AIAA 1st Intelligent Systems Technical Conference
20 - 22 September 2004, Chicago, Illinois

AIAA 2004-6229

A Hybrid A*/Automaton Approach to On-Line Path Planning with Obstacle Avoidance

Nathan D. Richards*, Manu Sharma†,
and David G. Ward‡
Barron Associates, Inc., Charlottesville, VA 22901

Autonomous path planning capability is a critical requirement for future unmanned vehicle operations. This paper presents a novel path planning approach for rapid construction of feasible, yet agile, trajectories. The approach relies upon separating the planning task into an on-line and an off-line component. The off-line component consists of building a library of motion primitives using nonlinear simulation, the primitives are then sequenced on-line using an A* search. The main benefit of this separation is that it shifts the computationally intensive work to the off-line component and leaves the on-line task relatively light. A major benefit of using A* is that the search algorithm is unaffected by the changes to the library of trim primitives. Thus in the event that a UAVs capabilities are degraded, e.g. due to damage, any unavailable motion primitives are simply removed from the library and A* will not included them in the path construction. To speed up the on-line search time, a sub-optimal modification is made to the standard A* algorithm that delivers a feasible sub-optimal path quickly with relatively small increase in total path cost. Examples of waypoint planning and obstacle avoidance are given to illustrate the path planning approach.

I. Introduction

There has been significant interest in autonomous path planning over the past few years. While this technology has been applied in a number of application areas, it has received particular attention from the aircraft community for use on unmanned aerial vehicles (UAVs). In this application, autonomous path planing capability can greatly reduce operator workload and even allow operators to control multiple UAVs by focusing only on the higher, mission-level, tasks.

Some of the earliest work on path planning was performed by Dubins,[1] who addressed the problem of constructing an optimal planar path, with constraints on maximum path-curvature, to move a "car" from an initial location and orientation to a subsequent location and vehicle orientation. He showed that the optimal solution for this problem has a bang-bang form consisting of at most three path segments and takes either the form CCC or CSC, where C represents circular arcs of maximum curvature, and S represents straight lines. The main benefit of Dubins' solution for 2D path planning is that it is a geometric construction and lends itself to rapid on-line construction of minimum-time paths. However, this approach assumes that the car can switch instantaneously between path types, and therefore ignores the transitions between them, i.e., maneuvers. It should also be noted that the Dubins's solution does not consider obstacles or cost functions other than minimum time.

A common approach for path planning in the presence of obstacles uses potential fields to "attract" the agent to the goal while "repelling" it from any obstacles, see Khatib.[2] Two key issues arise in the

*Research Associate, richards@bainet.com
†Research Scientist, Member AIAA, sharma@bainet.com
‡Senior Research Scientist, Member AIAA, ward@bainet.com

potential field approach. First, there is a possibility of local minima in which the search can get stuck. Secondly, constraints on vehicle motion are not readily included, i.e. the vehicle may not be capable of tracking the path that is constructed. Janabi-Sharifi and Vinke,[3] have tackled the local minima problem by using simulated annealing to escape and permit the algorithm to continue searching for the global minimum however there is no guarantee a global minimum will be found in a tractable amount of time.

McLain and Beard[4] approached the UAV path planning problem by constructing Voronoi diagrams, searching for the shortest path to the goal along the edges of the diagram, then smoothing the path to make it "flyable". However, as pointed out by Howlett et al,[5] this approach is computationally expensive and difficult to complete on-line. Howlett et al[5] developed a method of path-planning for sensing targets at known locations by assembling arcs and straight line segments using a learning real-time A* (LRTA*) search, this method requires a computationally expensive initialization and also assumes linear vehicle dynamics although the arc radii are determined by the vehicle maximum turn rate. Richards and How[6] formulated waypoint path planning as a mixed integer linear programming (MILP) problem, their approach allows the inclusion of obstacles but also suffers from high computational complexity. Although consideration is given to vehicle capability limits (i.e. max turn rate), these approaches do not explicitly account for nonlinear vehicle dynamics.

Frazzoli et al[7,8] approached a the path planning problem by discretizing the vehicle dynamics into two types of motion primitives: trims and maneuvers. They define trims as steady-state vehicle operating conditions (airspeed, turn-rate, flight-path-angle), and maneuvers as the transient motion between the trims. The trims and maneuvers are stored in a library, referred to as an *automaton*, that is generated off-line via a high-fidelity simulation of the vehicle then accessed as part of the on-line planning. This discretization is attractive for two key reasons. First, the vehicle flight envelope is taken into account by the limits of the trim conditions, i.e. only trims which the vehicle can actually achieve are included. Also, maneuver information acquired through observation of high-fidelity simulation explicitly accounts for nonlinear motion between the trim trajectories. The trajectory planning task is then reduced to assembling trim and maneuver primitives to achieve the desired objective. The main benefit of this approach is that it explicitly accounts for the nonlinear transient motion between trim trajectories that the previous work assumed to be instantaneous.

