2017-05-01

Robust Object Tracking: A Path-Planning Approach

Bryant Eldon Chandler
Brigham Young University

Robust Object Tracking: A Path-Planning Approach

Bryant Eldon Chandler

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Michael A. Goodrich, Chair
Jacob W. Crandall
Seth R. Holladay

Department of Computer Science

Brigham Young University

ABSTRACT

Robust Object Tracking: A Path-Planning Approach

Bryant Eldon Chandler
Department of Computer Science, BYU
Master of Science

When attempting to follow ground-based moving objects (hereafter referred to as "waldos") using an unmanned air vehicle (UAV), occlusion can become a significant problem for computer vision algorithms designed to track the object. When a waldo is occluded, the computer vision algorithm loses the track and the UAV's ability to predict movement degrades. We propose a path-planning and replanning method that moves a UAV to a location that maximizes the important waldos that can be seen while accounting for occlusion, and attempts to maximize the area it can see during travel. The proposed work moves beyond state-of-the-art algorithms designed to follow a single waldo while accounting for occlusion to enable tracking multiple prioritized waldos.

Keywords: path-planning, rapid replanning, RRT*, FMT*, ORRT*, OFMT*, target tracking

ACKNOWLEDGMENTS

This work was funded by the Center for Unmanned Aircraft Systems (C-UAS), a National Science Foundation-sponsored industry/university cooperative research center (I/UCRC) under NSF Award No. IIP-1161036 along with significant contributions from C-UAS industry members.

Table of Contents

List of Figures

Chapter 1

Introduction

Tracking moving objects ("waldos") on the ground from a UAV is useful for a number of applications. The obvious application is for aerial reconnaissance in military and police operations. In those situations it might be used to monitor terrorist or gang activity. Another application is in search and rescue. UAVs can be used to search for a missing person, complementing search from people on foot or in a manned aircraft because UAVs are able to move much lower than manned aircraft and more quickly than searchers on foot. There are less obvious applications including filming footage for TV or cinema, herding animals, or crowd monitoring. These less obvious applications benefit from the same advantages gained in military, police, and search and rescue operations. All applications would benefit from having path-planning and replanning that helps a human expert focus on performing a mission rather than controlling a UAV.

Tracking waldos with a UAV requires significant time, attention, cognitive workload, and expertise from a human (see [5] for example). Because of these manpower demands, UAVs can be difficult to deploy and use in many practical applications. Automating some of the tasks can reduce human workload and result in a more functional system. To potentially reduce the cognitive workload on humans, we chose to look at automating waldo following. Algorithms exist for making a UAV autonomously follow a waldo (or even multiple waldos) [24, 34], but there are still many holes to be filled before it can be done robustly.

This thesis presents an algorithm that plans and replans a path for a UAV such that it maximizes the ability to persistently track high importance waldos while accounting for

occlusion. The problem is solved in several parts: (a) path end-points are chosen based on current knowledge about the waldos and predictions about their future locations, (b) paths are planned using a cost function that penalizes for putting the UAV in locations that don't provide good visibility, and (c) path-planning algorithms are presented which afford rapidly replanning to new end-points while attempting to match the aforementioned cost function. The usefulness of the algorithms presented is validated in the Chapters that follow. The end-point selection methods and cost function are evaluated using a score derived from the amount of time waldos were inside the view radius scaled by their importance. path-planning algorithms are validated for computational efficiency, and memory efficiency, as well as optimality with respect to this cost function.

The path-planning algorithms presented have applications beyond the problem that this thesis aims to solve. They provide rapid replanning for new path end-points and adaptation to time-varying cost with relatively little overhead. These features apply to any application where either a robot needs to change its end-point frequently or travel costs are dynamic rather than static.

This effort is part of a more complete tracking project that aims to perform robust multi-object tracking from multiple UAVs. The project uses Recursive-RANSAC (R-RANSAC) to visually track waldos and maintain track persistence [26]. The role of path-planning is to place the UAV such that the R-RANSAC algorithm has the best chance of success in tracking waldos. Future work should extend the presented work to enable multiple UAVs to prioritize and position themselves cooperatively.

This thesis is organized in chapters for each portion of the solution. Chapter 2 discusses related work. Chapter 3 is a conference paper currently under review for IROS which presents the algorithms for rapid replanning with dynamic cost. Chapter 4 discusses the approach for selecting path-planning end-points at each replanning time step. Chapter 5 presents an attempted approach for avoiding tracking-loss due to occlusion. Finally, Chapter 6 presents

conclusions and possible avenues for future work. Validations for the various components of our algorithm are presented and discussed in their respective chapters.

Chapter 2

Related Work

This chapter explores four areas of related research: discrete-based planning, sampling-based planning, visual multi-target tracking, and waldo-following.

2.1 Discrete Planning

There are many algorithms in the literature for discrete multi-objective path-planning, one of the most notable being A* [33], which is an alternative to Dijkstra's algorithm [4] when the goal is known. These algorithms build a graph in a configuration space that has been discretized into a grid with known available transitions between grid cells. They then perform search on those graphs. This allows the algorithms to operate quickly, but discretization reduces the likelihood of finding a truly optimal solution; the algorithms might not find paths through narrow passages and the location of grid cells might keep it from finding an optimal path. D* [30], AD* [25], and D* Lite [19] expand on Dijkstra and A* to get dynamic graphs that can handle unexpected obstacles, but they suffer from the same discretization issues.

Moving Target Search (MTS) [11] is a discrete path-planning algorithm that is geared toward the application of reaching a moving target. Its main claim is that the searcher will reach the target eventually, as long as the searcher moves faster than the target. Its underlying planning algorithm is LRTA* [20], which is a discrete planning approach that learns a heuristic as it travels. We choose not to use MTS for our application because of discretization issues, and the fact that it is designed for finding a single waldo.

2.2 UAV Motion Planning

Another element of planning for UAVs is motion planning, which is planning that accounts for the dynamics of the vehicle. There are several algorithms that plan paths to search for and follow targets with a focus on vehicle dynamics; we list only a few representative algorithms. The first motion-planning algorithm [9] supplements particle filtering with search trees to plan motion. The algorithm's combination of filtering and discrete search is innovative, but it focuses on searching and doesn't anticipate target movement and planning paths to track a target once it has been found. Rather, the algorithm implicitly follows a target because once the target has been seen, the search probability for its actual location will be high, causing the discrete search to plan paths to reach the vehicle.

The second motion-planning algorithm tracks a single target with multiple UAVs [18]. It attempts to maximize the visibility of a target while a tracking vehicle circles the target by picking the center of a minimum turning radius circle. This motion-planning algorithm can include multiple UAVs as follows: if more than one UAV is available, the algorithm picks an optimal angular spacing of UAVs all moving on the same circle. This motion-planning algorithm was expanded in [17] by an algorithm that clusters many targets into large groups to be tracked by multiple UAVs. Once the groups have been formed, the revised algorithm uses motion planning from [18] to fly the UAVs in a minimum circular path.

The third motion-planning algorithm [10] combines discrete search and motion-planning by using LRTA* to visit targets optimally. The technique plans a path that minimizes the path length to see a set of targets using LRTA* and Dubins curves. The approach is interesting because it leverages more conventional path-planning, while still catering to fixed wing dynamics. We draw inspiration from this approach, but we cannot naively adopt the approach because we are interested in following targets rather than seeing them a single time.

2.3 Sampling-Based Planning

Sampling-based path-planning helps solve some of the problems created by discretizing the configuration space. Sampling-based path-planning has been justified with several arguments: it is effective for path planning in configuration spaces with complex obstacles [22], it can quickly find sub-optimal solutions to complex problems [6], the algorithms are relatively easy to implement [31], etc. There is room for discussion in support of or in opposition to these assertions, but even a cursory review of the literature reveals that sampling-based algorithms are being successfully used in many areas. RRT [21] and PRM [16] are classic algorithms in the area, from which many sampling-based algorithms are derived. RRT builds a rapidly exploring tree from random sampling and a cost function that can combine multiple objectives. Once the tree is built, one simply needs to select an end-point and trace back up the tree to the root, which is located at the start point. PRM builds a graph from randomly sampled points, and then paths are queried using another algorithm such as A*. The time complexity for constructing search trees is the same for both RRT and PRM, but RRT is $O(n)$ in query and space complexity, whereas PRM is $O(nlog(n))$ in both query and space complexity [14]; the reduced query time and memory requirements generally make RRT the favorite of the two. Additionally, RRT includes the ability to plan kinodynamically (accounting for vehicle dynamics and velocity) [21], which is a great benefit to any real application.

