JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 3, March 2006

Learning Real-Time A* Path Planner for Unmanned Air Vehicle Target Sensing

Jason K. Howlett
Ames Research Center, Moffett Field, California, 94305

and

Timothy W. McLain[*] , Michael A. Goodrich[†]
Brigham Young University, Provo, Utah 84602

This paper presents a path planner for sensing closely-spaced targets from a fixed-wing unmanned air vehicle (UAV) having a specified sensor footprint. The planner is based on the learning real-time A* (LRTA*) search algorithm and produces dynamically feasible paths that accomplish the sensing objectives in the shortest possible distance. A tree of candidate paths that span the area of interest is created by assembling primitive turn and straight sections of a specified step size in a sequential fashion from the starting position of the UAV. An LRTA* search of the tree produces feasible paths any time during its execution and minimum length paths if run to completion. The running time and path-length performance of the search are directly influenced by the operating parameters of the LRTA* algorithm. To improve the running time of the planner, a modified LRTA* search that terminates when there is no improvement in the path for a predefined number of iterations is implemented. The result is a path planner that produces short-distance paths in acceptably short running times.

I. Introduction

PATH planning is an essential activity for mobile autonomous vehicles. For autonomous fixed-wing aircraft, the path-planning problem is particularly difficult because the dynamic limitations of the aircraft, such as minimum turning radius, must be considered during the path-planning process. Aerial sensing of ground based targets adds further difficulty to the problem by requiring the footprint of a downward-looking sensor to pass over a group of targets. The path-planning process must not only consider the aircraft dynamics, but also the movement of the sensor footprint as the aircraft follows a specified path. If the targets are located far apart with respect to the minimum turning radius of the vehicle, then the path-planning approach is straightforward.[1,2] When the targets are close together, however, traditional path-planning approaches are not well suited for producing paths that effectively utilize the vehicle's sensor footprint.

The problem of focus for this work is to develop a path-planning method for sensing a group of closely-spaced targets that fully utilizes the planning flexibility provided by the sensor footprint, while operating within the dynamic constraints of the aircraft. The path planning objective is to minimize the path length required to view all of the targets. This problem is illustrated in Figure 1 where there are three targets. In addressing problems of this nature, three technical challenges must be addressed: coupling between path segments, utilization of the sensor footprint, and determination of the viewing order of the targets. Each of these challenges will be briefly discussed.

A standard approach for planning paths is to plan path segments that are optimal from one point to another. For multiple targets, the approach involves planning an optimal path from the starting point to the first target, from the first target to the second target, and so on, until all targets are visited. If targets are widely spaced relative to the turning

Received 08 December 2005; revision received 06 October 2005; accepted for publication 08 December 2005. Copyright © 2006 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved. Copies of this paper may be made for personal or internal use, on condition that the copier pay the $10.00 per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923; include the code 1542-9423/04 $10.00 in correspondence with the CCC.
[*] Department of Mechanical Engineering, Brigham Young University, Provo, Utah 84602
[†] Department of Computer Science, Brigham Young University, Provo, Utah 84602

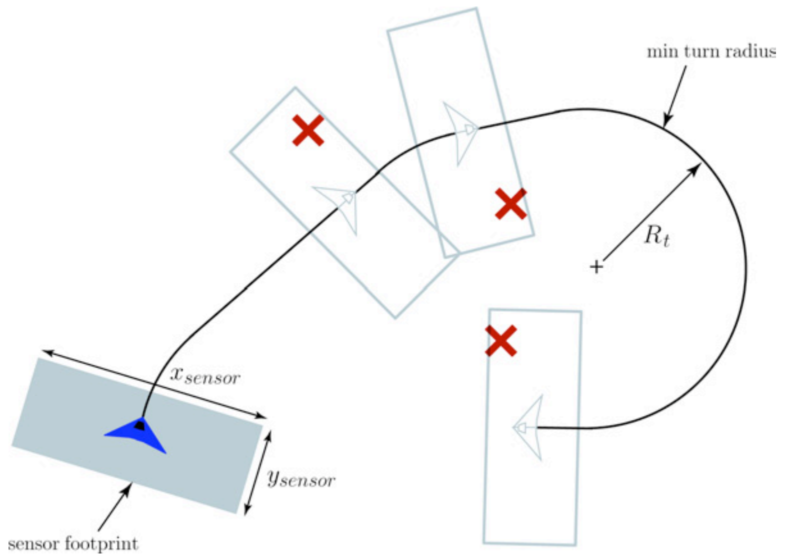


Fig. 1 Schematic of three-target sensing problem.

capabilities of the vehicle, then this approach works well provided that the optimal visit order can be specified. If targets are closely spaced, then coupling between the path segments due to the dynamic limitations of the vehicle can result in a poor outcome for the composite path. The negative effect of this coupling is illustrated in Figure 2(a) where an optimal path is planned to the first target. The heading of the vehicle as it flies over the first target puts it in a poor position to view the second target. Similar coupling exists between the second and third segments of the path. Although segments two and three are also optimal with respect to their initial position and heading and final position, the composite path is clearly suboptimal. Figure 2(b) illustrates the optimal multi-target path. Clearly the path segment from the initial position to the first target is longer than the myopic optimal segment in (a), but at the completion of the first segment, the vehicle is well situated to view the subsequent targets leading to a minimum path length for viewing all the targets.

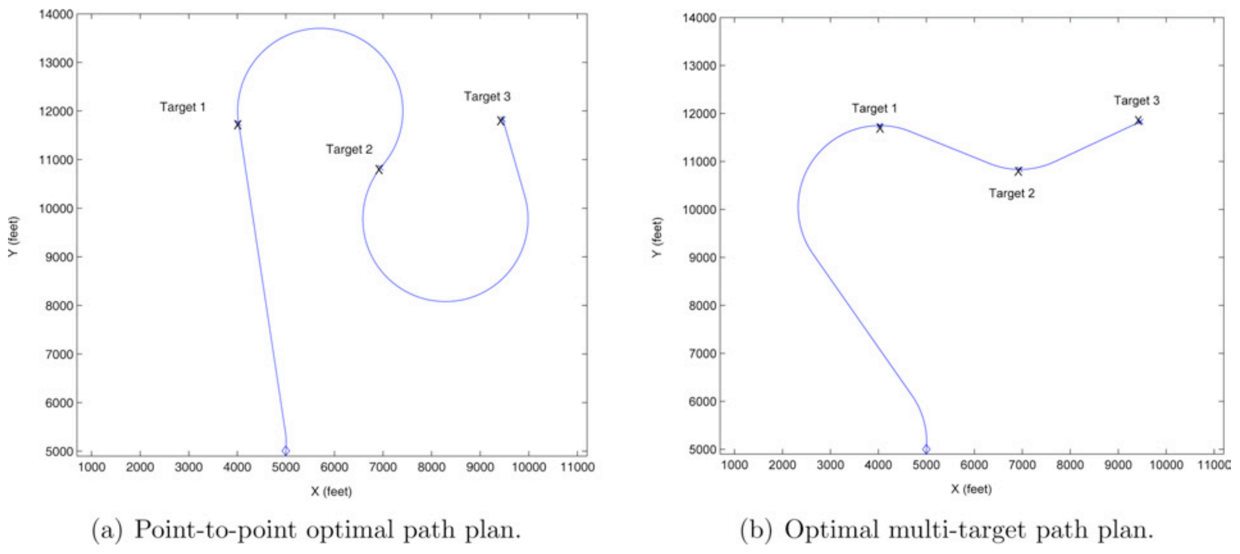


Fig. 2 Coupling between path segments.

