University of California

Merced

Hierarchical Search with Probabilistic Quadtrees Applied to Single and Multi-Agent Systems

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Electrical Engineering and Computer Science

by

Derek Burch

2012

© Copyright by

Derek Burch

2012

Abstract of the Thesis

Hierarchical Search with Probabilistic Quadtrees Applied to Single and Multi-Agent Systems

by

Derek Burch

Master of Science in Electrical Engineering and Computer Science

University of California, Merced, 2012

Professor Stefano Carpin, Chair

We consider the problem of searching for one or more targets in an environment using a noisy multi-scale sensor. This work utilizes the recently introduced probabilistic quadtree (PQ) framework and provides several improvements and extensions. This framework allows searchers to maintain a compact representation of their belief of the world, while allowing them to consider sensing at various scales. Improvements herein include: reduced computational complexity on our previously used objective function, an alternative quadtree updating technique, and a new searcher objective function. In addition to theoretical contributions, this work considers the problem of coordinating multiple searchers using the PQ structure. Several experiments are included which demonstrate the performance of the algorithms under various configurations and conditions.

The thesis of Derek Burch is approved.

David Noelle

Marcelo Kallmann

Stefano Carpin, Committee Chair

University of California, Merced

2012

Dedicated

to my parents for all the support they have given me

Table of Contents

Acknowledgments

First, I would like to thank my advisor, Stefano Carpin, for his support during my graduate work. He continually provided insight and direction for my research. I would also like to thank my other committee members, David Noelle and Marcello Kallmann, for reviewing my thesis and providing useful suggestions. Additionally, I thank these professors for all of the times that they were lecturers in my UC Merced courses. Over the course of my undergraduate and graduate career at UC Merced they have taught me a great deal and I will surely do my best to put their knowledge to good use.

I would also like to thank my lab mates, Benjamin Balaguer and Gorkem Erinc, and my fellow EECS students for their advice and help during my time as a graduate student. A special thanks to Nicola Basilico for his help with the partitioning portion of this work. He never gave up, even when presented with difficult challenges.

Additionally, I thank my friends, girlfriend, and family for all of their support. They allowed to take breaks from my work and prevent myself from burning out. Without their help I am sure that I would not have been able to finish. Additionally, my parents gave me considerable support and placed me in environments that allowed me to accel academically.

Lastly, my research was made possible by funding I received from a Eugene Cota-Robles Fellowship.

CHAPTER 1

Introduction

Robots are playing an increasing roll in the lives of humans. Example areas of influence include manufacturing, search and rescue, personal assistance, and others. While the robots of today may not be quite as advanced as people dreamed them to be, they are still progressing and providing assistance to humans. With devices like the Roomba vacuum cleaner, robots have even been able to enter and function in peoples' homes. This thesis focuses on robotic search with the important application of search and rescue. Every time a disaster occurs rescue workers put their lives on the line to try and save others. By introducing robots to this area, we can hope to both increase the amount of lives that can be saved and also reduce the risk to rescue workers. One such event where robots were used to provide assistance was the Fukushima nuclear power plant disaster after the earthquake in Japan in 2011. By using robots to gather data in the facility, clean up crews did not have to enter into areas with very harmful levels of radiation [12]. In other disaster scenarios such as building fires, robots could be sent in to give rescue workers an accurate look into the building and see if there are people needing rescue inside. They could also provide accurate maps to help rescue teams navigate buildings after damage from fire or earthquake. In search and rescue, the chances of finding a person alive drops significantly after 72 hours and is often referred to as the "golden 72 hours" [25]. This highlights the importance of using whatever methods possible to accelerate the rescue process.

In general for robots to perform well at their assigned tasks, they need to be able to search for people and objects effectively. This thesis focuses on how robots can search when they are given a sensor that can be used at multiple resolutions. An example situation is using an aerial vehicle to search for a lost hiker. Recently, vertical take off and landing (VTOL) vehicles are becoming more popular. They generally are very mobile robots that use 4 rotors to take off vertically and can hover in place making them excellent for surveillance and data collection. Robots of this type can be equipped with a camera and by changing their altitude they can collect images of varying fields of view. The trade off for the searcher is that by flying high to get a larger view of the environment it reduces the small details visible to the camera and thus reduces the accuracy of its sensing. It is important in this case that the robot balance this ability to search at different altitudes depending on its need for course or more fine grained information.

This hierarchical form of data collection is already being validated in practice via research done by the Robotics lab at UC Merced. Data was collected using our Air Robot quad rotor platform at the Camp Roberts testing facility. Various objects were placed in a large environment and images from the quad rotor were taken at several locations and altitudes. Using this data, an image classifier could be trained to detect cars or people. This type of data collection is important because it allows validation of the theoretical models and simulations being tested in our lab.

There are multiple ways to try and improve the performance of robots. One such way is to improve the hardware they use by giving them better/more sensors and computational power. The downside of this strategy is that it is costly and by using only one robot if it fails the mission is over [1]. Alternatively, robots can be improved by using better algorithms for search. Examples of this will be seen in later chapters in this thesis. A final way to improve robotic performance is to increase the number of robots used. This often involves using robots that on their own are inferior, but can work effectively as a team. The downside to this strategy is that creating algorithms to work with teams of robots is more difficult as it generally involves coordination and communication much like teams of humans. Utilizing teams of robots and getting them to perform effectively with minimal communication is a difficult prospect and is something that will be discussed later in the thesis. Lastly, this work contains benchmarks for teams of varying numbers of robots in an attempt to measure the presented algorithms' ability to scale based on the number of searchers.

In summary, this thesis provides the following contributions to our previously researched hierarchical search structure.

- A reduction in the computational complexity when searching for a single target in an environment.
- An alternative method to update the search structure when searching for multiple targets.
- An improved objective function when patrolling an environment.
- Extensions to multi-agent search.

CHAPTER 2

Related Works

2.1 Probabilistic Search

In operations research, the theory of search is a well studied topic and its use dates back to World War II. The goal of search theory is to define a strategy to find the location of an object or the locations of multiple objects in an optimal manner. In addition to having objects with unknown location, the way in which these objects are detected has a degree of uncertainty as well. Works by Koopman [16], Stone [23], and others have provided solutions to these problems that are grounded in mathematical theory. In general, the problem is solved by using prior knowledge to construct a probability distribution over the possible locations for the object or objects and applying sensor readings to locate those objects. The search effort is applied to locations that provide the greatest chance of finding the object. In one example application, Koopman discusses how the work was used to help find optimal defensive placement of U.S Navy ships during World War II.

One way to formulate the search problem is using a partially observable Markov decision process (or POMDP). The POMDP framework allows the searcher to deal with many forms of uncertainty. It is able to handle cases where the searcher has error in its own motion model, the search target can move, and cases where the sensor used is noisy. In short, to use a POMDP one models the position of the searcher and object as a state of the world. The exact state of the world is unknown to the searcher so it maintains a probability distribution over all possible states. Then through various actions the searcher and object transition to other states. These transitions behave based on a known probability distribution based on actions taken by the searcher. Additionally, the searcher is provided with observations to help identify the current state of the world. In a work by J.N. Eagle [11] a POMDP is used to perform optimal search for moving targets with constrained search paths. By modeling the problem with a POMDP, the author is able to handle moving targets, while finding optimal search paths. The downside of the POMDP framework is that it has difficulty scaling to large problem sizes.

One could also frame the search problem in the context of game theory. In work by B. Sujit and colleagues [24], the authors approached the problem of multi-agent search in an unknown environment using game theory. This allows well known concepts such as Nash equilibrium to be applied and determine the optimal strategies for the search process. Each of the searchers maintain an uncertainty map of the environment and it is their goal to select actions that maximize the reduction in uncertainty. The work considers the case where the searchers do not communicate with each other (and thus can be thought as competing with one another). It also considers the case where the searchers work together and cooperate on the search mission. It is the goal of the searchers to maximize their own reward based on the worst case actions of the other search agents.

Another framework is one for cooperative search with distributed agents by M.M. Polycarpou and colleagues [21]. In their work, they considered the case where the environment of the searchers is unknown. Additionally, it is assumed that the searchers communicate their sensor information with each other so that they can intelligently cooperate. This means that the searchers must learn the features of their environment, while also searching for targets with no prior information. This scenario requires knowledge to be incorporated during the mission and is solved using a combination of on-line learning and optimization. On-line learning is used to interpret readings from the environment and optimization is used to find the best search path through the environment.

It is not always the case that the prior probability distributions provided to searchers are known with high certainty. One method for handling uncertainty in prior belief distributions was proposed by I. Sisso and colleagues [22]. In their work, they utilize a technique base on info-gap theory which allows for decision making when faced with large uncertainty in prior information. The work provides an alternative to just using the search method that provides the highest expected utility. Instead, the work describes strategies that maximize the robustness of the search strategy. This allows the searcher to use a robust search method when uncertainty in the prior information is high and a utility maximization method when the uncertainty in the prior is low. Searchers can also handle the problem of uncertainty in their prior by altering their search strategies throughout the mission as they obtain more sensor readings.

For a more complete history of the theory of search refer to a the survey by S.J. Benkoski [3]. For another summary on more recent work and challenges one can also refer to a survey by T.H Chung [10].

2.2 Bayesian Search and Rescue

Bayesian search is a type of probabilistic search which uses evidence to modify probability values. One important application of Bayesian search is search and rescue. In work by L. Lin and M.A. Goodrich [19], the authors consider searching for a lost person in the wilderness. When searching for a lost person one often creates a probability distribution that tries to identify possible locations for the person. In order for Bayesian search to work effectively, one must have prior distributions that are as accurate as possible. Their work attempts to generate good prior distributions for the person's location by incorporating prior knowledge of lost-person behavior along with environment features. This information allows for a more accurate prediction of the target's location and can also provide information for how the missing person may move over time. Overall, the work provides tools that can be used by rescue teams to find people faster and increase the chance of finding lost people alive.

2.3 Searching with UAVs

Search theory is now being applied in the field of robotics for autonomous search. Some work that has strong influences in the work of this thesis was done by Furukawa and colleagues [5] [6] [28]. In their works, they considered searching with unmanned aerial vehicles (UAVs). The authors investigated applying Bayesian techniques to searching for lost targets using grid representations of the environment. Scenarios with both multiple searchers and multiple targets are considered. The methods used allowed for the use of noisy sensors and provided an objective function to guide the search. Their work focused on searching with fixed altitude UAVs which means that the sensor accuracy remained constant throughout the mission. In the works, greedy methods are applied since the optimal searcher trajectories are too computationally difficult to compute. In [18] the authors lift the assumption of a fixed grid like environment and allow it to be dynamically reconfigured during the mission. The goal of this reconfiguration is to allow better tracking and better handling of the underlying probability density function in the case of moving targets.

Another work that that provides foundation for our work is that done by T.H. Chung [8]. In this work, Chung also applied Bayesian techniques to search for a single lost target with a searcher constrained to move on a grid. The single target assumption induces correlations between cells in the grid. In addition, the work allows for the existence of false positives and missed sensor detections rather than just the usual missed detections. Finally, the work also considers the expected time to find the target in relation to the motion constraints of the searcher. The work shows that a relationship exists between the decision time of the searcher and the second smallest Laplacian eigenvalue.

In work by S. Waharte and colleagues [26], the authors consider the case of searching with agile UAV's. These agile UAV's move at low speeds and are able to easily alter their altitude. The authors discuss ways to combine sensor readings from different altitudes into one belief about the environment. In their work, they use a uniform grid representation, but the sensor readings are allowed to partially cover cells and/or cover multiple cells. The searcher is then able to control its sensor coverage and accuracy by controlling its altitude. The work provides control algorithms to balance the trade off between sensing more with less accuracy and sensing less with higher accuracy. Results from the work showed that sensing at multiple altitudes improved the performance of the searcher. One important difference between the work of these authors and the work in this thesis is that this thesis utilizes a hierarchical data structure to make the search process more efficient.

One can also consider the case of using UAV's to search in an environment with imprecise prior probability distributions. This scenario is considered in work by L. F. Bertuccelli and J. P. How [4], where they use a beta distribution to predict the minimum number of sensor reading needed per cell for a required amount of certainty. This distribution allows the authors to encode uncertainty about the prior and allows integration of imperfect binary sensor readings. Additionally, their framework allows the searchers to determine the minimum number of looks to achieve a desired confidence for the occupancy of a cell in the environment. This work provides an alternative to the methods used by Sisso and colleagues [22] mentioned previously.

In work by M. Flint and colleagues [15], the cooperative control problem for multiple UAV's is handled using a dynamic programming approach. The authors propose using a stochastic decision model along with approximation techniques to handle the computational complexity of the problem. Rather than compute the optimal paths for each searcher, the searchers model each other as stochastic elements. This results in sub-optimal trajectories, but their results showed good performance. This work was also extended to handle scenarios with risky environments [14]. In this scenario, it is possible for the searcher to be destroyed during its mission depending on how it explores its environment. Incorporation of prior information was also added to their dynamic programming framework. A summary work [13] provides a combined discussion on the two previous works. In the combined work, performance comparisons between informed and uninformed searchers were explored.

A decentralized cooperative search framework was considered in a work by Y. Yang [29] and colleagues. Their idea was to have each searcher view each other as soft obstacles that needed to be avoided. Like the previous work an approximate dynamic programming method was used to guide the searchers. This was done using various approximated reward functions that balance immediate, exploration, thread avoidance, and cooperation reward. In the work, the cooperation reward was larger the further searchers are from each other. This creates what they refer to as a rivaling force that pushes searchers away from each other. Additionally, another reward function was added to help searchers also try for long term goals. This mixture of objective functions allows the searchers to consider many search factors while keeping the computational complexity low. Overall, the proposed method was efficient and managed to provide a working framework for decentralized cooperative search.

2.4 Work on Probabilistic Quadtrees

A major weakness in the works previously mentioned is that they almost all rely on the usage of some form of uniform grid. These uniform grids result in a large search space that cause issues with many of the proposed search strategies. Recent work by S. Carpin and T.H. Chung [9] has been done to extend the previous techniques towards hierarchical representations. The work considers using a mobile UAV searcher with the ability to change its altitude to vary its sensor coverage and accuracy. Similar to the work by Chung mentioned above, this work considered the case of looking for a single object in an environment. Additionally, they introduced the probabilistic quadtree data structure which allowed the searcher to sense at different altitudes and maintain a compact belief representation. This compact representation also allowed the searcher to compute search locations more efficiently.

This work was later extended by S. Carpin, T.H. Chung and myself [7] to handle the case where multiple objects may be located in an environment. This extension required modifying the updated strategies used on the probabilistic quadtree and additionally the criterion used to decide when to stop the search. Lastly, the work focused on improving the search decisions using an entropy based approach.

Later work by N. Basilico and S. Carpin applied the probabilistic quadtree framework to the problem of patrolling an environment [2]. In this scenario, intruders enter the environment over time and it is up to the searcher to locate them in a timely manner to reduce the damage sustained.

Lastly it should be mentioned that work by G.K. Kraetzschmar [17] was the first to introduce a probabilistic quadtree. While this work shares the same name and a similar data structure as the one described in this thesis, ours considers the case of search where the other considers the problem of mapping. Additionally, the structure discussed in this thesis has different properties and different sensor reading integration. In short, both works consider applying the well known quadtree data structure to probabilistic beliefs, but differ on the implementation details.

CHAPTER 3

Probabilistic Quadtrees

3.1 Quadtrees

In general, a quadtree is a method for data representation that aids in compression and/or data partitioning. It is a recursive structure composed of nodes each with potentially four children. Quadtrees are mostly used to partition 2 dimensional data, but there are other variants that deal with higher dimensions such as octrees. An example usage is compression in a 2D black and white image. One could split the image in four regions and mark the nodes as completely filled, completely empty, or mixed color. If the nodes are mixed color four children would be added to this node. Otherwise, no children are added and the node is marked as filled or empty. This process would continue for these new nodes or until a maximum depth is hit. Using this method, the image can potentially be represented with fewer nodes than the number of pixels in the original image. Refer to Figure 3.1 for a quadtree example dealing with image compression.

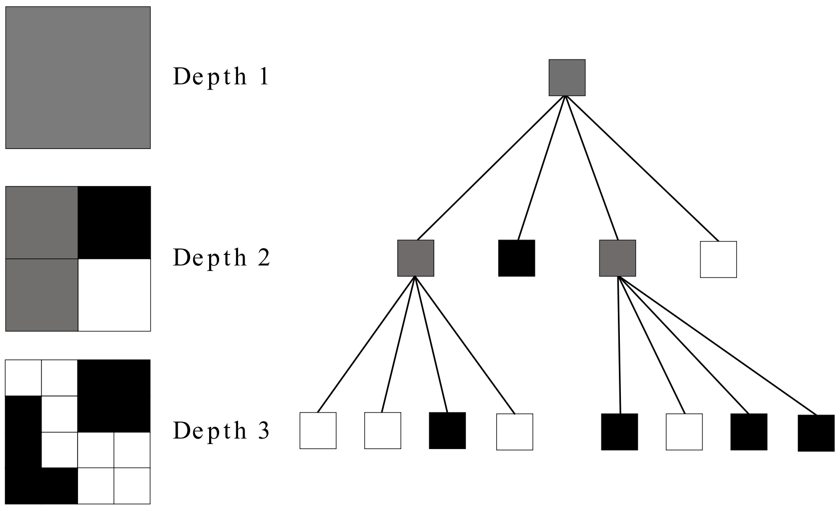

Figure 3.1: Simple Quadtree Example: The 16 pixel image is represented with 13 nodes.

3.2 Probabilistic Quadtrees

Rather than have the nodes of a quadtree represent filled, empty, or mixed one can instead have the nodes contain the probability that an object is present within a region. In areas where the probability is uniform, fewer nodes are needed to represent the underlying probability distribution which reduces the amount of information needed to represent the belief space. More importantly, this type of representation can also reduce the size of the search space when selecting areas to sense. Our work focuses on the idea that the different levels in a quadtree can be mapped to different sensing altitudes of an aerial vehicle. By changing its altitude, the searcher can change its sensor coverage and accuracy. When the searcher increases its altitude it can see more of an environment, but its sensor will detect objects at a reduced accuracy. In order to make updates to the probabilistic quadtree, it is assumed that the rate at which the sensor makes false positives and missed detections is known. By using this structure, searchers can balance the trade-off of scanning larger areas (areas at higher levels in the quadtree) versus scanning smaller areas (lower levels in the tree). The exact management of this tree structure depends on the assumptions made about the search process and will be explained in sections below.

3.3 Formal Definition

Given a rectangular search domain $\mathcal{A}$, a *probabilistic quadtree*(PQ) $\mathcal{T}$ can be constructed. Each node in $\mathcal{T}$ is associated with a square or rectangular region inside of $\mathcal{A}$. The region covered by node n will be referred to as $R(n)$. Additionally, for each node n in the tree a binary random variable X_n is assigned. This variable indicates the event that at least one intruder is present in the area $R(n)$. For convenience we also define:

$$p_n = \Pr[X_n = 1]$$

which is the probability that something is present in the region associated with node n.