RRT has been expanded to RRT* [14], which guarantees asymptotic optimality as more points are sampled by rewiring the existing tree to get path cost reductions. An advantage of this approach is that a sub-optimal path can be found quickly with a sparse tree. This thesis assumes that the algorithm will run for a long time, so it is more important to generate a dense (and therefore closer to optimal) tree, than to find a single path quickly. FMT* [13] constructs a tree with the same structure and optimality guarantees as RRT*, but it builds out from the start point densely. It is able to complete a dense and optimal tree more quickly than RRT because it doesn't use rewiring. Additionally, the final node

distribution has been subjectively observed to be more uniform for the same number of nodes, which leads to a tree that fills the configuration space more fully than RRT*.

There have been many improvements proposed to RRT*, but a particularly interesting one is Informed RRT* [8]. This algorithm performs standard RRT* until a path is found, then it fits an ellipse to the found path and only samples within that ellipse. This is possible because any path that beats the current path for cost must fall within that ellipse. Informed RRT* is very effective at reducing the number of iterations required to find the asymptotically optimal path, but it only works when searching for a shortest path. Furthermore, the tree created by Informed RRT* is only dense near the optimal path, which makes it less useful for tree reuse.

Several solutions have been developed to allow RRT* to function in an online/anytime fashion. Anytime RRT* [15] takes advantage of the time it takes for a robot to travel a path by improving the path during travel. It does this by pruning the part of the tree that has already been traversed and continuing to sample. The Anytime RRT* algorithm doesn't meet the requirements of this thesis because it destroys part of the tree as the robot moves; the application in this paper requires replanning, and replanning benefits from reusing information that could be destroyed by the Anytime RRT* algorithm. Another notable online solution is RRT^X [1], which switches the start and end-points, so that the end-point is now fixed and enables the algorithm to find optimal paths to all other points in the space. Thus, when the start point moves, it can simply use one of the other paths. When obstacle conditions change, a cascading rewire is performed to update the tree. This is a good approach to being online, but it still requires one of the points to be fixed, which isn't the case for the application in this thesis. This thesis requires an algorithm that can adapt when both the start-point and end-point move as the UAV and waldos both move; it is less concerned about unexpected obstacles.

The sampling-based planning approaches depend on uniform random sampling, but random sampling can be slow to converge because it is prone to clumping and open areas.

Uniformity isn't reached until a large number of points have been sampled. Because of these weaknesses, deterministic sampling methods, such as lattices and Halton sequences have been explored [2][12] and found to make promising improvements to convergence rates for PRM. This improvement might be especially beneficial in the effort to use RRT* in an anytime fashion, and should be explored in future work.

2.4 Visual Multi-Target Tracking

There are a number of well known visual multi-target tracking (MTT) algorithms in use. These include MHT [29], JPDA [7], MCMCDA [27], and GM-PHD [32]. A recently proposed MTT algorithm is Recursive-RANSAC [26], which is a visual MTT algorithm developed at Brigham Young University. Among its strengths are its ability to maintain track continuity through crossing tracks, infer the number of tracks instead of knowing them upfront, and reasonably quick execution time per frame. Its weaknesses are that the waldos need to be moving, and it is limited in its ability to reacquire waldos that go out of frame whether that's due to occlusion or limited field of view. A well-planned path can limit the number of times one of those weaknesses is encountered, thus helping the vision algorithm have more robust performance.

2.5 Waldo Following

Following and tracking waldos requires that the target stay in frame, or be brought back into frame quickly when it leaves. If future waldo positions can be predicted, then the UAV can make more informed decisions about where to fly.

One algorithm that performs waldo following uses probabilistic methods to combine the data between both UAVs and unmanned ground vehicles in an attempt to search for, and track, a single moving target [34]. Another algorithm [24] seeks to position the observing agent so that it maximizes the minimum time it would take for the target to leave the frame.

The approach wasn't feasible for a lookahead of more than one timestep when the paper was written, but a similar approach may be more tractable on modern hardware.

In order to accurately track waldos, an algorithm needs to be able to predict their motion. In Reference [3], the authors chose to fuse a learned sequence model and a kinematic motion model to predict future waldo location. Human motion prediction has been studied [35], with agent-based, entity-based, and flow-based models being a few of the main models being researched. Agent-based and entity-based models might be of particular interest to the application in this thesis, as they model the crowd as individuals rather than a single fluid flow.

Chapter 3

Online RRT* and Online FMT*: Rapid Replanning with Dynamic Cost

Bryant Chandler and Michael A Goodrich. Online RRT* and Online FMT*: Rapid Replanning with Dynamic Cost. In *International Conference on Intelligent Robots and Systems (IROS), 2017.* IEEE, 2017. — Under Review

3.1 INTRODUCTION

There are many scenarios in which a robot might need to change goals in the middle of a task or adapt to changes in its environment. The naive approach would be to plan an entirely new path from scratch, but this is computationally expensive and does not take advantage of the information already learned about the configuration space. Replanners exist that can adapt for unexpected obstacles [1, 19, 25, 30], but they do not adapt to changing cost functions, and do not support replanning to new end points. Another replanner [15] improves the path as it travels, but it does not support changing cost functions or replanning to new end points. We propose two algorithms, Online RRT* (ORRT*) and Online FMT* (OFMT*), that adjust online as the environment and robot positions change. The algorithms facilitate (a) rapid replanning when goals change, (b) adapting paths when the cost function or environment changes, and (c) planning for multiple objectives; all while maintaining memory efficiency.

For this paper our distance unit will be the diameter of the robot. We assume that a robot moves at a constant rate of 0.15 units per timestep. A map is 4900 square units with randomly generated rectangular (non-overlapping) obstacles. Obstacle sizes range from 0.01 square units to almost as large as the map. We do not allow obstacles that take up the

entire map, because it would become infeasible to plan paths. An example map can be seen in Figure 3.7. Note that we use rectangular obstacles for simulation, but our algorithms can function on an arbitrary occupancy grid.

ORRT* and OFMT* make two key additions to RRT*: (1) the location of the RRT* root changes to match the robot's location when the robot moves, and (2) new nodes are sampled up to a predefined density, and after that point *online sampling* only samples and rewires without adding new nodes.

We empirically validate the algorithms by comparing computation efficiency to FMT*, A* on a visibility graph, and A* on a grid. Additionally, we empirically verify that the algorithms can adapt to time-varying cost functions by comparing the resulting path cost to an approximately optimal "ground truth" path computed using PRM*.

3.2 RELATED WORK

This section describes three areas of related research: discrete-based planning, sampling-based planning, and replanning.

Discrete Planning. There are many algorithms in the literature for discrete multi-objective path planning, one of the most notable being A* [33] which is an alternative to Dijkstra's algorithm [4] when the goal is unknown. These algorithms build a graph in a configuration space that has been discretized into a grid with known available transitions between grid cells. They then perform search on those graphs. This allows the algorithms to operate quickly, but discretization reduces the likelihood of finding a truly optimal solution; the algorithms might not find paths through narrow passages and the location of grid cells might keep it from finding an optimal path. D* [30], AD* [25], and D* Lite [19] expand on Dijkstra and A* to get dynamic graphs that can handle unexpected obstacles, but they suffer from the same discretization issues.

Sampling-Based Planning. Sampling-based path planning addresses some of the problems created by discretizing the configuration space. Algorithms like RRT [21] and PRM

[16] randomly sample a continuous configuration space, allowing for more path options with better performance than would be gained by simply increasing the resolution of discretization. RRT builds a rapidly exploring tree from random sampling and a cost function that can combine the multiple objectives. PRM builds a graph from randomly sampled points, and then paths are queried using another algorithm such as A*. The time complexity for constructing search trees is the same for both RRT and PRM, but RRT is $O(n)$ in query and space complexity whereas PRM is $O(nlog(n))$ in both query and space complexity; the reduced query time and memory requirements generally make RRT the favorite of the two. Additionally, RRT includes the ability to plan kinodynamically [23] (accounting for vehicle dynamics and velocity).

RRT has been expanded to RRT* [14], which guarantees asymptotic optimality as more points are sampled by rewiring the existing tree to get path cost reductions. An advantage of RRT* is that a sub-optimal path can be found quickly with a sparse tree. For the problems considered in this paper, a robot may be traversing a long distance and therefore the algorithm must be capable of running for a long time. Consequently, quickly generating a dense (and therefore closer to optimal) tree is more important than finding sub-optimal solutions on a sparse tree. FMT* [13] was designed to generate dense trees; FMT* constructs a tree with the same structure and optimality guarantees as RRT*, but samples all node locations before beginning, and builds out from the start point densely.