Most path planning approaches produce paths that pass directly over the targets of interest. These approaches are unnecessarily restrictive. Many downward looking sensors (e.g., EO video, SAR, LADAR) utilized by UAVs have sensor footprints of significant size. Viewing a target typically does not require flying directly over the top of it, thus the direct fly-over restrictions imposed by most path planners do not utilize the range and flexibility provided by the sensor footprint. The benefit of being able to utilize the sensor footprint when viewing closely spaced targets is clearly demonstrated in Figure 3. While having a sensor with a significant footprint size is beneficial, utilizing it increases the complexity of path planning considerably.

A final challenge when viewing targets in close proximity is to determine the viewing order. The task of determining the viewing order is similar to the traveling salesman problem (TSP),[3] which is known to be *NP* hard. In addition to determining the viewing order, the path planning problems considered here have the added complexity of a limited turn radius and a sensor footprint to exploit. Because of the difficulty of this problem, this work will only consider constant altitude, constant velocity flight with a downward looking sensor.

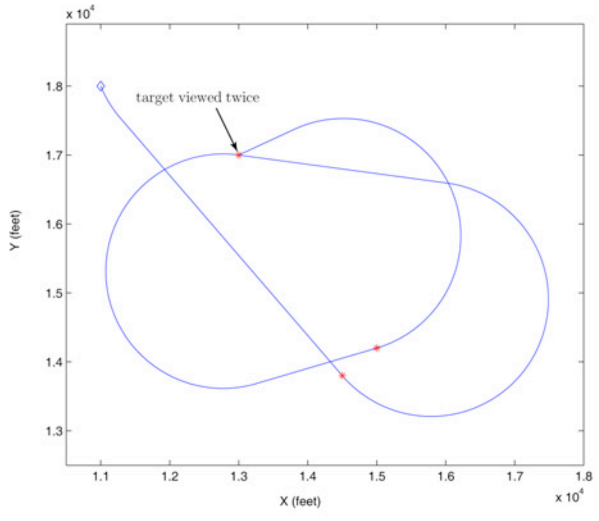
An effective path planning approach for viewing closely-spaced targets from a UAV requires a planner with the following capabilities:

- Plans optimal or near-optimal paths through multiple targets by overcoming the effects of coupling between path segments
- Utilizes fully the sensor footprint to view targets, thus eliminating the need to fly directly over them
- Determines the viewing order of targets yielding minimum-length paths.

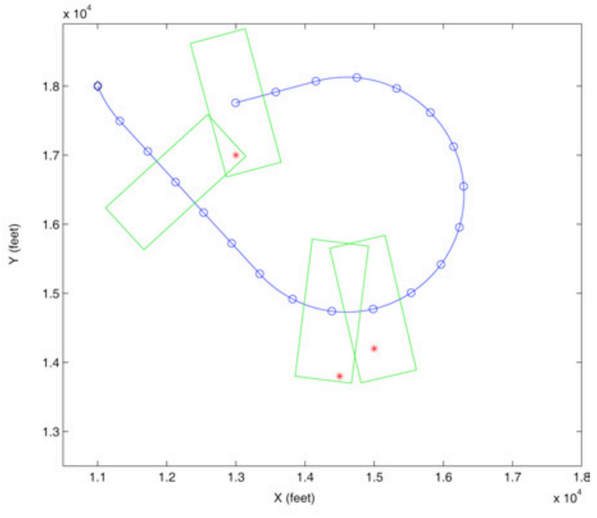
By utilizing a novel implementation of the learning real-time A* search, a path planner has been developed that meets these technical requirements and approximates optimal paths. The path planner requires only a list of targets as input and automatically provides the order and times in which the targets should be visited. The algorithm can be extended to accommodate any sensor-footprint geometry or additional constraints and goals for the path.

Trajectory planning for vehicle systems and mobile robots has been an area of active research for decades. Much of the early work in optimal control centered around the planning of optimal trajectories for aerospace systems.[4] Path planning for robotic systems has been the focus of significant attention resulting in the development of numerous techniques.[5] Most of the methods developed treat the situation where a start state and end state have been defined and the objective is to plan a path between them. This paper treats a different scenario where the end state is unspecified, but the UAV must pass within the vicinity of several targets for viewing.

This work draws ideas from several different types of path-planning approaches. Dubins[2] proves that the minimum-length, curvature-constrained path connecting some initial and final position with specified initial and final headings



(a) Optimal point-to-point segments.



(b) Optimal plan utilizing sensor footprint.

Fig. 3 Path shortening benefit of properly utilized sensor footprint.

will consist of a turn, followed by a straight segment, followed by a turn. Turns are made at the minimum turning radius of the vehicle. The traveling salesman problem for a Dubins' vehicle (i.e., a vehicle constrained to move forward along paths of bounded curvature) has been investigated by Savla et al.,[6] They develop an algorithm that builds on the optimal solution of the TSP to produce a sub-optimal Dubins' TSP tour with bounded worst-case length. Richards et al., have presented a hybrid A*/automaton approach to online UAV path planning that also utilizes Dubins vehicle turn primitives.[7]

Yang and Kapila use the concept of Dubins paths and vector calculus to pose multiple-target path planning as a parameter-optimization problem.[8] Their algorithm maintains the vehicle's dynamic constraints, as well as considering various tactical constraints. The resulting path is optimal, but requires the order in which the targets are to be visited to be specified prior to calculating the path. Caveney and Hedrick treat the closely-spaced target problem addressed in this paper, but pursue a different approach.[9] Their objective was not to plan a continuous path through the targets, but rather to determine target groupings to facilitate efficient viewing of the targets.

Naturally occurring potential fields have motivated the research of potential-field based path-planning methods. These methods can produce good results, but are inherently problematic,[10] and therefore not entirely successful in solving the path-planning problem. McLain and Beard present a potential-force based method for cooperative path planning of UAVs.[11] The path is treated as a chain made of discrete segments which are repulsed by the threats. The segments are also repulsed by each other to smooth the path and make it flyable. Unfortunately, this approach is too slow for real-time path planning.

Frazzoli et al.,[12] develop a real-time randomized path-planning algorithm that maintains the dynamic constraints of the vehicle. The algorithm, however, considers the path to only one target. This approach could be adapted for use in planning a path that visits multiple targets. A related area of path-planning research is that of randomized probabilistic search, also known as probability road mapping (PRM).[13,14] PRM randomly selects configurations from the configuration space, and plans local paths to those configurations. After randomly searching for some time, a road map can be constructed that can be searched for the shortest path to the goal. The underlying idea of PRM is that the probability of finding the optimal path will converge to one as the time spent building the road map goes to infinity. More importantly for real-time applications, PRM methods demonstrate rapid convergence to feasible solutions.

Related to probability road maps are rapidly-exploring random trees (RRTs).[15] By randomly selecting points in the search space, a tree of dynamically feasible nodes can be constructed. The tree is grown until the goal state is reached. By keeping track of the control inputs required to move between nodes, an open-loop control input sequence to move from the start node to the goal node can be found. Lavalle and Kuffner have developed several variations to the RRT algorithm to improve search speed.[16] Saunders et al., have implemented a version of the RRT algorithm to plan 3-D waypoint paths for small UAVs, and have successfully used the algorithm in simulation and flight tests.[17]

The planning of paths for UAVs operating as sensor platforms has been treated by Rysdyk.[18] This work simulates paths that maintain a constant line-of-sight with a ground-based target from a fixed-wing UAV. Camera gimbal control is discussed, but the emphasis is on UAV path planning.