Additionally, it is assumed that the tree $\mathcal{T}$ is restricted to a maximum depth $\mathcal{D}$. If all the leaves of the tree are at depth $\mathcal{D}$ then the tree represents the same partition of $\mathcal{A}$ as a uniform grid. Lastly, let $\mathcal{L}(\mathcal{T})$ be the set of leaves of $\mathcal{T}$ and $\mathcal{N}(\mathcal{T})$ be the set of all nodes in the tree. It is important to note that the tree may not and in general will not be fully expanded. This means that its leaves will not all be at depth $\mathcal{D}$ and that $|\mathcal{N}(\mathcal{T})|$ will change throughout the search mission. Some other sets that are used are listed below:

- Let $\mathcal{C}(n)$ be the set of nodes that have n as their parent node. In a quadtree a node either has 0 or 4 children.
- Let $\mathcal{IN}(\mathcal{T})$ be the set of internal nodes in the tree where

$$\mathcal{IN}(\mathcal{T}) = \mathcal{N}(\mathcal{T}) \setminus \mathcal{L}(\mathcal{T})$$

- let $\mathcal{S}(n)$ be the subtree in $\mathcal{T}$ rooted at node n

Lastly, there are important relationships between the event X_n and the events of descendant nodes of n. Since the descendants of n are contained within the region associated with n it must be the case that

$$X_n = 1 \Leftrightarrow \exists i \in \mathcal{S}(n) \cap \mathcal{L}(\mathcal{T}) \text{ s.t. } X_i = 1$$

This states that if the event $X_n = 1$ there must be a target inside of n and thus a target inside at least one of its descendant leaves. This is true because the leaves descendant from n form a complete partition of n. The opposite relationship is also true. Meaning that when a descendant i of n has a target within it, $X_i = 1$, then it must be the case that $X_n = 1$. This is true because the region associated with a descendant of n is always within the region of n. Similar relationships also hold for the case when node n is empty, $X_n = 0$, and when all of its descendants are empty:

$$X_n = 0 \Leftrightarrow \forall i \in \mathcal{S}(n) \;\; X_i = 0$$

Refer to Figure 3.2 for an example PQ with its associated X variables.

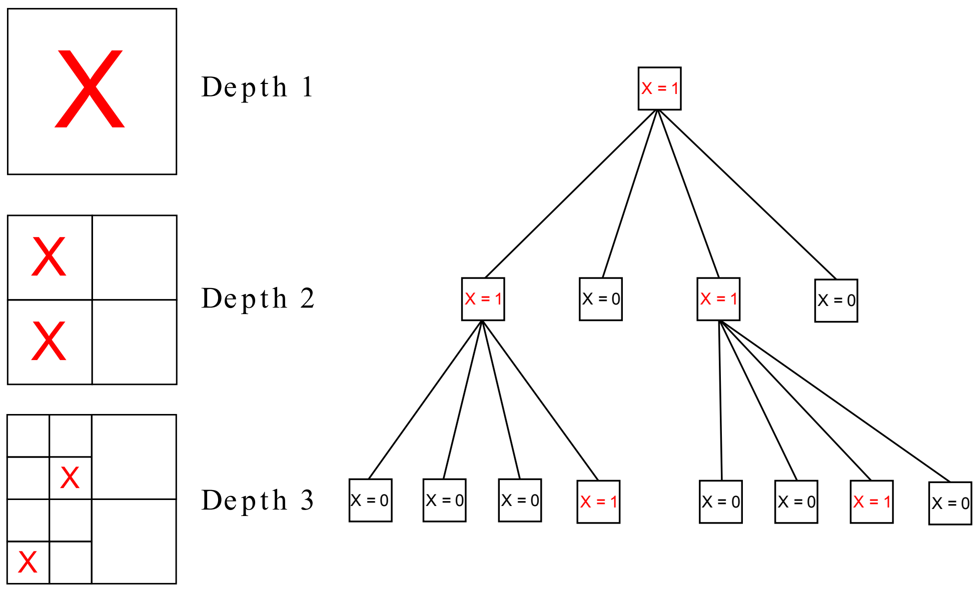

Figure 3.2: Event Example. On the left the red X's indicate the presence of a target in the region. The tree on the right shows the relationships of the event variables in the tree.

3.4 General Sensor Model

For each of the search scenarios in this thesis, the following assumptions are made about the sensor used by the searcher. First, as mentioned above the sensor is subject to varying levels of accuracy based on its altitude. Additionally, this sensor returns a binary value with 1 indicating that a target is present in the scanned area and a 0 otherwise. The event of sensing at node n in $\mathcal{T}$ will be referred to as Z_n where $Z_n = 1$ indicates a positive reading and $Z_n = 0$ indicates a negative reading. This sensor is also subject to both false positives (returns a 1 when no target is present) and missed detections (returns a 0 when a target is actually there). The chance of a false positive will be referred to as α and the chance of missed detections as β. Both of these rates vary depending on the depth of the sensor reading in $\mathcal{T}$ which is related to the altitude of the searcher.

Additionally these probabilities will be referred to as:

$$\Pr[Z_n = 1 | X_n = 0] = \alpha(d(n))$$

$$\Pr[Z_n = 0 | X_n = 1] = \beta(d(n))$$

Where $d(n)$ is the depth of node n. Additionally, the sensor does not change its accuracy based on the number of targets present in the scanned area. Lastly, the quadtree is structured in such a way that sensor readings at a node scan the area covered by the node and thus each of its children. Refer to the figure 3.3 for a graphic on multi level sensing.

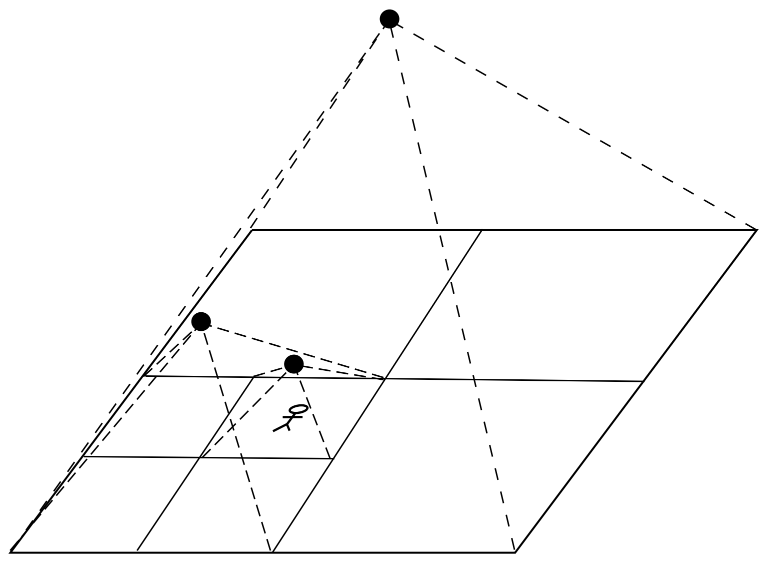

Figure 3.3: Multi Level Sensing. The dot at the top of the figure represents scanning at the root of the tree $\mathcal{T}$. At this location the entire search region $\mathcal{A}$ is visible. The other dots represent scan locations deeper in the tree. Take note that by going deeper into the tree the size of the area scanned is reduced by a factor of 4 for each level.

3.5 Environment and Prior Belief Representation

For each of the search scenarios below, we assume that the search environment $\mathcal{A}$ can be divided into a grid of cells and each of these cells can contain at most one target. Additionally, the searcher is given a prior probability distribution over this grid that indicates the probability that a target is located in each of the cells in the environment. The searcher takes this prior distribution and converts it to a quadtree based on the rules described in the following sections. Using this initial prior, also referred to as a belief, the probabilities for the random variables X_n at each node can be initialized. Lastly, the tree can be compressed using a technique that will be described in a later section.

3.6 The Type1 Scenario

One way to use the probabilistic quadtree structure is in a scenario where the searcher is looking for at most a single target in an environment. This scenario will be referred to as Type1 and its associated probabilistic quadtree as a Type1 PQ. The single target assumption creates dependencies between the leaves of the quadtree because if the target is not in one node then it must be in another or outside of the environment. For this case, an additional node that cannot be sensed is added to the tree which represents the event that the target is outside of the search region. This node will be referred to as the null node and the probability of a target being in this node as $p_{\varnothing}$.

$$p_{\varnothing} = 1 - \sum_{l \in \mathcal{L}(\mathcal{T})} p_l$$

3.6.1 Belief Representation

Each node in the tree represents the probability that the one target is located with the area covered by the node. This means the node at the root of the tree indicates the probability that the target is present in the overall search area. This idea works for all internal nodes and yields the property that the probability that any node in the tree contains a target is the sum of the probabilities stored in its children.

$$p_n = \sum_{c \in \mathcal{C}(n)} p_c \quad \forall n \in \mathcal{IN}(\mathcal{T}) \tag{3.1}$$

An example Type1 quadtree can be seen in Fig 3.4

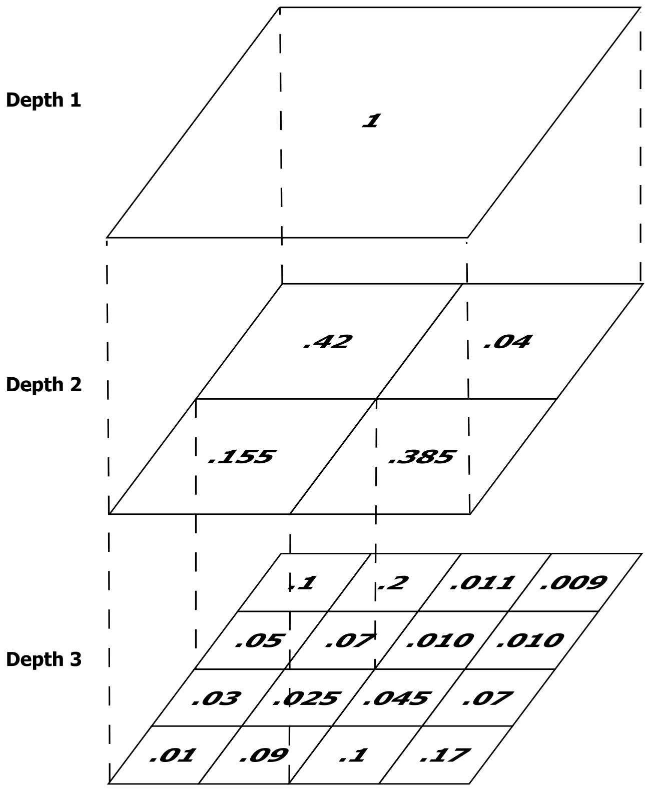

Figure 3.4: Type1 Probabilistic Quadtree: In the above figure, it is the case that there is a 100% chance that a target is located within the entire search region. This probability is then divided as the tree goes down levels in depth. Also, in this case the quadtree is shown to be fully expanded to depth 3.

3.6.2 Integrating Sensor Readings: Top Down

In the original Type1 work [9], the tree was updated using a top down strategy with Bayesian estimation:

$$p_n^t = \Pr[X_n = 1 | Z^1, ..., Z^t]$$

The worked built on the equations by T.H. Chung in [8]. These equations are included and explained below:

$$\Phi(Z_n^t) = (1 - Z_n^t)(1 - \alpha(n)) + Z_n^t \alpha(n)$$

$$\Psi(Z_n^t) = (1 - Z_n^t)\beta(n) + Z_n^t(1 - \beta(n))$$

$$\Theta(i) = \begin{cases} \Psi(Z_n^t) & : n = i \\ \Phi(Z_n^t) & : n \neq i \end{cases}$$

Note that the above equations are for the case where the searcher is working on a uniform grid and not on a probabilistic quad tree. The n in the above equations refers to the cell in the grid being sensed. Φ indicates the probability of a correct no detection in the case Z_n^t is 0 and the probability of a false positive in the case Z_n^t is 1. Ψ indicates the probability that a detection was missed when Z_n^t is 0 and the probability a detection was correctly called when Z_n^t is 1. Θ represents the sensor model of the searcher and allows the correct values to be used depending if the cell being updated is within the area covered by n. Different values are used because of the correlations between cells due to the Type1 assumption. For example, if $Z_n = 1$ is read in a cell n different from the one being updated, cell i, then the sensor reading should be considered as a false positive when updating cell i with Bayes rule $\Pr[X_i = 1 | Z_n = 1]$. Since it is assumed that there is only one target within the environment that sensor reading must have been a false positive if $X_i = 1$ with $i \neq n$. However, if $i = n$ then the cells are the same and $\Pr[X_i = 1 | Z_n = 1]$ represents the probability of a correct detection. With the above formulas a Bayesian update can be written in a general sense for any cell i given a sensor reading Z_n^t.

$$p_i^t = \frac{\Theta(Z_n^t)p_i^{t-1}}{\Phi(Z_n^t)(1-p_i^t)+\Psi(Z_n^t)p_i^t} \tag{3.2}$$

By expanding on these definitions, the Type1 probability updates after receiving a sensor reading Z_n at node n can be done with the following top down strategy.

1. First, update the scanned node n using Eq. (3.2)

2. Next, this probability can be propagated downwards to its children c proportionally using the following relationship:

   $$p_c^t = p_n^{t-1} + \left(\frac{p_c^{t-1}}{p_n^{t-1}}\right)\delta p \quad \forall c \in \mathcal{C}(n)$$

   where

   $$\delta p = p_n^t - p_n^{t-1}$$

   Propagating in this manner is done because it assigns credit for the sensor reading proportionally based on the existing children probabilities. This rule is applied recursively for each child until all leaves descendant from n are updated.

3. Then update all leaf nodes in $\mathcal{T}$ that were not updated by the previous step. They are updated using the case of $\Theta(i)$ where $i \neq n$.

4. Lastly, update all internal nodes in $\mathcal{T}$ using the parent child relationship Eq. (3.1).

3.6.3 Integrating Sensor Readings: Bottom Up

This thesis proposes doing all updates in a completely bottom up manner. The new update method is introduced because it can better handle the Type2 case that will be described in sections below. In the Type1 case the update results in the same probabilities as the top down approach. If one is to assume that at most only one target is present in the entire quadtree area, correlations occur between nodes in the tree. This is because if there is an object in one area of the environment it cannot be located in any other area as well. Sensor readings can be integrated into a Type1 PQ as follows assuming a sensor reading z at node n:

1. First, use Bayes rule to update the leaf nodes i of the tree in the sensing area of the searcher.

   $$\Pr[X_i = 1 | Z_n = z] = \frac{\Pr[Z_n = z | X_i = 1] \Pr[X_i = 1]}{\Pr[Z_n = z]} \tag{3.3}$$

   $\Pr[Z_n = z | X_i = 1]$ is the sensor model for the searcher and indicates the probability of either getting a correct detection or a missed detection.

2. Next, use Bayes rule to update the leaf nodes o and the *null* node of the tree not in the sensing area of the searcher.

   $$\Pr[X_o = 1 | Z_n = z] = \frac{\Pr[Z_n = z | X_o = 1] \Pr[X_o = 1]}{\Pr[Z_n = z]} \tag{3.4}$$

   In this case, $\Pr[Z_n = z | X_o = 1]$ is the probability that the sensor incorrectly returned a detection or correctly indicated no detection.

3. Using the total probability theorem, $\Pr[Z_n = z]$ is equal to

   $$\Pr[Z_n = z] = \Pr[Z_n = z | X_n = 1] \Pr[X_n = 1] + \Pr[Z_n = z | X_n = 0] \Pr[X_n = 0] \tag{3.5}$$

   Also note that $\Pr[X_n = 0] = 1 - \Pr[X_n = 1]$ since X_n is a binary random variable.

4. Finally, propagate the changes in all of the leaf nodes up the their parents under the assumption that the probability of the parent node is the sum of its children.

All Type1 experiments were done using a bottom up approach unless stated otherwise

Refer to Algorithm 1 for a summary on the update process.

```
Algorithm 1: Type1Update
  Input: probabilistic quadtree $\mathcal{T}$, sensed node $n$, sensor reading $z$
  $inside \leftarrow \mathcal{S}(n) \cap \mathcal{L}(\mathcal{T})$
  $outside \leftarrow \mathcal{L}(\mathcal{T}) \setminus inside$
  foreach $i \in inside$ do
      Update $p_i$ using Eq. (3.3)
  foreach $o \in outside$ do
      Update $p_o$ using Eq. (3.4)
  $internal \leftarrow \mathcal{T} \setminus \mathcal{L}(\mathcal{T})$ //ordered deepest first
  foreach $i \in internal$ do
      Update $p_i$ using Eq. (3.1)
```

3.6.4 Determining Where to Search

The objective function that the searchers would like to minimize is the expected time to detect the target in the environment. The optimal way to compute this value is to try all sensing sequences and determine which one identifies the target with the most certainty the fastest. Unfortunately, this method is intractable so heuristic methods must be used. One way to reduce the computational burden is to use a greedy strategy and compute the best node for the next time step only without explicitly accounting for what may happen in the future. With these adjustments to the problem there are a variety of ways to determine where to search next. The first method that was applied to probabilistic quadtrees was to use a function based on the probability stored in each node [9]. A quality function was devised that divided each nodes probability by the distance the searcher would need to travel to get to this node. The idea behind this method is that it tries to drive the searcher to areas with high probability of containing an intruder, while also balancing travel cost and the benefits of hierarchical sensing. The exact formula was first described in [9] and is included here:

$$J(n') \triangleq \frac{p_{n'} 4^{d(n')}}{cost(n, n')}, n, n' \in \mathcal{N}(\mathcal{T}) \tag{3.6}$$

where d is returns the depth of a node. This objective function J can be computed in constant time. The next node n' can then be selected by taking the argmax of the objective function J above.

$$n' = \underset{n}{\operatorname{argmax}} J(n)$$

Note that using this method it takes linear time in the size of the tree to compute the next search location.