Replanning. Several solutions have been developed to allow RRT* to function in an online/anytime fashion. Anytime RRT* [15] takes advantage of the time it takes for a robot to travel a path by improving the path during travel. It does this by pruning the part of the tree that has already been traversed and continuing to sample. Anytime RRT* does not meet the requirements of this paper because it destroys part of the tree as the robot moves and our application benefits from reusing information that could be destroyed. Another notable solution is RRT^X [1], which switches the start and end points, so that the end point is now fixed and finding optimal paths to all other points in the space. Thus, when

the start point moves, it can simply use one of the other paths. When obstacle conditions change, a cascading rewire is performed to update the tree. This is a reasonable approach to being online, but it still requires one of the points to be fixed, which is not the case for our application. We need to be able to move both the start and end point as the UAV moves, and we are less concerned about unexpected obstacles.

3.3 APPROACH

This section details the Online RRT* (ORRT*) and Online FMT* (OFMT*) algorithms. A tree-based path planning algorithm might plan a path as seen in Figure 3.1. The robot starts at the red square, and plans a path to the purple circle, but when it reaches the green circle the robot realizes that it actually wants to get to the blue point. The naive solution is to make a completely new plan to get to the new goal, but that is computationally expensive. If the robot can update the graph as it goes to take advantage of existing information, it will afford a shorter replanning time when the goal changes.

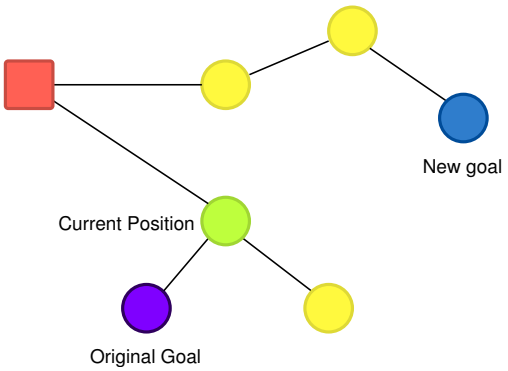


Figure 3.1: Need to replan to new goal after the robot has moved

3.3.1 Algorithm

ORRT* and OFMT* adapt to start and end point changes without needing to expensively build an entirely new tree. Instead, ORRT* and OFMT* reuse the existing tree with minimal new memory allocation and manageable processor utilization. In order to accomplish this,

they depend on the asymptotic optimality guarantee of RRT*, which guarantees that the paths in the tree will approach optimality as the number of sampled nodes increases. The guarantee is possible, because every time a new node is added to the tree its neighbors are rewired, thus improving the tree to better fit the cost function. If we can rewire and improve the tree without linearly increasing the number of nodes, then we can adapt to a changing environment and changing cost functions while maintaining constant memory usage. A side benefit of using these tree-based algorithms is that the optimal path from the start point to every other point in the configuration space is embedded in the tree, so we can rapidly replan to new goals.

The problem is solved in three parts. The first part is *online sampling*, which varies slightly between ORRT* and OFMT* during initialization. ORRT* (see Algorithm 1) adds nodes to the tree until a "node add threshold" is reached. OFMT* (see Algorithm 2) samples nodes up to the "node add threshold", and then densely builds a tree with the sampled nodes. Once tree building is complete, both algorithms begin *online rewiring* by sampling and rewiring without adding nodes (see Algorithm 3). In both ORRT* and OFMT*, the online sampling and online rewiring approach allows the tree to re-optimize when the start point moves, or the cost function changes.

The second algorithm is *start-point moving* (see Algorithm 4), which (a) adds a node at the new start point and then (b) rewires appropriate neighbors to that point.

The third algorithm is *online pruning*, which balances for the newly added node by removing a leaf node in the vicinity of the new root.

The next three subsections discuss start-point moving, online sampling and rewiring, and online pruning, respectively.

3.3.2 Start-Point Moving

RRT* and FMT* are able to handle changing the end point of the path, because they find the asymptotically optimal path to all of the points in the configuration space. What they

Algorithm 1 ORRT* Online Sampling

```
1: procedure SampleOnline
2:     if number of nodes in tree < nodeAddThresh then
3:         sample and add a node as per RRT*
4:     else
5:         RewireOnline()
```

Algorithm 2 OFMT* Online Sampling

```
1: procedure SampleOnline
2:     if tree is not initialized then
3:         randomly generate nodeAddThresh nodes
4:         run FMT* to completion on sampled nodes
5:     else
6:         RewireOnline()
```

Algorithm 3 Online Rewiring

```
1: procedure RewireOnline
2:     randPoint ← randomly sample a point
3:     neighbors ← all nodes in radius of randPoint
4:     nbr* ← best neighbor ∈ neighbors
5:     update cost from nbr* to its parent
6:     update cost for all children of nbr*
7:     for each  nbr ∈ neighbors do
8:         potCost ← nbr*.tCost + cost(nbr*, nbr)
9:         if potCost < nbr.tCost then
10:            rewire nbr to nbr∗
11:            update cost for all children of nbr
```

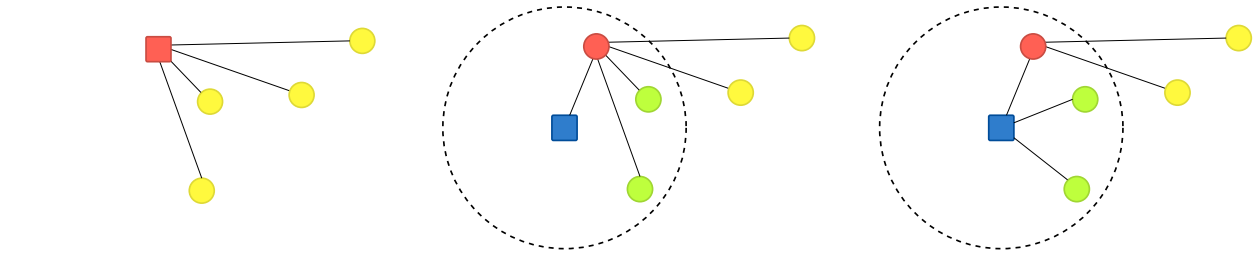
Algorithm 4 Move Start

```
1: procedure MoveStart
2:     newRt ← new node at newStartPoint
3:     wire oldRoot to newRt
4:     update cost for all children of oldRoot
5:     neighbors ← all nodes within a radius of newRt
6:     for each nbr ∈ neighbors do
7:         potCost = newRt.tCost + cost(newRt, nbr)
8:         if potCost < neighbor.tCost then
9:             rewire neighbor to newRoot
10:            update weights for all children of neighbor
11:    for each nbr ∈ neighbors do
12:        if numNodes > nodeAddThreshold then
13:            if dist(newRt, nbr) < pruneDist then
14:                if nbr.isLeaf then
15:                    remove nbr
16:        else
17:            RETURN
```

are not able to do is change the goal after the robot has started traveling; the end point has to remain fixed. Because a robot's goal can change after the robot starts traveling, the replanning algorithm must be able to move the start point and update the tree. As illustrated in Figure 3.2, ORRT* and OFMT* create a new node at the current location of the robot and make that the parent of the original start node. All nodes in the neighborhood of the new node are rewired subject to the constraint that a new edge does not intersect an obstacle. Empirical results have been omitted in the interest of space. These omitted empirical results are intuitive; the tree can be rewired quickly enough as long as the robot moves slowly (not too far between time samples) and continuously. Jumps larger than the rewire neighborhood would destroy the integrity of the tree.

3.3.3 Online Sampling and Rewiring

In order to re-optimize the tree after structure changes, the RRT* and FMT* algorithms need to continue sampling points and rewiring the tree. Naively continuing to sample new points in the configuration space does not work because the size of the tree continues to



(a) Original tree with red square as start point

(b) New blue start node as parent of old red start node, with neighbors in green

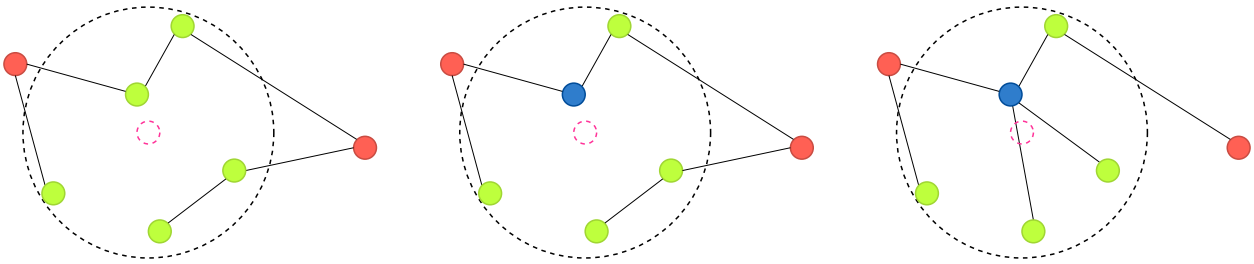
(c) Rewire green neighbors to blue new start point

Figure 3.2: Moving start point

grow to the point that the algorithm becomes intractable in space and time. ORRT* and OFMT* solve this problem by continuing to sample and rewire after we reach a chosen level of saturation (the total number of nodes required to "cover" a configuration space), but not adding new nodes to the tree. Instead of adding a new node to the tree, online rewiring is performed. A new sample is used as the center of a nearest neighborhood search; nodes within the neighborhood are rewired as shown in Figure 3.3. The lowest cost node from the neighbors near the new sample is selected, and all other nodes in the neighborhood are rewired to that best node as their parent, subject to the constraint that the new edge does not intersect an obstacle. This keeps the execution time per iteration approximately the same, if not less than it was when the algorithm reached the node add threshold. Memory utilization stays fixed at the threshold level since no new nodes are being added, and each iteration the tree improves to better fit the current cost function.