Path-planning techniques using Mixed Integer Linear Programming (MILP) have been developed for aircraft applications.[19–21] MILP methods are capable of producing collision free paths between starting and ending states, but require specialized software for solving the MILP optimization problem. MILP methods are well suited to problems addressing task assignment and path planning simultaneously. Evolutionary algorithms for UAV path planning have been explored by Jia and Vagners.[22] Problems with premature convergence were addressed by creating a framework of parallel evolutionary algorithms in which several populations evolve simultaneously and compete with one another. The real-time heuristic search[23] has been the basis for many path planning and other intelligent-search algorithms.[24,25] There has been a great deal of effort expended in increasing the speed and efficiency of the real-time search.[26–28] The text by Weiss,[29] provides an overview of the literature concerning real-time search and the learning real-time A* algorithm.

II. Discrete-Step Paths

A successful path-planning algorithm should produce a type of path that is not constrained by end points or headings, that utilizes the full capability of the vehicle's sensors, and that satisfies the dynamic constraints on the vehicle. These capabilities can be provided by discrete-step paths, which are built by assembling primitive turn and straight segments to form a flyable path. The choice of which primitive to use at each step is driven not by getting from

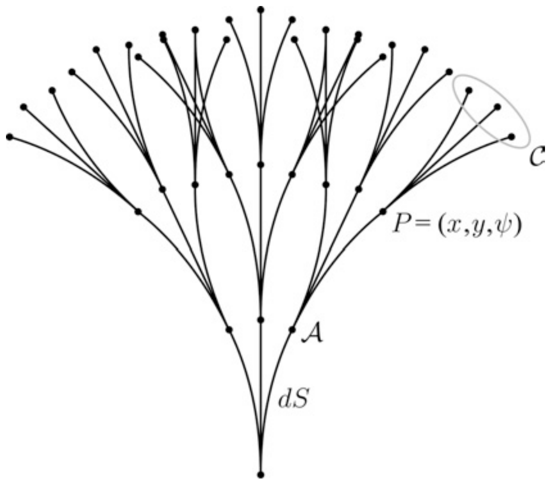


Fig. 4 Primitive turn and straight segments assembled to form a tree of flyable paths.

point A to point B, but instead by meeting a set of specific objectives for the path, such as sensing a group of targets. For this work, each primitive segment in a discrete-step path is of a specified length, dS, and is either a turn, made at the minimum turning radius of the vehicle, or a straight line (thus following the Dubins path concept). Normalizing dS by the minimum turning radius R_t gives the vehicle's maximum heading change at each step: $d\psi = dS/R_t$. Since the step size is constant for each primitive, the number of steps in a path is given by $N = P_n/d\psi$, where P_n is the normalized path length $P_n = P/R_t$. Assembling the left turn, right turn, and straight primitives creates a tree of flyable paths as shown in Figure 4. Thus, the objective for the path planner is to search the path tree for the branch that accomplishes the desired objectives in the shortest distance. Nodes in the path tree have a parent node, $\mathcal{A}$, and a set of child nodes, $\mathcal{C}$. Each node also has a record of the targets that have been sensed by its ancestors, which are denoted by the set $\mathcal{S}$. The set of known targets is denoted as $\mathcal{T}$.

The path tree represented in Figure 4 has a depth of three. It is intuitively clear that shorter step lengths increase the resolution of the path tree and that deeper trees allow paths of greater length, increasing the coverage area of the tree. Given that the number of nodes in a tree grows exponentially with the depth of the tree, there are practical limitations to tree depth imposed by computer memory limits. As step length and tree depth significantly influence the speed of the path planning tree search and quality of the resulting path plan, their role will be investigated further in Section III, which details the tree search. In considering the construction of the path tree, the simplest approach

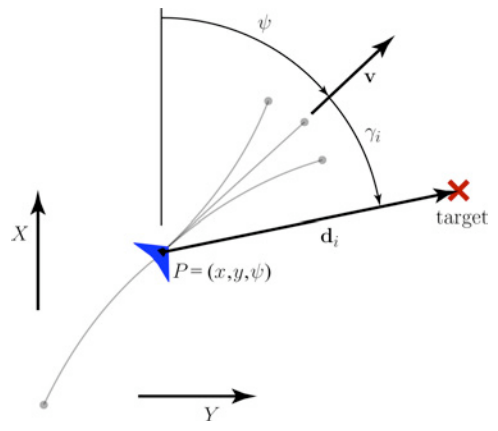


Fig. 5 Notation for the path-planning problem.

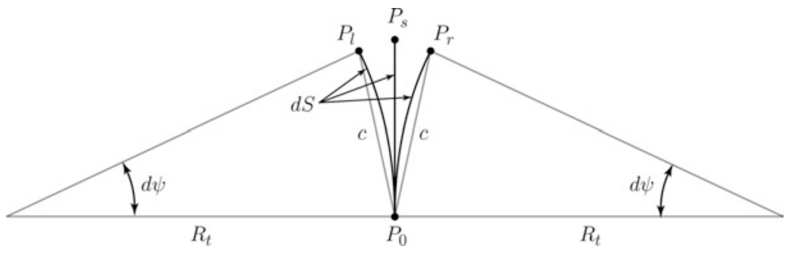


Fig. 6 Geometry of nodes in the path tree.

is to generate a fixed-depth tree that utilizes the existing computer memory resources. For desktop computers used currently, tree depths between 30 and 50 can be implemented feasibly.

The position and attitude of the vehicle are characterized using standard aircraft notation, as shown in Figure 5. The configuration of the vehicle at a node is given by the triplet $P = (x, y, \psi)$, where x and y represent the inertial position, and ψ is the heading of the vehicle measured from the North. The vector from the vehicle to the i^{th} target is denoted as $\mathbf{d}_i$.

The configuration of a child node is determined by calculating the change in heading and position relative to the parent configuration, as shown in Figure 6. The equations for calculating the child configurations, P_l, P_s, P_r, are provided below.

$$c = R_t\sqrt{2(1-\cos(d\psi))}$$

$$P_l = P_0 + \begin{bmatrix} c\sin(\psi_0 - 0.5d\psi) \\ c\cos(\psi_0 - 0.5d\psi) \\ -d\psi \end{bmatrix}$$

$$P_s = P_0 + \begin{bmatrix} dS\sin(\psi_0) \\ dS\cos(\psi_0) \\ 0 \end{bmatrix}$$

$$P_r = P_0 + \begin{bmatrix} c\sin(\psi_0 + 0.5d\psi) \\ c\cos(\psi_0 + 0.5d\psi) \\ d\psi \end{bmatrix}$$

Targets are sensed by the vehicle whenever they are inside the vehicle's sensor footprint. The algorithm presented can accommodate sensor footprints of arbitrary size and shape. For this work, the sensor footprint is rectangular with $x_{sensor} = 1.2R_t$ and $y_{sensor} = 0.5R_t$, and its center is located directly beneath the vehicle[*]. The discrete-step paths described and implemented are limited to two dimensions to reduce computational complexity. The concepts of building discrete-step path trees and searching them, as described subsequently, can be extended to three dimensions in a straightforward manner.

III. LRTA* Tree Search

This work applies the learning real-time A* (LRTA*) algorithm in a novel way to learn which branch of a defined path tree best accomplishes the desired path planning objectives. The LRTA* algorithm is well established, but has typically been applied to path-planning problems in grid based worlds[29]. In general, the LRTA* algorithm may be applied to any type of world: grid, tree, directed graph, or other. The LRTA* algorithm is chosen over the faster A* algorithm because the limiting factor on the performance of the path-planner is the memory space required to store the expanded nodes. By using LRTA*, execution speed is sacrificed to reduce the spatial complexity of the search.

[*] The sensor footprint size relative to the turn radius is based on the capabilities of a specific vehicle known as a Low Cost Autonomous Attack System (LOCAAS).