Alternatively, one can instead search nodes where the highest amount of expected information will be gained. This allows the searcher to find areas where it can reduce its uncertainty about the environment the most.

$$J \triangleq I(n'), n' \in \mathcal{N}(\mathcal{T}) \tag{3.7}$$

Using concepts from information theory, the expected information gain I can be computed as follows:

$$I(n) = H(\mathcal{T}) - E_{Z_n}\left[H(\mathcal{T}|Z_n)\right]$$

where $H(\mathcal{T})$ represents the entropy of $\mathcal{T}$, E_{Z_n} is the expectation with respect to possible sensor readings Z_n, and $H(\mathcal{T}|Z_n)$ represents the entropy of $\mathcal{T}$ after the inclusion of sensor reading Z_n. It can be shown that the entropy of a Type1 probabilistic quadtree is equal to the following:

$$H(\mathcal{T}) = -p_{\varnothing} \log_2 p_{\varnothing} - \sum_{n \in \mathcal{L}(\mathcal{T})} p_n \log_2 p_n \tag{3.8}$$

This formula is simply the entropy of a random variable with k different values where in this case k is the number of possible target locations. In the tree, the target must be located in one its leaves or the *null* cell outside of it. Since the target can only be in one of those locations, only k options are possible. The formula for the entropy of a random variable Y with k outcomes $\{y_1, y_2, ...y_k\}$ is included below.

$$H(Y) = -\sum_{i=1}^{k} p_i \log p_i$$

When computing the information gained by scanning a node, the entropy after the scan must be calculated. If the formula above is used then it takes linear time to compute this updated entropy and thus quadratic time to compute the information gain for all nodes in the tree. However, for the Type1 case with some manipulation the expected information gained by scanning at each node for all nodes in the tree can be computed in linear time rather than quadratic. This is a contribution of the thesis and is explored in greater detail in the next subsection.

The last method used for Type1 is a balance of information gain and distance. The searcher weighs the expected information gain from scanning a node with its distance away from the searcher using the formula below:

$$J \triangleq \left[\gamma \frac{I(n')}{max_{n^\star \in \mathcal{N}(\mathcal{T})} I(n^\star)} - (1-\gamma) \frac{D(n, n')}{max_{n^\star \in \mathcal{N}(\mathcal{T})} D(n^\star, n')} \right], n' \in \mathcal{N}(\mathcal{T}) \qquad (3.9)$$

Where D is a function that returns the euclidean distance between the center of two nodes and γ is a factor that weights the importance of information gain and distance. The idea behind this formula is to encourage the searcher to scan areas that are close to it. Since the searcher has limited search time it has to balance the trade off for traveling far to get lots of information and traveling a shorter distance to get less. The exact value for γ can be tuned depending on the importance of distance.

For completeness, non myopic techniques (i.e. techniques also considering future actions) were explored, but in our brief testing did not seem to yield much better search performance and required significantly more computation time to evaluate.

3.6.5 Expected Information Gain in Linear time

When deciding where to sense next the expected information gain must be computed for each node using the formula below.

$$I(n) = H(\mathcal{T}) - E_{Z_n}\left[H(\mathcal{T}|Z_n)\right]$$

where E_{Z_n} is the expectation with respect to the value Z_n sensed when scanning node n, and $H(\mathcal{T})$ is the entropy of $\mathcal{T}$ defined as

$$H(\mathcal{T}) = -p_{\varnothing}\log_2 p_{\varnothing} - \sum_{n\in\mathcal{L}(\mathcal{T})} p_n \log_2 p_n$$

Because of the correlations between leaf nodes in $\mathcal{T}$, when computing $H(\mathcal{T}|Z_n)$ one must compute new probability values for all nodes in $\mathcal{T}$. This probability update requires $\mathcal{O}(|\mathcal{N}(\mathcal{T})|)$ operations and thus the naive approach to computing $H(\mathcal{T}|Z_n)$ requires linear time. Since this computation has to be done for every node in $\mathcal{T}$ the required complexity is $\mathcal{O}(|\mathcal{N}(\mathcal{T})|^2)$. It is possible to reduce this complexity by exploiting properties of the PQ structure. We do this by first rewriting the entropy of the tree after a scan in two parts. One part containing all sensing locations covered by a scan and those outside of the sensing region.

$$H(\mathcal{T}|Z_n) = -\sum_{i\in\mathcal{I}_n(\mathcal{T})} p_i' \log_2 p_i' - \sum_{i\in\mathcal{O}_n(\mathcal{T})} p_i' \log_2 p_i'$$

Where p_i' is the probability of a target being located in node i after a scan at node n, $\mathcal{I}_n(\mathcal{T})$ is the set of leaves of $\mathcal{T}$ that are within the region covered by node n and $\mathcal{O}_n(\mathcal{T})$ is the set of leaves of $\mathcal{T}$ that are outside of the region covered by node n. The null node $p_{\varnothing}$ is also considered to be part of the outside set. By using Bayes rule, the probability at node i after a scan can be computed as

$$p_i' = \frac{\Pr[Z_n = z|X_i = 1]\Pr[X_i = 1]}{\Pr[Z_n = z]}$$

From the section above about bottom up updates, it was shown that $\Pr[Z_n = z|X_i = 1]$ has different meanings and values depending if the node being updated is inside or outside of the sensing region.

$$\Pr[Z_n = z|X_i = 1] = \Pr[Z_n = z|X_n = 1], \quad \forall i \in \mathcal{I}_n(\mathcal{T})$$

$$\Pr[Z_n = z|X_i = 1] = \Pr[Z_n = z|X_n = 0], \quad \forall i \in \mathcal{O}_n(\mathcal{T})$$

One situation is where node i is within the sensing area which means this is the probability that the sensor returns a correct detection or it misses the detection. This case will be captured by the constant η_n

$$\eta_n = \frac{\Pr[Z_n = z|X_i = 1]}{\Pr[Z_n = z]} = \frac{\Pr[Z_n = z|X_n = 1]}{\Pr[Z_n = z]}, \quad \forall i \in \mathcal{I}_n(\mathcal{T})$$

The other case is when node i is outside of the sensed area. This means that $\Pr[Z_n = z|X_i = 1]$ represents the probability of a correct no detection and a false positive. This case will be captured by the constant ζ

$$\zeta_n = \frac{\Pr[Z_n = z|X_i = 1]}{\Pr[Z_n = z]} = \frac{\Pr[Z_n = z|X_n = 0]}{\Pr[Z_n = z]}, \quad \forall i \in \mathcal{O}_n(\mathcal{T})$$

In both cases when applying Bayes rule, this update can be considered to be a multiplication of one of the constants above times the probability that node i contains a target. Thus two situations arise:

The updated node may be inside the scanned area

$$p_i' = \eta_n p_i \quad \forall i \in \mathcal{I}_n(\mathcal{T})$$

or outside of it

$$p_i' = \zeta_n p_i \quad \forall i \in \mathcal{O}_n(\mathcal{T})$$

By substituting these values into the equation above the following result is obtained:

$$H(\mathcal{T}|Z_n) = -\sum_{i \in \mathcal{I}_n(\mathcal{T})} \eta_n p_i \log_2(\eta_n p_i) - \sum_{i \in \mathcal{O}_n(\mathcal{T})} \zeta_n p_i \log_2(\zeta_n p_i)$$

By using the property that $\log(ab) = \log(a) + \log(b)$

$$\begin{aligned} H(\mathcal{T}|Z_n) = &- \sum_{i \in \mathcal{I}_n(\mathcal{T})} [\eta_n p_i \log_2(\eta_n) + \eta_n p_i \log_2(p_i)] \\ &- \sum_{i \in \mathcal{O}_n(\mathcal{T})} [\zeta_n p_i \log_2(\zeta_n) + \zeta_n p_i \log_2(p_i)] \end{aligned}$$

This can be divided further into a sum of 4 terms

$$\begin{aligned} H(\mathcal{T}|Z_n) = &- \eta_n \log_2(\eta_n) \sum_{i \in \mathcal{I}_n(\mathcal{T})} p_i - \eta_n \sum_{i \in \mathcal{I}_n(\mathcal{T})} p_i \log_2(p_i) \\ &- \zeta_n \log_2(\zeta_n) \sum_{i \in \mathcal{O}_n(\mathcal{T})} p_i - \zeta_n \sum_{i \in \mathcal{O}_n(\mathcal{T})} p_i \log_2(p_i) \end{aligned}$$

At this point, two of the sums can replaced with quantities that are known before the update. The sum of the probabilities p_i inside the sensing region is the same as the probability that the target is located in node being sensed. This is true because of the Type1 assumption that only 1 target is present in the entire search region. Since scanned regions correspond to nodes on the quadtree, this probability sum can be replaced with the value p_n assuming the scan occurs at node n.

$$\begin{aligned} H(\mathcal{T}|Z_n) = &- \eta_n \log_2(\eta_n) p_n - \eta_n \sum_{\mathcal{I}_n(\mathcal{T})} p_i \log_2(p_i) \\ &- \zeta_n \log_2(\zeta_n)(1 - p_n) - \zeta_n \sum_{\mathcal{O}_n(\mathcal{T})} p_i \log_2(p_i) \end{aligned}$$

The outside sum was replaced as well since the probability that a target is found outside a node must be $1 - p_n$ because the target is either inside or outside of node n. The remaining sums can also be replaced in a similar manner by pre computing possible values for the sums. This means that they can be computed before evaluating the information gain at each node. These values can be handled similarly to probabilities in a Type1 tree. By defining a new term for leaf nodes in the tree:

$$h_i = p_i \log_2(p_i) \quad \forall i \in \mathcal{L}(\mathcal{T})$$

Substituting this into the above equation yields

$$\begin{aligned}H(\mathcal{T}|Z_n) = &- \eta_n \log_2(\eta_n)p_n - \eta_n \sum_{\mathcal{I}_n(\mathcal{T})} h_i \\ &- \zeta_n \log_2(\zeta_n)(1-p_n) - \zeta_n \sum_{\mathcal{O}_n(\mathcal{T})} h_i\end{aligned}$$

In order to remove the remaining sums, the values of h_i can be precomputed for internal nodes. This is done by defining

$$h_i = \sum_{c \in \mathcal{C}(i)} h_c \quad \forall i \in \mathcal{IN}(\mathcal{T})$$

for all internal nodes. This means that the values of h_i can be computed for all nodes in the tree in linear time in the size of the tree before any information gain computations take place. One way to compute h_i for all nodes in linear time is to use a simple recursive algorithm 2 starting from the root of the tree. The algorithm works by first computing h for each leaf. Then each internal node gets its h value set to the sum of its children's h values. Computing the h values in this way requires visiting each node in the tree one time, resulting in a linear computation for the values of h in $\mathcal{T}$.

Algorithm 2: Compute_h

```
Input: probabilistic quadtree node $i$
if $i \in \mathcal{L}(\mathcal{T})$ then
    $h_i \leftarrow p_i \log_2(p_i)$
else
    $h_i \leftarrow 0$
    foreach $c \in \mathcal{C}(i)$ do
        $h_i \leftarrow h_i + Compute\_h(c)$
```

Using the computed h values above, the entropy equation can be simplified further. The sums of h_i can be replaced by applying similar logic as in the p_i case. The main difference is that the sum over the outside regions is now equal to

$-H(\mathcal{T}) - h_n$ because the sum over all leaf h_i values is equal the negative entropy of the tree. The portion of entropy outside must be whatever portion that is left over after removing the portion covered by h_n. This yields the final equation for computing the entropy given a sensor reading at node n

$$H(\mathcal{T}|Z_n) = -\eta_n \left[\log_2(\eta_n)p_n + h_n\right] - \zeta_n \left[\log_2(\zeta_n)(1-p_n) - (H(\mathcal{T}) + h_n)\right] \quad (3.10)$$

The resulting equation yields a computation that can be done in constant time for each node in the quadtree. By using this technique, the node maximizing expected information gain can be determined in linear time using Algorithm 3.

```
Algorithm 3: Compute_InformationGain
  Input: probabilistic quadtree $\mathcal{T}$
  Compute $H(\mathcal{T})$ using Eq. (3.8)
  Compute $h_n \ \forall n \in \mathcal{N}(\mathcal{T})$ using Algorithm 2
  Compute $H(\mathcal{T}|Z_n = z) \ \forall n \in \mathcal{N}(\mathcal{T}), \ \forall z \in \{0,1\}$ using Eq. (3.10)
  Compute $I(n) = H(\mathcal{T}) - E_{Z_n}[H(\mathcal{T}|Z_n)] \ \forall n \in \mathcal{N}(\mathcal{T}), \ \forall z \in \{0,1\}$
  Return $I$
```

In Algorithm 3, each step operates in $\mathcal{O}(k)$ where k is the number of nodes in the probabilistic quadtree $\mathcal{T}$. Thus the entire method takes $\mathcal{O}(k)$ time to compute the expected information gain from scanning at any node inside of $\mathcal{T}$.

3.6.6 Expanding the Tree

In our tests, the searcher would only add new nodes to its quadtree when receiving a sensor reading at a leaf node indicating a target was present. To add new nodes, the searcher marks the leaf node as internal and adds 4 children to this node. It is assumed that none of these new children are more likely than any of the others to contain a target so their probabilities are each initialized to:

$$p_c = \frac{p_n}{4} \quad \forall c \in \mathcal{C}(n) \quad (3.11)$$

3.6.7 Stopping the Search

In the Type1 case, searchers will continue searching until a cell at the maximum depth of the quadtree contains a probability that is above a given threshold. This threshold could be set to 1 if it is desired that the searcher searches for the maximum mission duration. In general, this threshold represents the confidence required for the searcher to stop its search.

3.6.8 Algorithm

```
Algorithm 4: Type1_Searcher
Input: prior belief distribution
$\mathcal{T} \leftarrow constructTree(prior)$
while not searchDone do
    Compute $J(n) \quad \forall n \in \mathcal{N}(\mathcal{T})$ using Eq. (3.9)
    $n' \leftarrow \text{argmax}_{n \in \mathcal{N}(\mathcal{T})} J(n)$
    Travel To $n'$
    $z \leftarrow \text{Scan}(n')$
    Type1Update($\mathcal{T}, n', z$) using Algorithm 1
    if $z = 1 \wedge depth(n) < \mathcal{D}$ then
        Add 4 new children to $\mathcal{T}$ each with probability $\frac{p_n}{4}$. See Eq. (3.11)
    foreach $n \in \mathcal{L}(\mathcal{T})$ do
        if $depth(n) == \mathcal{D} \wedge p_n > threshold$ then
            SearchDone
            Report $n$
```

Algorithm 4 is formed by putting together the pieces from the previous subsections and provides a method to search on a PQ. First the searcher constructs a tree based on the provided prior target distribution. Then as long as the search is not done the searcher computes a good sensing location. It then moves to this location, senses, and integrates this reading into its PQ. Once the sensor reading is integrated into the searcher's tree, the searcher decides if it has found the target with enough certainty. If the searcher is confident enough, it can stop the search. Otherwise, it continues by picking a next sensing location and repeating the process. Note that the searcher must find the target at the deepest level in the tree.

3.7 The Type2 Scenario

In another scenario that we call Type2, the searcher is searching an area with an unknown number of targets and assumes that each cell of the environment is independent from each other. Since each cell is independent, the additional null node that was added in Type1 for the case where the target is outside of environment is not needed. This is because the tree no longer represents the possible location for a single target and thus the probabilities no longer sum to 1.

3.7.1 Belief Representation

The above Type1 methods can be extended to work in the case of multiple targets. The first change is that nodes in the tree store the probability that at least one target is present in the area. Also, since the probabilities for each child are independent from each other and each child may contain a target the probability of a parent node can be determined using the following formula:

$$p_n = 1 - \prod_{c \in \mathcal{C}(n)} (1 - p_c) \quad \forall n \in \mathcal{IN}(\mathcal{T}) \tag{3.12}$$

This formula states that the probability that a parent node contains at least one target is equal to 1 minus the probability that all of its children nodes are empty.

3.7.2 Integrating Sensor Readings: Top Down

In the first Type2 paper [7], sensor readings were propagated downwards when sensing occurred at an internal node. First, the probability of the scanned internal node would be updated using the same Bayes rule formula as Type1 Eq.(3.2). Then the probability change would be propagated uniformly to its children using the following ideas:

Let $q_n = 1 - p_n$ and $q_n = q_1 q_2 q_3 q_4$ where q_i represents the probability that child i is empty. Assume that q'_n is the probability that node n is empty after a scan at node n. It follows that $q'_n = q_n^k$ for some value k. Where k is the power needed to raise q_n to the new probability after applying Bayes rule. This value k is useful because it allows propagation of probability changes downwards in the tree. It also means that

$$q'_n = q_n^k$$

$$q'_n = [q_1 q_2 q_3 q_4]^k$$

$$q'_n = q_1^k q_2^k q_3^k q_4^k$$

Meaning that each child's probably after an update can be set to $q'_i = q_i^k$. These probability changes can be propagated all the way down to the leaves in this manner and then propagated back up to the root of the tree using the parent child relationships mentioned above (3.12).

3.7.3 Integrating Sensor Readings: Bottom Up

By using a bottom up approach, the update formula can be very similar to Type1 except that in the Type2 scenario only the leaves in the tree that are under the scan are updated rather than all leaves in the tree. This is because in the Type2 case there is no longer a correlation between leaf nodes. The following formula can be used to update the scanned leaf nodes i after a sensor reading:

$$\Pr[X_i = 1|Z_n = z] = \frac{\Pr[Z_n = z|X_i = 1]\Pr[X_i = 1]}{\Pr[Z_n = z]} \tag{3.13}$$

$\Pr[Z_n = z|X_i = 1]$ is the sensor model for the searcher and indicates the probability of either getting a correct detection or a missed detection. Unlike the Type1 case the outside leaves do not need to be updated since they are independent from those scanned. These probability changes are then propagated back upwards using the parent child relationship mentioned in the section above. The advantage of the bottom up update approach is that it uses properties of the Type2 PQ to perform the update rather than making the uniform assumption that is made for top down updates when propagating changes downward. An additional advantage is that the bottom up update method proves a more conservative change in the probabilities. When using the top down update method the probabilities tend to jump quickly to the extremes, 0 or 1, even though the sensor cannot be sure which cell contains or does not contain a target. An example of how the probabilities change in the two update methods is shown in Figure 3.5.

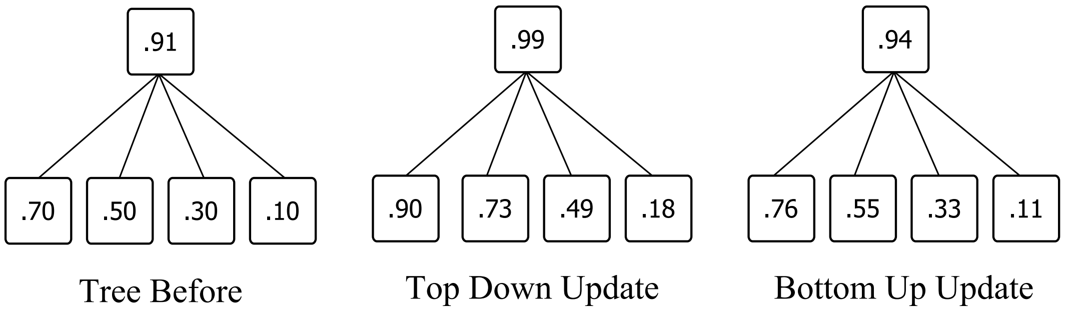

Figure 3.5: Update Example. This figure shows an example update on Type2 PQ with 5 nodes. The sensor reading occurs at the root of the tree with $\alpha = 0.1$ and $\beta = 0.1$. The probabilities are rounded for display.