Online rewiring can refine the tree as long as the cost function is fixed because it will always try to reduce cost. If the cost function is dynamic, however, the cost along a path might need to increase to match the cost function. To support dynamic cost functions, we add 2 steps once the best neighbor of the sampled point has been found, but before rewiring other neighbors. First, we update the cost between the best neighbor and its parent. Second, we recursively propagate the new cost to all children of the best neighbor. This does not

immediately make the whole tree match the new cost function, but it does shift the overall tree a little closer.



(a) Neighborhood of sampled point

(b) Selecting best neighbor

(c) Rewire other neighbors to best neighbor if it reduces their cost

Figure 3.3: Online rewiring

3.3.4 Online Pruning

By continually adding a new node when the robot moves, the start-point moving algorithm creates a problem, as illustrated in Figure 3.4. *Both ORRT* and OFMT* grow search trees based on a fixed number of nodes.* Adding a new node each time the robot moves means that the number of nodes will increase linearly forever. Eventually, the number of nodes will become intractable in time and space. We could solve this problem by occasionally pruning the tree, but that would have an effect similar to how a garbage collector functions in software; it would have to pause execution occasionally in order to prune extra nodes. Instead of a "pause and prune" approach, we prune a single leaf node that is very close to the new root so that distant parts of the tree are not affected. This keeps the tree at a constant number of nodes without leaving holes (which would occur if we pruned branches or nodes far from the root). Because of the short distance, there is a reasonable chance that the leaf node removed will be the old root, but that is not required.

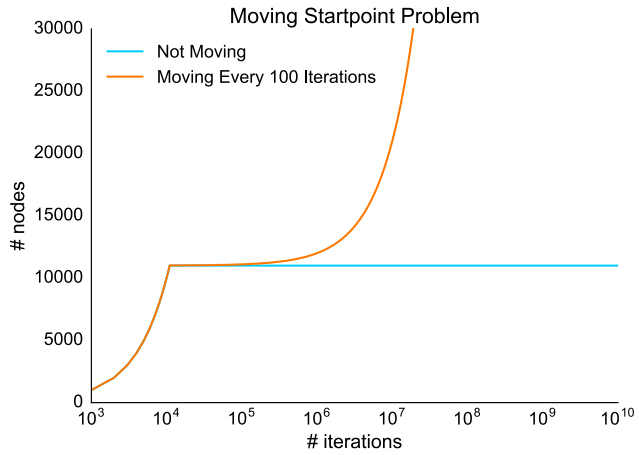
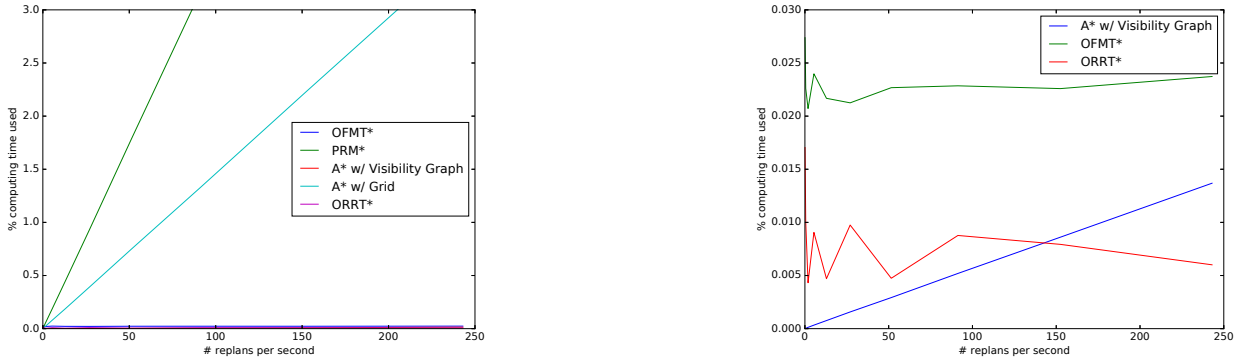


Figure 3.4: Moving the start point adds nodes, which eventually becomes a significant problem

3.4 VALIDATION

3.4.1 Computational Efficiency: Algorithm Comparison

This section empirically compares the computational efficiency of ORRT* and OFMT* to A* over a visibility graph, PRM*, and A* over a grid discretization. Results are shown in Figure 3.5 for all algorithms, as well as for the top three algorithms, namely ORRT*, OFMT*, and A* on a visibility graph. Results are computed for a 30 second simulation with frequent replanning over 50 randomly generated maps. Results use a shortest-path problem, and the optimal path is generated using A* running on a visibility graph (which is guaranteed to find the true shortest path). Each algorithm was simulated with the start point moving at 30 Hz, and replanning frequencies were varied from 1 to 250 Hz. The time for each algorithm to compute a new path was computed. Results are reported as the percentage of the 30 seconds available (the duration of the simulation) that was used for planning; higher values indicate that the algorithm is using more of the available time. Algorithms were allowed to use more than the theoretically available time, that is, they could use more than $1/$*replan frequency* seconds each time they replanned, to demonstrate any inefficiency. All simulations were performed on a desktop workstation with an Intel Core i7-6700 3.4 GHz CPU and 15.6 GB DDR4 RAM.



(a) All tested algorithms

(b) Three best algorithms

Figure 3.5: Percentage of 30 s run time used for path computation

As can be observed in Figure 3.5, PRM* and A* with a grid quickly began to perform very poorly with relatively low replan frequencies and reached far over 100% of the available time. A* with a visibility graph performed the best, despite the fact that it grows linearly with replans per second. Note, however, that A* over a visibility graph only works for shortest-path problems; additionally, as the map becomes more complex the visibility graph becomes more complex and therefore A*, which has an exponential worst-case computational complexity, becomes less efficient. ORRT* and OFMT* had relatively constant results with a low percentage of compute time being used. They continue to take the same amount of computation time as replanning becomes very frequent, because they adjust the tree as the start point moves, and the optimal path to any new end point is embedded in the tree.

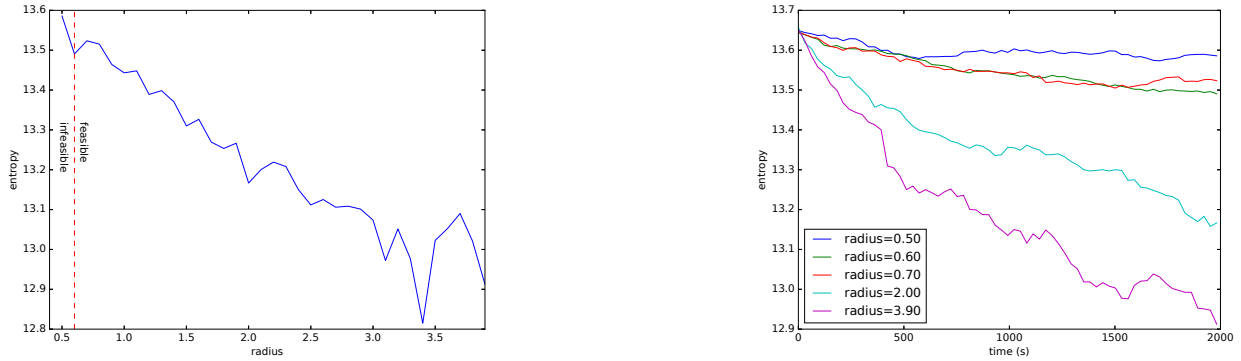
3.4.2 Memory Efficiency

The memory efficiency of ORRT* and OFMT* is best understood by treating a node as a unit of memory. RRT* would add a new node every iteration, which equates to memory usage increasing linearly. ORRT*, by contrast, adds one node per iteration until the threshold is reached. Similarly, OFMT* builds a tree to a predetermined number of nodes. At that point, the number of nodes remains constant at the threshold for both algorithms as long as the pruning radius is set appropriately.