Algorithm Details

The LRTA* algorithm itself is simple and elegant. Each node has a heuristic estimate, h, of the remaining distance that must be traveled to complete the unfinished objectives. At each step of the search, the current node calculates $f_c = k_c + h_c \; \forall \; c \in \mathcal{C}$, where h_c is the child's heuristic estimate, and k_c is the cost of moving to the respective child. In other words, f_c is the estimated remaining travel distance if a move were to be made to child c. The current node updates its heuristic value with $h = \min_{c \in \mathcal{C}} f_c$, and then moves to the corresponding child. The search continues to move down the tree until either all the objectives are accomplished, or the path becomes longer than the current best path, at which point the search begins again from the root node. At each step of the search, the heuristic value for the current node is updated with a better estimate of the distance to the goal, and after some number of iterations, the updated heuristics converge to the actual path lengths. At this point the search has learned the minimum-length path that accomplishes the desired objectives. In other words, when $h_0^* - h_0 = 0$, where h_0^* is the actual path length from the root node, then the search has found the optimal path. The LRTA* tree search algorithm is outlined in Algorithm 1, where Δh_{total} is the total heuristic change for the current run, $count$ is the length of the current path, $bestcount$ is the length of the best path found thus far, and $allSensed$ indicates whether all the targets have been sensed.

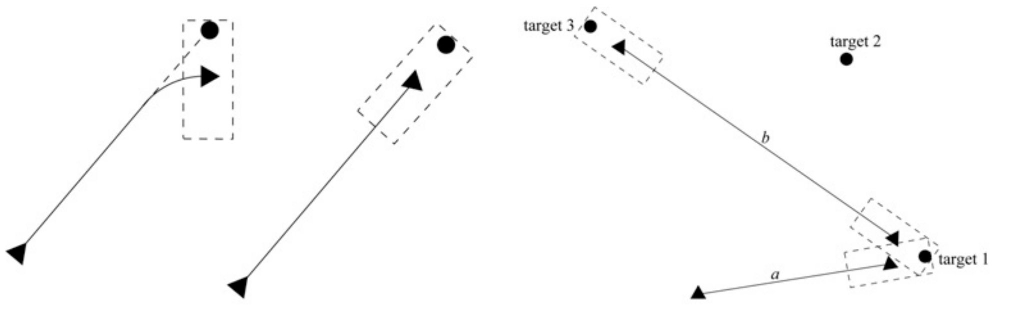
```
Algorithm 1: LRTA* Tree Search
Input: Set of targets $T$, Initial configuration $P_0 = (x_0, y_0, \psi_0)$
Output: End node of the path branch
LRTA($T$, $P_0$)
(1)  while $\Delta h_{total} > 0$
(2)      $P = P_0$
(3)      while $count < bestcount$ & $!allSensed$
(4)          $f_c = h_c + dS \; \forall \; c \in \mathcal{C}$
(5)          $h \leftarrow \min_{c \in \mathcal{C}} f_c$
(6)          $P = \arg\min_{c \in \mathcal{C}} f_c$
(7)          $count = count + 1$
(8)      if $count < bestcount$
(9)          $bestcount = count$
```

It should be noted that the algorithm described above only allows moves to children. It is feasible to move back up the tree to a node's parent and continue the search from there, as in the traditional A* search, but doing so results in large portions of the tree being explored without finding a better path. It is more efficient to continue down a branch until either all the objectives are complete, or the branch is terminated for some other reason, and then start the next search iteration at the root node. Doing so allows the search to explore different parts of the tree to find a better path, instead of staying in just one area.

Initial Heuristic

The primary requirement of the LRTA* search is that $h_i \leq h_i^*$ is always true for any node i. The reason for this requirement is simple: if a node on the optimal path overestimates the distance to the goal, the search may never move to that node and hence never find the optimal path. Heuristics that conservatively estimate the true path length are termed admissible heuristics[29] and the calculation of appropriate initial values for these heuristics is essential to the performance of the LRTA* search. It would be permissible to initialize the heuristics to zero, but the search would require a long time to learn the optimal path. Therefore, the initial heuristic estimates should be as high as possible while still guaranteeing that they are admissible.

Heuristics in the LRTA* tree-search problem are estimates of the distance that must be traveled from the current node to complete the remaining tasks. The simplest method for calculating heuristics is to use the distance to the furthest target as the initial heuristic value since we know the vehicle must travel at least that far to complete its objectives. As shown in the left of Figure 7(a), however, the vehicle can swing the sensor around to the target by making a turn, and thus the vehicle need not go completely to the target. Since it is impossible to determine when



(a) Distance to furthest remaining objective. (b) Distance to nearest objective plus distance between nearest and furthest.

Fig. 7 Calculation of initial heuristics.

this can be done, the approximation illustrated in the right of the figure is used and the initial heuristic value is calculated as

$$h = \max_i \|\mathbf{d}_i\| - \frac{y_{sensor}}{2}.$$

These initial heuristic values are guaranteed to be admissible, but are not necessarily very close to the actual path length, and therefore the search may converge slowly.

An alternative initial heuristic value is to find the distance to the target closest to the vehicle, and then add the distance to the target furthest from this first target, as is illustrated in Figure 7(b). To guarantee that the heuristic is admissible, one-half the sensor width is subtracted from the distance to the nearest target, and the full sensor width is subtracted from the second distance. The discrete nature of the path tree must also be accounted for, and so, to be conservative, the step size is subtracted from the first segment and twice the step size is subtracted from the second segment. The resulting heuristic is

$$h = \min_i \|\mathbf{d}_i\| + \max_j \|\mathbf{t}_j\| - \frac{3}{2} y_{sensor} - 3\,dS$$

where $\mathbf{t}_j$ are the vectors from the nearest target to the other targets. These initial heuristic values are guaranteed to be admissible, and they provide better path-length estimates than the previous method. This method is used by the LRTA* algorithm.

Terminating Conditions

The method used to terminate the tree search directly affects the speed and path-length performance of the tree search. On one extreme the search runs until the heuristic change is zero, thus guaranteeing that the minimum-length path has been found, but possibly requiring a significant amount of running time. On the other extreme the search runs until a maximum running time is reached, thus guaranteeing termination by a certain time, but with the resulting path possibly being far from optimal. Neither of these extremes is very attractive for the path-planning problem. These two extremes, however, may be combined to create a terminating condition that encourages continual improvement of the path length, while providing the ability to terminate the algorithm if no improvement is being made. This condition is to stop the search after there has been no improvement in the path length for some specified number of iterations. In other words, the search continues as long as the path length is improving, with the trade off between speed and path-length performance being controlled by the number of non-improving iterations that must lapse before termination. The more iterations that are required, the better the resulting path lengths may be, but with increased running time. This terminating condition is successful because the optimal path is typically found fairly quickly and the majority of the search time is spent confirming that it is indeed the optimum. The tree search algorithm with this terminating condition is referred to as the Non-Improving LRTA*, or NILRTA* algorithm. The NILRTA* algorithm terminates when there is no improvement in path length, whereas the LRTA* algorithm terminates when there is no improvement in the heuristic estimate. Figure 8 shows sample paths generated by the LRTA* and NILRTA* algorithms for the same test conditions.