All Type2 and patrolling experiments were done using a bottom up approach unless stated otherwise

Algorithm 5 provides a brief method for updating nodes in a Type2 PQ after a sensor reading.

Algorithm 5: Type2Update

```
Input: probabilistic quadtree $\mathcal{T}$, sensed node $n$, sensor reading $z$
$inside \leftarrow \mathcal{S}(n) \cap \mathcal{L}(\mathcal{T})$
foreach $i \in inside$ do
    Update $p_i$ using Eq. (3.13)
$internal \leftarrow \mathcal{T} \setminus \mathcal{L}(\mathcal{T})$ //ordered deepest first
foreach $i \in internal$ do
    Update $p_i$ using Eq. (3.12)
```

3.7.4 Determining Where to Search

In our tests, only the information gain based method was used to search in the Type2 case. Searching based only on probability is more difficult because if the searcher favored an area with high probability it would find one target and continually sense that node since it would have the highest chance of containing a target. Once the searcher has identified a target it should not continue to scan that cell since other cells need to be explored. One can get around this problem by assigning thresholds for when to ignore cells, but the entropy based methods get around this problem without needing to tune such parameters. This is because entropy based methods already favor searching areas with the most uncertainty. As in the Type1 case, the information gain for scanning at node n in a Type2 PQ can be computed as

$$I(n) = H(\mathcal{T}) - E_{Z_n}\left[H(\mathcal{T}|Z_n)\right]$$

The main difference from the Type1 case is in the way the entropy of the tree is calculated. For a Type2 tree the entropy is equal to

$$H(\mathcal{T}) = -\sum_{l \in \mathcal{L}(\mathcal{T})} \left[p_l \log_2 p_l + (1-p_l)\log_2(1-p_l)\right]$$

This entropy is the sum of the entropies of the individual independent leaf binary random variables. Note that the information gain calculation for all nodes cannot be done in linear time as in the Type1 scenario. In short, the strategy used for pre computing in the Type1 scenario fails due to the $\log_2(1-p_l)$ term. Instead, when calculating the information gain for scanning at a node, the entropy must be calculated using the equation above which takes linear time. Thus resulting in a $\mathcal{O}(|\mathcal{N}(\mathcal{T})|^2)$ computation for calculating the information gain for every node in the tree.

3.7.5 Expanding the Tree

Much like the Type1 case the tree is only expanded when a positive reading occurs. The difference is how the probabilities for the children nodes are assigned. We again assume that no child node is more likely to contain a target. This yields 4 new children each with the probability:

$$p_c = 1 - \sqrt[4]{1-p_n} \tag{3.14}$$

3.7.6 Stopping the Search

Unlike the Type1 case, the number of intruders in the environment is unknown so the searcher cannot stop once it has found all the targets. Instead, the searcher is allowed to stop once it has reached a certain confidence about its knowledge of the environment. To achieve this, the stopping criterion used is the one proposed in [27]. Let U be defined as:

$$U(\mathcal{T}) = \frac{\sum_{l\in\mathcal{L}(\mathcal{T})} H(l)}{|\mathcal{L}(\mathcal{T})|\, H_{max}} \tag{3.15}$$

Where H is the entropy defined above and H_{max} is the largest occurring entropy in the leaves of the tree. The search ends when the searcher runs out of time or the entropy for each leaf falls below $\varepsilon U(\mathcal{T})$. By decreasing ε, the searcher can be forced to search for more time before ending its search. This stopping criterion prevents the searcher from having leaf nodes with entropies that are much higher than other leaves.

3.7.7 Declaring Targets

Rather than use a threshold method as in the Type1 case we use a cost based approach to determine if a target is present at a node.

For each node, a binary decision must be made D_n indicating if a target is present $D_n = 1$ or no target $D_n = 0$.

With this formulation, the cost associated with making a decision at node n can be formulated as

$$\begin{aligned} C_n = &\Pr[D_n = 1|X_i = 0]\Pr[X_i = 0]C_{10} \\ &+ \Pr[D_n = 0|X_i = 1]\Pr[X_i = 1]C_{01} \\ &+ \Pr[D_n = 0|X_i = 0]\Pr[X_i = 0]C_{00} \\ &+ \Pr[D_n = 1|X_i = 1]\Pr[X_i = 1]C_{11}. \end{aligned} \tag{3.16}$$

Where C_{ij} is the cost to the searcher when decision $D_n = i$ is made when $X_n = j$. For the simplest case of only one sensor reading $D_n = 0$ should be chosen when the following is true [20]

$$\frac{\Pr[Z|X_n = 0]}{\Pr[Z|X_n = 1]} > \frac{(C_{11} - C_{01})\Pr[X_n = 1]}{(C_{00} - C_{10})\Pr[X_n = 0]} \tag{3.17}$$

$D_n = 1$ should be selected otherwise. In our case, many sensor readings are taken over the course of the mission. Let $Z_n^1, ..., Z_n^m$ represent the sequence of m sensor readings over a node n. The above decision rule can be changed to the following.

$$\frac{\Pr[Z_n^m|X_n = 0]}{\Pr[Z_n^m|X_n = 1]} > \frac{(C_{11} - C_{01})\Pr[X_n = 1|Z_n^1, ..., Z_n^{m-1}]}{(C_{00} - C_{10})\Pr[X_n = 0|Z_n^1, ..., Z_n^{m-1}]}. \tag{3.18}$$

Rather than compute this value at the end of a search mission, one can show that it is simpler to compute the decision after each sensor reading using 3.17.

This removes the need to remember each individual sensor reading and allows the searcher to have a decision at any point during the mission.

3.7.8 Algorithm

Algorithm 6: Type2_Searcher

```
Input: prior belief distribution
$\mathcal{T} \leftarrow constructTree(prior)$
while not searchDone do
    Compute $J(n) \quad \forall n \in \mathcal{N}(\mathcal{T})$ using Eq. (3.9)
    $n' \leftarrow \text{argmax}_{n \in \mathcal{N}(\mathcal{T})} J(n)$
    Travel To $n'$
    $z \leftarrow \text{Scan}(n')$
    Type2Update($\mathcal{T}, n', z$) using Algorithm. 5
    if $depth(n) = \mathcal{D}$ then
        Update Decision for node $n$ using Eq. (3.17)
    if $z = 1 \wedge depth(n) < \mathcal{D}$ then
        Add 4 new children to $\mathcal{T}$ each with probability $\frac{p_n}{4}$. See Eq. (3.14)
    Compute $U(\mathcal{T})$ using Eq. (3.15)
    SearchDone $\leftarrow true$
    foreach $n \in \mathcal{L}(\mathcal{T})$ do
        if $-p_n \log_2 p_n - (1 - p_n) \log_2(1 - p_n) > \epsilon U(\mathcal{T})$ then
            SearchDone $\leftarrow false$
Report leaves at depth $\mathcal{D}$ with $D_n = 1$
```

Algorithm 6 shows the general search algorithm for the Type2 scenario formed by putting together all the pieces in the previous sub sections. The algorithm works similar to the Type1 case in that the searcher computes the best node to sense, travels to it, reads from its sensor, and integrates the reading into its PQ. The difference is that the decision variables are updated after each sensor reading and the value of U is computed and compared to each leaf node to determine if the search should continue.

3.8 Patrolling Scenario

By expanding on the Type2 scenario, we can also introduce the notion of intruders that come in to the environment during the mission. In this scenario, it is the job of the searcher to locate these intruders as quickly as possible to reduce the damage or loss in an area. For this case the PQ is modified to include a few new pieces of information for each node: loss and detrimental. The added loss values allow the searcher to consider some nodes as more important than others and the detrimental informs the searcher about the rate that intruders are entering a particular node. Both these values will be described in greater detail in the next section.

3.8.1 Formulation and objective

The following derivations in this subsection originate from [2]. It is assumed in the patrolling scenario that the search region $\mathcal{A}$ is divided up into cells that correspond to regions covered by the leaf nodes in a fully expanded quadtree. Each of these cells in the environment has an associated loss which is a penalty that is paid each time step that an intruder is in that cell. Let l be a function that maps a cell to its loss value. Let another function a indicate if a target is in cell c at time t. Over the time period [0,T] one can write the accumulated loss as

$$\rho = \sum_{t=0}^{T} \sum_{c \in \mathcal{A}} a(c,t) l(c)$$

It is the goal of the searcher to minimize this function by identifying targets and deciding when to remove them from the environment. In addition to a loss value, each cell in the environment has an associated probability that an intruder will enter the area. Let π_c be the probability of having an intruder enter cell c within one unit of time. This intruders appear into cell c without travelling and do not move once they enter the environment. Additionally, once an intruder enters the environment it does not leave until removed by the searcher. The searcher must pay a cost to remove targets so it will only attempt to remove them once it is certain enough about their presence. Also, note that once the searcher decides to remove a target the node can be considered clear of any intruders as removal is assumed to not fail. The value of π for each node in the quadtree can then be computed by multiplying the probably that no attacks occur in any of the cells contained within node n

$$\pi_n = 1 - \prod_{c \in R(n)} (1 - \pi_c)$$

This means that the probability of an intruder being in any node within the probabilistic quadtree changes over time. This is done by computing 1 minus the probability that the node in the tree remains empty over Δt units of time. To compute the probability of the node remaining empty, multiply its probability of being empty times the chance a target does not enter for each time step. Thus, over Δt units of time the probability in a node changes to

$$p_n^{t+\Delta t} = 1 - (1 - p_n^t)(1 - \pi_n)^{\Delta t}. \tag{3.19}$$

3.8.2 Determining Where to Search

In this scenario, methods based completely on entropy do not work without modification. This is because the searcher may spend most of its time reducing the uncertainty in the PQ and never be sure enough of a particular intruder location. This constant attempt to reduce uncertainty can prevent the searcher from removing any intruders and thus can cause the searcher to pay a large penalty during the mission. To combat this, the searcher needs to scan in areas that are likely to contain an intruder. One proposed strategy in this domain is searching areas weighted by loss, probability, and area [2]. First define a function j which measures the value of a node n

$$j(n) = \frac{l(n)\mu(n)}{A(n)}$$

where $\mu(n)$ is the expected number of targets in node n and $A(n)$ is the area of the region associated with node n. Using this function, a new objective function weighted by the distance D can be written

$$J \triangleq \gamma \frac{j(n')}{max_{n^\star \in \mathcal{N}(\mathcal{T})} j(n^\star)} - (1-\gamma)\frac{D(n,n')}{max_{n^\star \in \mathcal{N}(\mathcal{T})} D(n^\star, n')} \tag{3.20}$$

The benefit of weighing by distance is to encourage the searcher to look at nearby nodes rather then potentially travelling far to scan a slightly better node. This reduction in time spent travelling is intended to help increase the amount of time the searcher can spend making sensor reads and removing targets. In this thesis, we also propose another search method based on weighing probability, entropy, and distance. The idea is that internal nodes will be scored based on their entropy and leaf nodes will be scored by their probability. This allows the searcher to consider areas inside the tree where it can scan to gain lots of information, while also allowing it to sometimes scan leaves with high probability and remove any intruders at those cells. By tuning the weights, one can control how much the searcher favors exploring versus trying to remove intruders. The formula used is shown below:

$$\begin{aligned} J \triangleq & \gamma \left[\omega \frac{P(n')}{max_{n^\star \in \mathcal{N}(\mathcal{T})} P(n^\star)} + (1-\omega)\frac{H(n')}{max_{n^\star \in \mathcal{N}(\mathcal{T})} H(n^\star)}\right] \\ & - (1-\gamma)\frac{D(n,n')}{max_{n^\star \in \mathcal{N}(\mathcal{T})} D(n^\star, n')} \\ & n' \in \mathcal{N}(\mathcal{T}), \omega \in [0,1], \gamma \in [0,1] \end{aligned} \tag{3.21}$$

Where ω controls how much the searcher favors gathering information versus removing targets and γ controls the importance of distance. The other values in the equation are

$$P(n) = \begin{cases} p_n l_n & : n \in \mathcal{L}(\mathcal{T}) \\ 0 & : otherwise \end{cases}$$

$$H(n) = \begin{cases} l_n \left[-p_n \log_2 p_n - (1-p_n)\log_2(1-p_n)\right] & : n \in \mathcal{IN}(\mathcal{T}) \\ 0 & : otherwise \end{cases}$$

3.8.3 Expanding the Tree

Tree expansion is done identically to the Type2 scenario.

3.8.4 Stopping the Search

In the patrolling scenario, the searcher continues searching for the entire mission duration because it assumed that a target can enter the environment at any moment and the number of intruders is potentially unlimited.

3.8.5 Algorithm

```
Algorithm 7: Patrolling_Searcher
Input: prior belief distribution
$\mathcal{T} \leftarrow constructTree(prior)$
while not searchDone do
    Compute $J(n) \quad \forall n \in \mathcal{N}(\mathcal{T})$ using Eq. (3.21)
    $n' \leftarrow \text{argmax}_{n \in \mathcal{N}(\mathcal{T})} J(n)$
    Travel To $n'$
    $z \leftarrow \text{Scan}(n')$
    Type2Update($\mathcal{T}, n', z$) using Algorithm. 5
    Detrimental Update: $p_n \quad \forall n \in \mathcal{N}(\mathcal{T})$ using Eq.(3.19)
    if $depth(n) = \mathcal{D}$ then
        Update Decision for node $n$ using Eq. (3.18)
        if $D_n = 1$ then
            Attempt to remove intruder at $n$
    if $z = 1 \wedge depth(n) < \mathcal{D}$ then
        Add 4 new children to $\mathcal{T}$ each with probability $\frac{p_n}{4}$. See Eq. (3.14)
```

Algorithm 7 shows the method used to control the patrolling searcher formed by putting together all the pieces in the previous sub sections. It behaves similar to the Type2 scenario with a few modifications. First after performing a Type2 update the detrimental effect must be accounted for. This increases the probability that intruders are inside of the nodes in the PQ. Additionally, if a decision value is ever 1 for a leaf node in the tree at the maximum depth then the searcher will remove any intruder in the node from the environment.

3.9 Probabilistic Quadtree Compression

Compression of a probabilistic quadtree can be done in several ways. The first proposed method in [9] utilizes the idea of disparity of a node where

$$disparity(n) = max_{c \in \mathcal{C}(n)} p_c - min_{c \in \mathcal{C}(n)} p_c$$

The goal of disparity is to remove nodes where the probability values differ greatly. The idea is that areas with smaller differences lose less information when compressed in the PQ structure. The method compresses a tree by first building a full tree given an initial belief distribution. Then nodes with the lowest disparity are removed until the tree is of the desired size. An alternative approach is proposed in this thesis that considers all children of a node rather than just its min and max. First take an initial prior and construct the full quadtree for that prior. Then as long as the tree contains more than the desired amount of nodes, greedily remove nodes that impact the error the least. More explicitly the error of a node n is defined as follows:

$$Error(n) = \sum_{l \in \mathcal{I}(n)} (ExpandedProb(n, l) - p_l)^2 \tag{3.22}$$

Where $\mathcal{I}(n)$ is the set of leaves that are descendants of node n. If two nodes share the same error the node with the greater depth in the tree is removed. The function $ExpandProb$ returns the probability that would be in leaf l if node n was split evenly to the same depth as l. For the Type1 case the expanded probability is calculated as:

$$ExpandedProb(n, l) = \frac{p_n}{4 * [depth(l) - depth(n)]}$$

Where as in the Type2 or patrolling case it is:

$$ExpandedProb(n, l) = (1 - p_n)^{-4 * [depth(l) - depth(n)]}$$

Note that since the above method is greedy it will not return the optimal compression of the probabilistic quadtree, but it worked well enough in practice.

The method also has the benefit or working in either of the Type1, Type2, and patrolling cases. The only difference is in the way the probabilities of the expanded tree are computed.

Algorithm 8: TreeCompression

```
Input: probabilistic quadtree $\mathcal{T}$, maximum number of nodes k
foreach $n \in \mathcal{N}(\mathcal{T})$ do
    Compute Error($n$) using Eq. (3.22)
$\mathcal{T}' \leftarrow \mathcal{T}$
while $|\mathcal{T}'| > k$ do
    $n \leftarrow \text{argmax}_{n \in \mathcal{IN}(\mathcal{T})} Error(n)$
    $\mathcal{T}' \leftarrow \mathcal{T}' \setminus \mathcal{C}(n)$
return $\mathcal{T}'$
```

Example compression results for the Type1 PQ in Figure 3.6 be seen in Figure 3.7 and Figure 3.8. Note that in this case the results are very similar for the two methods.

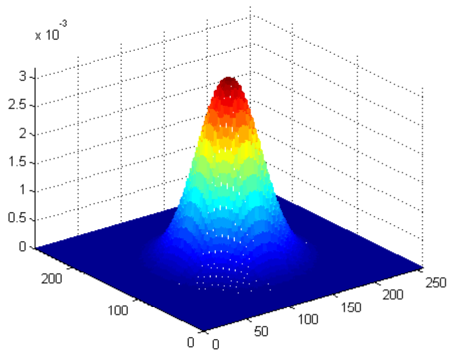

Figure 3.6: A full tree with 5461 Nodes

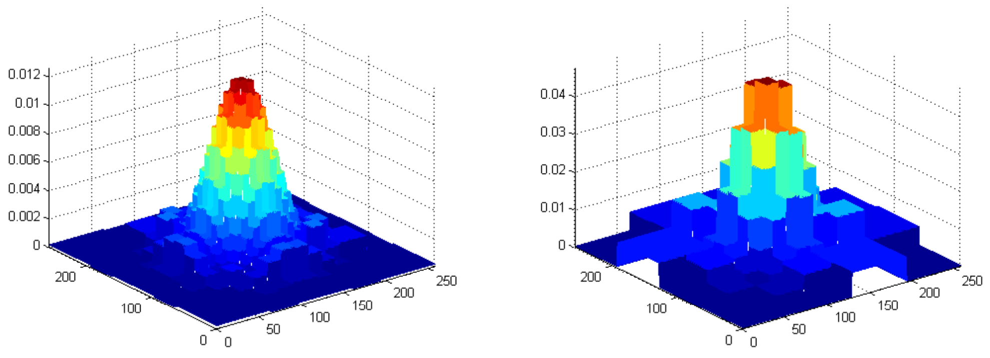

Figure 3.7: Compression using the error based approach with 400 and 100 nodes respectively

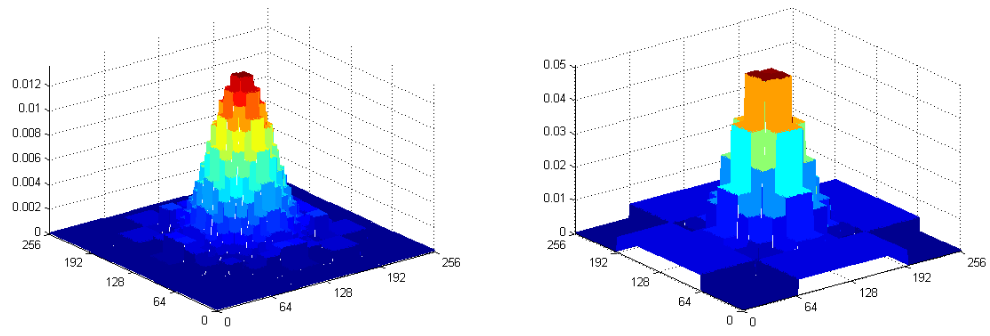

Figure 3.8: Compression using the disparity based approach with 400 and 100 nodes respectively

CHAPTER 4

Multi-Agent Strategies

All of the above tree scenarios can be expanded to handle the case of multiple searchers in the environment. The main difficulty is determining how they should coordinate. On one side of the spectrum there is the idea of a single controller which knows all information about the searchers and makes decisions for all searchers. This controller has the most information and can potentially yield the best performance. The downside is the solution requires full communication and is less resilient to failure due to its reliance on a single controlling agent. If the controlling agent fails then the entire search mission is over. At the other end of the spectrum is a fully distributed solution where the searchers each run their own controller and do not communicate with one another.

