Faculty Publications

2003-8

Learning Real-Time A* Path Planner for Sensing Closely-Spaced Targets from an Aircraft

Jason K. Howlett

Brigham Young University - Provo

Michael A. Goodrich

Brigham Young University - Provo, mike@cs.byu.edu

Timothy W. McLain

Brigham Young University - Provo, mclain@byu.edu

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

Part of the [Computer Sciences Commons](#), and the [Mechanical Engineering Commons](#)

Original Publication Citation

Jason Howlett, Michael A. Goodrich, and Tim McLain. "Learning Real-Time A* Path Planner for Sensing Closely-Spaced Targets from an Aircraft". AIAA Guidance, Navigation, and Control Conference and Exhibit, (August 14, 2003). DOI: 10.2514/6.2003-5338

BYU ScholarsArchive Citation

Howlett, Jason K.; Goodrich, Michael A.; and McLain, Timothy W., "Learning Real-Time A* Path Planner for Sensing Closely-Spaced Targets from an Aircraft" (2003). *Faculty Publications*. 1512.
<https://scholarsarchive.byu.edu/facpub/1512>

This Peer-Reviewed Article is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Faculty Publications by an authorized administrator of BYU ScholarsArchive. For more information, please contact ellen_amatangelo@byu.edu.

Learning Real-Time A* Path Planner for Sensing Closely-Spaced Targets From an Aircraft

Jason K. Howlett¹ Michael A. Goodrich² Timothy W. McLain¹

¹Department of Mechanical Engineering, Brigham Young University, Provo, Utah 84602

²Department of Computer Science, Brigham Young University, Provo, Utah 84602

August 14, 2003

Abstract

This work develops an any-time path planner, based on the learning real-time A* (LRTA*) search, for generating flyable paths that allow an aircraft with a specified sensor footprint to sense a group of closely-spaced targets. The LRTA* algorithm searches a tree of flyable paths for the branch that accomplishes the desired objectives in the shortest distance. The tree of paths is created by assembling primitive turn and straight sections of a specified step size. The operating parameters for the LRTA* search directly influence the running time and path-length performance of the search. A modified LRTA* search is presented that terminates when there has been no improvement in the path for some number of iterations, resulting in a path planner that provides short-distance paths in short running times.

1 Introduction

Path planning is an essential activity for mobile autonomous vehicles. For autonomous aircraft, the path-planning problem is particularly difficult because the dynamic limitations of the aircraft

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-space targets that fully utilizes the planning flexibility provided by the sensor footprint, while maintaining the dynamic constraints of the aircraft. This problem is illustrated in Figure 1 where there are three targets. By utilizing a novel implementation of the learning real-time A* search, a path planner that meets these constraints and produces near-optimal paths has been developed. The path planner requires only a list of targets as input and automatically provides the order and times in which the targets are visited. The algorithm can be extended to accommodate any sensor-footprint geometry or additional constraints and goals for the path.

This work draws ideas from several different types of path-planning approaches. In [2], Dubins proves that the minimum-length, curvature-constrained path connecting some initial and final position with specified initial and final headings 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. In [3], Yang and Kapila use the concept of Dubins paths and vector calculus to pose multiple-target path planning as a parameter-optimization problem. Their algorithm maintains the vehicle’s dynamic constraints, as well as considering various tactical constraints. The resulting path is optimal, but,

Figure 1: Schematic of three-target sensing problem.

as with other path-planning approaches, the order in which the targets are to be visited must be specified prior to calculating the path.

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 [4], 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 in [5]. 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.

In [6] Frazzoli, et al. 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 may 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) [7, 8]. 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 to 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.

Path-planning techniques using Mixed Integer Linear Programming (MILP) have been developed in [9, 10]. 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. The real-time heuristic search presented in [11] has been the basis for many path planning and other intelligent-search algorithms (see e.g., [12, 13]. There has been a great deal of effort expended in increasing the speed and efficiency of the real-time search [14, 15, 16]. In [17], an overview of the literature concerning real-time search and the learning real-time A* algorithm is given.

2 Discrete-Step Paths

To develop a successful path-planning algorithm, a type of path is needed that is not constrained by end points or headings, that utilizes the full capability of the vehicle’s sensor to sense targets, and that maintains the dynamic constraints of 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 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. Normalizing

Figure 2: Primitive turn and straight path segments are assembled to form a tree of flyable paths.

dS by the minimum turning radius R_t gives the vehicle's maximum heading change at each step: $d\psi = \frac{dS}{R_t}$. Since the step size is constant for each primitive, the number of steps in a path is given by $N = \frac{P_n}{d\psi}$, where P_n is the normalized path length $P_n = \frac{P}{R_t}$. Assembling the left turn, right turn, and straight primitives creates a tree of flyable paths as shown in Figure 2. Thus, the objective for the path planner is to search this path tree for the branch that accomplishes the desired objectives in the shortest distance.