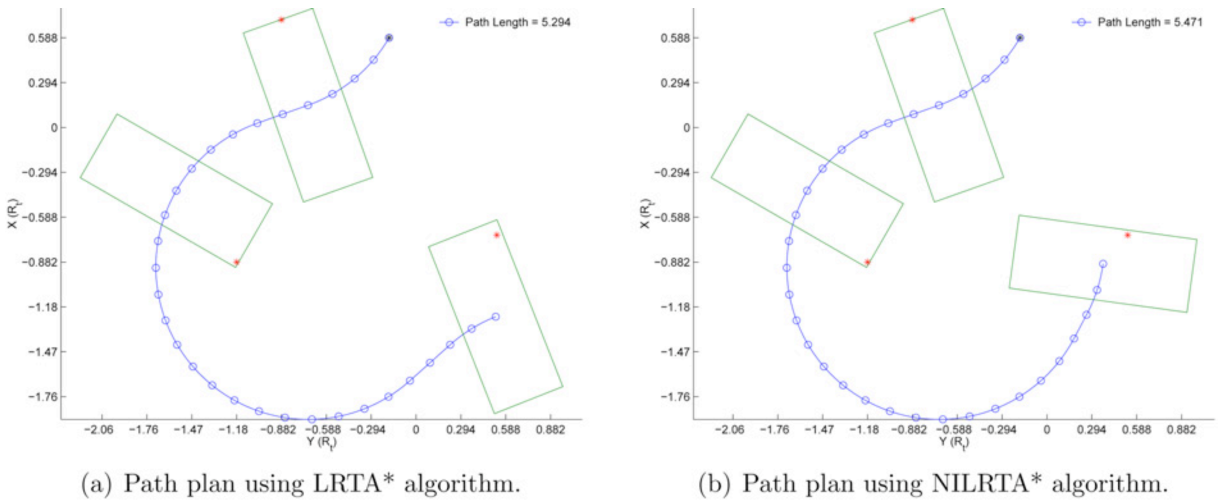


Fig. 8 Sample paths generated by the LRTA* and NILRTA* algorithms.

Branch Terminals

Memory management and pruning of the path tree are essential to good performance of the LRTA* tree search. When the currently-explored branch becomes longer than the present best path, the current node is terminated by deleting its children and setting its heuristic value to infinity and the search is restarted at the root node. If a node's children all have heuristic values of infinity, then the node is a dead-end and it can be terminated similarly. Also, since the heuristic value of a node is always less than or equal to the actual path length, the heuristic can be used to cull nodes from the tree. If at a given node, $h + count\ dS > bestcount\ dS$, where $count$ is the current depth in the tree and $bestcount$ is the number of nodes in the currently known best path, then all paths extending from that node are longer than the best path. Therefore the node and all its children can be safely terminated.

Path Tree Size

A significant problem with searching the path tree is the size of the tree. As the step size decreases, the size of the tree increases very quickly, which makes exploring the tree a lengthy process. The full path tree has a size of 3^d, where d is the number of levels in the tree. This means that for a tree with 30 levels, there are approximately 3.1×10^{14} nodes. Although the search will not need to explore all the nodes, there will be a large number that must be evaluated, meaning the search will be slow, and require a lot of memory space to store the visited nodes.

One method for decreasing the size of the path tree is to prune out branches that result in paths that frequently switch directions. For example, if a right turn has been made, there is no point in making a left turn on the next step if the step size is reasonably small. Therefore, after a turn has been made, the only choices are to go straight or to turn in the same direction again. This look-back may be extended to consider the last n moves, thus effectively reducing the size and complexity of the tree. An example tree structure with a two-step look-back is shown in Figure 9. For a tree with thirty levels, the size of the tree is 5.42×10^{10} nodes, which is less than 0.02 percent of the size of the full path tree. Pruning the tree in this manner is particularly helpful when using small step sizes. For larger step sizes, it is permissible to allow the turn direction to change between segments, and thus the full path-tree structure may be used.

The most effective means of limiting the size of the path tree is to limit the depth of the tree before the search begins. This may be done by setting some maximum depth that is used in every problem. A more efficient approach, however, is to precede the LRTA* tree search with some other path-planning method that provides a flyable path that accomplishes the desired objectives, but not necessarily in an optimal manner. Based on the length of this initial feasible path and the step size, a suitable tree depth for the particular scenario can be established. In addition to providing a maximum tree depth that is suited to the particular problem, the initial feasible path provides the ability to terminate the tree search at any time and still have a feasible path that accomplishes the desired objectives.

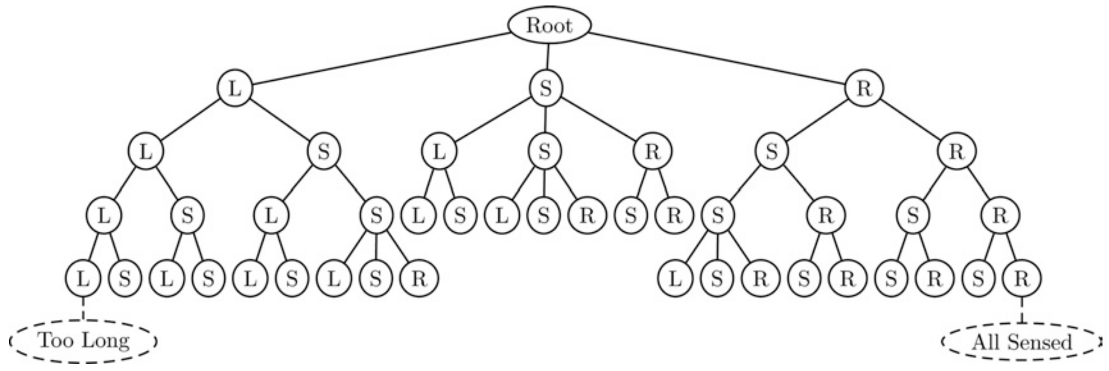


Fig. 9 Two-step look-back tree structure. Previous two moves limit possible next moves.

For this research, the LRTA* tree search is initiated using the results of a single-source potential-field path planning algorithm[1]. Using this method, the target closest to the vehicle is chosen as the source and the potential field algorithm plans a path to that target. Once the first target is reached, the next nearest target is selected as the source. This procedure continues until all targets have been visited. Although the resulting path is suboptimal, this method reliably produces a feasible initial path for the LRTA* algorithm. Starting with this initial path, the LRTA* algorithm finds the optimal minimum-length path through the targets. In finding the optimal path, the LRTA* algorithm determines the best tour order, which is often different from the order of the path used to initialize the LRTA* search.

One advantage of the LRTA* algorithm is that additional goals and constraints can easily be imposed on the path. Such constraints may include viewing a target from a specified heading or range of headings, or avoiding a known obstacle or no-fly zone. Implementing additional goals and constraints is simply a matter of identifying and pruning infeasible branches during the LRTA* tree search. Avoidance of unknown obstacles that are sensed during flight would require augmentation of the LRTA* planner with reactive capabilities. Simple Dubins paths could form the basis for real-time avoidance maneuvers that would momentarily take the vehicle off the LRTA* path to avoid the obstacle.

Step Size Selection

The LRTA* tree search is guaranteed to find the minimum-length path from the discrete-step path tree. This path is not necessarily the globally optimal path, but a discrete approximation to the optimal path. In theory, if the step size in the LRTA* tree search were decreased to be infinitesimally small, then the true global optimum would be found. Unfortunately this is not possible in practice because the tree size would be too large to search effectively. The key to making the LRTA* tree search work well, is choosing a step size that is small enough to best approximate the global optimum, but without making the tree too large to search quickly. Results from this work show that a trade off must be made between speed and path-length performance when selecting a step size, and that larger step sizes greatly improve the running time of the algorithm with only a slight increase in the resulting path lengths.

In its present implementation, the LRTA* path planner operates with a fixed step size, dS. Based on an initial feasible path length and this fixed step size, a maximum tree depth is determined. This approach minimizes the computational complexity by limiting the size of the tree. An alternative approach that would provide paths of increased resolution and possible shorter lengths would be to fix the depth of the tree at the maximum value within the capability of the computer. From this maximum depth and the initial feasible path length, a minimum feasible step size could be determined for building the path tree. Approaches to adapt the step size based on the density of the targets in close proximity could allow larger areas to be searched with greater efficiency.