4.1 Multi-Agent Type1

4.1.1 Independent Searchers

The simplest multi-agent search strategy is to have each searcher preform the search independently. This means that no information is shared between the searchers. The search mission ends as soon as at least one of the searchers has identified the location of the target within a confidence threshold. This strategy is used as a baseline for other controllers because it is the simplest method for scaling to multiple searchers and can be improved in various ways.

4.1.2 Single Controllers

Alternatively, a few single controller strategies were tested to see how much performance can be improved using a centralized controller. This meant that one agent was controlling the actions of the other agents and it could then coordinate the searchers efforts intelligently. By incorporating knowledge of other searchers readings and locations, the search effort can be spread out more effectively and reduce wasted scans. For each of the single controllers discussed in this thesis, searchers were never assigned the same sensing location as another searcher to prevent wasted search effort. Each of the following three strategies were implemented:

- One controller would wait to give searchers new goals until all searchers reached their goal. When assigning new locations, it selected the top r goals where r is the number of robots and assigned them to team members. One should note that these goal locations may overlap and thus not be the true optimal assignment of the r searchers.
- In order to reduce the time wasted by searchers waiting for the others to finish, an additional controller was introduced that would immediately assign a new goal to a searcher once it finished a scan. This new goal could not be the same as any of the other searchers.
- The last controller performs a full re plan whenever any searcher completes a scan. This causes each searcher to take advantage of every scan and causes any searcher in flight to potentially reconsider their goal if a better goal location is now available.

It is likely possible to improve on these controllers further, but they served as target performances for the other more distributed controllers.

4.1.3 Partitioning Based Controllers

One simple way to divide the work among the searchers is to partition the search area and assign searchers to particular regions. The advantage of this method is that the searchers do not need to communicate during the mission and can still divide up the search effort. In the Type1 senario, partitioning is done by assigning searchers to a sub tree of the original quadtree used in the single searcher Type1 controller. For cases where the number of searchers is not a power of four, a searcher can be assigned to multiple sub trees. Similar to the multi-agent strategies above the mission ends once any one of the searchers find the target with enough confidence. The downside of this method is that without communication and a more complicated control scheme the searchers' search regions do not change as the mission progresses. Since the target can only be in one of the search regions, this means that eventually search effort will be wasted because most of the searchers are stuck searching areas with a very low probability of finding the target. In order to alleviate this issue, repartitioning via communication can be introduced to their controllers, but it is not explored in this thesis.

4.1.4 Communicating Searchers

Another strategy without partitioning is to allow the searchers to communicate with each other. This was done by allowing searchers within a given radius to receive information whenever a nearby searcher performs a scan. Information passed included the scan result and its next goal location. This allows the searchers to incorporate the data of others and consider the locations of other searchers so the search effort is not wasted by sensing the same nodes as each other. In the experimental section of this thesis, the influence of communication radius on search performance is evaluated. Additionally, experiments were done to test the importance of sharing location versus sharing scan information.

4.2 Multi-Agent Type2

Given that the leaf nodes of a Type2 tree are independent the scenario lends itself very well to partitioning if no information can be shared between searchers. If searchers are able to communicate, one way to take advantage of this is to repartition throughout the mission such that the searchers better cover areas where little information is known; however, no experiments with repartitioning were explored in this thesis.

4.3 Multi-Agent Patrolling

4.3.1 Independent Searchers

Similar to the Type1 case mentioned above, patrolling tests were done with searchers that shared no information between each other and patrolled the environment independently.

4.3.2 Partitioning Methods

More advanced partitioning was attempted for the patrolling problem. An integer linear programming optimization problem was solved to determine optimum partitions for searchers based on a given loss. The equations below were derived by Nicola Basilico and are included here for completeness. Note that in this form of portioning the searchers are each given their own quadtree to search and that the areas covered by these trees can potentially overlap.

Assumptions:

- $q \in Q$ quadtree roots, each one characterized by an altitude $h(q)$

- $g \in G$ grid cells, each one corresponding to a leaf of the quadtree and characterized by a loss $l(g)$
- $\gamma(q, g) = 1$ if g falls in the visibility region of q
- $r \in R$ set of robots with $|R| = k$

Decision variables

- $x_{q,r}$, binary, equal to 1 iff robot r operates on the quadtree with root in q
- $y_{g,r}$, binary, equal to 1 iff robot r's quadtree has a leaf in g

$$\max \sum_{g \in G} u(g)$$

s.t.

$$\sum_{q \in Q} x_{q,r} = 1 \qquad \forall r \in R \tag{4.1}$$

$$\sum_{r \in R} y_{g,r} \geq 1 \qquad \forall g \in G | l(g) > 0 \tag{4.2}$$

$$\sum_{r \in R} x_{q,r} \leq 1 \qquad \forall q \in Q \tag{4.3}$$

$$y_{g,r} \geq x_{q,r} \quad \forall r \in R, g \in G, q \in Q, \gamma(q, g) = 1 \tag{4.4}$$

$$y_{g,r} \leq \sum_{q \in Q : \gamma(q,g)=1} x_{q,r} \qquad \forall r \in R, g \in G \tag{4.5}$$

$$u(g) = \sum_{r \in R} \sum_{q \in Q : \gamma(q,g)=1} x_{q,r} \frac{l(g)}{h(q)} \tag{4.6}$$

The first constraint provides the restriction that each searcher should only have 1 tree root. This prevents searchers from being assigned to multiple search partitions. The second constraint imposes that every cell with non zero loss should be covered by at least one of the searchers. One could relax this constraint a bit by having the searchers cover all nodes with loss greater than some constant k

rather than 0. The third constraint states that no searchers can share the same location for the root of their PQs. The fourth and fifth constraints bind x and y together. In the fourth constraint, it assigns the y variables to 1 for all the leaves of the quadtree q for robot r. The fifth constraint states that if y is 1 then it must be visible by one of the robot roots. The last constraint is the main function being optimized which tries to maximize the loss covered while keeping the overall area small by reducing the objective value based on altitude of the PQ root location. Example partitions output from this formulation can be seen later in the experiments chapter.

CHAPTER 5

Experiments

5.1 Matlab Testing Framework

All of the experiments below were done using custom code developed in Matlab. The software simulates each time tick of the search missions based on an initial configuration. Both sensor readings and searcher movement are controlled by the Matlab simulation. Within the simulation it is also assumed that the searchers have perfect localization and can fly to their targets in a straight line without hitting obstacles. The software was designed to be flexible and allow for many different mission types and configurations to be tested.

5.2 General Test Settings

For each of the tests below the search trees were allowed to have a maximum depth of 9. This means that the leaves of the fully extended quadtrees corresponded to a uniform grid with 256x256 cells. Note that the root of the quadtree is considered to be depth 1.

5.3 Common Type1 Settings

Each of the Type1 tests used the following configuration settings unless specified otherwise. Leaves in the tree were assumed to be of unit size with side lengths of 1. This means that the overall size of the search region was a square with sides

256 units long. Altitudes in the tree increased by powers of two starting from 2 at the lowest depth. The sensor accuracy was relatively poor and had performance characteristics shown in Figure 5.1. The sensor profile was chosen in a way that the accuracy increased as the depth in the tree increased. The performance was considered to be linear, but other profiles could be used.

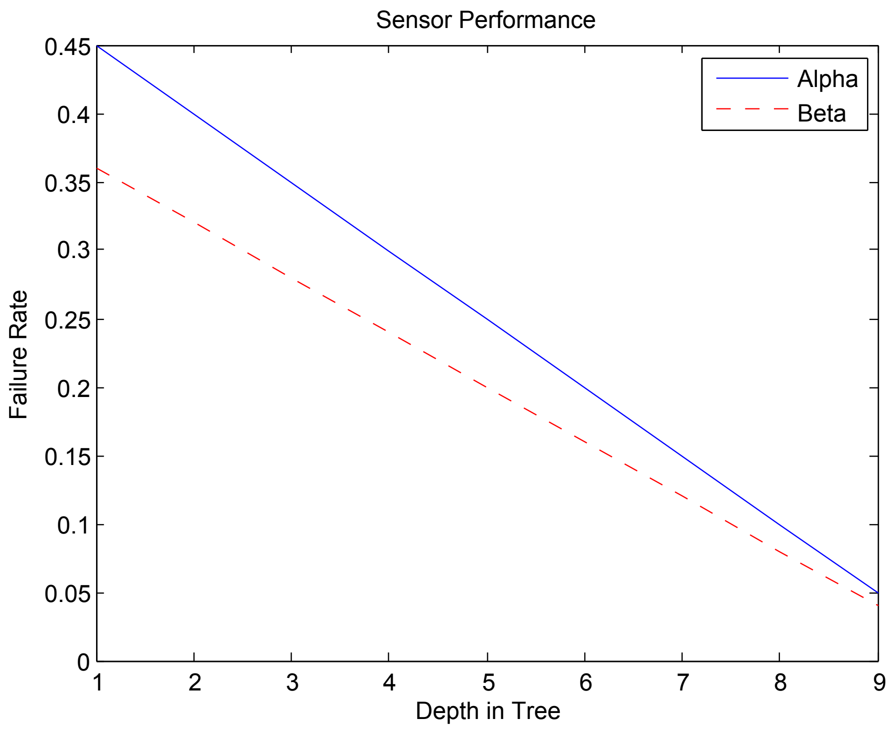

Figure 5.1: Searcher initial belief distribution

As a reminder, α and β are the false positive and missed detection rates of the sensor respectively. When determining when to stop, the searcher had to obtain a probability in a leaf node with at a value of at least 0.95. The length of a search mission was set to 15,000 units of time. In all simulations, the searcher is able to move 1 unit of distance in 1 unit of time. Additionally, sensing requires 1 unit of time.

5.4 Type1: Objective Functions

5.4.1 Experiment Configuration

The goal of this experiment was to test the performance of the various objective functions for the Type1 PQ. Each of the following objective functions were tested against two different intruder distributions: the original high probability based objective function (Eq. (3.6)), information gain (Eq. (3.7)), and information gain combined with distance(Eq. (3.9)). In the weighted case, information gain accounted for 80% of the objective function, while distance accounted for the other 20%. Changing this number may improve the performance, but other values were not explored for this test. For half of the tests, target locations were distributed based on the same probability distribution as the searcher's initial prior and in the other half the targets are placed uniformly in the environment. For both test types, a Gaussian prior centered near the origin was provided to the searcher (see Figure 5.2). Which means that for half the tests the searcher is misinformed.

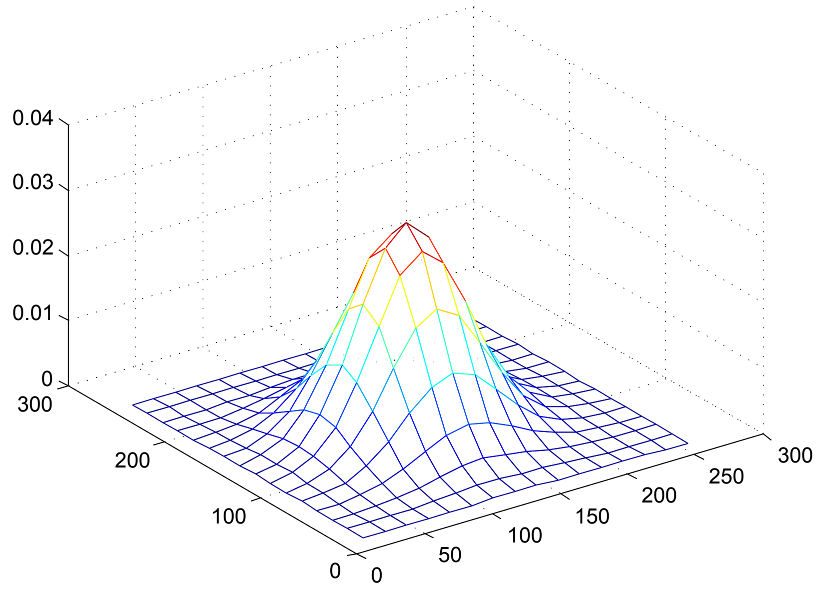

Figure 5.2: Searcher initial belief distribution

For the uniform target placement 256 missions were run, while in the case where targets matched the prior 100 missions were run. In both cases, the results were obtained by taking average of all performed runs.

5.4.2 Results

Type	MTTD	Correct Detections	Out of Time
Probability(3.6)	11504	90	162
IG(3.7)	5288	251	5
IG + Distance(3.9)	4082	254	1

Table 5.1: Results from running 256 missions with a uniform target placement. MTTD (Mean Time To Detection) represents the time till the searcher found the target over the missions. Note that MTTD also includes the mission times where the searcher ran out of time. Correct detections are the number of missions where the searcher identified the target location and out of time is the number of missions where the searcher ran out of time before deciding on a target location.

Type	MTTD	Correct Detections	Out of Time
Probability(3.6)	5537	86	14
IG(3.7)	2295	100	0
IG + Distance(3.9)	1902	100	0

Table 5.2: Results from running 100 missions with a target placement based on prior probability distribution given to the searcher. The other columns have the same meaning as Table 5.1

In both cases (Tables 5.1 and 5.2), information gain performed better than the previous objective function based purely on probability and distance. One should also note that in the case where the prior did not match the placement of targets the original probability based method ran out of time over 50% of the time. Whereas the information gain based methods very infrequently ran out of time.

This seems to indicate that the information gain methods are better at handling scenarios where bad information is given to the searcher. Also, one can see that objective function that weighed distance with information gain performed better than information gain alone. It is probable that by tweaking the weighting value between the two the results could be improved further.

5.5 Type1: Multi-Agent

5.5.1 Experiment Configuration

The first set of multi-agent testing was done using a Gaussian prior centered at the middle of the environment. For these experiments the target placement did not match the prior and were placed uniformly in the environment similar to the previous Type1 Experiment. Agents factored in distance when selecting the next target using a weighting between information gain and distance. Information gain accounted for 80% of the node value, while distance accounted for the other 20%. 256 missions were ran with a single target placed uniformly in the environment. Additionally, tests were run for varying numbers of searchers to measure a method's ability to scale up to larger numbers of agents. This experimental configuration was used on each of the following methods:

1. Fully independent searchers
2. Single Controller with replanning only when all searchers reach their goal
3. Single Controller where individual searchers re plan once they reach their goal
4. Single Controller where all searchers are given a new goal destination when any scan is made.
5. Partitioned Searchers each using their own independent planning.

Note that each of the above methods is discussed in greater detail in the chapter on multi-agent search.

5.5.2 Results

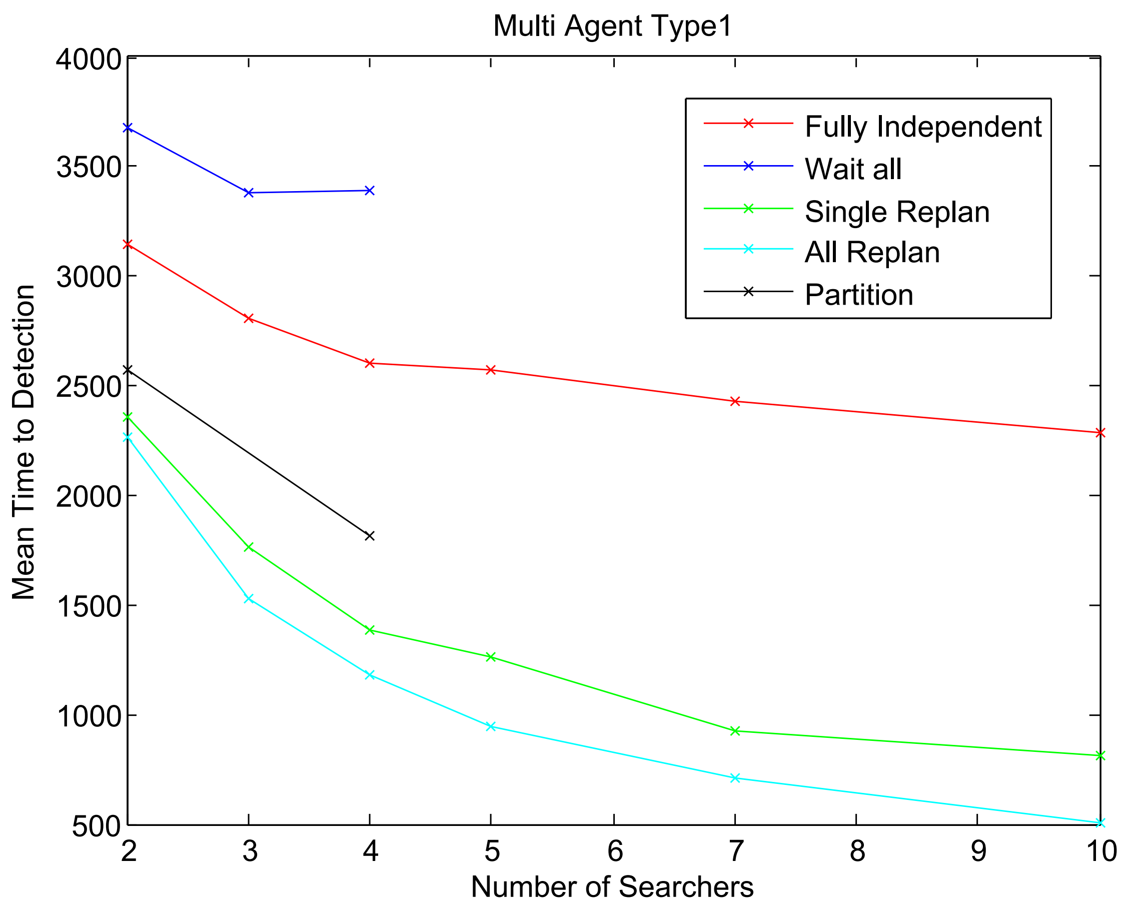

Figure 5.3: MTTD for the various methods over 256 runs per method per searcher amount. The Wait all controller only has 3 samples because its performance was deemed too poor to run for additional searchers. The partitioning planner was tested with 2 and 4 searchers since this number mapped well to partitioning by sub trees.