This paper borrows the idea of discretizing the vehicle dynamics into motion-primitive libraries from Frazzoli et al,[7,8] but focuses on rapidly assembling the trajectories, consisting of only currently achievable primitives, on-line using an A* search.[9] It also addresses 3D motion planning whereas the majority of the previous work is in 2D. The benefits of using A* include straightforward integration with motion-primitive libraries and an optional algorithm modification, discussed herein, which trades optimal solutions for solutions which may be found in real-time or near-real-time.

Section II discusses the discretized motion-primitives, the A* search algorithm, and the integration of motion-primitives with A* for UAV path planning. Section III presents results for 2D trajectory planning including the case of deteriorated vehicle performance. Following the 2D planning discussion, section IV expands concepts from the 2D case to the higher dimension 3D problem and incorporates obstacle avoidance.

II. A* Based Path Planning

The formulation of a path planning problem in the context of A* requires definition of the search space, possible agent actions, optimization variable, and heuristic function. For digital implementation, the search space must also be discretized using a carefully selected resolution. Under refinement may result in excessive terminal state errors while over refinement leads to a large discrete search space that may cause solutions to be inefficient or even intractable due to CPU speed and memory limitations. Real-time or near-real-time implementation is a significant concern for this work.

A. Motion Primitives

The main motivation for using motion primitives for path planning is that it allows the simplification of complex vehicle dynamics in the continuous domain into a simpler discrete form more amenable to path construction techniques that sequence primitives together into a useful trajectory. At the lowest level, the

most basic type of motion primitive for a dynamical system is a trim. For aerospace vehicles a trim can be considered to be constant-velocity trajectories, e.g., straight and level flight, in-plane turns, spirals, etc. The concept of trims is commonly used for flight control; however, trims are generally viewed as being independent of each other, and transitioning from one trim to another requires another trajectory primitive. Thus *maneuvers* are defined as finite-time transitions between two trim states. The collection of trims and maneuvers, referred to as a *maneuver automaton* forms a description of the aircraft motion, and can be used for path construction.[7]

One of the primary benefits of the maneuver automaton is that it is a compact representation of the system dynamics that does not rely on approximations or simplifications of the underlying dynamical system. The maneuver automaton is model-based (or a flight-test based) and, as such, it can describe full-order, complex, nonlinear dynamics. However, to reduce complexity, the automaton does not include every possible trim and maneuver, but rather a finite set of all possible trims and maneuvers - with each trim or maneuver representing a possible decision the automaton can make. This introduces a degree of suboptimality, and difficulty representing the "smooth" motion of the original system; however, these simplifications allow for real-time solutions of complex path-planning problems that take into account the full nonlinear dynamics of the original system.

A graphical depiction of the concept of trims and maneuvers is given in Figure 1.[8]

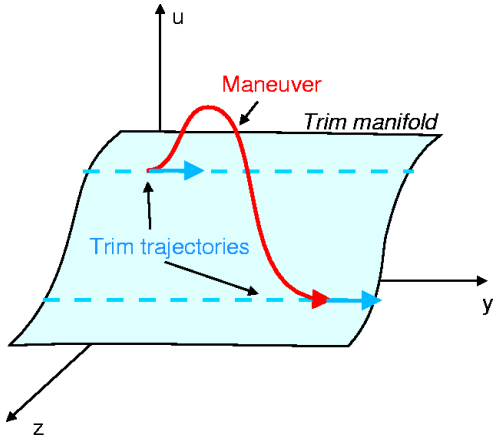

Figure 1. Trims and Maneuvers as Trajectory Primitives

B. A* Search Algorithm

A* is considered a form of best-first search. This type of search selects nodes based on the cost to reach them from the starting point, i.e. those reached with the lowest cost are expanded first. The main shortcoming of the standard best-first search is that it uses no knowledge of the cost to continue onward to the goal, and is therefore susceptible to expanding nodes that appear cheap but have a high hidden cost. A* is an enhancement of best-first search that incorporates a heuristic function as an estimate of the cost to continue on to the goal. An A* search selects nodes to expand based on the cost to reach them from the starting point *plus* the estimated cost to reach the goal from there. The evaluation function used for an A* search is given by,

$$f(n) = g(n) + h(n), \tag{1}$$

where n represents the node in question, $g(n)$ is the cost to reach n from the starting point, and $h(n)$ is the heuristic cost to go from n to the goal. Thus at each step, the node with the lowest evaluation function value, $f(n)$, is expanded next.[9]

When a node is expanded, all admissible actions from that node explored to see whether it leads to another node. Certain actions may lead outside the user-specified search space, and are discarded. This characteristic can be exploited for obstacle avoidance, which is discussed later. Each of the admissible nodes

is then evaluated using the function in (1), and the one with lowest cost is expanded next. This process continues until the goal node is reached and no other nodes have a potentially lower cost to the goal.

