, **Erase**, **Increase**, and **Decrease**, then paints on the map using the paintbrush to make modifications.

If **Erase** is selected, painting an area on the map means completely erasing the probability in that area. This can be useful when an area has already been thoroughly searched by the UAV and/or ground searchers in the previous episode and the user is confident that the missing person is not in that area. The user can also move the cursor to paint with freehand. The brush size can be set using the brush size slider on the control panel. The big

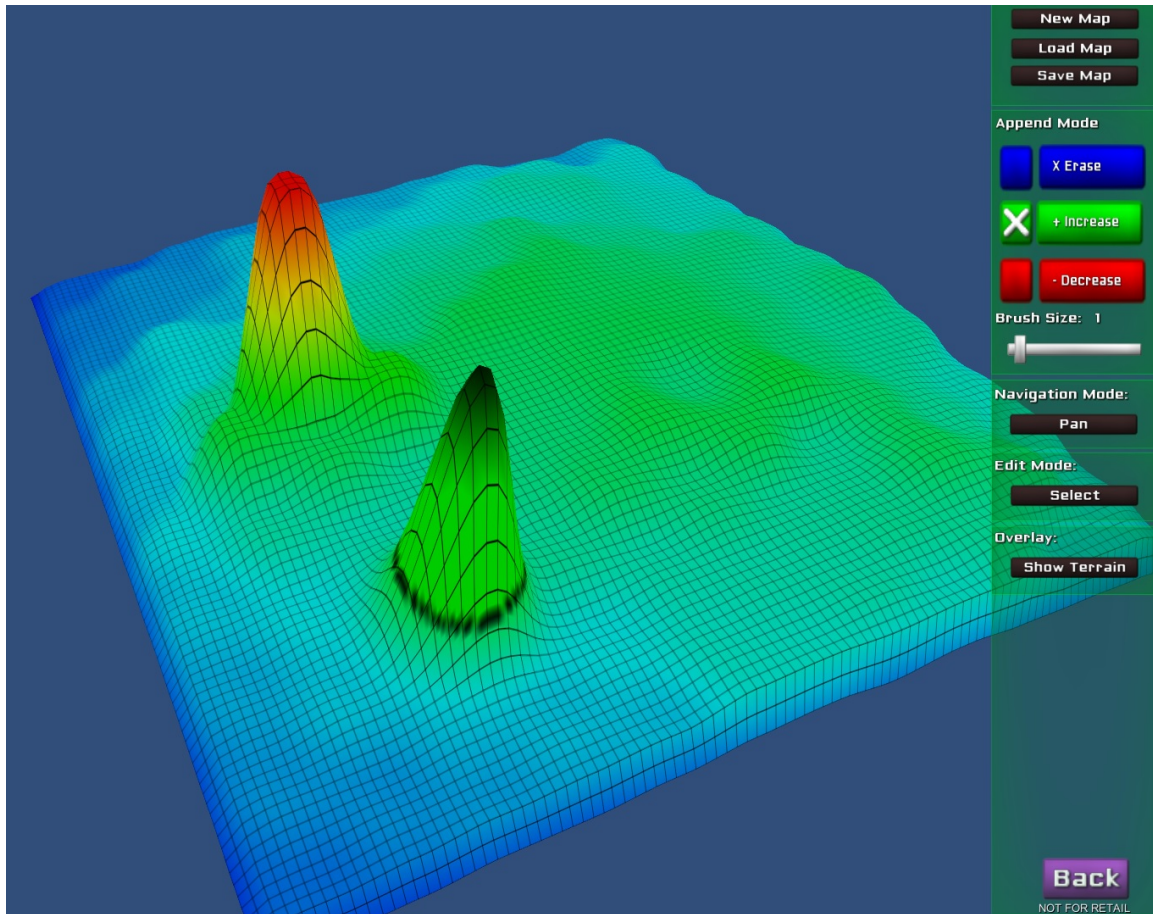


Figure 7.8: A Gaussian is added to the systematically generated *probability distribution map* shown in Figure 7.6.

circular hole in Figure 7.7 shows an example of an erased area created by using a paintbrush of the same size.

If **Increase** is selected, The brush size determines the standard deviation (with a radius equivalent to three times the standard deviation) of a symmetric Gaussian to be added to the existing *probability distribution map*. The mouse click (or finger press gesture) position determines the mean of the Gaussian distribution, and the duration of the click (or finger press gesture) determines the scale (height relative to other parts of the distribution) of the Gaussian distribution. The probability hill in Figure 7.7 shows an example of a Gaussian added to the *probability distribution map*. Figure 7.8 shows how another probability hill can

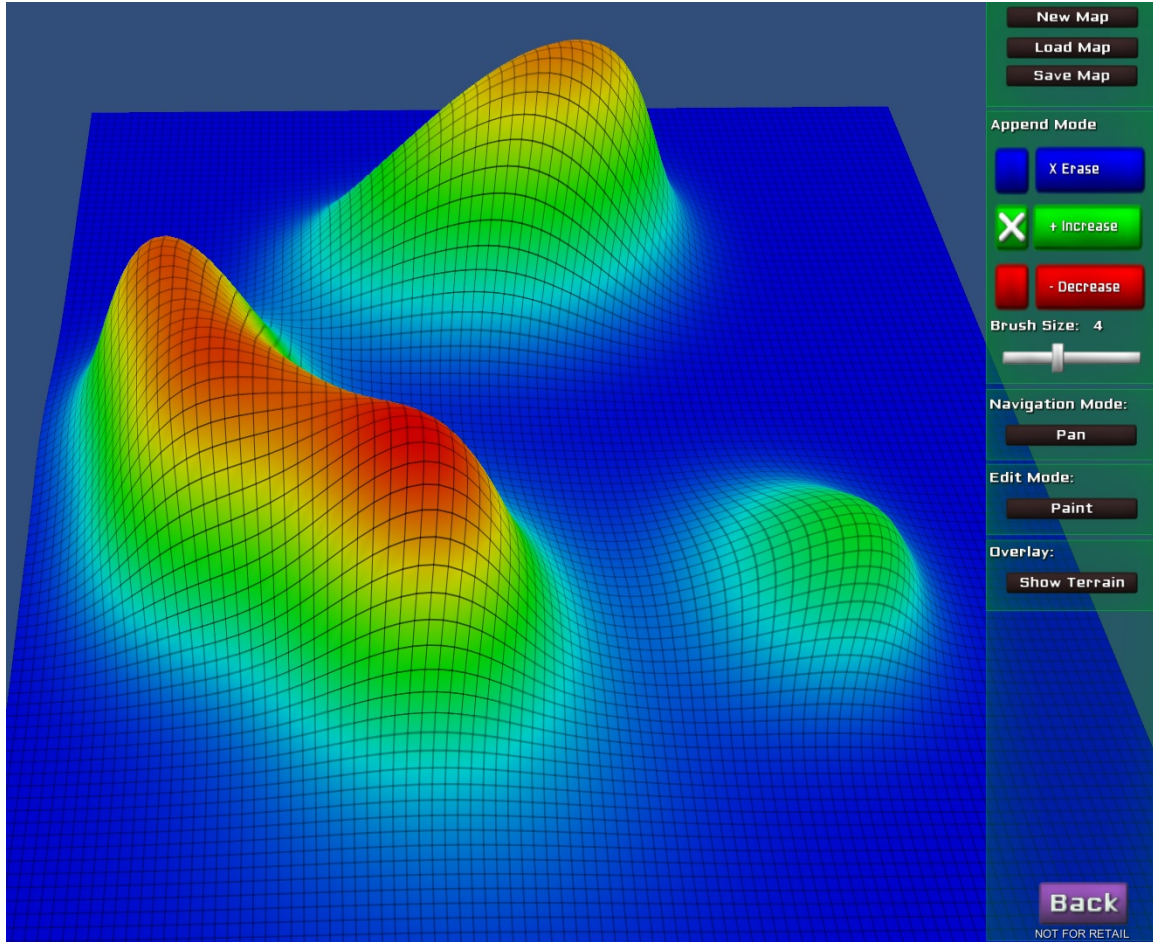


Figure 7.9: Examples of elliptically-symmetric and bivariate Gaussians and asymmetric bivariate distributions approximated using the **DistEdit** tool.

be added to the systematically generated *probability distribution map* from the **strategic** scale. The black circle shows the brush size.

If **Decrease** is selected, the effect is just the reverse of **Increase**. Instead of adding a Gaussian to the existing distribution map, a Gaussian is subtracted. The mean, standard deviation, and the scale of the Gaussian is determined the same way as mentioned above. The small basin in the middle of Figure 7.7 shows an example of a Gaussian subtracted from the *probability distribution map*.

The blue circle on the lower left part of the map in Figure 7.7 shows the paintbrush cursor projected onto the 3D surface from above, so the user can see directly the brush size with respect to the entire map. The color of the circle matches the color-coded buttons in



the control panel, so it is easy to tell which action (erase, increase, and decrease) is currently selected. The user can move the brush size slider in the control panel (see the right side of Figure 7.4) to select a desired brush size between 1 and 10. The probability hill in Figure 7.10 is created with brush size 10.

It is worth mentioning that although in the **paint** mode, only circularly-symmetric bivariate Gaussians (standard bivariate Normals) can be added to or subtracted from the *probability distribution map*, elliptically-symmetric bivariate Gaussians and asymmetric bivariate distributions can also be approximated using the paintbrush tool. For example, the user can create three circularly-symmetrical Gaussians where the means of these Gaussians are on a straight line with equal distance, the two Gaussians on the outside have identical scales, and the Gaussian in the middle have a larger scale. The mixture of these three Gaussians approximates the shape of an elliptically-symmetrical bivariate Gaussian. The user can also move the cursor in a straight line but in varying speed while adding a Gaussian to the map to approximate asymmetric bivariate distributions. Figure 7.9 shows examples of such approximations.

Similar to the function in the **DiffEdit** tool, in the **select** mode the user can drag a freehand selection around the desired area. The tool will automatically connect the starting point and the end point of the line to form a closed selection. Probability in the selected area is automatically erased. The white line in Figure 7.10 shows an area selected by a user. When satellite imagery of the search area is overlaid on top of the *probability distribution map*, the user can zoom in using 3D controls and then trace an area on the satellite imagery using this method.

With both the **paint** mode and the **select** mode, when a touch-screen device is used, the user can use a finger to paint or select on top the *task-difficulty map*.