3.4.3 Online Pruning

This section provides evidence that the moving start point algorithm allows a robot to move while still finding optimal paths. The algorithm was executed for 33 minutes and measured the particle entropy of the nodes in our tree. Particle entropy is a measure that helps determine the uniformity of the distribution of nodes. A high entropy value indicates that the distribution is uniform, and therefore the asymptotic optimality guarantees of RRT* hold. We performed the experiment for several different pruning radii as seen in Figure 3.6, and found that 0.7 units was the best radius in our scenario. It can be observed that a radius of 0.5 units had a higher entropy, but that radius does not prune well enough to keep the number of nodes from growing.

Future work should consider how best to select the pruning radius. One potentially important relationship is the ratio of velocity to prune radius, which determines whether or not the old root will fall within the prune radius. The ratio in the simulations above was empirically and subjectively set to 3/14, but better ratios might exist for other environments and other algorithm parameters. A second important parameter which might have influence on the ideal pruning radius is the sampling density. Sampling density can affect the ideal pruning radius because it influences the number of nodes that might fall within a given radius.



(a) Entropy as it relates to pruning radius

(b) Entropy over time with various pruning radii

Figure 3.6: Online pruning

3.4.4 Support for Multiple Objectives

Figure 3.5 illustrated that the only algorithm which might be faster than ORRT* and OFMT* for rapid replanning is A* on a visibility graph. Importantly, A* on a visibility graph only works when the objective function is shortest path. Many scenarios require a more complex cost function, and therefore need to use a different algorithm. When the objective functions have structures that can be exploited, similar to how a visibility graph extracts shortest path structures, then objective-specific efficient algorithms may be possible, but it we claim that ORRT* and OFMT* can work for a wide range of objectives.

To provide evidence for this claim, Figure 3.7 presents an an example of OFMT* using a cost function that seeks to avoid being "seen" by the pink and blue circles in the world. The selected path is obviously not the shortest path, and the tree shows a lot of curvature as it tries to avoid the pink and blue circles. The path also tends to maximize the time that the robot is hidden behind obstacles and, when not hidden behind obstacles, try to be far from the pink and blue circles to make the probability of being seen smaller. Space does not allow a full presentation of how many objectives are compatible with the algorithms, but note that simulation results generate subjectively acceptable paths for many convex blends of the shortest path objective and the "stealth" objective of avoiding being seen.

3.4.5 Time-Varying Cost

In many real-world scenarios, assuming a fixed cost function is unrealistic. ORRT* and OFMT* are capable of adjusting to changes in the cost function as long as those changes are gradual. Future work needs to characterize what constitutes a "gradual" change. To provide evidence in support of the claim that the algorithms adjust to gradual changes in cost functions, we simulated with fixed start and end points on 4 different maps. The cost function was based on the "stealth" objective of hiding from moving enemies. Cost is high for points where enemies can see the robot and are close to it. We allow the enemies to move over time, which gives us a time-varying cost function.

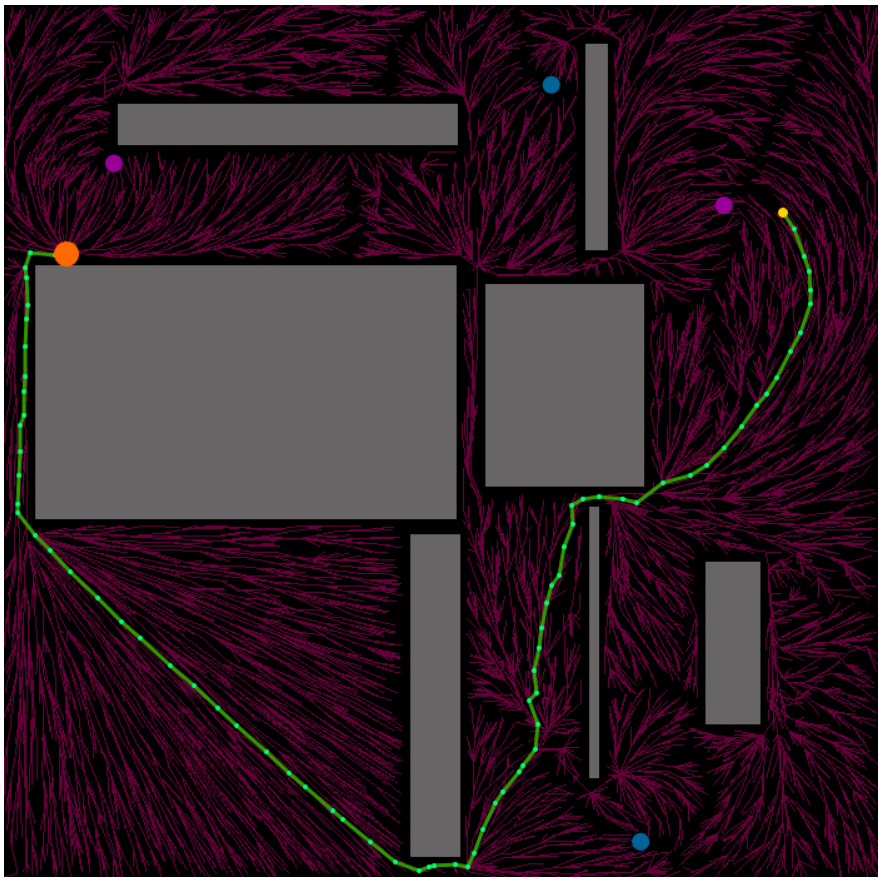
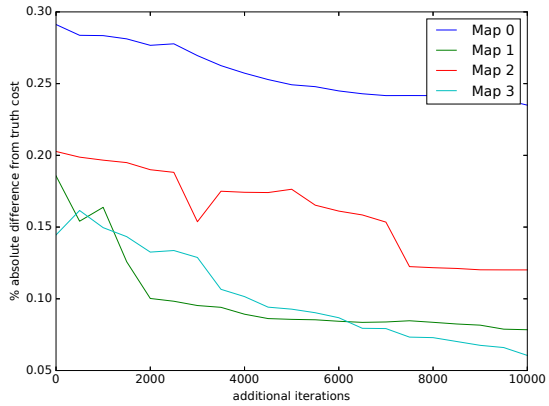
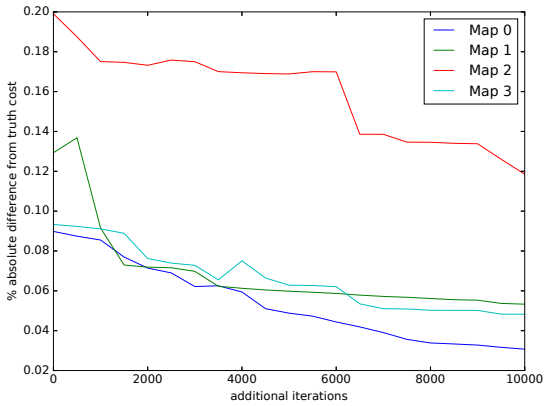


Figure 3.7: A tree and path created using a cost function that attempts to avoid the pink and blue circles

For 5 randomly generated scenarios, ground truth optimal paths at 14 time steps over a 30 second trial were found using PRM*. For ORRT* and OFMT*, we paused the simulation at the same 14 time steps and ran the algorithms for an additional 10000 iterations. Figure 3.8 illustrates how ORRT* and OFMT* adapt their paths in such a way that path costs approach the optimal cost after the cost functions change. The trend toward decreasing absolute difference between the true optimal and the adapting path illustrate that ORRT* and OFMT* can adjust for time-varying cost and be used in an anytime fashion; given faster processors they would be able to compute fully optimal paths in real-time. Further research is required to assess how the algorithms perform with different cost functions as computation of cost can be a significant computational load.



(a) ORRT*



(b) OFMT*

Figure 3.8: Anytime percent absolute difference from truth path cost with time-varying cost function

3.5 SUMMARY AND FUTURE WORK

This paper illustrates that the ORRT* and OFMT* algorithms rapidly replan in dynamic environments. Replanning to new end points occurs in real-time, and computational efficiency is good enough to scale to rapid replan frequencies. Additionally, the algorithms can adjust to gradual time-varying objective functions in an anytime fashion. This allows for scenarios, for example, where the cost depends on the positions of other moving agents.

The strength of ORRT* is to rapidly replan for new end points and adapting to changing cost functions, but that might not be its only application. We plan an extension to make ORRT* work for very large worlds by applying a sliding window to the planned tree. This approach would delete nodes that fall outside of the window due to movement, and add new nodes when new areas are exposed. The same framework would allow for handling moving obstacles.