The differences in the controller performances can be seen in Figure 5.3. The completely independent strategy scales poorly as the number of robots increases where as the single controllers are able to scale and perform better. The single controller that waits for each searcher to finish its assigned task tends to not do as well since it wastes search effort while searchers are waiting on each other to finish. The method where every searcher receives a new goal on a scan works the best, but requires more communication effort especially for larger numbers of robots since whenever one reaches their goal all searchers must get a new goal. While partitioning the searchers did not out perform the better single controllers, it worked better than the independent case without requiring communication between searchers. The naive partitioning case may be dangerous however because if the searcher assigned to the region with the target malfunctions then no searcher will be able to locate the target. In cases where the searchers may have a high rate of failure and communication is not easily available, a more independent based approach may be required.

5.6 Type1: Information Sharing Tests

5.6.1 Experiment Configuration

Additional experiments were run to measure the importance of sharing goal positions versus sharing sensor readings. Using the same configuration as above, we ran one set of missions where the searchers used independent controllers, but shared their goal positions with each other. The controllers would then avoid picking the same goal as another searcher. Overlapping nodes were allowed. Another set of missions were run where the searchers would share their sensor readings with each other, but not their goal locations. The sensor readings were shared by sharing the area sensed rather than the specific node. The searchers could integrate these readings by expanding their trees till their trees contained a node with the same area. Note that for these experiments all the searchers shared the same tree root location.

5.6.2 Results

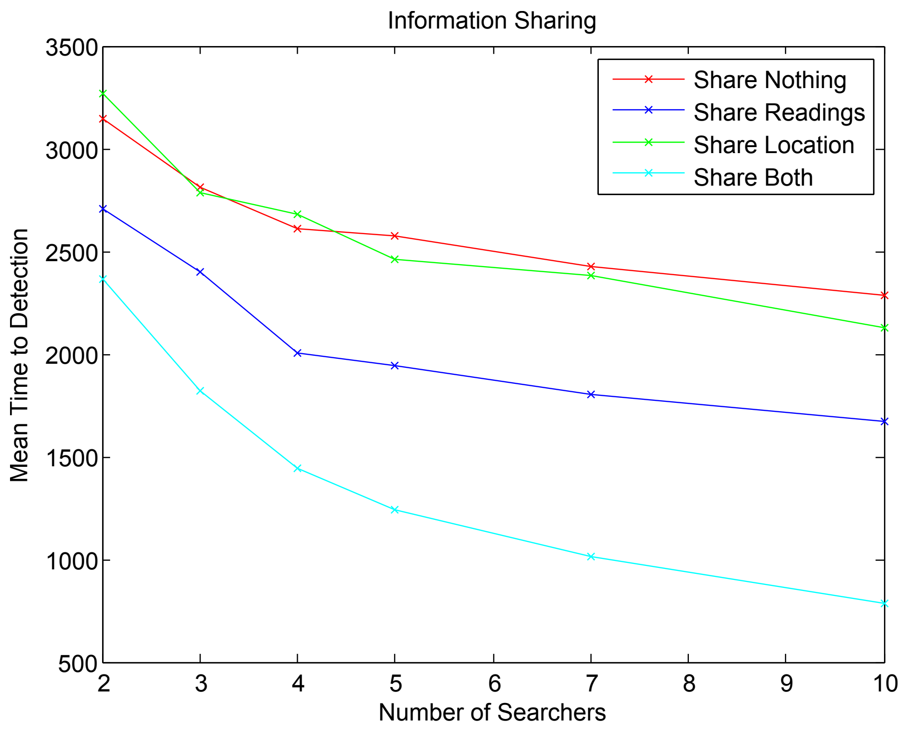

Figure 5.4: Average mean time to detection for the different sharing types over 256 runs per method per searcher amount.

The results in Figure 5.4 show that sharing location alone does not tend to benefit the searchers. This makes some sense as sharing location only prevents searchers from sensing the exact same node in the quadtree. Since each searcher starts in different locations, the chances of them sensing the same node very often are low. Sharing sensor readings helped the searchers find their targets more quickly, but not as fast as when both location and sensor readings were shared. This is possibly because when sensor readings are shared the searchers will have the same probabilities in their PQ. Thus it is more likely that robots will go to the same nodes in the tree as they share the same posterior probability distribution for the target location.

5.7 Type1: Communicating Searchers

5.7.1 Experiment Configuration

Another Type1 test was designed to test the effect of communication range on the performance of the searchers. We ran missions for each of the following communication ranges: 30%, 25%, 20%, 15%, and 10% of the diagonal of the environment. In our simulations, searchers would communicate only once after they had scanned and would broadcast both the scan result and their goal location to all nearby searchers. No relaying of information took place and messages were assumed to be successfully delivered as long as the searchers were within communication range of each other. No modeling of dropped messages or signal strength were done in these tests. Results for these experiments are on the next page.

5.7.2 Results

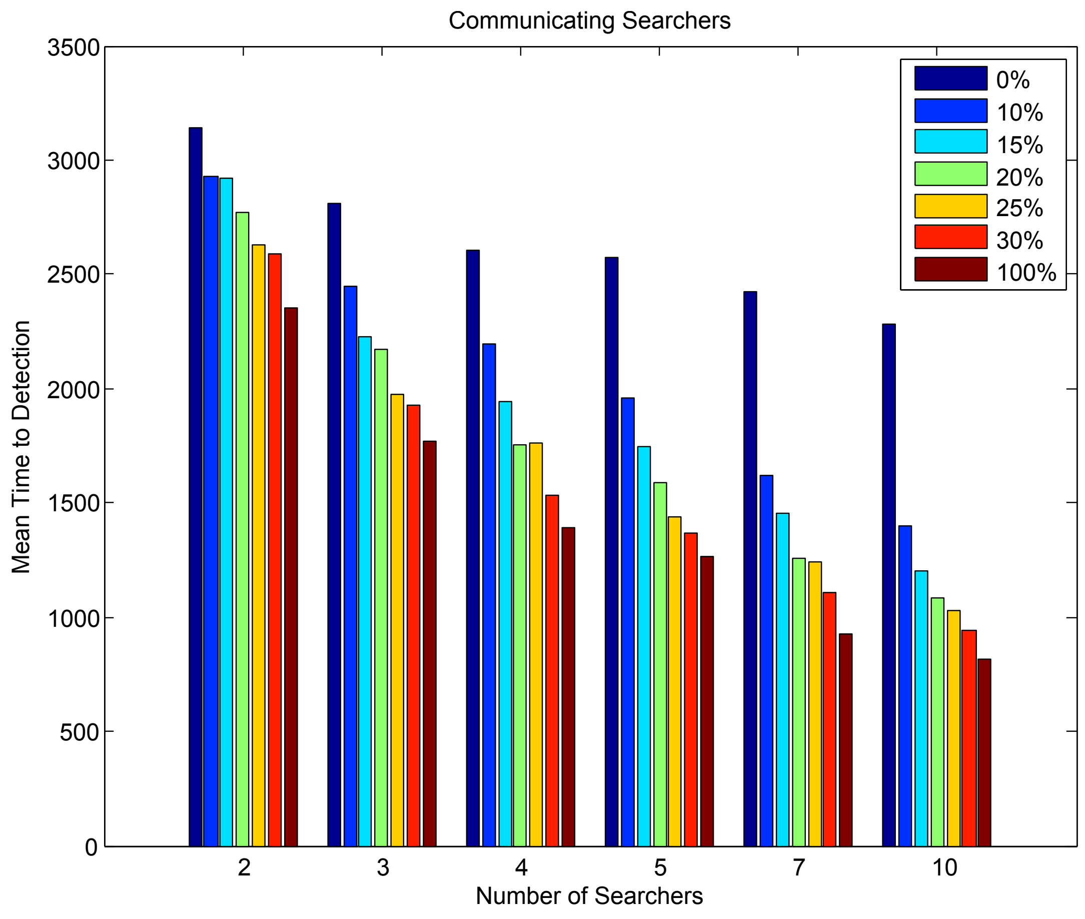

Figure 5.5: Average mean time to detection for various communication ranges over 256 runs per range per searcher

Refering to Figure 5.5, a communication radius of 30% worked almost as well as a full 100% radius. Additionally, as expected performance degraded as the communication radius decreased. Also note that by even enabling a small amount of communication over a short radius (as little as 10%) provides good performance increases over the no communication case. This is especially true as the number of searchers increases.

5.8 Type1: Data Transmission

5.8.1 Experiment Configuration

In order to evaluate the usefulness of a centralized controller, the amount of information passed between the controller and agents was measured. The controller considers sending a goal position to a searcher to be one message and also one message when it receives a sensor reading from a searcher.

5.8.2 Results

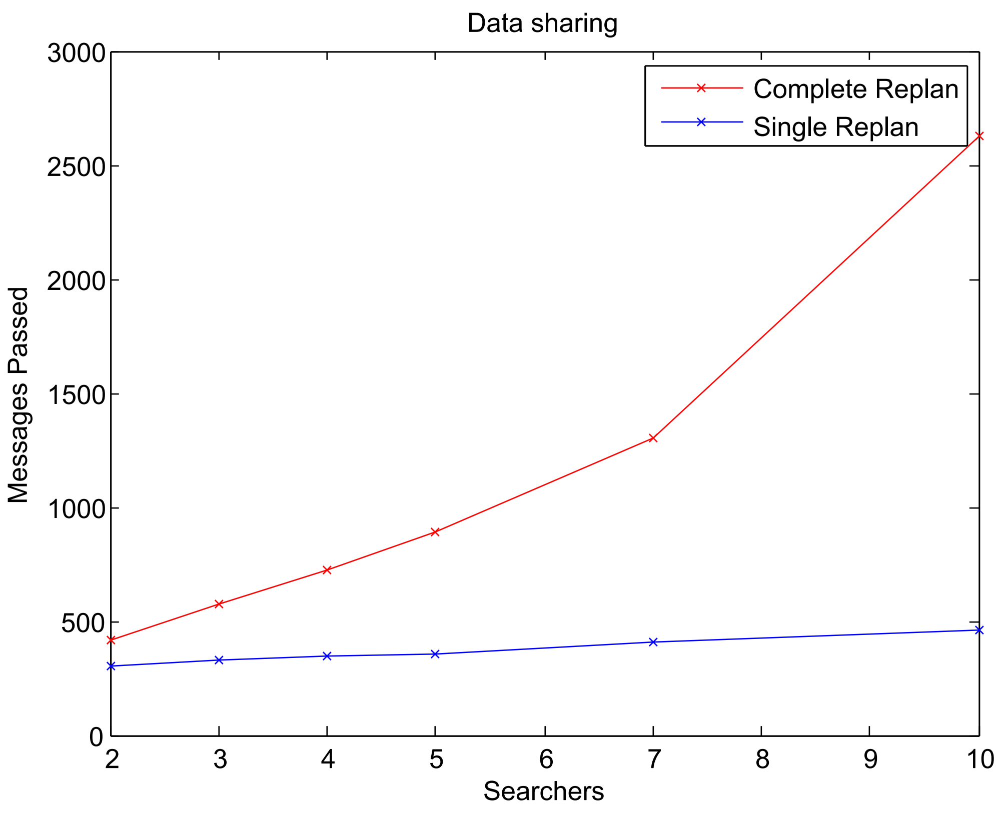

Figure 5.6: Average number of messages passed over 256 runs per controller type per searcher amount.

The interesting result with Figure 5.6 is that the amount of data transmitted for the single replan controller does not increase much as the number of searchers is increased. This is not true for the complete replan controller and the amount of data required increases quickly as more searchers are added. It is possible when implementing a single controller system that these data costs may be prohibitive and the performance improvement of the single replan controller is not justified.

5.9 Common Type2 Settings

For the Type2 experiments, the size of the environment is the same as the Type1 tests above. Additionally the same sensor performance from the Type1 experiments was used. The prior in Figure 5.7 was provided to all Type2 searchers. In these experiments, targets were placed based on the prior distribution. The prior was used to test the searchers ability to cope with targets that may be spread out in separate regions of the environment.

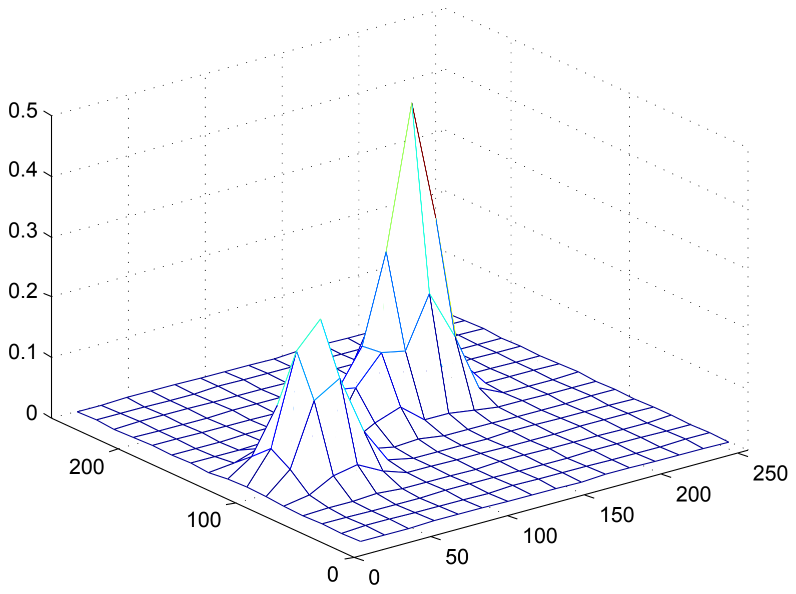

Figure 5.7: Type2 prior target distribution

5.10 Type2: Top Down vs Bottom Up Updates

5.10.1 Experiment Configuration

Using the initial belief prior from above, the performance of the two update methods was compared. 100 experiments were run each with a random number of targets placed within the environment based on the prior distribution. The number of targets ranged uniformly from 2 to 7. Note that an additional set of 100 tests was performed with the sensor accuracy increased by 2x in order to see if the methods differed.

5.10.2 Results

Type	MTTD	Accuracy	Total False Positives
Top Down	6868	0.87	3
Bottom Up	5735	0.81	3
Top Down x2	2743	0.90	25
Bottom Up x2	2636	0.86	19

Table 5.3: Data in this table was collected over 100 runs. MTTD represents the mean time to detection over the 100 runs where accuracy represents the number of targets successfully located divided by the total number of targets over all missions. The x2 signifies that the missions were run with a sensor with alpha and beta reduced by half.

The results in Table 5.3 indicate that methods seemed to give very similar performance. The main difference may be that the bottom up approach had a bit fewer false positives than the alternative top down approach and also a shorter mission time, but with lower accuracy. The accuracy for both update types could likely be improved by modifying the confidence needed to terminate a mission.

5.11 Type2: Effects of Gamma

5.11.1 Experiment Configuration

To test the importance of weighting by distance in the Type2 scenario, several missions were run with varying values of gamma in Eq. (3.9)

5.11.2 Results

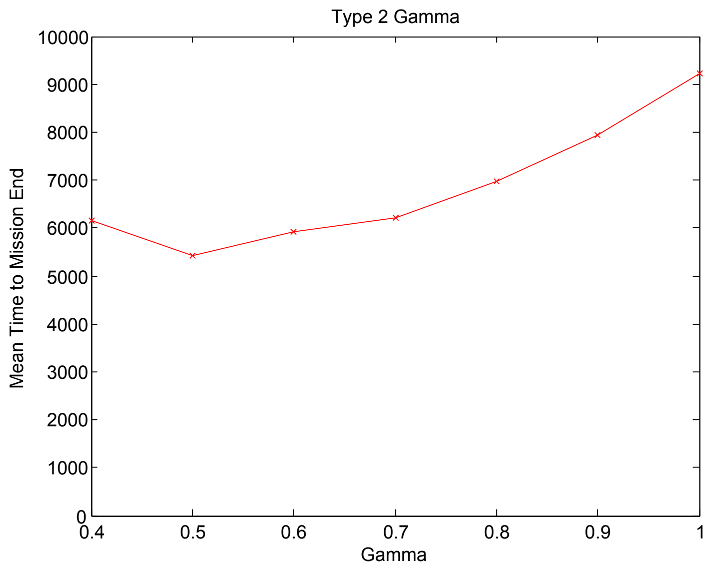

Figure 5.8: Mean time to mission end over 100 runs for varying values of gamma.

One can see in Figure 5.8 that by decreasing gamma a greater emphasis is placed on nearby nodes and the searcher is able to finish its mission quicker as less time is spent traveling. However, if gamma is decreased too low the searcher will put too much emphasis on close by nodes and not enough on those providing information, causing the search mission to take longer.

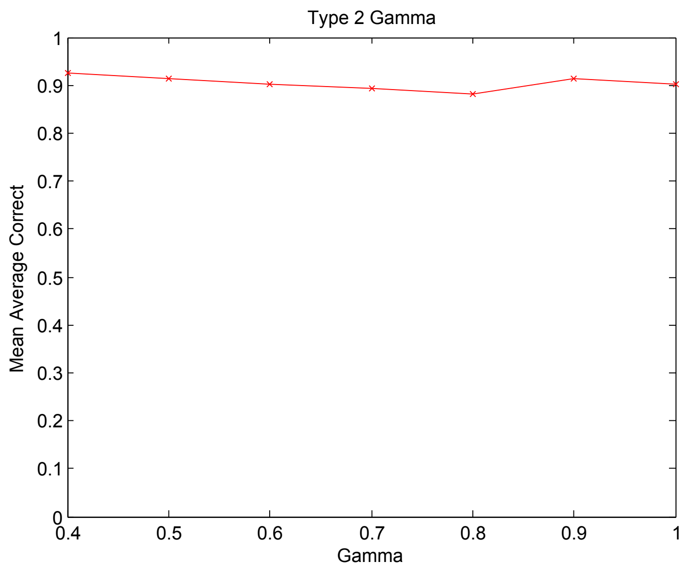

Figure 5.9: Mean average correctly identified targets as a percentage over 100 runs for varying values of gamma.

Note that in Figure 5.9 that the accuracy remains relatively constant for all values of gamma. This idea of balancing gamma is very similar to the idea of exploration versus exploitation in AI. In both cases, a parameter is tuned in order to control how eager the searcher is to take a risk and travel to new locations versus exploiting the more easily available locations.