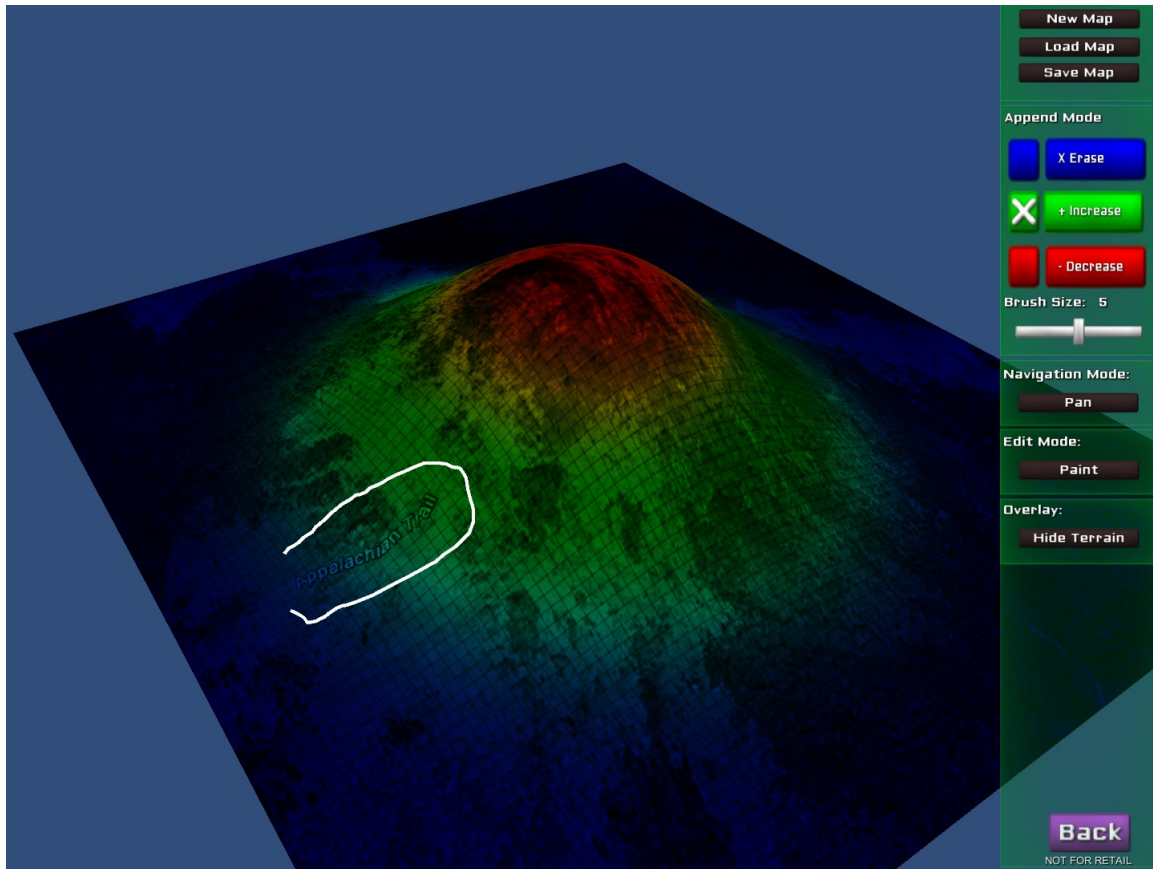


Figure 7.10: An example *probability distribution map* with the satellite imagery overlaid on top where an area is selected in the lasso **select** mode.

### 7.3.4 Cross-Platform Support

Both the **DiffEdit** tool and the **DistEdit** tool are developed using the free version of the Unity Game Engine. This means that these tools are cross platform and can support Windows, Mac, Unix, and can be ported to mobile devices such as iPhone, iPad, Android tablets, and Android phones. Both tools also have web versions that can run under typical web browsers such as Chrome, Firefox, IE, and Safari. The platform independence adds value to these tools and make it more feasible to integrate the tools into real WiSAR operations.

## Chapter 8

### Conclusions and Future Work

The research presented in this dissertation develops and evaluates tools that allow users to manage autonomy by managing information. This autonomy management approach is applied to the application domain of using a UAV to support Wilderness Search and Rescue. Autonomous components and autonomy management tools operate at three distinctive temporal scales, **strategic**, **between-episodes**, and **within-episode**. These scales represent three different temporal spans that are relevant to many problems: a long-term span that uses data and models from similar problems generates behavior that exploits common trends; a medium-term span that uses data and models obtained from prior attempts to solve the specific task to shape a new attempt to solve the problem; and a short-term span that uses real-time data and insights obtained from performing the task to shape the behavior of the autonomy moving forward, respectively. By managing two information representations, a *probability distribution map* and a *task-difficulty map*, at each temporal scale, the domain expert user becomes an “intelligent sensor”, “digesting” information from various sources and then feeding the “filtered” information to the system in forms the system can understand. Doing so enables the user to influence the behavior of the autonomous subsystems without understanding the statistical models or complex algorithms used in the autonomous components.

## 8.1 Conclusions

Challenges of integrating autonomy into an intelligent system can be characterized along two dimensions: *attributes of an intelligent system* (capability, information management, performance evaluation) and *organizational scale* (individual, collaborative agents, distributed system). These attributes can serve as guidelines when designing autonomous components and autonomy management tools.

We proposed a new autonomy management approach where the user influenced the behavior of an autonomous system (or subsystem) by hierarchically managing information at different temporal scales through two information representations: a *probability distribution map* and a *task-difficulty map*. We designed autonomous components and autonomy management tools for a wilderness search and rescue problem, and tailored the autonomy and tools so that they could operate across the different temporal scales. We then presented evidence that these tools enabled the user to create and modify these maps by incorporating their domain knowledge and information.

The key to making it possible for the autonomy algorithms to work across temporal scales was to endow them with the ability to use the information provided by the human to create real-time plans. These plans were then presented to the human, allowing him or her to modify the plan by modifying the type of information provided to the planner. For the wilderness search and rescue problem, this required the creation of multiple real-time UAV-based path planning algorithms that used the maps as input.

Human-interaction with the path planners came in two forms: modifying the maps and providing constraints on the autonomy. The maps are consistent with a Bayesian approach to decision making, yielding algorithms that outperform the state-of-the-art approaches for problems that require real-time feedback and support partial object detection. Constraints were implemented using a sliding autonomy interface where the user influenced the planner by specifying path segment endpoint constraints (spatial) and flight segment durations (temporal).

	Probability Distribution Map	Task-Difficulty Map
Strategic	<b>DistCreate</b> for map creation	<b>DiffCreate</b> for map creation
Between-Episodes	<b>DistEdit</b> for map update and info management	<b>DiffEdit</b> for map update and info management
Within-Episode	<b>IPPA path planning algorithms</b> that support: <b>real-time feedback</b> and <b>partial detection</b>	
	<b>SlidingAutonomy</b> interface for autonomy management	

Figure 8.1: Autonomous components and autonomy management tools of the dissertation work at each temporal scale/hierarchy.

A user study provided evidence that this autonomy management approach enabled the human-autonomy team to outperform the human or autonomy working alone. Additionally, the user study provided evidence that this performance increase was accompanied by a reduction in human cognitive workload and an improvement in the human's perception of the human-automation experience.

## 8.2 Future Work

This section presents a few of the natural extensions from current research that can be pursued as future work. We list them in the order of the three temporal scales and show how they relate to the various components of the dissertation work (see Figure 8.1).

### 8.2.1 At the Strategic Scale

Presently, the **DistCreate** component uses a set of default prior belief parameters (transitional probabilities between pairs of terrain features) that can be changed by the domain expert user based on the user's domain expertise and information only the user

can interpret. It would be beneficial if the system could automatically suggest transitional probability values based on statistics from past incidents [60] after the user provides some initial lost person profile information (such as age, gender, profession, etc.). When the user wants to modify the suggested parameters, instead of typing in values, it would be helpful to provide a tool that allows the user to visually see the prior belief distributions. By moving two sliders to change the mean and standard deviation values, the user could see how the shapes of the distributions changes. Ideally, the user could also see how this affects the shapes of the final prior/posterior predictive probability distributions with instant feedback. This immediate visual feedback would allow the user to understand causal effects and therefore help the user form a mental model of the system that is similar to how the system truly works. Figure 8.2 shows a mock up screen of such a tool. Computationally, instant feedback would require that we perform complex matrix computations on the GPU using CUDA (Compute Unified Device Architecture) architecture. To evaluate how this would affect the human-autonomy interaction and the performance of the human-autonomy team, a user study could be performed.

The **DistCreate** component considers three terrain features: topography, elevation, and vegetation density. It is probably beneficial to incorporate more factors that affect lost-person behaviors into the network. Such factors include but are not limited to direction of travel, trail following, missing person profile, panicking factor, weather conditions and season of the year. Such factors could be included into the existing Bayesian network as prior nodes. Additional utility tools might be needed to generate geographic data when it is not directly available. For example, if trail data cannot be automatically extrapolated, such utility tools would allow the search to manually mark trails on a map.

Currently the user can only specify whether to use past-human behavior data or not with the **DistCreate** component. In the future, as more past-human behavior data become available, the data could be stored in a database, and the user could further decide which subset of the behavior data to use by querying the database. For example, the user could

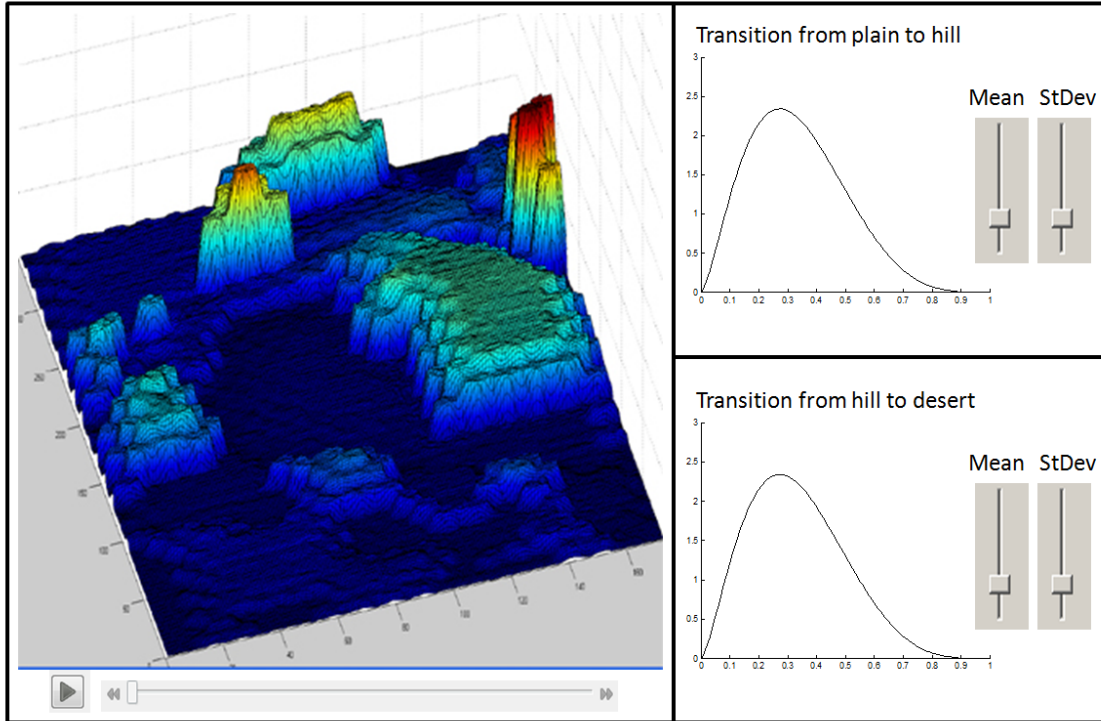


Figure 8.2: A mock up screen for the management tool interface at the strategic scale.

choose to only use data of the same search region, season of the year, or only data from people who have similar profile (age, gender, profession, etc.) with the missing person.

The **DiffCreate** component uses vegetation coverage data from USGS satellite imagery data to extrapolate vegetation density and determines task difficulty (detection probably). A more advanced model could be designed to include additional factors such as UAV height above ground, time of the day, season of the year, “seeability” [82], and sensor specific properties (e.g., an infrared multi-spectrum camera). Additional constraints such as no-fly zone or dangerous areas could also be included in the *task-difficulty map*.

### 8.2.2 At the Between-Episodes Scale

The **DistEdit** and **DiffEdit** components at this temporal scale let the user use mouse and finger gestures to edit the *probability distribution map* and the *task-difficulty map* generated from the strategic scale in a 3D environment. In this dissertation we only validated

the usefulness of these two tools by demonstration. To better evaluate the usefulness of the two tools, a well-designed user study should be performed. Ideally, these tools should be given to real searchers and rescuers to generate maps for real WiSAR scenarios. The user should be able to modify these maps as new information becomes available. For example, the maps should be modified when a piece of clothing or candy wrapper is found during the search, or when ground searchers have thoroughly searched an area and confirmed that the missing person is unlikely located at certain regions. These tools could also be integrated with the Sliding Autonomy component so the two maps can be updated in real time while the UAV is in the air during the mission.

### 8.2.3 At the Within-Episode Scale

At this temporal scale we designed multiple intelligent path planning algorithms that support real-time feedback and partial detection. These algorithms assume the missing person is stationary and the probability distribution map and the task-difficulty map are static. Because these algorithms are very fast, they should be improved to deal with moving targets, changing probability distributions and a task-difficulty map that changes over time. Then to further expand the problem, these algorithms could be improved to work with multiple targets or support multiple UAVs. We leave these to future work.