All results in this paper were gathered assuming 2D worlds, but the same algorithms theoretically work for 3D configuration spaces. A possible limitation is that adding another dimension would increase the number of nodes required and increase the amount of time

required to compute cost functions. Future work should evaluate how well these algorithms work in 3D.

Further future work needs to be done with the online pruning radius. The simulations showed that the ideal pruning radius depends on decisions about other algorithm parameters such as maximum segment length and UAV travel velocity. Future work would involve running many more simulations in this combinatorial space, and analyzing the relationship between the various parameters to establish rules of thumb.

Chapter 4

End-Point Selection

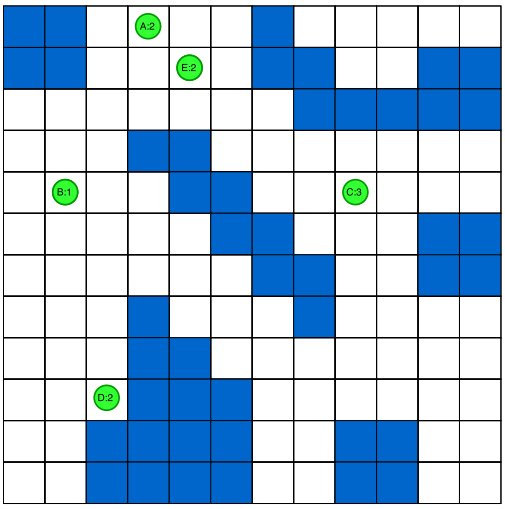
Chapter 3 presented two path-planning algorithms, ORRT* and OFMT*, that afford rapid replanning to new end-points while the UAV is moving. Such algorithms are necessary because the end-point selection method presented in this chapter executes at every timestep, and the selected end-point can vary dramatically between timesteps.

This chapter presents an algorithm that selects an end-point based on the expected utility for having one or more waldos in frame. This allows the UAV to follow a subset from a large group of waldos with a prioritized approach rather than naively following only the most important waldo.

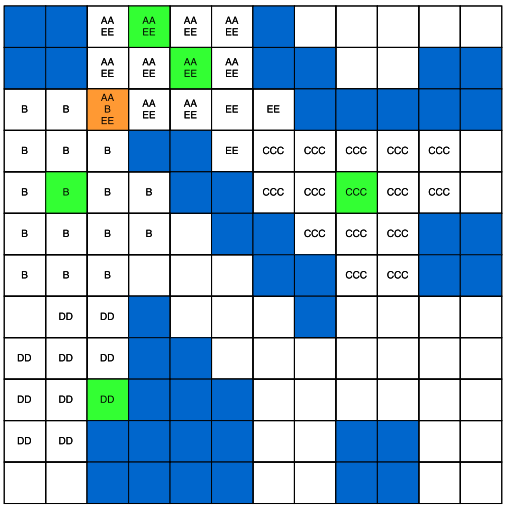
4.1 Selection Method

The end-point of the ORRT* or OFMT* should be a location that maximizes the important waldos that can be seen at that location. A well-chosen end-point allows the vision algorithm to more effectively and accurately track waldos, and should attempt to preempt occlusion.

The first step in choosing an end-point is to discretize the space into a grid of bins. For this thesis, the importance of a waldo is randomly selected from integer range $\{0, 1, 2\}$ when the simulation is initialized. In practice, these values would be assigned by a human operator and the range of importance values could be increased to provide more control. The important thing is that the zero-value represents being unimportant and the two-value represents greatest importance. A value is assigned to each poential search bin by adding the importance scores for each waldo that can be seen from the center of the grid cell.



(a) Waldos as green circles with a letter as a label, and number as importance



(b) Scores tallied by waldo label, with chosen end cell in orange

Figure 4.1: End-point selection

The score for the waldos is scaled by distance between the waldo and the UAV, as encoded in Equation 4.1.

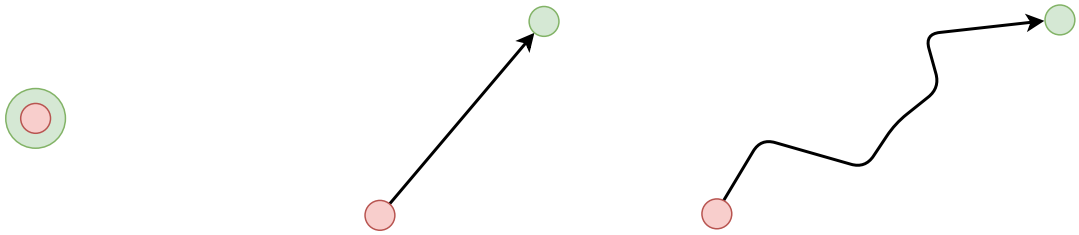
$$score = max(0, importance * (viewRadius - dist(waldoLocation, cell))) \tag{4.1}$$

This equation creates a bin score with a peak at predicted waldo location, scaled by importance, and decreasing from the center to a maximum distance away (the view radius). The max function assures that the values are greater than zero. We assume that the UAV cannot see through walls, so the visibility region is computed for each possible UAV end-point. Within each cell, we assume that detection probability is uniform, meaning that the distance-weighted probability is measured from center of one cell to the center of another.

Weighting a waldo's importance by the probability of seeing it (the distance scaling) produces a cell value that represents the expected utility of the UAV being located in the given cell. The cell with the maximum expected utility is selected as the end-point for path-planning.

To improve the end-point selection we not only accumulate votes for the current location of waldos, we use what we know about their historic movement to predict future locations and accumulate votes for those locations. We discuss the algorithm for predicting locations in the next section.

4.2 Future Prediction



(a) Center (b) Constant velocity (c) Directed random walk

Figure 4.2: Prediction methods

The three prediction methods are illustrated in Figure 4.2; the red circle represents the current location of the waldo, and the green circle represents the predicted future location of the waldo. The naive prediction method (Figure 4.2a) is to assume that the Waldo will stop at its current location. Hereafter, this method will be referred to as "center".

The second prediction method (Figure 4.2b) is a moving average filter of frame to frame velocities. We can use the velocity vector from the moving average filter to predict into the future assuming that the Waldo travels at constant velocity. This method will likely fail for long look-ahead because a Waldo is unlikely to have constant velocity for a long period of time. This method will be referred to hereafter as "constant velocity".

The final prediction method (Figure 4.2c) uses the heading ψ and magnitude K_v of the velocity vector gathered from moving average filter to constrain a random walk. At each prediction time step the x and y positions move at a rate of K_v in the direction ψ as seen in

Equations 4.2 and 4.3.

$$x(t+1) = x(t) + K_v \Delta t \cos(\psi(t)) \tag{4.2}$$

$$y(t+1) = y(t) + K_v \Delta t \sin(\psi(t)) \tag{4.3}$$

$$\psi(t+1) = \psi(t) + \eta \quad \text{where} \quad \eta \sim U(-1,1) \tag{4.4}$$

The value of $\psi(t+1)$ is calculated in Equation 4.4 by adding the value at $\psi(t)$ to a random angle selected from a uniform distribution, $\eta \sim U(-1,1)$, where the unit is radians. The reasoning behind using a directed random walk is that it might be more helpful in cluttered environments because it allows a prediction that goes around a corner without allowing a waldo to double back on itself. This method will be referred to hereafter as "directed random walk".

4.3 Results

To compare the methods for predicting future waldo location, simulations were conducted with several different prediction method combinations over different numbers of obstacles. The parameters used can be found in Table 4.1 in addition to the description provided in this paragraph. To account for variance, 120 trials were performed at each obstacle density level with each trial lasting 3 minutes. The obstacle densities where 5, 10, 15, and 20 obstacles. For each map the obstacles were placed randomly with the constraint that no obstacles could overlap. Scores were recorded at 30 Hz, with a score being the sum of the importance values for all waldos within the viewing radius of the UAV. The importance values of the waldos were 0 for unimportant, 1 for semi-important, and 2 for very important. Each of the prediction methods was tested on the same 120 trials with recorded obstacles and waldo positions.

For all development and experiments, waldo paths were randomly simulated. This was accomplished using a unique RRT* planner for each waldo. The planner selects a random end-point and samples until a path is found, but doesn't continue refining the path after

Obstacle Densities	5, 10, 15, 20
Trials per Density	120
Trial Duration	3 minutes
Scoring Frequency	30 Hz
Importance Values	0, 1, 2

Table 4.1: Parameters for endpoint selection simulations

that point. The waldo then travels along the path at constant velocity. When the end-point is reached, the process is repeated. This approach provides purposeful waldo movement, without having the paths be too direct. We have observed that the simulated waldos seem to have realistic human movement, but future work should include tests with paths sampled from real humans.