The notation and geometry for nodes in the discrete-step path tree are shown in Figure 3. The configuration of the vehicle at the 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 $\mathbf{d}_i$. 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 configuration of a child node is determined by calculating the change in heading and position relative to the parent configuration as shown in Figure 4. The equations for calculating the child configurations, P_l , P_s , P_r , are provided below.

Figure 3: Notation and geometry for the path-planning problem.

$$\begin{aligned}
 c &= R_t \sqrt{2(1 - \cos(d\psi))} \\
 P_l &= P_0 + \begin{bmatrix} c \sin(\psi_0 - 0.5d\psi) \\ c \cos(\psi_0 - 0.5d\psi) \\ -d\psi \end{bmatrix} \\
 P_s &= P_0 + \begin{bmatrix} dS \sin(\psi_0) \\ dS \cos(\psi_0) \\ 0 \end{bmatrix} \\
 P_r &= P_0 + \begin{bmatrix} c \sin(\psi_0 + 0.5d\psi) \\ c \cos(\psi_0 + 0.5d\psi) \\ d\psi \end{bmatrix}
 \end{aligned}$$

Targets are sensed by the vehicle whenever they are inside the vehicle's sensor footprint. Determining whether a target is being sensed is independent of any path-planning algorithm, thus

Figure 4: Geometry of nodes in the path tree.

any sensor footprint shape may be substituted into the algorithm presented below. For this work, the sensor footprint is rectangular with $x_{sensor} = 1.18R_t$ and $y_{sensor} = 0.48R_t$, and located directly beneath the vehicle.

3 LRTA* Tree Search

This work applies the learning real-time A* 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 [17]. In general, however, 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.

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 $h = \min_c f_c$, 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 currently 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 $h^* - h = 0$, where h^* is the actual path length, 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.

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, 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 and find a better path, instead of staying in just one area.

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 BranchLRTA(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) $h \leftarrow \min_c f_c$
- (6) $P = \arg \min_c f_c$
- (7) $count = count + 1$
- (8) **if** $count < bestcount$
- (9) $bestcount = count$

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 [17] and the calculation of the 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 heuristics 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 5(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 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 5(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

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

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 is running the search 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 is running the search 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

Figure 5: (a) Initial heuristic value using the distance to the furthest remaining objective. (b) Initial heuristic value using the distance to the nearest objective plus the distance between the nearest and the furthest.

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. Figure 6 shows sample paths generated by the LRTA* and NILRTA* algorithms for the same test conditions.

Figure 6: Sample paths generated by the LRTA* (a) and NILRTA* (b) algorithms.

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 an infinitesimally small amount, then the true global optimum would indeed be found. Unfortunately this is not possible in practice because the tree size would be too large to search effectively. The key, then, 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.

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 presently best path, the

Figure 7: Two-step look-back tree structure. The previous two moves limit the possible next moves.

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.

Complexity

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^n , where *n* is the number of levels in the tree. This means that for a tree with 30 levels, there are approximately 2.1×10^{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. 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 7. This tree has an approximate size of $e^{0.79524n+0.85865}$. Following the above example of thirty levels, the size of the tree is 5.42×10^{10} nodes, which is a 99.97% decrease. 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. Doing so provides a maximum tree depth that is suited for the particular problem, as well as the ability to terminate the tree search at any time and still have a feasible path that accomplishes the desired objectives. For this research, the LRTA* tree search is initiated using the results of a single-source potential-field path planning algorithm [1].

One advantage of the 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 requiring the vehicle to over-fly a target and perform an attack. Implementing additional goals and constraints is simply a matter of identifying and pruning infeasible branches during the LRTA* tree search.

4 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. 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. Because of the complexity of the LRTA* tree search, step sizes smaller than 0.2 radians were not considered. Algorithm testing has demonstrated that using step sizes below 0.2 radians does not significantly decrease the path lengths produced by the algorithm.

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 8(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% longer. Resetting the sensor to its original dimensions produced the path shown in Figure 8(b). This path is 33%, 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.

Figure 8: (a) The LRTA* algorithm is capable of producing nearly the same optimal path through a set of targets as the brute-force global search. (b) The LRTA* algorithm utilizes the vehicle's full sensing capabilities and produces a 33% shorter path.

Step Size

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. The following tests use step sizes between 0.2 and 0.65 radians, in 0.05 radian increments.