The **SlidingAutonomy** interface allows the user to affect the behavior of the path planning autonomy by setting temporal and spatial constraints. The user study we performed was a short-term study where each user had only minimal training before using this interface. Because a human's trust in autonomy can change over time, it would be interesting to research how the user's trust gets calibrated when the user uses this autonomy management approach for a long period of time. Would the user be able to gradually identify the weaknesses of the path planning autonomy and remedy correctly? Would the user overtrust autonomy and perform worse in the long run? Or would the user undertrust autonomy because autonomy makes obvious mistakes in certain scenarios? In our user study, the two scenarios used are both



relatively easy scenarios. When more complex *probability distribution maps* and *task-difficulty maps* are used, the benefit of using the **SlidingAutonomy** interface might be more obvious, and it would be interesting to investigate how users would react to that. It is also possible to let the user specify the number of top regions through the **SlidingAutonomy** interface for the Top2 and TopN algorithms and see how that would affect the human-autonomy interaction. These questions can only be answered with a long-term user study, which we leave for future work.

At this temporal scale, while the UAV is in the air during mission, as information is collected and processed by the collective search and rescue team, situation could arise when the *probability distribution map* and/or the *task-difficulty map* become incorrect. If these two maps cannot be updated in real-time, how could the user use the **SlidingAutonomy** interface to influence path planning autonomy in order to address the information change? The user might want the UAV to avoid certain regions or force the UAV to visit other regions repeatedly. How well does the **SlidingAutonomy** interface support such interaction could be another interesting research topic. A user study could be performed to evaluate the human-autonomy interaction experience and the performance of this sliding autonomy approach.

#### 8.2.4 The Overall Autonomy Management Approach

In this dissertation we applied the proposed autonomy management approach to the application domain of using a UAV to support Wilderness Search and Rescue. It is worth hypothesizing that this approach could be generalized and applied to many application domains. For example, when using an assistive robot to help a therapist treat children with autism, robot autonomy could also be managed by hierarchically managing information at different temporal scales.

At the **strategic** scale, before a child with autism begins clinical treatments, the therapist performs a series of evaluations. Information collected is analyzed to determine the

deficiencies, then a treatment plan is created identifying areas the therapist should focus on (e.g., joint attention, turn taking). Here this *areas of focus* map is analogous to the *probability distribution map* we discussed before. It is possible to develop a model to assist the therapist in generating this initial plan at the general trend scale. The therapist could decide what model parameters and dataset to use to train the model or to affect the plan. Similarly, a *task-difficulty map* could be created identifying areas where the therapy treatment might not be very effective.

At the **between-episodes** scale, a child with autism may receive one or two treatments each week. At different stages of the treatment the therapist might focus on different areas or reinforce certain behaviors in each session. The therapist may prefer the robot to have exaggerated facial expression and movement in one session but appear calmer and more verbose in another; or the therapist might want the robot to demonstrate a higher degree of reliance on the therapist. Ideally, by adjusting the areas of focus and the task-difficulty map at the between-sessions scale, the therapist could take advantage of special knowledge or experience and indirectly affect the robot's autonomous behaviors by managing what information to provide.

At the **within-episode** scale, during a clinical session a child with autism might not behave as the therapist expects (due to fatigue, previous events, or unexpected events). The therapist needs to be able to manage the robot's autonomous behaviors in real-time in order to improve or maximize the potentials of the treatment. The ability to modify the areas of focus and the task-difficulty map in real time affords the therapist the desired levels of control. The therapist could also strategically plan out the order of activities (targeted to different deficiencies) and desired intensity (time allocated to each activity) during the session to improve the efficiency of the treatment. At the within-session scale, the therapist is really actively collecting information (the child's behavior and reaction), digesting the information, and then deciding what information to provide to the system, in forms the system can understand, in order to manage the autonomous behavior of the robot.

Applying our proposed autonomy management approach to a completely different application domain and then evaluating how well the approach generalized is an important part of future work.

## References

- [1] J.A. Adams, C.M. Humphrey, M.A. Goodrich, J.L. Cooper, B.S. Morse, C. Engh, and N. Rasmussen. Cognitive task analysis for developing unmanned aerial vehicle wilderness search support. *Journal of Cognitive Engineering and Decision Making*, 3(1):1–26, 2009.
- [2] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.
- [3] L. Bainbridge. Ironies of automation. *Automatica*, 19(6):775–780, 1983.
- [4] K.P. Balanda and H.L. MacGillivray. Kurtosis: a critical review. *The American Statistician*, 42(2):111–119, 1988.
- [5] R. Beard, D. Kingston, M. Quigley, D. Snyder, R. Christiansen, W. Johnson, T. McLain, and M.A. Goodrich. Autonomous vehicle technologies for small fixed-wing UAVs. *AIAA Journal of Aerospace Computing, Information, and Communication*, 2(1):92–108, 2005.
- [6] R.E. Bellman. Combinatorial processes and dynamic programming. In *Proceedings of the Tenth Symposium in Applied Mathematics of the American Mathematical Society*. Rand Corporation, 1960.
- [7] D.P. Bertsekas. *Dynamic programming and optimal control*, volume 2. Athena Scientific, Belmont, MA, 1995.
- [8] S.A. Bortoff. Path planning for UAVs. In *Proceedings of the American Control Conference*, volume 1, pages 364–368. IEEE, 2000.
- [9] F. Bourgault, T. Furukawa, and H.F. Durrant-Whyte. Coordinated decentralized search for a lost target in a Bayesian world. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.
- [10] F. Bourgault, A. Goktogan, T. Furukawa, and H.F. Durrant-Whyte. Coordinated search for a lost target in a Bayesian world. *Advanced Robotics*, 18(10):979–1000, 2004.

- [11] F. Bourgault, T. Furukawa, and H.F. Durrant-Whyte. Optimal search for a lost target in a Bayesian world. In *Field and Service Robotics*, pages 209–222. Springer, 2006.
- [12] F. Bourgault, A. Chokshi, and M. Compbell. Human-computer augmented nodes for scalable mobile sensor networks. In *Proceedings of the IEEE SMC International Conference on Distributed Human-Machine Systems*, 2008.
- [13] J.M. Bradshaw, M. Sierhuis, A. Acquisti, P. Feltovich, R. Hoffman, R. Jeffers, D. Prescott, N. Suri, A. Uszok, and R. Van Hoof. Adjustable autonomy and human-agent teamwork in practice: An interim report on space applications. *Agent Autonomy*, pages 243–280, 2003.
- [14] J.M. Bradshaw, P.J. Feltovich, H. Jung, S. Kulkarni, W. Taysom, and A. Uszok. Dimensions of adjustable autonomy and mixed-initiative interaction. *Agents and Computational Autonomy*, pages 235–268, 2004.
- [15] J.M. Bradshaw, R.R. Hoffman, M. Johnson, and D.D. Woods. The seven deadly myths of autonomous systems. *Intelligent Systems, IEEE*, 28(3):54–61, 2013.
- [16] J. Brookshire, S. Singh, and R. Simmons. Preliminary results in sliding autonomy for coordinated teams. In *Proceedings of The 2004 Spring Symposium Series*. Carnegie Mellon University the Robotics Institute, March 2004.
- [17] J. Casper and R.R. Murphy. Human-robot interactions during the robot-assisted urban search and rescue response at the World Trade Center. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 33(3):367–385, 2003.
- [18] E. Cawi, N. Jones, and C. R. Twardy. MapScore: Probability map evaluation for search & rescue. In *Virginia Search & Rescue Conference*. Conference Presentation presented at the Virginia Search & Rescue Conference, Holiday Lake, VA, 2012.
- [19] I. Chao, B.L. Golden, E.A. Wasil, et al. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88(3):475–489, 1996.
- [20] A.H.W. Chun. Optimizing limousine service with AI. In *Proceedings of the Innovative Applications of Artificial Intelligence Conference*. AAAI, 2010.
- [21] S. Clark and M.A. Goodrich. A hierarchical flight planner for sensor-driven UAV missions. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication*, pages 509–514. IEEE, 2013.

- [22] J.L. Cooper and M.A. Goodrich. Towards combining UAV and sensor operator roles in UAV-enabled visual search. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*, 2008.
- [23] J.W. Crandall and M.L. Cummings. Identifying predictive metrics for supervisory control of multiple robots. *IEEE Transactions on Robotics*, 23(5):942–951, 2007.
- [24] J.W. Crandall and M.A. Goodrich. Principles of adjustable interactions. In *AAAI Fall Symposium Human-Robot Interaction Workshop*, 2002.
- [25] M.B. Dias, B. Kannan, B. Browning, E.G. Jones, B. Argall, M.F. Dias, M. Zinck, M.M. Veloso, and A.J. Stentz. Sliding autonomy for peer-to-peer human-robot teams. In *Proceedings of the Intelligent Conference on Intelligent Autonomous Systems*, page 332, 2008.
- [26] G. Dorais and D. Kortenkamp. Designing human-centered autonomous agents. In *Advances in Artificial Intelligence. PRICAI 2000 Workshop Reader*, pages 321–324. Springer, 2001.
- [27] G.A. Dorais, R.P. Bonasso, D. Kortenkamp, B. Pelland, and D. Schrechenghost. Adjustable autonomy for human-centered autonomous systems on Mars. In *The First International Conference of the Mars Society*, 1998.
- [28] P.F. Drucker. *The Practice of Management*. Harper Paperbacks, NYC, NY, October 2006.
- [29] J.L. Drury, J. Scholtz, and H. Yanco. Awareness in human-robot interactions. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 912–918. IEEE, 2003.
- [30] D. Feillet, P. Dejax, and M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205, 2005.
- [31] D. Ferguson. GIS for wilderness search and rescue. In *ESRI Federal User Conference*, February 2008.
- [32] M. Fischetti. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148, Feb 1998.
- [33] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

- [34] T. Fong, C. Thorpe, and C. Baur. Collaborative control: A robot-centric model for vehicle teleoperation. In *AAAI Spring Symposium: Agents with Adjustable Autonomy*, 1999.
- [35] A. Gelman, J.B. Carlin, H.S. Stern, and D.B. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC, Boca Raton, FL, 2nd edition, 2004.
- [36] B.L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.
- [37] M.A. Goodrich and E.R. Boer. Multiple mental models, automation strategies, and intelligent vehicle systems. In *Proceedings of the IEEE/IEEEJ/JSAI International Conference on Intelligent Transportation Systems*, pages 859–864, 1999.
- [38] M.A. Goodrich and E.R. Boer. Model-based human-centered task automation: A case study in ACC system design. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 33(3):325–336, 2003.
- [39] M.A. Goodrich and A.C. Schultz. Human-robot interaction: A survey. *Foundations and Trends in Human-Computer Interaction*, 1(3):203–275, 2007.
- [40] M.A. Goodrich, J.L. Cooper, J.A. Adams, C. Humphrey, R. Zeeman, and B.G. Buss. Using a mini-UAV to support wilderness search and rescue: Practices for human-robot teaming. In *Proceedings of the IEEE International Workshop on Safety, Security, and Rescue Robotics*, 2007.
- [41] M.A. Goodrich, B.S. Morse, D. Gerhardt, J.L. Cooper, M. Quigley, J.A. Adams, and C.M. Humphrey. Supporting wilderness search and rescue using a camera-equipped miniUAV. *Journal of Field Robotics*, 25(1-2):89–110, 2008.
- [42] M.A. Goodrich, B.S. Morse, C. Engh, J.L. Cooper, and J.A. Adams. Towards using unmanned aerial vehicles (UAVs) in wilderness search and rescue: Lessons from field trials. *Interaction Studies*, 10:453–478(26), 2009.
- [43] B. Gorman, C. Thureau, C. Bauckhage, and M. Humphrys. Bayesian imitation of human behavior in interactive computer games. In *Proceedings of the International Conference on Pattern Recognition*, pages 1244–1247, 2006.
- [44] M.R. Gupta and Y. Chen. *Theory and Use of the EM Algorithm*. Now Publishers Inc, Hanover, MA, 2011.