IV. Testing and Comparison

The LRTA* and NILRTA* algorithms were tested on a set of 2000 target scenarios generated from a $2R_t$ by $2R_t$ world. Each scenario included three randomly-selected targets and a random initial position and heading for the vehicle. For heading changes of 0.5 radians and below, a simplified path tree was used that did not include segments composed of alternating left and right turns. Tests were conducted using an Athlon XP 1500+ (1.33 GHz) processor with 512 Mb of RAM running the Linux operating system. The running time for the algorithms was limited to one minute and data from early-terminated runs were used in the analysis. The terminating condition for the NILRTA* algorithm was 10,000 iterations of no improvement.

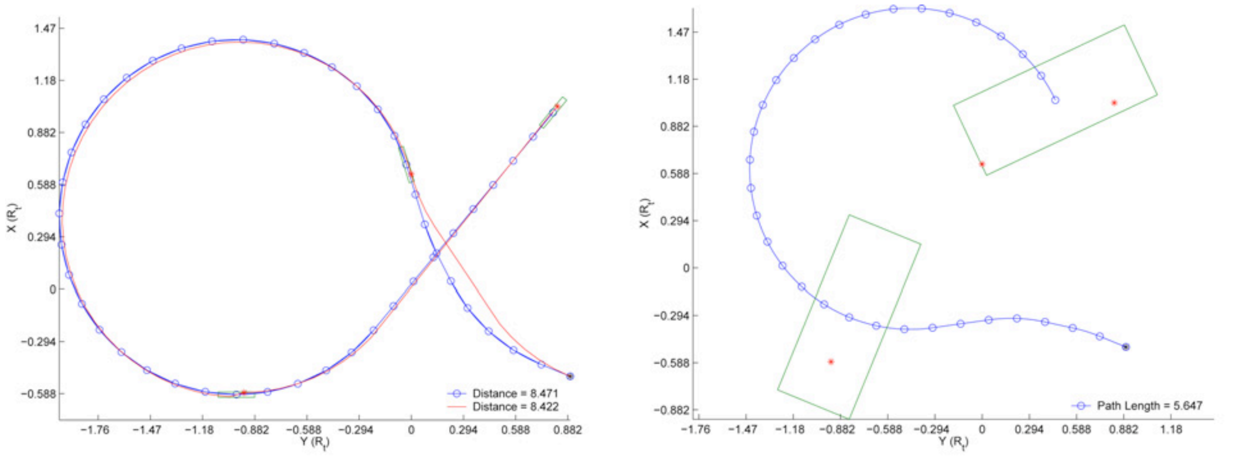
The step size used for the discrete-step path tree directly influences the running time and path-length performance of the LRTA* tree-search algorithms. It can be shown that increasing the step size significantly reduces the running time of the algorithm with only a moderate increase in the resulting path lengths. Furthermore, algorithm testing has demonstrated that using step sizes below 0.2 radians does not significantly decrease the path lengths produced by the algorithm. The following tests use step sizes between 0.2 and 0.65 radians, in 0.05 radian increments.

LRTA* Tree-Search Validation

To demonstrate the advantages of the LRTA* tree search, a comparison was made between the path generated by the LRTA* algorithm and the path found through a brute-force global search based on Dubins paths. The size of the sensor footprint for the LRTA* algorithm was reduced to approximate the vehicle passing through the target points. Figure 10(a) shows that the LRTA* tree search produces approximately the same minimum-length path as the brute-force global search, with the LRTA* path being only 0.5 percent longer. Resetting the sensor to its original dimensions produced the path shown in Figure 10(b). This path is 33 percent shorter than the path that is constrained to pass directly over the targets and demonstrates the ability of the LRTA* tree-search algorithm to both utilize the full sensor footprint and to learn the viewing order of the targets.

Path Length

The path-length performance for the two algorithms at the various step sizes is presented in Figure 11. The mean path length increases approximately 19 percent, over the range of step sizes, indicating that smaller step sizes are better. The increase in the mean path length between 0.2 and 0.4 radians, however, is only 6.8 percent, which demonstrates the diminishing utility of using smaller step sizes. The figure also shows that the mean path-length



(a) With pin-point sensor footprint, LRTA* closely approximates optimal path obtained by brute-force global search.

(b) LRTA* utilizes large sensor footprint to produce a 33 percent shorter path.

Fig. 10 LRTA* search paths for pin-point and large sensor footprints.

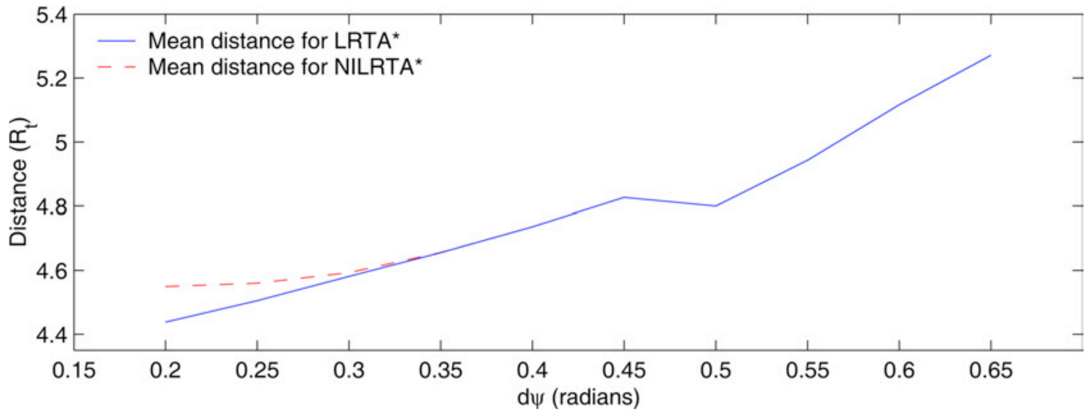


Fig. 11 Mean path lengths for the two algorithms at various step sizes.

performance of the NILRTA* algorithm is comparable to that of the LRTA* algorithm. At $d\psi = 0.2$ radians, the mean NILRTA* path lengths are 3.8 percent longer than those for the LRTA* algorithm. For step sizes of 0.35 radians and above, the path lengths resulting from two algorithms are similar. At a step size of 0.5 radians, the algorithms switch to searching the full path tree instead of the simplified path tree. The path-length performance improves slightly, but does not provide a significant benefit for increasingly larger step sizes.