In Figure 4.3 we see the average score over the 120 trials on the y-axis, and the number of obstacles on the x-axis. Each line represents the results for a different combination of prediction methods. Observe that the best results were derived using a combination of the current waldo center and the constant velocity predicted location with a two second look-ahead. Also observe that the three prediction methods with the lowest scores are the ones which include a directed random walk, likely indicating that a directed random walk is not a useful prediction method given the "purposeful" way that the waldos planned their paths. Finally, observe that the remaining prediction methods where nearly indistinguishable.

Future work should include exploring longer look-ahead, as these trials were all conducted with a two second look-ahead. A longer look-ahead might change the value of the various prediction methods. Other future work should include methods for modeling the purposes used by the waldos to plan paths.

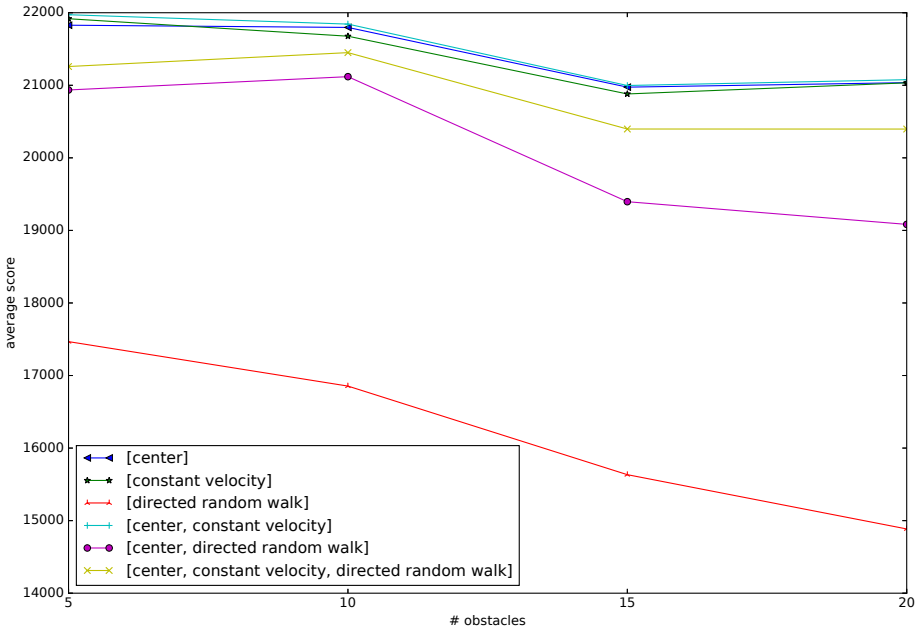


Figure 4.3: End-point selection scored for different prediction methods

Chapter 5

Cost Function

Chapter 3 and Chapter 4 presented algorithms that can rapidly replan to end-points that attempt to maximize the number of waldos within the viewing radius of the UAV. A limitation of the algorithms in the previous chapters is that they do no consider how much information is gained while the UAV moves along its path to the end-point. Instead, the end-point is chosen to maximize the expected utility of seeing important waldos *once the end-point is reached*, and the path is chosen that minimizes some objective function so that the path to reach the end-point is optimal. This chapter addresses this limitation by designing a cost function that enables the UAV to increase information gain while traveling to the planned end-points.

We hypothesized that maximizing visible area during travel will minimize the number of places that the waldo could "go to hide", that is, the number of places where the waldo could go in the near future and be be occluded from the UAV. Minimizing the places to hide would reduce the chance of losing an important waldo being tracked. As the results will demonstrate, a cost function that maximizes information game produces intuitive behaviors such as swinging wide at corners so the waldo doesn't abruptly become occluded when it turns a corner. We design a cost function that has a low value for areas that have high visibility and a high value for areas with low visibility. Exact computation of this cost is expensive, so we present a technique to approximate travel cost, thus making the approach computationally feasible.

5.1 Maximizing Visible Area

The polygon that is visible from any point in a 2D map can be found using many different ray-tracing techniques. A ray sweeping visibility polygon algorithm [28] is used in this thesis because it is relatively efficient and forms a simple and accurate polygon. Having a simple polygon is advantageous because it is easy to compute the area of such a polygon, and having an accurate polygon will lead to a cost function with low noise. Given the area of the polygon, the *unseen open area* can be computed, denoted by $u(i)$, that tells us how much of the map isn't visible from point i, excluding the space taken up by obstacles.

Equation 5.1 calculates the unseen open area u for point i.

$$u(i) = area(map) - totalObstacleArea - area(visiblePolygon(i)) \tag{5.1}$$

The components are as follows: $area(map)$ is the total area of the map, $area(visiblePolygon(i))$ finds the polygon visible from point i and computes its area, and $totalObstacleArea$ is the total area occupied by obstacles. Subtracting $totalObstacleArea$ from $area(map)$ gives the open area of the map not occupied by obstacles. Finally, $area(visiblePolygon(i))$ is subtracted from the open area, leaving the area that isn't visible from point i.

The unseen open area u is used to influence the travel cost between nodes. The most accurate measure of the travel cost would be to integrate the unseen area at very small intervals along each edge of the path. This, however, is very slow and is compounded by the fact that we have to compute a travel cost for every edge we consider adding, not just the edges we actually add. As a solution to this problem, the unseen area across the edge is approximated by (1) computing the average of the unseen area u at the end-points of the edge and (2 multiplying that average by the length of the edge. Approximating the travel cost allows the use of dynamic programming to store and reuse the unseen area at each node.

The approach is similar to numerical integration using the midpoint method. The resulting equation for approximating the edge cost is given by

$$eu(i,j) = \frac{u(i) + u(j)}{2} dist(i,j) \tag{5.2}$$

The proposed approximation certainly has loss, but it is made more reasonable by the fact that segments are likely to be very short for reasonable sampling parameters for ORRT* and OFMT*. A short segment length allows us to assume that the rate of change between the two points is close to constant, thus making an averaged approximation reasonably accurate. Given the random sampling associated with these planners, a scenario can be imagined where the segments are very long with significant cost fluctuations between them. If the random sampling produced long edges, the approximation in Equation (5.2) would almost certainly fail. Fortunately, sampling densities can be controlled, making it more likely that edge segments are reasonably short.

5.2 Results

We hypothesized that using unseen area in the cost function will reduce the chance that the UAV will lose tracking of a waldo due to occlusion. To test this hypothesis, we run the same simulation that we used in Section 4.3, but we only use the high-performing prediction method that combines center, constant velocity, and directed random walk. The table of parameters has been repeated in Table 5.1 for the convenience of the reader. We simulated two cost functions, shortest path and unseen open area, on the same scenarios, and compared the total score for each scenario.

Obstacle Densities	5, 10, 15, 20
Trials per Density	120
Trial Duration	3 minutes
Scoring Frequency	30 Hz
Importance Values	0, 1, 2

Table 5.1: Parameters for cost function simulations

Figure 5.1 shows the average score over the 120 trials on the y-axis and the number of obstacles on the x-axis. We see that shortest path seems to do better for all but the lowest number of obstacles, but the actual score differences are very small. This might be due to the fact that unseen area doesn't have a strong influence on the path shape, thus the paths computed are almost the same as the shortest path. Based on these results, we have to conclude that the method proposed in this chapter didn't accomplish its purpose. Future work should explore other cost functions which can provide a path shape which provides more incentive for the robot to maximize what it can see.

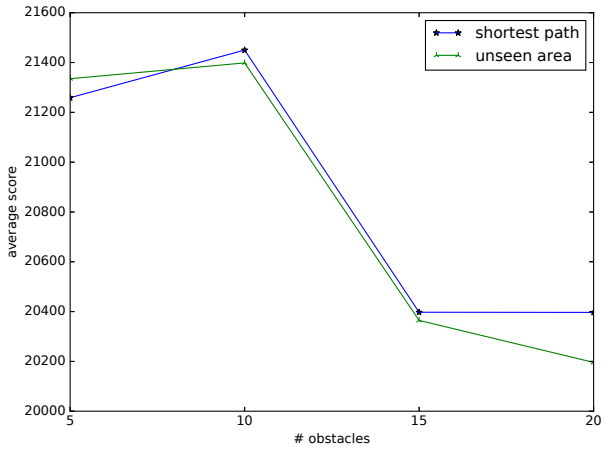


Figure 5.1: Scores with different cost functions

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Chapter 3 demonstrated that ORRT* and OFMT* can replan at a rate much faster than the 30 Hz required in the specification for the application. Chapter 4 demonstrated that the best endpoint selection used a combination of the center and constant velocity predictions. Therefore, we conclude that the combination of the algorithms in this thesis does work for following waldos. It rapidly replans in real-time using only a small portion of the available compute time, and the end-point selection method selects subjectively reasonable end-points.