- [45] G. Gutin and A.P. Punnen. *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers, Dordrecht, the Netherlands, 2002.
- [46] S.R. Hansen, T.W. McLain, and M.A. Goodrich. Probabilistic searching using a small unmanned aerial vehicle. In *AIAA Infotech@Aerospace*, 2007.
- [47] M.A. Hearst. Mixed-initiative interaction. *Intelligent Systems, IEEE*, 14(5):14–23, 1999.
- [48] C.D. Heth and E.H. Cornell. Characteristics of travel by persons lost in Albertan wilderness areas. *Journal of Environmental Psychology*, 18(3):223–235, September 1998.
- [49] K.A. Hill. *Lost Person Behavior*, chapter The Psychology of Lost. National SAR Secretariat, Ottawa, Canada, 1998.
- [50] R.R. Hoffman, M. Johnson, J.M. Bradshaw, and A. Underbrink. Trust in automation. *Intelligent Systems, IEEE*, 28(1):84–88, 2013.
- [51] R.C. Holte, M.B. Perez, R.M. Zimmer, and A.J. MacDonald. Hierarchical A\*: Searching abstraction hierarchies efficiently. In *AAAI/IAAI, Vol. 1*, pages 530–535, 1996.
- [52] P.S. Horn. A measure for peakedness. *The American Statistician*, 37(1):55–56, 1983.
- [53] J.K. Howlett, T.W. McLain, and M.A. Goodrich. Learning Real-Time A\* path planner for unmanned air vehicle target sensing. *Journal of Aerospace Computing, Information, and Communication*, 3(3):108–122, 2006.
- [54] C.M. Humphrey. *Information abstraction visualization for human-robot interaction*. PhD thesis, Vanderbilt University, 2009.
- [55] V.E. Johnson. A Bayesian  $\chi^2$  test for Goodness-of-Fit. *The Annals of Statistics*, 32(6): 2361–2384, 2004.
- [56] T.R. Jorris and R.G. Cobb. Three-dimensional trajectory optimization satisfying waypoint and no-fly zone constraints. *Journal of guidance, control, and dynamics*, 32(2):551–572, 2009.
- [57] D.B. Kaber, J.M. Riley, K.W. Tan, and M.R. Endsley. On the design of adaptive automation for complex systems. *International Journal of Cognitive Ergonomics*, 5(1): 37–57, 2001.



- [58] D.B. Kaber, M.C. Wright, L.J. Prinzel III, and M.P. Clamann. Adaptive automation of human-machine system information-processing functions. *Human Factors*, 47(4):730, 2005.
- [59] A. Khurshid, E. Hussain, and Masood ul Haq. A note on finding peakedness in bivariate normal distribution using Mathematica. *Pakistan Journal of Statistics and Operation Research*, 3(2):75–86, 2007.
- [60] R.J. Koester. *Lost Person Behavior: A search and rescue guide on where to look - for land, air and water*. dbS Productions LLC, Charlottesville, VA, August 2008.
- [61] B.O. Koopman. The theory of search. II: Target detection. *Operations Research*, 4(5): 503–531, 1956.
- [62] B.O. Koopman. The theory of search. III: The optimum distribution of searching effort. *Operations Research*, 5(5):613–626, 1957.
- [63] B.O. Koopman. *Search and Screening: General Principles with Historical Applications*. Pergamon Press, Oxford, NY, 1980.
- [64] G. Laporte. The vehicle routing problem: An overview of the exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.
- [65] G. Laporte and S. Martello. The selective travelling salesman problem. *Discrete Applied Mathematics*, 26(2-3):193–207, 1990.
- [66] J.D. Lee and K.A. See. Trust in automation: Designing for appropriate reliance. *Human Factors*, 46(1):50, 2004.
- [67] Y. Liang and A.E. Smith. An ant colony approach to the orienteering problem. *Journal of the Chinese Institute of Industrial Engineers*, 23(5):403–414, 2006.
- [68] L. Lin and M.A. Goodrich. A Bayesian approach to modeling lost person behaviors based on terrain features in wilderness search and rescue. In *Proceedings of the Conference on Behavior Representation in Modeling and Simulation*, pages 49–56, 2009.
- [69] L. Lin and M.A. Goodrich. UAV intelligent path planning for wilderness search and rescue. In *Proceedings on the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 709–714, 2009.

- [70] L. Lin and M.A. Goodrich. A Bayesian approach to modeling lost person behaviors based on terrain features in wilderness search and rescue. *Computational and Mathematical Organization Theory*, 16(3):300–323, 2010.
- [71] L. Lin, M. Roscheck, M.A. Goodrich, and B.S. Morse. Supporting wilderness search and rescue with integrated intelligence: Autonomy and information at the right time and the right place. In *Proceedings of the AAAI Conference on Artificial Intelligence, Special Track on Integrated Intelligence*, 2010.
- [72] L. Lin, M.A. Goodrich, and S. Clark. Hierarchical heuristic search using a Gaussian Mixture Model for UAV coverage planning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 2014. Submitted, under 2nd round of review.
- [73] K. Mardia. Measures of multivariate skewness and kurtosis with applications. *Biometrika*, 57(3):519, 1970.
- [74] N. Meuleau and R.I. Brafman. Hierarchical heuristic forward search in stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 7, page 2542, 2007.
- [75] F. Michaud, C. Côté, D. Letourneau, Y. Brosseau, J.M. Valin, E. Beaudry, C. Raievsky, A. Ponchon, P. Moisan, P. Lepage, et al. Spartacus attending the 2005 AAAI conference. *Autonomous Robots*, 22(4):369–383, 2007.
- [76] T.M. Mitchell. *Machine Learning*. McGraw-Hill, NYC, NY, 1997.
- [77] J. Mittenthal and C.E. Noon. An insert/delete heuristic for the travelling salesman subset-tour problem with one additional constraint. *The Journal of the Operational Research Society*, 43(3):277–283, 1992.
- [78] N. Moray. Designing for transportation safety in the light of perception, attention, and mental models. *Ergonomics*, 33(10):1201–1213, 1990.
- [79] N Moray. A lattice theory approach to the structure of mental models. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 327(1241):577–583, 1990.
- [80] N. Moray. *Attention and Performance XVII Cognitive Regulation of Performance: Interaction of Theory and Application*, chapter Mental Models in Theory and Practice, pages 223–258. The MIT Press, Cambridge, MA, 1999.

- [81] B.S. Morse, D. Gerhardt, C. Engh, M.A. Goodrich, N. Rasmussen, D. Thornton, and D. Eggett. Application and evaluation of spatio temporal enhancement of live aerial video using temporally local mosaics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.
- [82] B.S. Morse, C.H. Engh, and M.A. Goodrich. UAV video coverage quality maps and prioritized indexing for wilderness search and rescue. In *Proceeding of the ACM/IEEE International Conference on Human-Robot Interaction*, pages 227–234, 2010.
- [83] R.R. Murphy, E. Steimle, C. Griffin, C. Cullins, M. Hall, and K. Pratt. Cooperative use of unmanned sea surface and micro aerial vehicles at hurricane Wilma. *Journal of Field Robotics*, 25(3):164–180, 2008.
- [84] M. Naveed, D.E. Kitchin, and A. Crampton. A hierarchical task network planner for pathfinding in real-time strategy games. In *Proceedings of the International Symposium on AI & Games*, pages 1–7, 2010.
- [85] P. Niedfeldt, R. Beard, B.S. Morse, and S. Pledgie. Integrated sensor guidance using probability of object identification. In *Proceedings of the American Control Conference*, pages 788–793, 2010.
- [86] D. Norman. *Some Observations on Mental Models*. Lawrence Erlbaum Associates, Mahwah, NJ, 1983.
- [87] N.M. Oliver, B. Rosario, and A.P. Pentland. A Bayesian computer vision system for modeling human interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):831–843, 2000.
- [88] D.R. Olsen. Evaluating user interface systems research. In *Proceedings of the ACM Symposium on User interface Software and Technology*, pages 251–258, 2007.
- [89] D.R. Olsen and M.A. Goodrich. Metrics for evaluating human-robot interactions. In *Proceedings of PERMIS*, volume 2003, 2003.
- [90] R. Parasuraman, T.B. Sheridan, and C.D. Wickens. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 30(3):286–297, 2000.
- [91] P.O. Pettersson and P. Doherty. Probabilistic roadmap based path planning for an autonomous unmanned helicopter. *Journal of Intelligent and Fuzzy Systems*, 17(4): 395–405, 2006.

- [92] M. Quigley, B. Barber, S. Griffiths, and M.A. Goodrich. Towards real-world searching with fixed-wing mini-UAVs. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.
- [93] R. Ramesh, Y. Yoon, and M.H. Karwan. An optimal algorithm for the orienteering tour problem. *ORSA Journal on Computing*, 4(2), Spring 1992.
- [94] J. Rasmussen, A.M. Pejtersen, and L.P. Goodstein. *Cognitive Systems Engineering*. Wiley-Interscience, Hoboken, NJ, 1994.
- [95] G. Righini and M. Salani. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & Operations Research*, 36(4):1191–1203, 2009.
- [96] B. Robins, K. Dautenhahn, and P. Dickerson. From isolation to communication: A case study evaluation of robot assisted play for children with autism with a minimally expressive humanoid robot. In *Proceedings of the International Conferences on Advances in Computer-Human Interactions*, pages 205 –211, 2009.
- [97] W.B. Rouse. Adaptive aiding for human/computer control. *The Journal of the Human Factors and Ergonomics Society*, 30(4):431–443, 1988.
- [98] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, Upper Saddle River, NJ, 2009.
- [99] A. Ryan and J.K. Hedrick. Particle filter based information-theoretic active sensing. *Robotics and Autonomous Systems*, 58(5):574–584, 2010.
- [100] E. Salas and S.M. Fiore. *Team cognition: Understanding the factors that drive process and performance*. American Psychological Association, Washington, DC, 2004.
- [101] Adam Sanborn and Thomas Griffiths. Markov Chain Monte Carlo with people. In *Advances in Neural Information Processing Systems*, pages 1265–1272, 2008.
- [102] N. Sarter. Making coordination effortless and invisible: The exploration of automation management strategies and implementations. In *CBR Workshop on Human Interaction with Automated Systems*, 1998.
- [103] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2): 461–464, 1978.

- [104] T.J. Setnicka. *Wilderness Search and Rescue*. Appalachian Mountain Club, Boston, MA, 1980.
- [105] T.B. Sheridan. *Telerobotics, automation, and human supervisory control*. The MIT Press, Cambridge, MA, 1992.
- [106] T.B. Sheridan and W.L. Verplank. Human and computer control of undersea teleoperators. Technical report, MIT, Cambridge, MA, 1978.
- [107] R. Simmons, D. Goldberg, A. Goode, M. Montemerlo, N. Roy, B. Sellner, C. Urmson, A. Schultz, M. Abramson, W. Adams, et al. Grace: An autonomous robot for the AAI robot challenge. *AI Magazine*, 24(2):51, 2003.
- [108] M. Sniedovich. *Dynamic programming: Foundations and principles*. CRC press, Boca Raton, FL, 2010.
- [109] P.R. Sokkappa. *The Cost-Constrained Traveling Salesman Problem*. Doctoral dissertation, University of California, Berkeley, 1990.
- [110] M. Sonka, V. Hlavac, and R. Boyle. *Image processing, analysis, and machine vision*. Thomson-Engineering, Southwick, MA, 2007.
- [111] E. Soylemez and N. Usul. Utility of GIS in search and rescue operations. In *ESRI Users Group Conference*, 2006.
- [112] D.J. Spiegelhalter, N.G. Best, B.P. Carlin, and A. Linde. Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 64(4):583–639, 2002.
- [113] A. Steinfeld, T. Fong, D. Kaber, M. Lewis, J. Scholtz, A. Schultz, and M.A. Goodrich. Common metrics for human-robot interaction. In *Proceedings of the ACM SIGCHI/SIGART Conference on Human-Robot Interaction*, pages 33–40, 2006.
- [114] A. Stentz. Optimal and efficient path planning for partially known environments. In *Intelligent Unmanned Ground Vehicles*, pages 203–220. Springer, 1997.
- [115] L.D. Stone. *Theory of Optimal Search*. Academic Press, NYC, NY, 1975.
- [116] L.D. Stone, C.M. Keller, T.M. Kratzke, and J.P. Strumpfer. Search analysis for the underwater wreckage of Air France flight 447. In *Proceedings of the International Conference on Information Fusion*, pages 1–8, 2011.