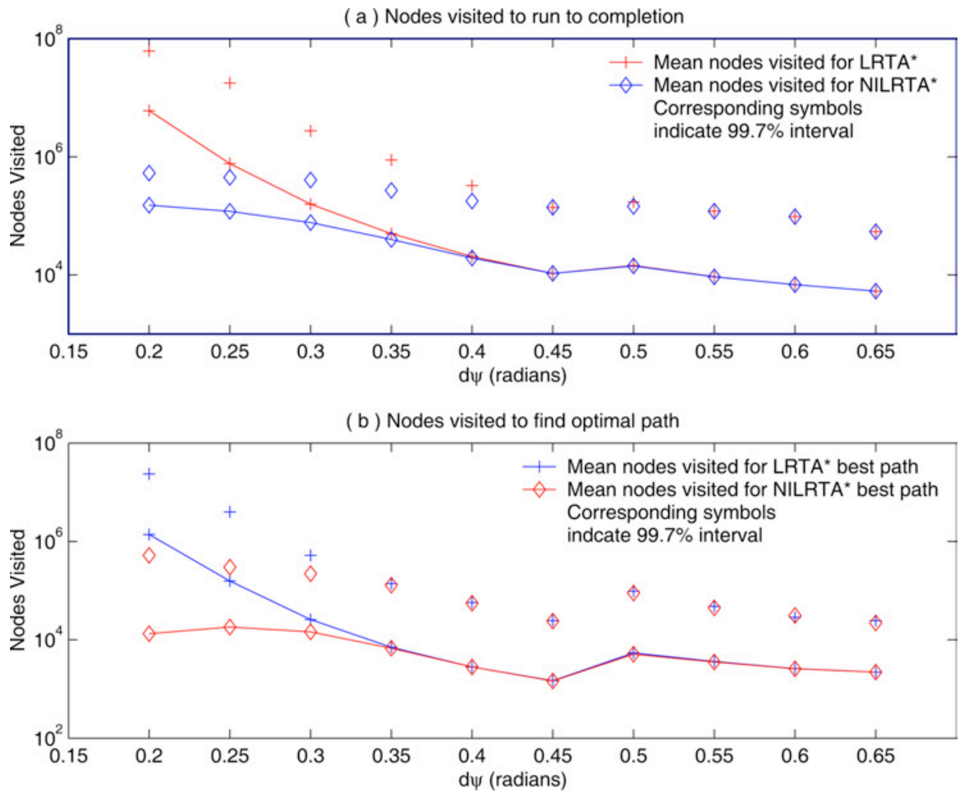


Fig. 12 LRTA* and NILRTA* running time comparison for increasing step sizes.

Running Times

To present running-time data in a platform-independent fashion, running times for the algorithms are specified by the number of nodes that are visited by the search. This is not the same as the number of nodes from the path tree that are expanded, but is the total number of moves made from node to node during the tree search. To provide a meaningful reference point, running times (in seconds) for the Athlon-based system are listed in parentheses. Figure 12 presents the running-time results for the LRTA* and NILRTA* algorithms, with the average total running times (in terms of nodes visited) for both algorithms at the various step sizes shown in plot (a). At $d\psi = 0.2$ radians, the average running time of the LRTA* algorithm is about 8,000,000 nodes (10 s), and about 150,000 nodes for the NILRTA* algorithm (0.3 s). At a step size of 0.3 radians, the mean running time for the LRTA* algorithm drops to about 200,000 nodes (0.5 s), which is a 98 percent decrease. For the The NILRTA* algorithm's running time drops by 47 percent to about 80,000 nodes (0.15 s). As the step size continues to increase, the mean running times decrease further, leveling off at about 7,000 nodes (0.015 s) for both algorithms. For step sizes of 0.4 radians and above, the two algorithms perform similarly.

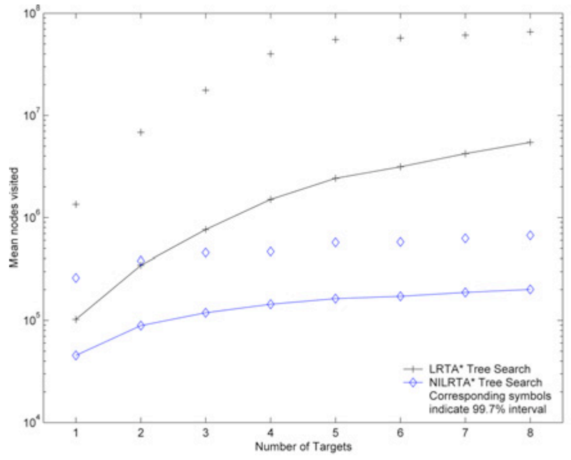
The results shown in plot (b) of Figure 12 are the mean running times at which the best paths are found, meaning that any additional running time is spent confirming the optimum path. For the LRTA* algorithm, roughly 85 percent of the total running time is spent confirming the optimal path. The NILRTA* spends about 93 percent of its total running time processing the 10,000 iterations of no improvement. Thus, if either algorithm is terminated prematurely, the likelihood of having found the best path is high.

The path-length and running-time results show that increasing the step size improves the running time without significantly decreasing the path-length performance. The test results also show that the NILRTA* algorithm runs significantly faster than the LRTA* algorithm, and has comparable path-length performance. In general, these results show that there is a trade off to be made between path-length performance and speed: shorter, highly optimized paths require more time to compute. Also, using the full path tree for larger step sizes does not significantly improve the path-length performance of the algorithms.

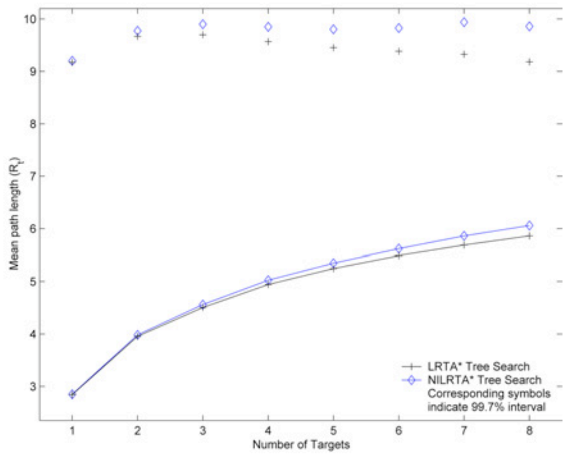
Differing Numbers of Targets

The results presented thus far were all gathered using scenarios with three targets. To test the algorithms' performance for different numbers of targets, two-thousand random target scenarios are used for each number of targets between one and eight, with a step size for the tests of 0.25 radians.

The running-time results in Figure 13(a) show that the running times for both the LRTA* and NILRTA* algorithms increase with the number of targets. For eight targets, the LRTA* algorithm visits an average of 5.5 million nodes (70 s), compared to the NILRTA* algorithm which visits an average 200,000 nodes (0.5 s). The NILRTA* algorithm



(a) Running time versus numbers of targets.



(b) Path length versus numbers of targets.

Fig. 13 The effect of increasing target numbers on running time and path length.

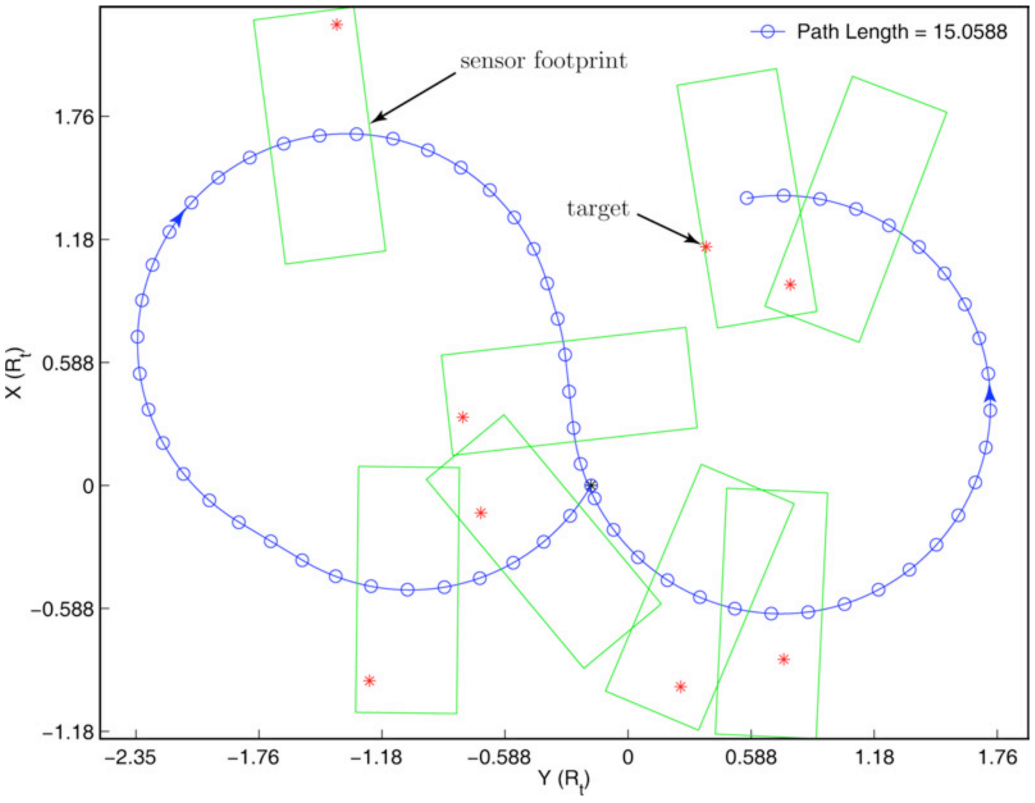


Fig. 14 Path produced by the LRTA* algorithm for sensing eight targets.

is consistently faster than the LRTA* algorithm for any number of targets, and as shown in Figure 13(b), has path length performance comparable to the LRTA* algorithm. Figure 14 shows an example of a path generated by the LRTA* algorithm for an eight-target scenario. A video of simulated UAV flight over the eight targets can be viewed by clicking here. The simulation models the full six degree-of-freedom dynamics of a small UAV and utilizes a path following approach designed specifically for small UAVs.