After observing the waldo following algorithm in simulations, we hypothesize that the same results might have be accomplished with a short horizon A* or PRM* planner rather than OFMT* or ORRT*. Such a switch is likely possible for a few reasons: (a) the lookahead is short so it is possible to plan for a short horizon, (b) planned paths almost never vary from shortest path for the objective functions used in this thesis, and (c) replanning occurs at 30 Hz and both A* and PRM* are capable of replanning at that rate as can be observed in Figure 3.5. The strengths of ORRT* and OFMT* for rapid replanning might be showcased more thoroughly if the waldo prediction lookahead was significantly increased, or if a different cost function that varied significantly from shortest path was employed.

OFMT* and ORRT* were useful in solving the problem presented in this thesis, but there are additional problems for which they are likely to be very useful. One example is for time-varying cost functions as presented in Chapter 3. Many real world scenarios do not allow a static cost assumption, such as worlds with partially known environments, moving

obstacles, and moving enemies. The conditions of the world might change at any time, and using a planner like ORRT* or OFMT* allows the UAV to rapidly adjust its plan to suit the conditions. While we have shown that adjusting for dynamic cost works in an anytime fashion, current hardware limits us from reaching real-time performance. We expect that near future hardware will be capable of pushing OFMT* and ORRT* into real-time performance. OFMT* is currently being used in a user study that aims evaluate the nature of user intent in a planning problem with dynamic costs. The ability of OFMT* to be used in a user study demonstrates its utility in a problem that requires adaption to dynamic cost while the robot is moving. Future work should explore applications of time-varying cost more fully.

6.2 Future Work

In Chapter 4, results were acceptable for the prediction methods with a 2-second lookahead, but there is room for expansion. Future work should explore additional prediction methods, including some of the models discussed in Section 2.5, as well taking user input about how a waldo might move. Future work should also explore longer look-ahead to better reduce the chance of tracking loss, with the additional benefit that a longer look-ahead would more fully utilize the benefits of ORRT* and OFMT*. Finally, future work should take a deeper dive into the outcomes in more cluttered environments with more unique object layouts.

We hypothesized that using a cost function based on unseen area might help the UAV to plan paths which swing wide at corners and increase the time it would take for a waldo to become occluded. Empirical results show that the proposed cost function does not improve the UAVs ability to keep waldos in frame, but it still seems that behavior such as swinging wide at corners should improve tracking. Future work should include experiments with a wider range of cost functions that can encode this type of reasoning. One possibility is to measure the distance to obstacle borders and penalize for being too close to obstacles, thus encouraging paths that are centered between obstacles.

The chapter on ORRT* and OFMT* discusses the use of those algorithms in 3D worlds. 3D worlds allow the UAV to make decisions about changing altitude to increase or decrease view radius, and it also provides the option to fly over buildings. Such an environment would be much more broadly applicable than assuming urban canyons with buildings tall enough to disallow flyover.

The algorithms that we have developed should generalize for use with any Multi-Target Tracking algorithm as long as it tracks in an anytime fashion. Future work should include validation of the claim that the approach presented in this thesis works with Recursive-RANSAC and other major MTT algorithms.

References

[1] Joshua Bialkowski, Michael Otte, Sertac Karaman, and Emilio Frazzoli. Efficient Collision Checking in Sampling-based Motion Planning via Safety Certificates. *The International Journal of Robotics Research*, 26:212–240, 2010.

[2] Michael S Branicky, Steven M LaValle, Kari Olson, and Libo Yang. Quasi-randomized path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 2, pages 1481–1487, 2001. doi: 10.1109/ROBOT.2001.932820.

[3] Kevin Cook, Everett Bryan, Huili Yu, He Bai, Kevin Seppi, and Randal Beard. Intelligent cooperative control for urban tracking. *Journal of Intelligent & Robotic Systems*, 74 (1-2):251, 2014.

[4] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[5] Stephen R Dixon and Christopher D Wickens. Control of multiple-UAVs: A workload analysis. Technical report, DTIC Document, 2003.

[6] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014.

[7] Thomas E Fortmann, Yaakov Bar-Shalom, and Molly Scheffe. Multi-target tracking using joint probabilistic data association. In *Proceedings of the 19th IEEE Conference on Decision and Control including the Symposium on Adaptive Processes*, pages 807–812, Dec 1980. doi: 10.1109/CDC.1980.271915.

[8] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004, Sept 2014. doi: 10.1109/IROS.2014.6942976.

[9] Christopher Geyer. Active target search from UAVs in urban environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2366–2371, May 2008. doi: 10.1109/ROBOT.2008.4543567.

[10] Jason K Howlett, Timothy W McLain, and Michael A Goodrich. Learning Real-Time A* path planner for unmanned air vehicle target sensing. *Journal of Aerospace Computing, Information, and Communication*, 3(3):108–122, 2006.

[11] Toru Ishida and Richard E Korf. Moving-target search: A real-time search for changing goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6):609–619, 1995.

[12] Lucas Janson, Brian Ichter, and Marco Pavone. Deterministic Sampling-Based Motion Planning: Optimality, Complexity, and Performance. *arXiv preprint arXiv:1505.00023*, 2015.

[13] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research*, 34(7):883–921, 2015.

[14] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[15] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime Motion Planning using the RRT*. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1478–1483, May 2011. doi: 10.1109/ICRA.2011.5980479.

[16] Lydia E Kavraki, Petr Svestka, Jean Claude Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[17] Jongrae Kim and John L Crassidis. UAV path planning for maximum visibility of ground targets in an urban area. In *Proceedings of the 13th Conference on Information Fusion (FUSION)*, pages 1–7. IEEE, 2010.

[18] Jongrae Kim and Yoonsoo Kim. Moving ground target tracking in dense obstacle areas using UAVs. *IFAC Proceedings Volumes*, 41(2):8552–8557, 2008.

[19] Sven Koenig and Maxim Likhachev. D*Lite. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 476–483, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. ISBN 0-262-51129-0.

[20] Richard E Korf. Real-time heuristic search. *Artificial intelligence*, 42(2-3):189–211, 1990.

[21] Steven M Lavalle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report, Computer Science Department, Iowa State University, 1998.

[22] Steven M LaValle. *Planning Algorithms*, chapter 5. Cambridge University Press, 2006.

[23] Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[24] Steven M LaValle, Héctor H González-Baños, Craig Becker, and Jean-Claude Latombe. Motion strategies for maintaining visibility of a moving target. In *Proceedings of International Conference on Robotics and Automation*, volume 1, pages 731–736, Apr 1997. doi: 10.1109/ROBOT.1997.620122.

[25] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime search in dynamic graphs. *Artificial Intelligence*, 172(14):1613–1643, 2008.

[26] Peter C Niedfeldt and Randal W Beard. Multiple target tracking using recursive RANSAC. In *Proceedings of the American Control Conference (ACC)*, pages 3393–3398. IEEE, 2014.

[27] Songhwai Oh, Stuart Russell, and Shankar Sastry. Markov chain Monte Carlo data association for multi-target tracking. *IEEE Transactions on Automatic Control*, 54(3): 481–497, 2009.

[28] Amit Patel. 2D Visibility from Red Blob Games. http://www.redblobgames.com/articles/visibility/, Jun 2012. [Online; accessed 29-Jul-2016].

[29] Donald B Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, Dec 1979. ISSN 0018-9286. doi: 10.1109/TAC.1979.1102177.

[30] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3310–3317, May 1994. doi: 10.1109/ROBOT.1994.351061.

[31] Konstantinos I Tsianos, Ioan A Sucan, and Lydia E Kavraki. Sampling-based robot motion planning: Towards realistic applications. *Computer Science Review*, 1(1):2–11, 2007.

[32] Ba-Ngu Vo and Wing-Kin Ma. The Gaussian Mixture Probability Hypothesis Density Filter. *IEEE Transactions on Signal Processing*, 54(11):4091–4104, Nov 2006. ISSN 1053-587X. doi: 10.1109/TSP.2006.881190.

[33] Wikipedia. A* search algorithm — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/w/index.php?title=A*_search_algorithm&oldid=719205928`, 2016. [Online; accessed 18-May-2016].

[34] Huili Yu, Randal W Beard, Matthew Argyle, and Caleb Chamberlain. Probabilistic path planning for cooperative target tracking using aerial and ground vehicles. In *Proceedings of the American Control Conference*, pages 4673–4678. IEEE, 2011.

[35] Suiping Zhou, Dan Chen, Wentong Cai, Linbo Luo, Malcolm Yoke Hean Low, Feng Tian, Victor Su-Han Tay, Darren Wee Sze Ong, and Benjamin D Hamilton. Crowd modeling and simulation technologies. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 20(4):20, 2010.