- [117] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, Cambridge, UK, 1998.
- [118] K. Sycara. Integrating agents into human teams. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 46, pages 413–417, 2002.
- [119] W.G. Syrotuck. *Analysis of Lost Person Behavior*. Barkleigh Productions, Mechanicsburg, PA, 2000.
- [120] W.G. Syrotuck. *An Introduction to Land Search Probabilities and Calculations*. Barkleigh Productions, Mechanicsburg, PA, 2000.
- [121] R. Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2(1):104, 2006.
- [122] M.F. Tasgetiren and A.E. Smith. A genetic algorithm for the orienteering problem. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 2, pages 910–915, 2000.
- [123] J.B. Tenenbaum. Bayesian modeling of human concept learning. In *Advances in Neural Information Processing Systems*, pages 59–65, 2000.
- [124] D. Thornton, B.S. Morse, and M.A. Goodrich. Unusual-object detection in color video for wilderness search and rescue. Master’s thesis, Brigham Young University, 2011.
- [125] S. Thrun, W. Burgard, D. Fox, et al. *Probabilistic Robotics*, volume 1. The MIT Press, Cambridge, MA, 2005.
- [126] K. E. Trummel and J. R. Weisinger. Technical note – the complexity of the optimal searcher path problem. *Operations Research*, 34(2):324–327, 1986.
- [127] K.S. Tso, G.K. Tharp, W. Zhang, and A.T. Tai. A multi-agent operator interface for unmanned aerial vehicles. In *Proceedings of the Digital Avionics Systems Conference*, volume 2, pages 6–A, 1999.
- [128] P. Vansteenwegen, W. Souffriau, and D.V. Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011.
- [129] K. Vicente. Should an interface always match the operators mental model? *CSERIAC Gateway*, 8(1):1–5, 1997.
- [130] A.R. Washburn. Search and detection, military applications section. *Operations Research Society of America, Ketron, Inc., Arlington, Va*, 1981.

- [131] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.
- [132] D.D. Woods and E. Hollnagel. *Joint cognitive systems: Patterns in cognitive systems engineering*. CRC Press, Boca Raton, FL, 2006.

# Appendix A

## Complexity Analysis of the UAV Path Planning Problem

In this appendix, we analyze why a heuristic approach is preferred to a dynamic programming / reinforcement learning approach by comparing the computational complexity of each approach.

### A.1 Computational Complexity of Dynamic Programming

Classical dynamic programming (DP) is a method for solving complex problems by breaking them down into simpler subproblems. Solution to subproblems can be stored to trade space for time. Problems that can be solved by DP must exhibit two key attributes: optimal substructure and overlapping subproblems. When DP is used to solve the traveling salesman problem (TSP), the complexity is still  $O(2^n n^2)$  [6]. DP method suffers the “curses of dimensionality” and does not scale well with complex problems.

The key issue is that the path planning problem that we are solving cannot be solved in polynomial time using dynamic programming unless  $P=NP$ <sup>1</sup>. The justification for this is as follows:

- The problem that we are solving is at least as computationally complex as what is known as the Orienteering Problem.
- The Orienteering Problem is at least as computationally complex as the Traveling Salesperson Problem.

---

<sup>1</sup>The path planning problem we are solving is similar to the Orienteering Problem (OP), which can be seen as a combination between the Knapsack Problem (KP) and the Traveling Salesman Problem (TSP) [128]. In a fully connected graph where each vertex has a certain score (prize), with fixed starting vertex and end vertex, the OP problem asks for the path that would achieve the highest score within a given time frame. That is why the OP problem is also called the Prize-Collecting TSP [45] problem and TSP with profits [30]. Scores are entirely additive and each vertex can only be visited once (not all vertices have to be visited). OP, like TSP, is an NP-hard problem.

Our path planning problem is different from the OP problem in the following aspects: (a) we use a grid representation because this is compatible with UAV flight, so the search is on a graph that is not a complete graph (fully connected); (b) the Bayesian sensing of the UAVs sensors require that the path planning be able to visit the same vertex repeatedly; and (c) this means that the “prize” rewarded for each visit to a vertex is only partially collected and is path dependent.



- The Traveling Salesperson Problem is in the complexity class f-NP, with its corresponding decision problem in the class NP-complete.
- By reduction, this means that our path planning problem is NP-hard and cannot be solved in polynomial time unless P=NP.
- Because the path-planning problem is NP-hard, we cannot solve it using dynamic programming in real time for the planning lengths that we consider (up to 900 planning steps).

Our path planning problem has a state space of 10,000 nodes and a flight path of 900 (possibly higher in real application) time steps, meaning that theoretically the same node could be visited 900 times. If we treat each visit to the same node as a separate node, the state space expands to 9,000,000 nodes, and tracking the connectivity of all these nodes (not complete) also becomes intractable.

In order to support real WiSAR operations, we need to have the path created within seconds. Also in practical Wilderness Search and Rescue scenarios, the search area could be much bigger than the 10,000 nodes we demonstrated. The UAV flight time can also be much longer depending on the type of UAV platforms used. That's why we chose a heuristic approximation approach in solving this problem. The complexity of our approach is  $O(n)$  once we have the Mode Goodness Ratio (MGR) heuristic [72], where  $n$  is the flight duration in time steps ( $n=900$  in our scenarios). This means our approach is very fast and scales very nicely with the NP-hard problem.

## A.2 Computational Complexity of Reinforcement Learning (Approximate Dynamic Programming)

Instead of solving for the exact solution, approximate dynamic programming / reinforcement learning (ADP/RL) are approximate methods to search for solutions that approximate the optimal solution for complex problems to avoid the “curses of dimensionality”. ADP/RL methods have four main sub-elements: a policy, a reward function (immediate payoff), a value function (long-term payoff), and optionally, a model of the environment. A policy defines the learning agent's behavior at a given time, a reward function defines the goal and indicates what is good in an immediate sense, a value function specifies what is good in the long run, and, the model of the environment mimics the behavior of the environment. The idea is to learn the optimal policy iteratively for each state, balancing exploration and exploitation. ADP/RL methods use Markov Decision Process (MDP) and can work with problems that have uncertainty in transition.

Reinforcement learning cannot learn an optimal solution to an NP-hard problem in polynomial time<sup>2</sup>. If it could, then  $P=NP$ . Moreover, reinforcement learning typically requires many, many iterations to reach convergence even for moderately sized problems, meaning it is likely to be significantly slower. Even dynamic programming-based approaches to reinforcement learning, like learning the transition model and applying policy iteration, cannot run in polynomial time on an exponential problem.

In addition to this fundamental limitation of what reinforcement learning can theoretically do, there is a second practical problem with using reinforcement learning for this problem. This practical limitation is that reinforcement learning approaches tend to get stuck in local minimal when there are multiple rewards in the problem. Indeed, the literature includes many papers that seek to resolve this problem by trading off exploration and exploitation [117]. These approaches work in practice for some problems, but there is no evidence that these approaches will work for problems that are exponentially complex.

Moreover, the state space of our path-planning problem grows exponentially because of the possibility of revisiting states. For each revisit, a new reward function must be defined because the Bayesian approach allows for partial collection of information. The reward function and the value function both become path dependent. This means that we end up with an exponentially hard problem with an exponentially large state space and a unique reward for each element of the state space. There is no known reinforcement learning algorithm that can solve such problems, let alone solve it in real-time.

---

<sup>2</sup>The approximate dynamic programming / reinforcement learning (ADP/RL) approach does not support (near) real-time solution. For example, Righini and Salanil show in [95] that it takes roughly 1000-3000 seconds to solve an OP type problem with 100 vertices/nodes. The ADP/RL approach also does not scale well due to its complexity. Vansteenwegen et al. surveyed different approaches to solving the OP [128]. Most of the approaches were heuristic approaches, and the only ADP/RL approach mentioned is [95].

## Appendix B

### Full Experiment Results for Chapter 5

Here we present the full experiment results of the four WiSAR scenarios described in Chapter 5. For each scenario, we generate paths with three flight durations ( $T = 300, T = 600,$  and  $T = 900$ ) and compare algorithm computation speed (in seconds) and path  $Efficiency_{LB}$  (in %) for BA, LHC-GW-CONV, Top2, and TopN algorithms (including Top2 and TopN algorithms where  $k = 5$  and  $N = 3$ ). All numbers shown are averages of 10 runs. Best performance results are displayed in bold font face. Standard deviations ( $\sigma$ ) are also shown for both computation speed and path  $Efficiency_{LB}$ .

Table B.1 shows the experiment results for the synthetic WiSAR scenario with a multi-modal distribution of the missing person location and a simple *task-difficulty map* with three difficulty levels (as shown in Fig. 5.2). The UAV path starts from a subregion with high task-difficulty (lower right corner).

	$T = 300$				$T = 600$				$T = 900$			
	Speed	$\sigma$	$E_{LB}$	$\sigma$	Speed	$\sigma$	$E_{LB}$	$\sigma$	Speed	$\sigma$	$E_{LB}$	$\sigma$
BA	-	-	27.59	-	-	-	43.54	-	-	-	59.56	-
LHC-GW-CONV	0.17	0.01	92.26	0.00	0.33	0.07	92.68	0.00	0.51	0.09	94.03	0.01
Top2 (1 layer)	0.12	0.05	87.49	0.03	0.14	0.04	91.66	0.04	0.15	0.04	91.02	0.03
TopN (1 layer)	<b>0.10</b>	0.05	91.28	0.02	<b>0.08</b>	0.05	91.93	0.04	<b>0.07</b>	0.03	95.24	0.01
Top2 (Hierarchy)	0.37	0.10	90.85	0.02	0.42	0.10	93.83	0.02	0.48	0.10	93.59	0.01
TopN (Hierarchy)	0.91	0.21	<b>92.27</b>	0.00	0.84	0.16	<b>95.50</b>	0.01	0.93	0.21	<b>95.56</b>	0.01

Table B.1: Algorithms speed and  $Efficiency_{LB}$  comparison for the multi-modal synthetic scenario.

Table B.2 shows the experiment results for the HikerPaul WiSAR scenario, in which an elderly couple was reported missing near the Grayson Highlands State Park in Virginia. Fig.5.10 shows the *probability distribution map* and the *task-difficulty map* for the scenario. Fig.5.11 shows example paths generated. Each UAV path starts from the Last Known Position (LKP), which is in the middle of the search region.

	$T = 300$				$T = 600$				$T = 900$			
	Speed	$\sigma$	$E_{LB}$	$\sigma$	Speed	$\sigma$	$E_{LB}$	$\sigma$	Speed	$\sigma$	$E_{LB}$	$\sigma$
BA	-	-	56.95	-	-	-	60.07	-	-	-	57.11	-
LHC-GW-CONV	0.30	0.16	60.18	0.13	0.47	0.03	56.76	0.00	0.98	0.16	55.18	0.00
Top2 (1 layer)	<b>0.24</b>	0.06	66.68	0.09	0.30	0.11	65.21	0.07	0.41	0.20	66.08	0.07
TopN (1 layer)	0.25	0.07	76.19	0.08	<b>0.24</b>	0.11	71.02	0.04	<b>0.22</b>	0.09	68.26	0.04
Top2 (Hierarchy)	0.73	0.11	78.67	0.03	0.84	0.14	73.81	0.04	1.19	0.36	72.75	0.02
TopN (Hierarchy)	1.52	0.15	<b>81.43</b>	0.03	1.73	0.25	<b>75.48</b>	0.02	1.68	0.26	<b>74.13</b>	0.02

Table B.2: Algorithms speed and  $Efficiency_{LB}$  comparison for the HikerPaul scenario.