V. Conclusions

This work presents the LRTA* and NILRTA* tree-search algorithms, which find the branch from a discrete-step path tree that best accomplishes a set of desired objectives. The LRTA* algorithm is guaranteed to produce the minimum-length path, but with longer running times. The NILRTA* algorithm successfully trades off some path-length performance for much faster running times. The LRTA* and NILRTA* algorithms plan optimal or near-optimal paths through multiple targets by overcoming the effects of coupling between path segments. They fully utilize the sensor footprint to view targets and eliminate the need to fly directly over them. They also determine the optimal viewing order as part of the planning process. In this way, both algorithms provide viable solutions to the multiple, closely-spaced-target sensing problem, along with the flexibility to incorporate a variety of goals and constraints.

VI. Acknowledgments

The authors would like to thank the AFOSR for their sponsorship under grant number F49620-01-1-0091. Further appreciation is extended to Phil Chandler and Steve Rasmussen at the AFRL Air Vehicles Directorate for helpful discussions and to Derek Nelson at BYU for his assistance with the flight simulation.

References

[1]Howlett, J. K., *Path Planning for Sensing Multiple Targets from an Aircraft*, Master's thesis, Brigham Young University, Provo, Utah, 2002.

[2]Dubins, L., "On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents," *American J. of Math*, Vol. 79, 1957, pp. 497–516.

[3]Lawler, E., Lenstra, J., Kan, A. R., and Shmoys, D., editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, 1985.

[4]Bryson, A. and Ho, Y.-C., *Applied Optimal Control*, Hemisphere Publishing Corporation, 1975.

[5]Latombe, J.-C., *Robot Motion Planning*, Kluwer Academic Publishers, 1991.

[6]Savla, K., Frazzoli, E., and Bullo, F., "On the point-to-point and traveling salesperson problems for Dubins' vehicle," *Proc. of American Control Conf.*, 2005, pp. 786–791.

[7]Richards, N., Sharma, M., and Ward, D., "A hybrid/A* automaton approach to on-line path planning with obstacle avoidance," *Proc. AIAA 1st Intelligent Systems Technical Conference*, September 2004, AIAA-2004-6229.

[8]Yang, G. and Kapila, V., "Optimal path planning for unmanned air vehicles with kinematic and tactical constraints," *Proc. IEEE Conf. on Dec. and Control*, December 2002, pp. 1301–1306.

[9]Caveney, D. and Hedrick, J., "Path planning for targets in close proximity with a bounded turn-rate aircraft," *Proc. AIAA Guidance, Navigation, and Control Conference*, August 2005, AIAA-2005-6185.

[10]Koren, Y. and Borenstein, J., "Potential field methods and their inherent limitations for mobile robot navigation," *Proc. IEEE Conf. on Robotics and Automation*, Sacramento, CA, April 1991, pp. 1398–1404.

[11]McLain, T. W. and Beard, R. W., "Trajectory planning for coordinated rendezvous of unmanned air vehicles," *Proc. AIAA Guidance, Navigation, and Control Conference*, Denver, CO, August 2000, AIAA-2000-4369.

[12]Frazzoli, E., Dahleh, M. A., and Feron, E., "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance, Control, and Dynamics*, Vol. 25, No. 1, January-February 2002.

[13]Kavraki, L. E., Švestka, P., Latombe, J.-C., and Overmars, M. H., "Probabilistic roadmaps for path planning high-dimensional configuration spaces," *IEEE Trans. on Robotics and Automation*, Vol. 12, No. 4, August 1996.

[14]Kavraki, L. E., Kolountzakis, M. N., and Latombe, J.-C., "Analysis of probabilistic roadmaps for path planning," *IEEE Trans. on Robotics and Automation*, Vol. 14, No. 1, Feburary 1998.

[15]LaValle, S. M., "Rapidly-exploring random trees: A new tool for path planning," TR 98-11, Computer Science Dept., Iowa State University.

[16]LaValle, S. M. and Kuffner, J. J., "Randomized kinodynamic planning, *International Journal of Robotics Research*," Vol. 20, No. 5, May 2001, pp. 378–400.

[17]Saunders, J., Call, B., Curtis, A., Beard, R., and McLain, T., "Static and dynamic obstacle avoidance for miniature air vehicles," *Proc. AIAA InfotechAerospace*, September 2005, AIAA 2005-6950.

[18]Rysdyk, R., "UAV path following for constant line-of-sight," *Proc. 2nd AIAA Unmanned Unlimited Conference*, September 2003, AIAA-2003-6626.

[19]Schouwenaars, T., Moor, B. D., Feron, E., and How, J., "Mixed integer programming for multi-vehicle path planning," *Proc. European Control Conference*, 2001, pp. 2603–2608.

[20]Richards, A., Bellingham, J., Tillerson, M., and How, J., "Coordination and control of multiple UAVs," *Proc. AIAA Guidance, Navigation, and Control Conference*, 2002, AIAA-2002-4588.

[21]Schouwenaars, T., Mettler, B., Feron, E., and How, J., "Hybrid model for trajectory planning of agile autonomous vehicles," *Journal of Aerospace Computing, Information, and Communication*, Vol. 1, No. 12, December 2004.

[22]Jia, D. and Vagners, J., "Parallel evolutionary algorithms for UAV path planning," *Proc. AIAA 1st Intelligent Systems Technical Conference*, September 2004, AIAA-2004-6230.

[23]Korf, R. E., "Real-time heurisitic search," *Artificial Intelligence*, Vol. 42, No. 2-3, 1990, pp. 189–211.

[24]Barto, A. G., Bradtke, S. J., and Singh, S. P., "Learning to act using real-time dynamic programming," *Artificial Intelligence*, Vol. 72, No. 1, 1995, pp. 81–138.

[25]Ishida, T., "Real-time search for autonomous agents and multiagent systems," *Autonomous Agents and Multi-Agent Systems*, Vol. 1, No. 2, 1998, pp. 139–167.

[26]Edelkamp, S. and Eckerle, J., "New strategies in real-time heuristic search," *AAAI-97 Workshop on On-Line Search*, AAAI Press, 1997, pp. 30–35.

[27]Furcy, D. and Koenig, S., "Speeding up the convergence of real-time search," *Proc. of the Nat. Conf. on Artifical Intelligence*, 2000, pp. 891–897.

[28]Ishida, T. and Shimbo, M., "Improving the learning efficiencies of realtime search," *Proc. of AAAI-96*, 1996, pp. 305–310.

[29]Weiss, G., editor, *Multiagent Systems*, chap. 4, The MIT Press, 2000, pp. 165–199.