5.12 Type2: Stop Criterion Tests

5.12.1 Experiment Configuration

In order to test the stop criterion with Type2, several tests were run with varying values for epsilon. Refer to the section of when to stop for Type2, Eq. (3.15), and Algorithm 6

5.12.2 Results

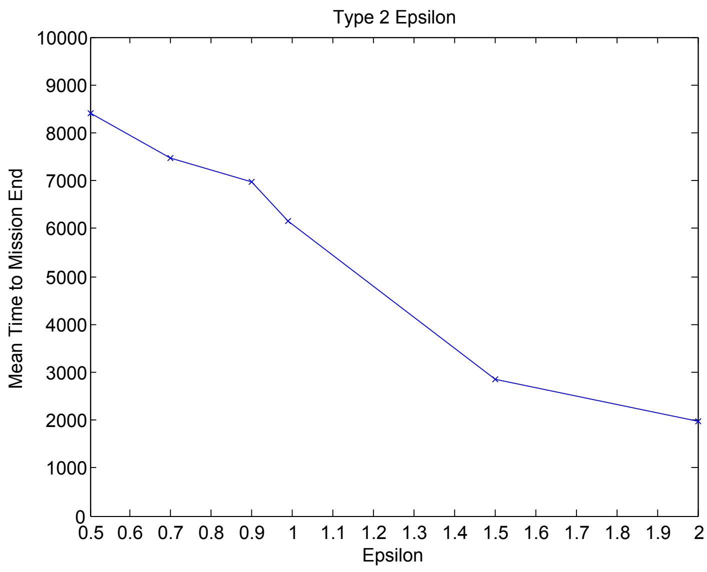

Figure 5.10: Mean time to mission end over 100 runs for varying values of epsilon.

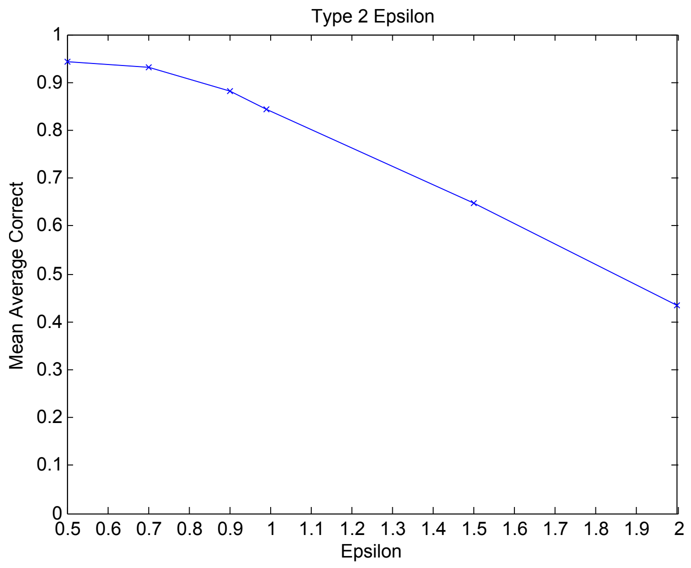

Figure 5.11: Mean average correctly identified targets as a percentage over 100 runs for varying values of epsilon.

It is clear in Figure 5.10 that by increasing epsilon the searcher has to be less confident about the environment and thus can stop its search earlier. This reduced confidence predictably reduces the searchers ability to successfully identify targets (see Figure 5.11). For instances where a high degree of confidence is needed, epsilon values at least under 0.7 should be used.

5.13 Type2: Multi-Agent Search

5.13.1 Experiment Configuration

These experiments compared the performance of completely independent Type2 searching versus partitioning the area. No communication is shared between any of the searchers during the mission. Different from the Type1 independent case the search mission is considered to be over once all of the searchers have terminated their search.

5.13.2 Results

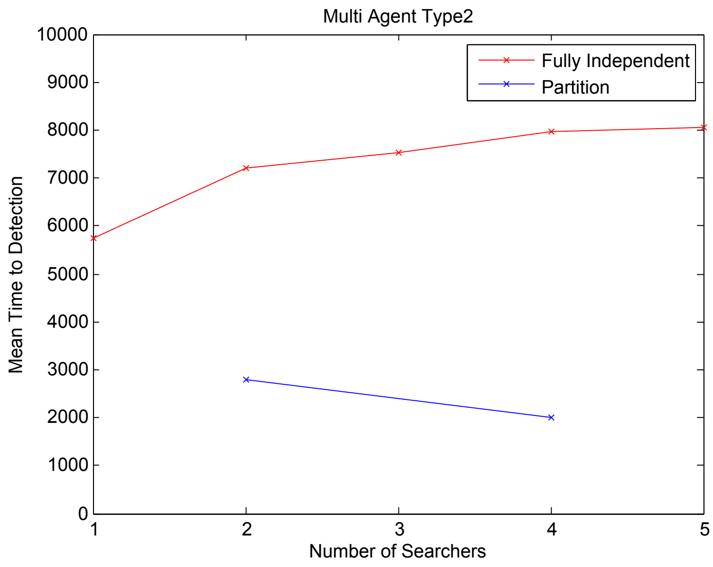

Figure 5.12: Mean time to mission end over 100 runs for varying amounts of searchers. Patrolling was only tested with 2 and 4 searchers.

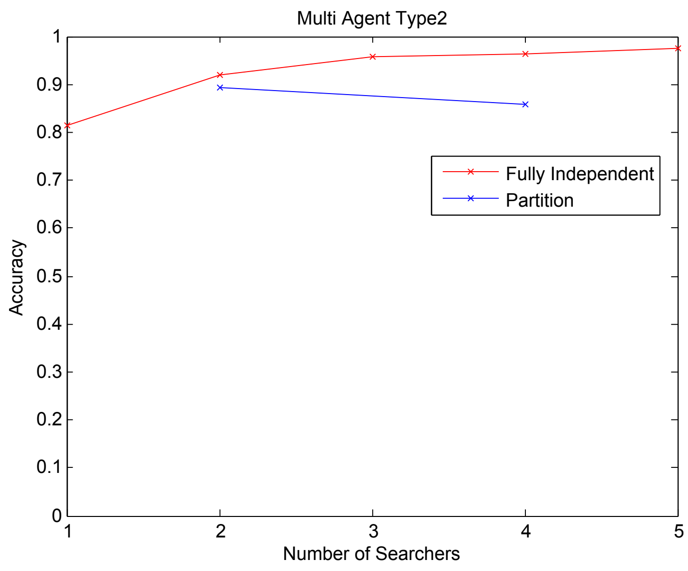

Figure 5.13: Accuracy over the 100 missions run for varying amounts of searchers. Accuracy is the total number of correctly detected targets divided by the total number of targets. Patrolling was only tested with 2 and 4 searchers.

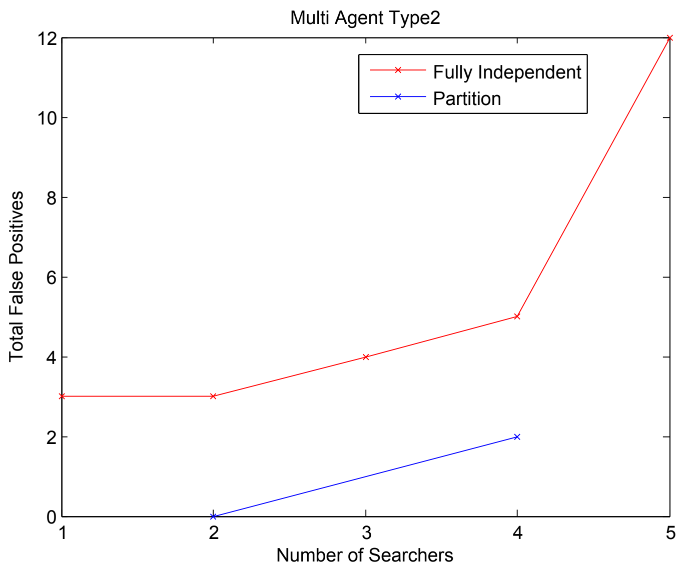

Figure 5.14: Total number of false positives over 100 runs for varying amounts of searchers. Patrolling was only tested with 2 and 4 searchers.

For fully independent searchers when increasing the number of searchers, the search time eventually stops increasing (Figure 5.12). This is because the search ends once all of the searchers are done searching, thus search time is controlled by the slowest searcher. Importantly, the search accuracy can be increased by using more searchers (Figure 5.13). The downside is that the number of false positives tends to rise as the number of searchers is increased (Figure 5.14). Additionally, as mentioned in the multi-agent chapter partitioning tends to work very well for the Type2 scenario.

5.14 Patrolling: Objective Functions

5.14.1 Experiment Configuration

This test was designed to measure performance of the density based patrolling function (Eq. (3.20) versus Entropy/Prob based method (Eq. (3.21) described earlier in the thesis. The search region was the same size as the experiments above. The loss function contained 3 peaks each with varying values of importance and is shown in Figure 5.15. The goal of this loss function was to challenge the searcher with different regions of importance. This requires the searcher to move from region to region when necessary.

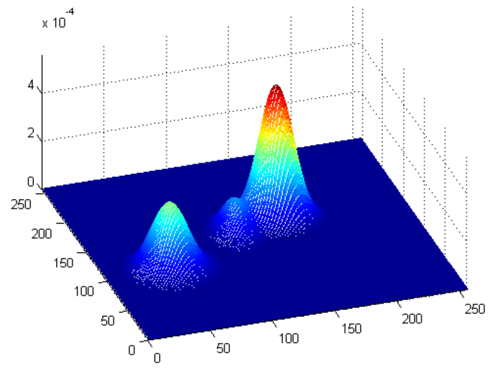

Figure 5.15: This is the loss function provided to patrolling searchers

100 missions were run where in each 100 targets were injected into the environment throughout the course of the mission and the performance of the searchers was measured in the accrued loss from those injected targets. Addtionally, the mission lengths for the patrolling tests were increased to 25,000.

5.14.2 Results

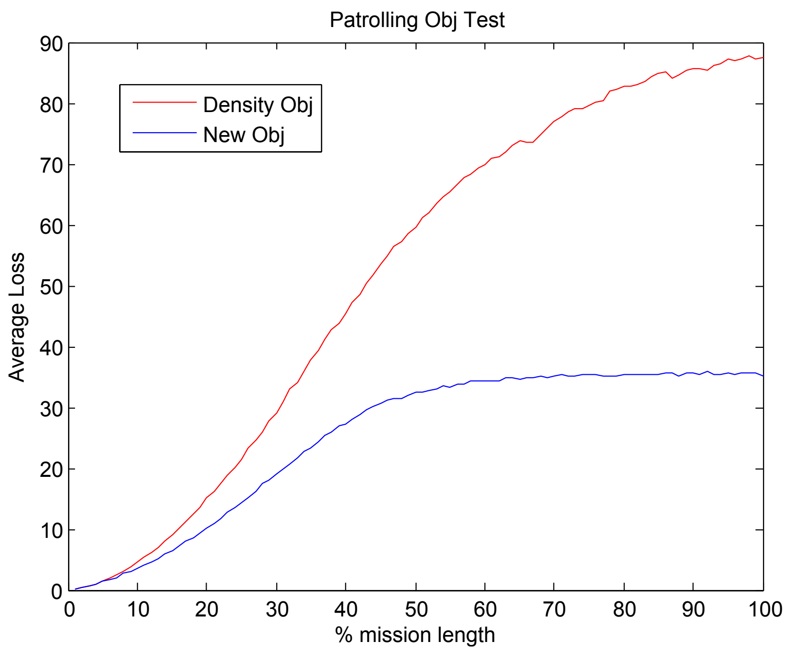

Figure 5.16: The average loss as a function of the percent of mission duration. The results are from an averaging of 100 missions

The results from Figure 5.16 show the the new method works well and is able to keep the loss to much lower values. One explanation for this result is that the searcher takes advantage of its ability to scan at multiple altitudes.

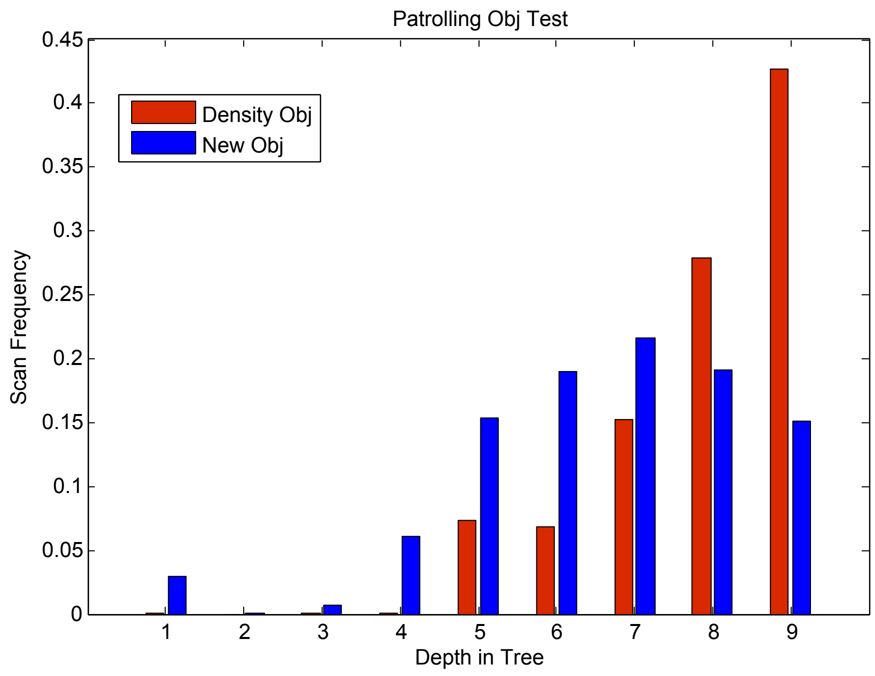

Figure 5.17: The results show the frequency that a particular depth in the tree was scanned over 100 patrolling missions.

When looking at the scan frequencies in Figure 5.17, the new objective function tends to scan more around depth 7 where as the density based obj function spends a large portion of its scans at depth 9.

5.15 Patrolling: Multi-Agent

5.15.1 Experiment Configuration

For a team of searchers, two strategies were used. One where the searchers were completely independent from each other and another where they were partitioned using the optimization method described in the earlier chapter. The computed partitions are shown in the figures on the next page.

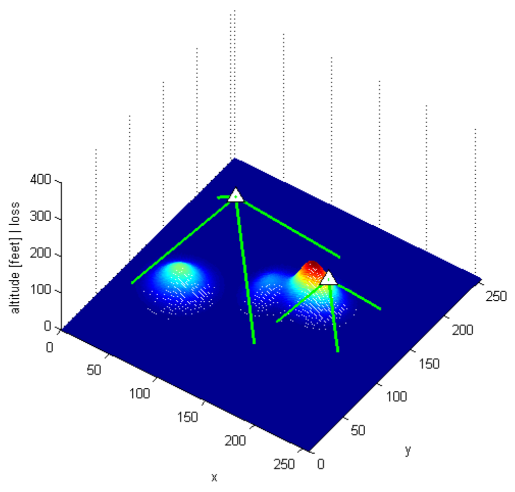

Figure 5.18: 2 Searchers

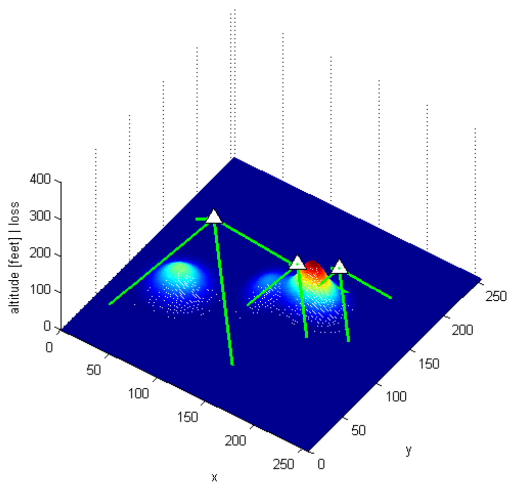

Figure 5.19: 3 Searchers

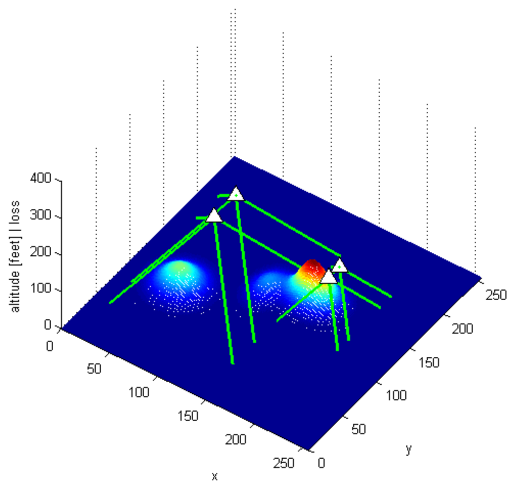

Figure 5.20: 4 Searchers

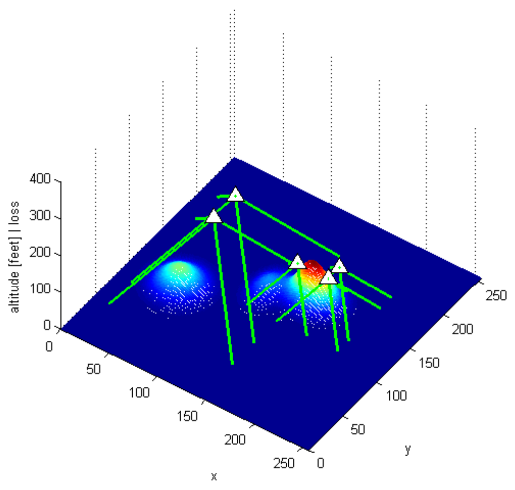

Figure 5.21: 5 Searchers

5.15.2 Results

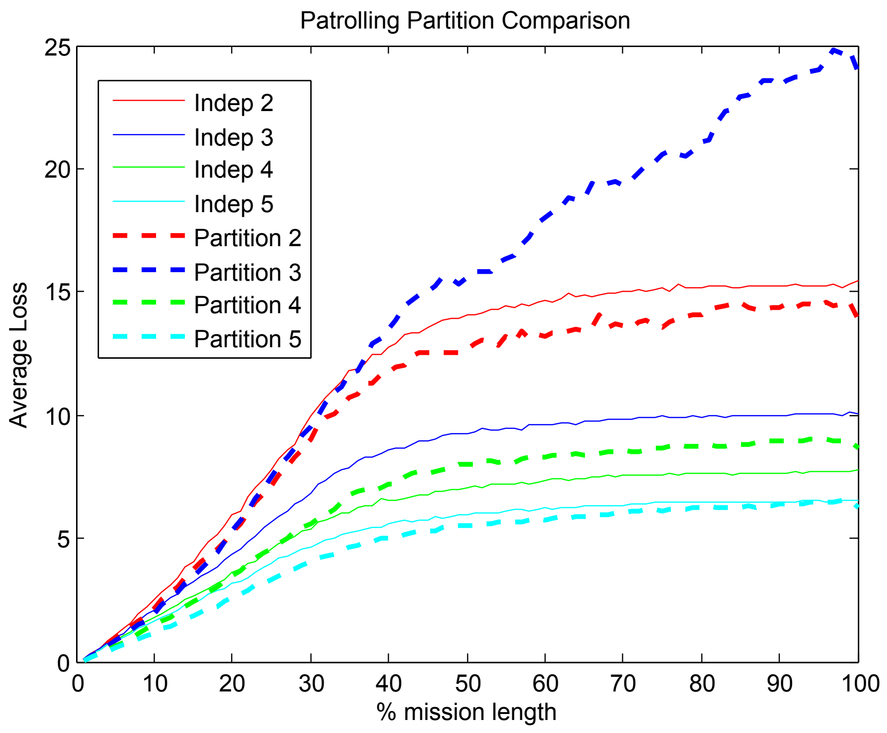

Figure 5.22: Average loss for the multi-agent missions as a function of the percentage of mission completion. Dashed lines represent patrolling missions with partitioned searchers and the solid lines represent fully independent searchers

The results in Figure 5.22 seem to indicate that partitioning the environment does not yield very strong performance benefits. In the case of 3 searchers, the performance was actually much worse which seems to indicate that either the partition for 3 searchers is not very good or the partition negatively interacts with the objective function somehow. For the remaining number of searchers, the performance was mostly the same. This hints that there is potentially some sort of implicit partitioning of the work done by the independent searchers. One hypothesis is that the independent searchers "communicate" with each other by removing targets from the environment. Once a target is removed, other searchers will not waste effort by sensing in the same area since they will not detect the removed intruder. This idea is explored in the experiment in the next section.