Table B.3 shows the experiment results for the NewYork53 WiSAR scenario, in which a 46 year old male camper was reported missing near Adirondack Park in upperstate New York. Fig.5.13 shows the *probability distribution map* and the *task-difficulty map* for the scenario. Fig.5.14 shows example paths generated. Each path starts from the Last Known Position (LKP), which is in the middle of the search region.

	$T = 300$				$T = 600$				$T = 900$			
	Speed	$\sigma$	$E_{LB}$	$\sigma$	Speed	$\sigma$	$E_{LB}$	$\sigma$	Speed	$\sigma$	$E_{LB}$	$\sigma$
BA	-	-	39.95	-	-	-	54.27	-	-	-	65.08	-
LHC-GW-CONV	<b>0.01</b>	0.00	38.47	0.00	<b>0.02</b>	0.00	56.91	0.00	<b>0.02</b>	0.00	67.38	0.00
Top2 (1 layer)	0.75	0.15	54.42	0.04	0.92	0.60	66.61	0.03	0.81	0.55	72.79	0.02
TopN (1 layer)	0.70	0.45	59.15	0.07	0.77	0.55	68.78	0.04	0.69	0.30	74.54	0.01
Top2 (Hierarchy)	1.87	0.23	57.18	0.03	2.06	0.34	69.29	0.02	1.92	0.33	74.44	0.01
TopN (Hierarchy)	5.01	0.67	<b>65.39</b>	0.03	5.76	0.96	<b>71.47</b>	0.02	5.32	1.12	<b>77.36</b>	0.02

Table B.3: Algorithms speed and  $Efficiency_{LB}$  comparison for the NewYork53 scenario.

Table B.4 shows the experiment results for the NewYork53 WiSAR scenario, in which two teenage female hikers were reported missing near West Chesterfield in Massachusetts. Fig.5.15 shows the *probability distribution map* and the *task-difficulty map* for the scenario. Fig.5.16 shows example paths generated. Each path starts from the Last Known Position (LKP), which is in the middle of the search region.

	$T = 300$				$T = 600$				$T = 900$			
	Speed	$\sigma$	$E_{LB}$	$\sigma$	Speed	$\sigma$	$E_{LB}$	$\sigma$	Speed	$\sigma$	$E_{LB}$	$\sigma$
BA	-	-	39.92	-	-	-	45.34	-	-	-	49.39	-
LHC-GW-CONV	<b>0.01</b>	0.00	41.38	0.00	<b>0.45</b>	0.06	52.88	0.00	<b>0.02</b>	0.00	52.61	0.00
Top2 (1 layer)	0.98	0.33	58.37	0.04	0.90	0.36	54.18	0.02	1.44	0.65	57.33	0.02
TopN (1 layer)	0.92	0.38	54.03	0.06	0.83	0.56	53.91	0.07	0.97	0.42	57.91	0.03
Top2 (Hierarchy)	2.42	0.39	<b>60.73</b>	0.02	2.52	0.67	55.91	0.01	2.50	0.23	57.94	0.01
TopN (Hierarchy)	6.81	1.10	59.60	0.02	6.59	0.98	<b>60.26</b>	0.01	7.42	1.11	<b>60.99</b>	0.02

Table B.4: Algorithms speed and  $Efficiency_{LB}$  comparison for the NewYork108 scenario.

## Appendix C

### Hierarchical Decision Making and Hierarchical Coarse-to-Fine Search

In several parts of our dissertation work, we used hierarchical methods to solve various problems. In this appendix we describe two main areas where we applied the hierarchical methods: 1) Choosing the appropriate path planning algorithm depending on the scenario using hierarchical decision making. 2) Speeding up algorithm computation by hierarchically searching through the parameter space in algorithms design.

#### C.1 Hierarchical Decision Making in Choosing the Appropriate Path Planning Algorithm

We designed multiple intelligent path planning algorithms to tackle the UAV coverage path planning problem at hand. According to the No Free Lunch Theorem [131], “for any algorithm, any elevated performance over one class of problems is offset by performance over another class.” Each path planning algorithm performs well with certain type of scenarios, but might not perform well with other types of scenarios. Therefore, given a scenario, we use a hierarchical decision tree to choose the appropriate path planning algorithm.

At the top level, the total amount of UAV **flight time** is evaluated. If flight time (in time steps) is much larger (e.g, 10 times) than the size of the search area (in number of cells), the Complete-Coverage (CC) algorithm can be used to just exhaustively search the entire area with lawnmower patterns.

At the next level, the amount of **computation time** allowed is evaluated. If there’s no rush to generate a UAV flight path within seconds, the Evolutionary (EA-Path) algorithm can be used to iteratively improve the flight path. The EA-Path algorithm can generate a final path in about 30 seconds. If it is necessary to generate a UAV flight within a fraction of a second (e.g. in the Sliding Autonomy interface), then the LHC-GW-CONV algorithm or the CC algorithm can be used, because they are both very fast algorithms.

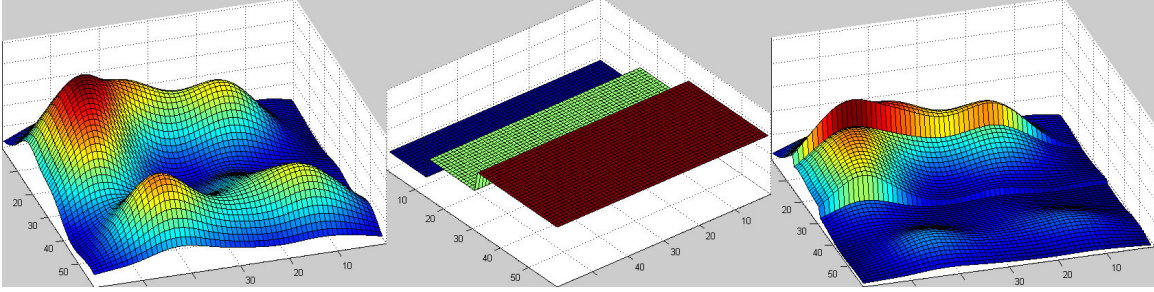


Figure C.1: A synthetic WiSAR scenario. Left: Multi-modal probability distribution. Middle: A simple *task-difficulty map*. Right: Probability collectible on first visit (combining *probability distribution* and *task-difficulty map*).

After combining the *probability distribution map* and the *task-difficulty map* (if one is used), we can compute a 3D surface indicating at each cell the amount of probability collectible when the UAV visit the cell the first time. The **shape of this surface** is considered at the next level of the decision tree. If the surface is completely flat like a uniform distribution, then the CC algorithm is the best candidate because a lawnmower pattern is the optimal path. If the surface is a unimodal surface, then the LHC-GW-CONV algorithm is selected, because it can generate a spiral-pattern path, which is optimal for this scenario. For a multi-modal surface, we move on to the next level of the decision tree.

At the last level, we check if a *task-difficulty map* is used. In other words, whether **partial detection** is considered. If the answer is no, then the LHC-GW-CONV algorithm is preferred, because the algorithm is fast and the average performance of the algorithm is quite good when 100% detection probability is assumed.

## C.2 Hierarchical Coarse-to-Fine Search in Parameter Space

In the LHC-GW-CONV and Top2 algorithms we used the same hierarchical coarse-to-fine search technique to speed up the search for the best UAV path. Here we describe the technique in detail using the Top2 algorithm as an example.

The Top2 algorithm is designed to generate paths that force the UAV to visit the top 2 subregions in the search area. First the Mode Goodness Ratio heuristic is used to identify the top 2 search subregions (represented by centroids). Then, shortest path segments from the start location and the end location (optional) to the nearest centroid are created. The algorithm then identifies a point (vertex) equidistant from the two centroids and launches two path planning tasks to plan path segments from each centroid to that point. By allocating different percentages of the remaining flight time to these two path planning tasks, the Top2

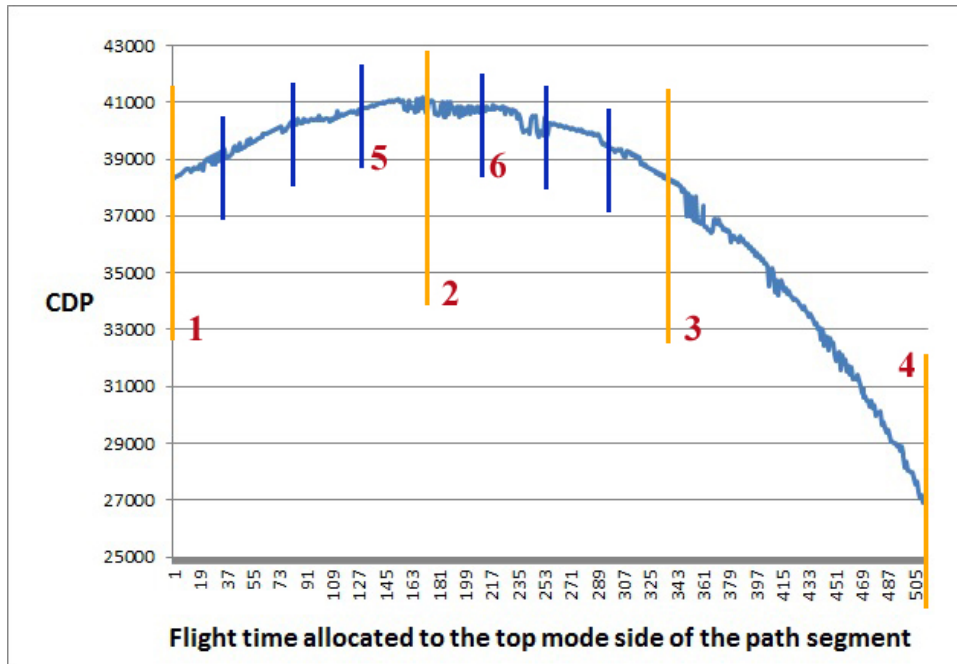


Figure C.2: Performance of the Top2 algorithm with the example WiSAR scenario when flight time allocated to first path segment varies.

algorithm can effectively search within a new dimension of time allocation. When searching in this new dimension, we used the coarse-to-fine search technique to improve search efficiency.

Figure C.1 shows an example synthetic WiSAR scenario, and Figure C.2 shows how search efficiency (CDP) changes when different amount of flight time (in time steps) is allocated to the first path segment (the total amount of flight time is static). The curve in Figure C.2 resembles a smooth curve with only one mode, meaning the local maximum would be the global maximum. This property allows us to use a coarse-to-fine search technique so we don't have to search exhaustively through the parameter space.

The coarse-to-fine technique is a recursive method with the number of recursive runs predetermined, depending on what resolution is needed. We start from a low resolution (large chunks of flight time transferred from one path planning task to the other) and gradually increase the resolution (smaller chunks) until the best path is found at the desired resolution.

First the total flight time is divided into equal chunks (3 chunks in our implementation), then four paths are generated with 0, 1, 2, and 3 chunks of flight time allocated to path segment 1 (remaining time allocated to path segment 2). The performance of these four paths are marked in Figure C.2 relatively by four orange lines labeled 1–4. Then the flight time allocation that generates the best path (maximum CDP) is identified (green line 2). In the next recursive run, the flight time chunk in the previous run is divided into smaller equal chunks, and three more paths are generated at each side of the green line 2 (marked

with shorter blue lines). Together with green line 2, performance of these seven paths are compared, and the one with the best path (still green line 2) is identified and will be used as the center for the next recursive round of search (between blue line 5 and 6). With a few recursive runs, the best time allocation point (between blue line 5 and green line 2) can be identified quickly without exhaustively searching through all the possible time allocation options.