Path Length

The path-length performance for the two algorithms at the various step sizes is presented in Figure 9. The mean path length increases approximately 19%, 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%, which demonstrates the diminishing utility of using smaller step sizes. The figure also shows that the mean path-length performance of the NILRTA* algorithm is comparable to that of the LRTA* algorithm. At $d\psi = 0.2$ radians, the mean NILRTA* path

Figure 9: Mean path lengths for the two algorithms at various step sizes.

lengths are 3.76% 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.

Running Times

Running times for the algorithms are determined 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. Figure 10 presents the running-time results for the LRTA* and NILRTA* algorithms, with the average total running times 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, and about 150,000 nodes for the NILRTA* algorithm. At a step size of 0.3 radians, the mean running time for the LRTA* algorithm drops to about 200,000 nodes, which is a 97.5% decrease. The NILRTA* algorithm's running time drops by 46.7% to about 80,000 nodes. As the step size continues to increase, the

Figure 10: LRTA* and NILRTA* running time comparison for increasing step sizes.

mean running times decrease further, leveling off at about 7,000 nodes 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 10 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% of the total running time is spent confirming the optimal path. The NILRTA* spends about 92.7% 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 paths require more time to compute.

(a)

(b)

Figure 11: (a) Running time comparison with different numbers of targets. (b) Distance comparison with different numbers of targets.

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 4(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, compared to the NILRTA* algorithm which visits an average 200,000 nodes. The NILRTA* algorithm is consistently faster than the LRTA* algorithm for any number of targets, and as shown in Figure 4(b), has path length performance comparable to the

Figure 12: Path produced by the LRTA* algorithm for sensing eight targets.

LRTA* algorithm. Figure 12 shows an example of a path generated by the LRTA* algorithm for an eight-target scenario.

5 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. 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.

References

- [1] Jason K. Howlett, “Path planning for sensing multiple targets from an aircraft”, Master’s thesis, Brigham Young University, Provo, Utah, 2002.
- [2] L.E. Dubins, “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, pp. 497–516, 1957.
- [3] Guang Yang and Vikram Kapila, “Optimal path planning for unmanned air vehicles with kinematic and tactical constraints”, Tech. Rep., Department of Mechanical, Aerospace, and Manufacturing Engineering, Polytechnic University, Brooklyn, NY 11201, February 2002.
- [4] Yoram Koren and Johann Borenstein, “Potential field methods and their inherent limitations for mobile robot navigation”, in *Proc. IEEE Conf. on Robotics and Automation*, Sacramento, CA., April 1991.
- [5] Timothy W. McLain and Randal W. Beard, “Trajectory planning for coordinated rendezvous of unmanned air vehicles”, in *Proc. AIAA Guidance, Navigation, and Control Conference*, Denver, CO., August 2000.
- [6] Emilio Frazzoli, Munther A. Dahleh, and Eric Feron, “Real-time motion planning for agile autonomous vehicles”, *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, January-February 2002.
- [7] Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars, “Probabilistic roadmaps for path planning high-dimensional configuration spaces”, *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, August 1996.

- [8] Lydia E. Kavraki, Mihail N. Kolountzakis, and Jean-Claude Latombe, “Analysis of probabilistic roadmaps for path planning”, *IEEE Trans. on Robotics and Automation*, vol. 14, no. 1, February 1998.
- [9] Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How, “Mixed integer programming for multi-vehicle path planning”, in *European Control Conference*, 2001.
- [10] Arthur Richards, John Bellingham, Michael Tillerson, and Jonathan How, “Co-ordination and control of multiple UAVs”, in *Proc. AIAA Guidance, Navigation, and Control Conference*, 2002.
- [11] R. E. Korf, “Real-time heuristic search”, *Artificial Intelligence*, vol. 42, no. 2-3, pp. 189–211, 1990.
- [12] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh, “Learning to act using real-time dynamic programming”, *Artificial Intelligence*, vol. 72, no. 1, pp. 81–138, 1995.
- [13] Toru Ishida, “Real-time search for autonomous agents and multiagent systems”, *Autonomous Agents and Multi-Agent Systems*, vol. 1, no. 2, pp. 139–167, 1998.
- [14] Stefan Edelkamp and Jürgen Eckerle, “New strategies in real-time heuristic search”, in *AAAI-97 Workshop on On-Line Search*. 1997, pp. 30–35, AAAI Press.
- [15] David Furcy and Sven Koenig, “Speeding up the convergence of real-time search”, in *Proc. of the Nat. Conf. on Artificial Intelligence*, 2000, pp. 891–897.
- [16] Toru Ishida and Masashi Shimbo, “Improving the learning efficiencies of realtime search”, in *Proc. of AAAI-96*, 1996, pp. 305–310.
- [17] Gerhard Weiss, Ed., *Multiagent Systems*, chapter 4, pp. 165–199, The MIT Press, 2000.