5.16 Patrolling: Environmental Influence

5.16.1 Experiment Configuration

One possible explanation for the good performance of independent searchers in the patrolling scenario is that there is some form of implicit communication through the environment. This occurs because once a searcher identifies a target and removes it from the area the other searchers will not detect this target when sensing the same location. To get around this, the above experiment is changed such that targets removed by one searcher can still be seen by other searchers. However, once a target has been "removed" by at least one searcher its presence is no longer counted towards the total penalty of the search team. This allows the results to be compared with those from the previous experiment.

5.16.2 Results

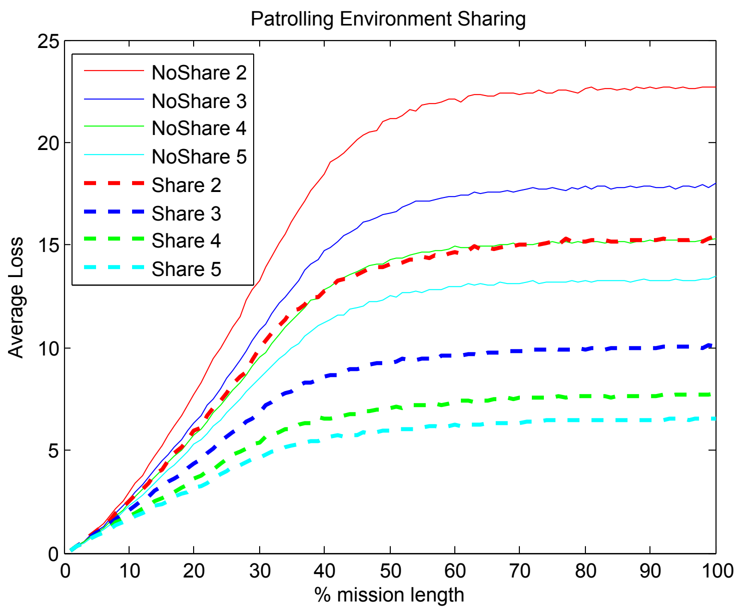

Figure 5.23: Average loss as a function of the percentage of mission completion for both environmental sharing and no sharing for varying number of independent searchers.

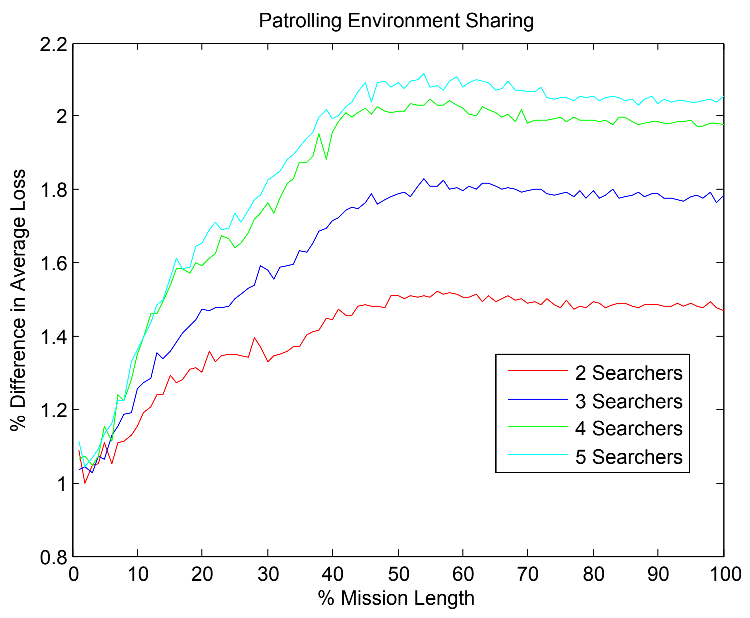

Figure 5.24: Difference in average loss between sharing and no sharing as a function of the percentage of mission completion for varying number of independent searchers. A value of 2 means that no sharing has a loss of 2x sharing.

As expected, the performance degrades with these changes and the amount at which this occurs increases as the number of searchers is increased (Figure 5.23). This is because more effort is wasted searching the same locations over and over again. The plots in Figure 5.24 indicate how many times worse the methods performed with these changes.

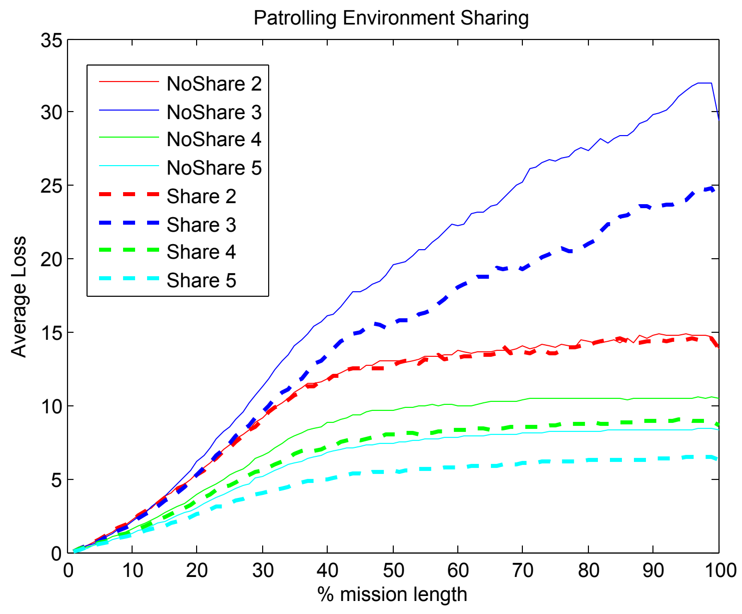

Figure 5.25: Average loss as a function of the percentage of mission completion for both environmental sharing and no sharing for varying number of partitioned searchers.

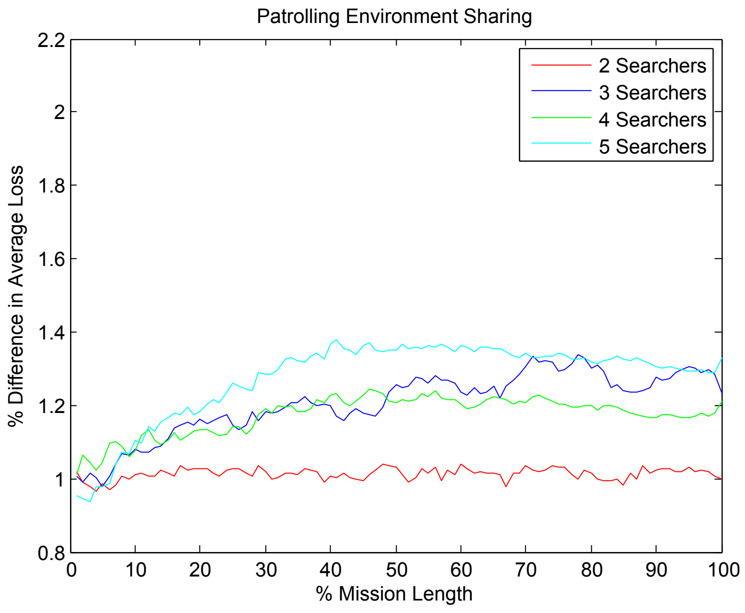

Figure 5.26: Difference in average loss between sharing and no sharing as a function of the percentage of mission completion for varying number of partitioned searchers. A value of 2 means that no sharing has a loss of 2x sharing.

When applying the environment changes the partitioning method suffers less than the independent searchers (Figure 5.25 and Figure 5.26). This is likely because there is less overlap between their search regions. These results show that the independent searchers benefit more from this environment sharing and this potentially allows them to perform well without direct communication.

5.17 Experimental Discussion

Each of the above experiments is included within this work to show that the probabilistic quadtree is a powerful framework for hierarchical search when using sensors with uncertainty. When considering the performance of the algorithms above it helps to compare them to the case where the searcher is forced to search on a uniform grid. In the above experiments, the trees used had a maximum depth of 9 which means their leaves correspond to a uniform grid of size 256 by 256 cells. With this environment the uniform grid searcher has to look at up to 65,536 possible locations to locate its target. The amount of time it would take to scan each of these locations once is far greater than the total time available to the above searchers. In the above tests, the searchers were given 15,000 time steps in the Type1 and Type2 tests, and 25,000 time steps in the patrolling tests. The speed at which the hierarchical methods are able to locate targets with an inaccurate sensor demonstrates their power. Another benefit of the quadtree representations is that the amount of space required to represent the searchers belief is much fewer than that of the uniform grid. The majority of the experiments above started with trees using around 100 nodes and ended with trees with around 150 nodes in the Type1 case and 350 in the Type2 case. These numbers are significantly lower than the 65,536 possible cells at the lowest depth.

Additionally, the results shed some light on the performance of multi-agent systems using quadtree representations. The results seem to show that while using fully independent searchers can be a viable strategy, the usefulness of such a method diminishes as more searchers are added. When the number of searchers gets larger it becomes more important for coordination to take place. One alternative to coordination is partitioning of the environment. This strategy was shown to work well in some cases, but did not seem to work extremely well in the patrolling case. Static partitioning has the advantage of using no communication between searchers, but appears to need good partition selections to function well.

CHAPTER 6

Conclusions

In this work, we have improved and evaluated the effectiveness of a hierarchical probabilistic search structure. The probabilistic quadtree is a structure that has been shown to work in several cases including: search for a single target, search for multiple targets, and patrolling for intruders. Additionally, it was shown that the method can be extended to work in the case of multiple search agents.

The major contributions of this work come from both theoretical findings and experimental results. In terms of theory, this thesis provided a method to reduce the complexity of the Type1 objective function from $\mathcal{O}(|\mathcal{N}(\mathcal{T})|^2)$ to $\mathcal{O}(|\mathcal{N}(\mathcal{T})|)$. Additionally, it provided a new way to perform the Bayesian updates to the PQ structure after sensor readings. This new bottom up approach allows the Type2 updates to be done precisely rather than using the approximate top down technique. Also the work included a new objective function for the patrolling scenario which was shown to provide much better performance over the original density based formulation. Lastly, the bulk of the experimental work dealt with teams of agents working on PQ's. The experimental results indicate that in general without partitioning or communication between searchers performance increases diminish as more searchers are added.

In terms of future work, there are several directions one could take. This thesis only touched on the possibilities for multi-agent cooperation using probabilistic quadtrees. One could investigate more complicated and complete single controllers or even look into creating sophisticated distributed controllers. The single searcher scenario also has many possible areas for improvement. Each of the proposed methods in this paper largely used myopic or greedy strategies for selecting which areas to search. One could spend time looking into ways to account for future sensor scans. Additionally, one could possibly find different objective functions that improve further those presented in this work. The sensor model could also be modified to allow sensors with varying accuracy based on the number of targets present. A challenging direction is to consider the case where targets within the environment are assumed to be moving based on a known motion model. With a moving target, the tree would need modifications to allow it to track targets effectively. Finally, one could also attempt to extend the work to different space partitionings such as triangulations and even partitions with overlapping regions.

Based on the experimental results Probabilistic Quadtree's are an effective search structure with plenty of research potential.

References

[1] T. Arai, E. Pagello, and L.E. Parker. Guest Editorial Advances in Multirobot Systems. *Robotics and Automation, IEEE Transactions on*, 18(5):655–661, October 2002.

[2] N. Basilico and S. Carpin. Online Patrolling Using Hierarchical Spatial Representations. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2163 –2169, may 2012.

[3] S.J. Benkoski, M.G. Monticino, and J.R. Weisinger. A Survey of the Search Theory Literature. *Naval Research Logistics (NRL)*, 38(4):469–494, 1991.

[4] L.F. Bertuccelli and J.P. How. Robust UAV Search for Environments with Imprecise Probability Maps. In *IEEE Conference on Decision and Control (CDC)*, pages 5680–5685, Seville, Spain, 12-15 December 2005.

[5] F. Bourgault, T. Furukawa, and H. Durrant-Whyte. Optimal Search for a Lost Target in a Bayesian World. In Shinichi Yuta, Hajima Asama, Erwin Prassler, Takashi Tsubouchi, and Sebastian Thrun, editors, *Field and Service Robotics*, volume 24 of *Springer Tracts in Advanced Robotics*, pages 209–222. Springer Berlin / Heidelberg, 2006.

[6] F. Bourgault, T. Furukawa, and H.F. Durrant-Whyte. Coordinated Decentralized Search for a Lost Target in a Bayesian World. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 48 – 53 vol.1, oct. 2003.

[7] S. Carpin, D. Burch, and T.H. Chung. Searching for Multiple Targets Using Probabilistic Quadtrees. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4536 –4543, sept. 2011.

[8] T.H. Chung. On Probabilistic Search Decisions under Searcher Motion Constraints. In GregoryS. Chirikjian, Howie Choset, Marco Morales, and Todd Murphey, editors, *Algorithmic Foundation of Robotics VIII*, volume 57 of *Springer Tracts in Advanced Robotics*, pages 501–516. Springer Berlin Heidelberg, 2009.

[9] T.H. Chung and S. Carpin. Multiscale Search Using Probabilistic Quadtrees. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2546 –2553, may 2011.

[10] T.H. Chung, G.A. Hollinger, and V. Isler. Search and Pursuit-Evasion in Mobile Robotics - A Survey. *Auton. Robots*, 31(4):299–316, 2011.

[11] J.N. Eagle. The Optimal Search for a Moving Target When the Search Path is Constrained. *Operations Research*, 32(5):1107–1115, 1984.

[12] Nuclear Power After Fukushima [Special Report]. *Spectrum, IEEE*, 48(11):30 –33, november 2011.

[13] M. Flint, E. Fernandez, and M. Polycarpou. Stochastic Models of a Cooperative Autonomous UAV Search Problem. *Military Operations Research*, 8(4):13–32, 2003.

[14] M. Flint, E. Fernandez-Gaucherand, and M. Polycarpou. Cooperative Control for UAV's Searching Risky Environments for Targets, 2003.

[15] M. Flint, M. Polycarpou, and E. Fernandez-Gaucherand. Cooperative Control for Multiple Autonomous UAV's Searching for Targets. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 3, pages 2823 – 2828 vol.3, dec. 2002.

[16] B.O. Koopman. Search and Its Optimization. *The American Mathematical Monthly*, 86(7):pp. 527–540, 1979.

[17] G.K Kraetzschmar, G.P. Gassull, and K. Uhl. Probabilistic Quadtrees for Variable-Resolution Mapping of Large Environments. In M. I. Ribeiro and Santos J. Victor, editors, *Proceedings of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, Lisbon, Portugal, July 2004. Elsevier Science.

[18] B. Lavis, T. Furukawa, and H.F. Durrant-Whyte. Dynamic Space Reconfiguration for Bayesian Search and Tracking with Moving Targets. *Auton. Robots*, 24(4):387–399, 2008.

[19] L. Lin and M. Goodrich. A Bayesian Approach to Modeling Lost Person Behaviors Based on Terrain Features in Wilderness Search andRescue. *Computational & Mathematical Organization Theory*, pages 1–24–24, July 2010.

[20] L.C. Ludeman. *Random Processes: Filtering, Estimation, and Detection.* Wiley, 2003.

[21] M.M. Polycarpou, Y. Yang, and K.M. Passino. A Cooperative Search Framework for Distributed Agents. In *Intelligent Control, 2001. (ISIC '01). Proceedings of the 2001 IEEE International Symposium on*, pages 1 –6, 2001.

[22] I. Sisso, T. Shima, and Y. Ben-Haim. Info-Gap Approach to Multiagent Search Under Severe Uncertainty. *IEEE Transactions on Robotics*, 26(6):1032–1041, 2010.

[23] L.D. Stone. *Theory of Optimal Search.* MAS of INFORMS, 2nd edition, 2007.

[24] P.B. Sujit and D. Ghose. Multiple Agent Search of an Unknown Environment Using Game Theoretical Models. *Proc. Amer. Control Conf.*, 6:5564–5569, 2004.

[25] S. Tadokoro. *Rescue Robotics.* Springer, 2010.

[26] S. Waharte, A. Symington, and N. Trigoni. Probabilistic Search with Agile UAVs. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2840–2845, May 2010.

[27] Y. Wang, I.I. Hussein, D.R. Brown, and R.S. Erwin. Cost-Aware Bayesian Sequential Decision-Making for Search and Classification. *Aerospace and Electronic Systems, IEEE Transactions on*, 48(3):2566 –2581, july 2012.

[28] E. Wong, F. Bourgault, and T. Furukawa. Multi-Vehicle Bayesian Search for Multiple Lost Targets. *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3169–3174, 2005.

[29] Y. Yang, A.A. Minai, and M.M. Polycarpou. Decentralized Cooperative Search by Networked UAVs in an Uncertain Environment. *American Control Conference, 2004. Proceedings of the 2004*, 6:5558–5563 vol.6, 2004.