## Appendix D

### Identifying Modes in a 3D Surface using Local-Hill Climbing with Memory

A probability density function over a 2D map encodes the probability of certain events in a specific region. For example, the probability density function created for a Wilderness Search and Rescue (WiSAR) operation can show the searchers areas where it is more likely to find the missing person. The distribution map can be used to allocate search resources and to generate flight paths for an Unmanned Aerial Vehicle (UAV). Figure D.1 shows an example *probability distribution map* with 4 modes.

Because different path-planning algorithms may be better suited for different probability distributions [131], identifying the type of distribution beforehand can help us decide what algorithm is appropriate for the specific path-planning task. In our decision process, we particularly care about how many modes the probability distribution has. So how can we automatically identify all the modes in a 3D probability distribution surface? In this appendix we describe the algorithm we use.

In our case, the 3D probability distribution surface is represented by a matrix/table where each value represents the height of the point. You can think of this distribution as a gray-scale image where the gray value of each pixel represent the height of the point.

The *local hill climbing with memory* algorithm proceeds as follows:

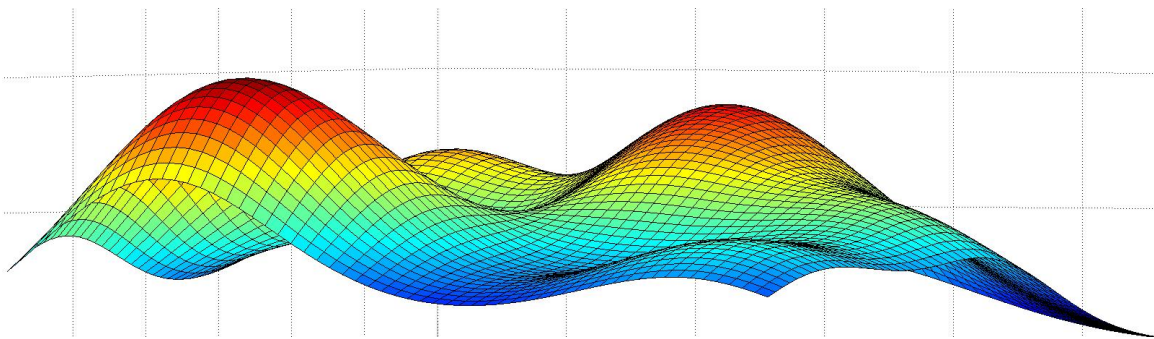


Figure D.1: An example 3D surface with 4 modes.

1. **Downsample and smooth the distribution:** If the distribution map is very large, it is useful to downsample the distribution to improve algorithm speed. For the results in this dissertation, we used  $100^2$  sample points over an  $100 \times 100$  grid. Since the algorithm assumes that the surface is free of noise, it may be necessary to smooth the surface using a Gaussian filter. The results in this dissertation assume a noise-free surface and, therefore, do not use smoothing.
2. **Check for a uniform distribution** (a flat surface): It is a good idea to check if the probability distribution is a uniform distribution. Just check to see if all values in the matrix are identical or are within  $\epsilon$  units of each other. If a uniform distribution is identified, we know the distribution has 0 modes and we are done.
3. **Local Hill Climbing with Memory:** Start from any point of the surface and then check its neighbors (8-connected). As soon as an unvisited neighbor with the same or better value is found, we “climb” to that point. As we “climb” and check neighbors, we mark all the points we visited along the way. When we check neighbors, we only check points we have not visited before. This way we avoid finding a mode we had found before. The “climb” process is repeated until we reach a point (hilltop) where all unvisited neighbors (if there is any) have smaller values. Once we find a “mode”, we can start from another unvisited point on the surface and do another Local Hill Climbing. Here I use quotes around the word mode because we are not sure if the “mode” we found is a real mode, meaning that we have actually found a local maximum of the probability surface.
4. **Make sure the “mode” we found is a real mode:** The “mode” we found using Local Hill Climbing might not actually be a real mode. It might be right next to a mode previously found and have a lower value (because we only checked unvisited neighbors in the previous step). It might also be part of another flat-surface mode where the mode consists of multiple points with identical values (think of a hilltop that looks like a plateau or think of a ridge). Things get even more complicated with special distributions such as the example in Figure D.2.

Moreover, the “mode” point we found might be connected to a previously found mode through other points with the same value (e.g, the “mode” point is the end point of the short branch in the middle of Figure D.2). Therefore, we need to keep track of all points leading to the final “mode” point that have identical values and check all the visited neighbors of these points, making sure this flat surface is not part of a previously found mode. If these points make up a real new mode, we mark these points with a unique mode count id (e.g, mode 3). If they are only part of a previously found mode,

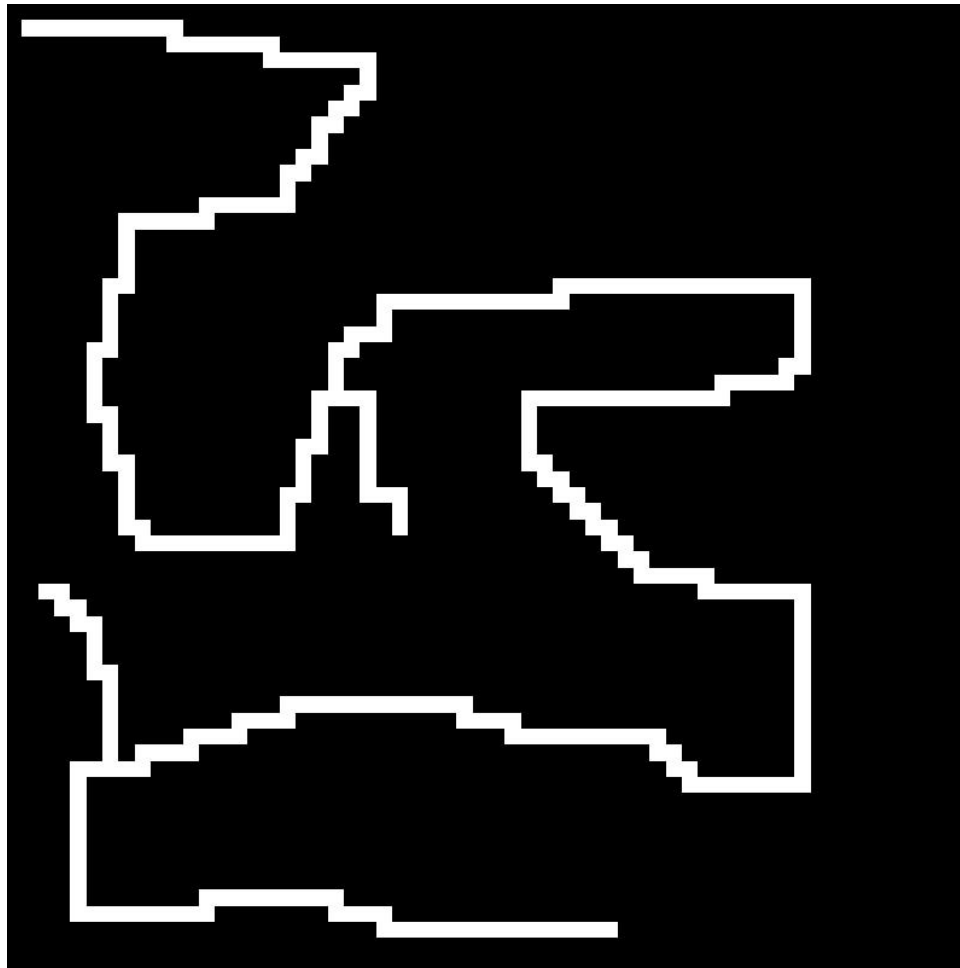


Figure D.2: An example path-type distribution resembling the Great Wall of China.

we mark these points with the previously found mode id (e.g., mode 2). If one of them is right next to a previously found mode but has lower value, we mark these points as non-mode points. This step is almost like performing a connected-component labeling operation in Computer Vision [110].

At the end of the algorithm run, we will have a count of how many modes the probability distribution has (maximum mode id) and also a map with all the mode points marked.

# Appendix E

## Sliding Autonomy User Study Design and Full Results for Chapter 6

### E.1 User Study Design

We performed a  $2 \times 3$  within-subject design with 2 scenarios (easy vs difficult) and 3 planning methods (manual, pattern, and sliding autonomy). All participants completed all 6 exercises. The order of the scenarios and planning methods is counterbalanced to reduce learning effect. Half of the participants started with scenario 1 (the other half scenario 2), and the order of the planning methods were randomly drawn without repeat from the permutation of all possible combinations (without following the same order in both scenarios).

#### E.1.1 Participants

After analyzing data collected from a pilot study with 6 volunteers, it was determined that 25 participants would likely produce significant test results. We recruited a total of 26 college students (14 males and 12 females) between the age of 19 and 30 (average 22.89). None has colorblindness. The majority has no experience with robots (57.69%), and 34.62% of them are slightly experienced with vacuuming robots.

#### E.1.2 Simulation Environment

The user study is conducted in a 3D simulation environment. The top portion of Figure E.1 shows a screen capture of the simulation interface. Both the *probability distribution map* and the *task-difficulty map* are displayed as 3D surfaces with a color map (red means high altitude and blue means low). The user can switch between the two maps at any time. The user can also rotate/pan a map and zoom in/out at will. The UAV in the simulation is a hexacopter that is capable of flying in all directions or hover in the same spot.

With the **manual** planning method, the user can fly the UAV around with arrow keys as the clock runs in a sped up fashion. The user can switch between two flying modes, turn mode and strafe mode, and four camera views, global view (always north up with

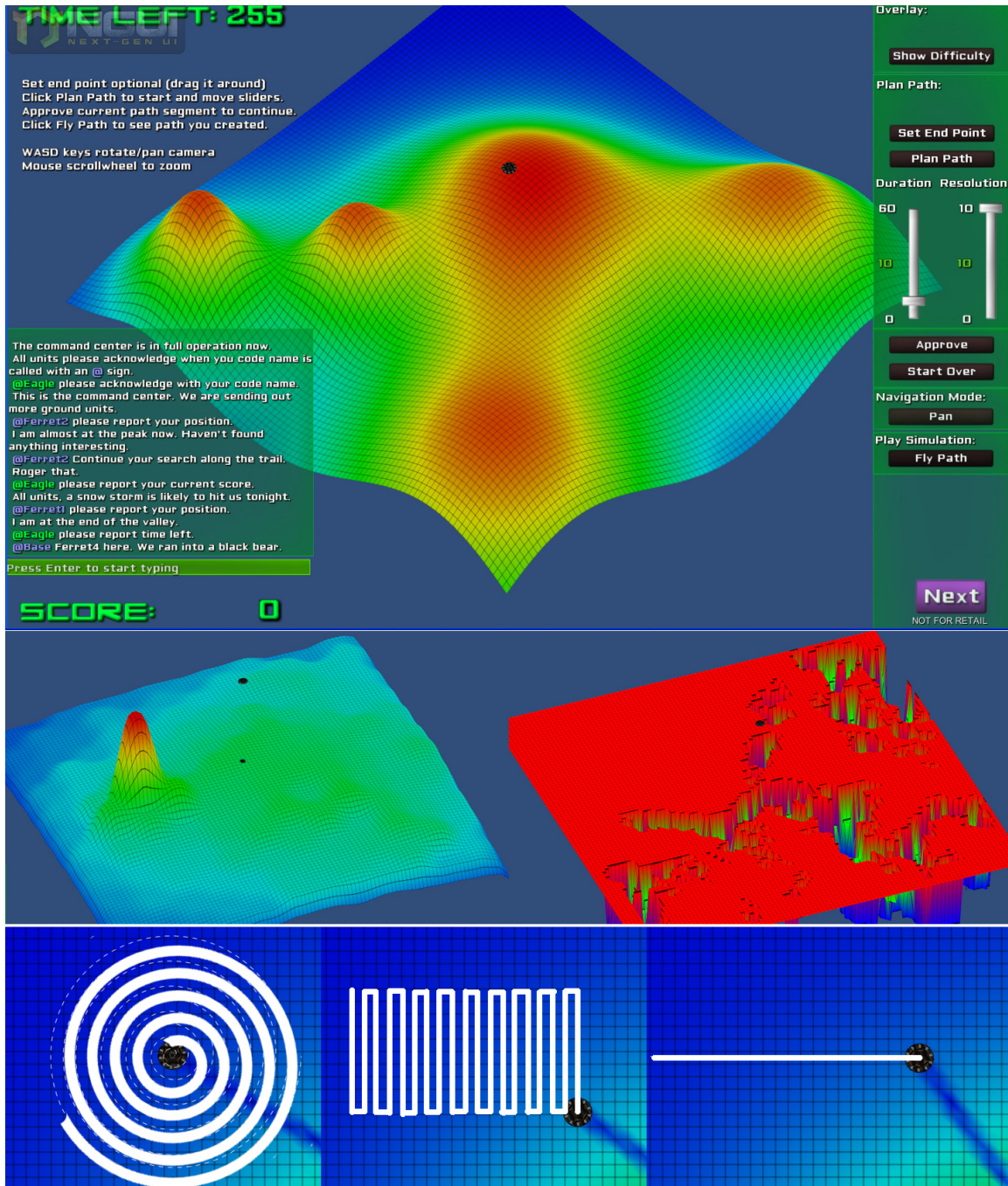


Figure E.1: Top: User study simulation interface with sliding autonomy method showing the *probability distribution map* for scenario 1. Middle left: *Probability distribution map* for scenario 2. Middle Right: *Task-difficulty map* for scenario 2. Bottom: The three patterns available to user in pattern planning mode, spiral, lawnmower, and line.

full view of the map), behind view, bird's eye view, and free form view (where the user can rotate/pan/zoom while flying). The user can pause/resume the flight to perform the secondary task or just to review the search area for better planning.

With the **pattern** planning method, the user can choose from three simple patterns (spiral, lawnmower, and line as shown in the bottom portion of Figure E.1) and join these patterns to form the final path. As the user moves the cursor around, the size of the pattern changes with the cursor position marking the end of the path segment (up to the remaining flight time to keep the path valid). Rotation of the lawnmower pattern can be achieved by rotating the map left/right instead. And rotating the map up/down turns the perfect spiral pattern into an ellipse pattern. The user can also undo the last path segment created all the way back to the start. This planning method is “semi-autonomous” because the patterns are generated automatically without manually setting waypoints.

With the **sliding autonomy** method (top portion of Figure E.1), the user can (optionally) set an end point anywhere on the map (reachable within remaining flight time) for the current path segment and then see the path suggested by autonomy. Then he/she can drag the knob of the left slider to change the amount of time allocated to autonomy, and see path suggested by autonomy instantly for each time allocation as the knob of the slider is dragged up or down. The slider's max value always reflects the remaining flight time (in minute) for the user to plan. If the user is happy with the current path segment, he/she approves it, the UAV moves to the end of the path segment, and the process repeats until all flight time has been planned. The path planning algorithm used in this planning method is the LHC-GW-CONV algorithm described in [69, 72]. We choose this algorithm because it is the fastest algorithm out of all the algorithms we designed.

The right slider is used to set the resolution or step value of the left slider and has a range between 1 and 10. For example, if the value of the right slider is set to 10, moving the left slider from bottom up will change time allocation to 10, 20, 30, ..., respectively. The purpose of the second slider is to improve the interaction experience when the user moves the left slider, because in order to provide instant feedback, paths with different time allocation need to be pre-computed by autonomy. Although each path only takes a fraction of a second to generate, for a 60-minute flight autonomy has to generate 60 paths for all possible time allocations, which can take a long time. When the value of the second slider is set high, only a small number of paths need to be pre-computed which enables the instant feedback. This feature turned out to have negative effect on users' interaction experience, which we will discuss later.

With all three planning methods, the user can choose to start over at any time during the exercise, and can restart as many times as exercise time allows, and we record the best

path out of all tries. Each user fully understands how the manual and pattern planning methods work, but does not know how path planning autonomy generates paths behind the scene in the sliding autonomy method.

### E.1.3 Scenarios

The user study contains two WiSAR scenarios. Scenario 1 is a synthetic case with a *probability distribution map* that is a mixture of five Gaussians (shown in the top portion of Figure E.1). No *task-difficulty map* is used in this scenario (uniform detection probability is assumed).

Scenario 2 comes from a real WiSAR scenario, in which An elderly couple was reported missing near the Grayson Highlands State Park in Virginia. The *probability distribution map* used for this scenario (Figure E.1 middle left) was generated using a Bayesian model [70]. The map has been evaluated at George Mason University’s MapScore web portal [18] and performed better than most other models evaluated<sup>1</sup>. This scenario also uses a *task-difficulty map*, and the map (Figure E.1 middle right) was generated using vegetation density data downloaded from the USGS web site and categorized into three difficulty levels (sparse, medium, and dense, with detection probability of 100%, 66.67%, and 33.33% respectively).

Scenario 2 is clearly more complicated than scenario 1 because the user also has to consider the different detection probability defined by the *task-difficulty map*. We refer to scenario 2 as the high information scenario and scenario 1 as the low information scenario.

### E.1.4 Secondary Task

In each exercise, we also ask each participant to perform a secondary task together with the main task of path planning. This way we can measure the mental workload of the user. The secondary task is in the form of a group chat window (see the lower left corner of the top picture in Figure E.1), and when the user’s code name “Eagle” is called, the user is asked to answer simple questions by typing in the chat window. Every 3 seconds (plus a random integer drawn from the uniform distribution  $[-2,2]$ ) a text message is sent to the chat window, and every 5th message asks the user a simple question. Therefore, every minute the user receives 20 messages and 4 of them are directed to the user. For the same scenario and the same planning method, all users use the same set of chat messages.

We choose to use a group chat window as the secondary task because this is typical in real SAR operations. We also designed the chat messages to simulate a real WiSAR search. The user is asked to acknowledge connection and report path planning status periodically.

<sup>1</sup>Scoring 0.8184 on a  $[-1,1]$  scale where the higher the score the better. <http://sarbayes.org/projects/>

This design ensures that the secondary task is ecologically valid [94, 129] and makes the experiment result more convincing.

### E.1.5 Procedure

Each participant first fills out a demographic survey after signing the IRB consent form, then he/she completes four training exercises. The first three training exercises teach the user how to plan paths with the three planning methods using a simple *probability distribution map* and no *task-difficulty map*. The fourth training exercises adds the *task-difficulty map* to the path planning problem, and the user gets to practice the manual planning method again. Each training exercise lasts 5 minutes and the user cannot skip it. A “cheat sheet” is provided to each participant during the entire user study to explain the simulation environment and key concepts. Each participant is also asked to read the instructions for the NASA TLX survey.

Then each participant completes the six exercises (2 scenarios and 3 planning methods each) in a counterbalanced order. For each exercise the user has up to 5 minutes to plan a path. Once the user is happy with the path generated, he/she can finish the exercise early. We choose this design because we do not want the user to put all effort into completing the secondary task once he/she considers the primary task completed, which would skew the measurements on secondary task performance.

After the user completes each exercise, we ask the user to complete a NASA TLX survey for the exercise. Then after all six exercises are completed, the user fills out a post user study survey describing his/her subjective preference with the three planning methods.

## E.2 User Study Results

We analyzed user study data with a mixed measures analysis of variance (ANOVA). In this section we list the complete user study result analysis.

### E.2.1 Comparing Across Scenarios

Table E.1 lists the user study results compared between scenario 1 (low information) and scenario 2 (high information) with statistically significant results highlighted in bold.

No statistically significant differences were found for percent score, time spent, try count, percent of questions missed, and chat latency across scenarios. However, mouse clicks per try and NASA TLX are significantly different between the two scenarios, indicating scenario 2 with high information load required more work and cognitive workload are needed for scenario 2.



Table E.1: Comparing across scenarios (SE stands for standard error)

	S1 Low	S2 High	SE	Significance
% Score	76.99	74.17	1.12	$F[1, 25] = 7.51, p = .01$
Time spent	224.01	250.47	12.06	$F[1, 25] = 8.35, p = .0079$
Try count	3.08	2.67	0.37	$F[1, 25] = 3.32, p = .80$
Clicks/try	15.95	33.54	2.59	<b><math>F[1, 25] = 28.65, p &lt; .0001</math></b>
NASA TLX	48.19	58.17	2.50	<b><math>F[1, 25] = 31.35, p &lt; .0001</math></b>
% Q. missed	54.88	54.90	5.18	$F[1, 25] = 0.00, p = .99$
Chat latency	10.88	10.77	0.56	$F[1, 25] = 0.03, p = .88$

### E.2.2 Comparing Across Planning Methods

Table E.2 lists user study results comparison among the three path planning methods (manual, pattern, and sliding autonomy). with statistically significant results highlighted in bold.

No statistically significant differences were found for time spent, try count, percent of questions missed, and chat latency across scenarios. However, sliding autonomy performed significantly better than pattern and pattern also performed significantly better than manual in both scenarios. Mouse clicks per try and NASA TLX are also significantly different among the three planning methods.

Table E.2: Comparing across planning methods (SE stands for standard error)

	M	P	SA	SE	Significance
% Score	59.40	72.75	94.60	1.39	<b><math>F[2, 50] = 223.03, p &lt; .0001</math></b>
Time spent	243.35	240.02	228.37	12.06	$F[2, 50] = 1.16, p = .32$
Try count	1.75	3.56	3.31	0.43	$F[2, 50] = 9.47, p = .0003$
Clicks/try	13.01	35.64	25.58	2.90	<b><math>F[2, 50] = 19.47, p &lt; .0001</math></b>
NASA TLX	61.51	49.18	48.86	2.81	<b><math>F[2, 50] = 14.15, p &lt; .0001</math></b>
% Q. missed	52.94	56.69	55.04	5.17	$F[2, 50] = 1.26, p = .29$
Chat latency	10.39	11.17	10.92	0.65	$F[2, 50] = 0.46, p = .63$

### E.2.3 Additional Factors

We also performed ANOVA analysis on some additional factors that might create differences between scenarios or among planning methods: gender, experience in video games, order of the scenarios, and whether participants used full autonomy with the sliding autonomy method. Table E.3 lists analysis results. No statistically significant differences were found

in any of these factors. There is also no significant correlation (-0.23) between percent of questions missed in the secondary task and the NASA TLX raw scores.

Table E.3: ANOVA Analysis Results for Additional Factors

	Overall	Scenario	Method
Gender	$F[1, 24] = 0.05, p = .83$	$F[1, 24] = 0.76, p = .39$	$F[2, 48] = 0.59, p = .56$
Video Game Exp.	$F[4, 21] = 0.78, p = .55$	$F[4, 21] = 1.64, p = .20$	$F[8, 42] = 1.13, p = .37$
Scenario Order	$F[1, 24] = 0.09, p = .77$	$F[1, 24] = 0.39, p = .54$	$F[2, 48] = 1.53, p = .23$
Full Autonomy	$F[1, 22] = 3.70, p = .07$	$F[1, 22] = 0.36, p = .56$	$F[2, 44] = 0.04, p = .96$

#### E.2.4 Comparing Against Autonomy Performance Markers

Table E.4 summarizes what percent of participants were able to perform better than autonomy working alone with each planning method in each scenario.

Table E.4: Percent of participants outperforming autonomy with each method

	Manual	Pattern	Sliding Autonomy
Scenario 1 (Low)	0%	0%	<b>88.46%</b>
Scenario 2 (High)	0%	19.23%	<b>92.31%</b>

Table E.5 lists what percentage of participants outperformed full autonomy, EA, and autonomy+1 human input.

Table E.5: Percent of participants outperforming autonomy performance markers

	Autonomy	EA	Autonomy+1
Scenario 1 (Low)	<b>88.46%</b>	<b>88.46%</b>	7.69%
Scenario 2 (High)	<b>92.31%</b>	26.92%	15.38%