One of the benefits of the A* search is that it is complete in the sense that if a solution exists the algorithm is guaranteed to find it. If the heuristic is perfect, or an underestimate of the actual cost to reach the goal (i.e., if it is an *admissible* heuristic) the search result is also optimal.

Computational time and memory requirements for A* search are strong functions of the heuristic quality. The number of nodes that are expanded, stored, and sorted is an exponential function of the amount by which the heuristic underestimates the true cost to goal. If the heuristic perfectly represents the cost to reach the goal, the search will only visit nodes that lie on the optimal solution path, and will therefore complete very rapidly. However, if the heuristic is an underestimate, additional paths will be explored and slow down the search execution. This increase in computational complexity highlights the advantage of A* over best-first search as the latter is simply an A* search with a zero heuristic. In certain applications an inadmissible heuristic, i.e., and *over*-estimate, can be used to decrease computation computational complexity. In these cases, the search will still be *complete*, but is no longer guaranteed to be *optimal*.

Discretization of the search space is required to implement the A* search on a digital computer as well as to minimize the memory and computational burden of applying the algorithm to a large, high dimension search space. The A* algorithm keeps track of which states in the search space have been visited and stores information about only the "best" of each state as determined by the evaluation function. Thus if a fine grid is used, the algorithm can rapidly run out of memory and/or computational time as it attempts to store and sort an excessive number of states. Coarser spacing reduces memory and computational requirements but increases terminal state errors as any location within the grid square containing the goal will be considered a solution. Thus, the lower and upper limiting factors on grid spacing are the computational/memory capacity of the CPU and the terminal error tolerance respectively.

C. State Connection to Reduce Quantization Error Build Up

To progress with this discussion, it is important to note the difference between node and state. In the context of A*, a node is a container for a state plus additional information required by the A* algorithm. A node contains a state, a pointer to the parent node, the action required to get from the parent node to the node, the cost to get to the node state from the initial state, and the estimated cost to continue on to the goal. The state referred to here is located at the center of the discretized search space encompassed by the node. The cost associated with a node is the sum of the cost to arrive at it from the initial state and the estimated cost to the goal, Equation 1.

To eliminate quantization error build up associated with discretization, an additional piece of information will be stored in each node. Referring to Figure 2, consider node B which has been encountered as a result of action 1 from the previous node, node A. Each node encompasses a region of the search space whose size is defined by the grid spacing, the state defining this node is the center of the encompassed region. Any subsequent action from node B will advance the search to another node, node C. In discrete implementation, the state change incurred by this action may be applied to either the defining state at the center of the grid region or to the state resulting from the previous action, referred to as the action result state. These options are represented as paths 1b and 2b and paths 1a and 2a, respectively. As may be observed in the figure, using the action result state does not build up quantization error as using the grid defining state would. Thus to eliminate build up of quantization error, each node will also contain the action result state and this state will be used as the initial state for each action taken from this node.

D. Suboptimal Modification

The A* search algorithm can be modified to operate more quickly if the requirement for an optimal solution is relaxed. Although the solution may not be optimal, the search requires considerably less memory and computation time. This is very desirable for on-line operation when there are hard limits on the time allowed for the search process.

The modification involves altering how the algorithm handles the situation when a node is encountered

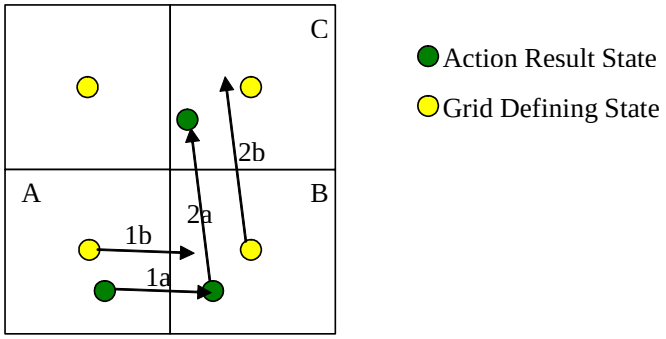

Figure 2. Grid vs. State Node Connection

more than once. In the regular A* algorithm, any new shorter path to a particular node replaces the previously stored shortest path and subsequently, assuming an admissible heuristic, advancement towards the goal from that node restarts. The suboptimal A* proposed here only retains information about the first path to encounter a particular node. This greatly reduces computation time for two key reasons. Firstly, nodes are not reallocated to new paths allowing the original path to proceed towards the goal. Secondly, since the goal node cannot be reallocated to a shorter path, the first path to the goal is taken as the solution instead of expanding all other potential paths before declaring a winner. Although the solution may not be optimal, the algorithm still favors low cost solutions.

E. Path Planner Reconfiguration

The need for reconfiguration arises when the vehicle has suffered a deterioration in nominal performance due to a failure or damage sustained in combat. The task or reconfigurable path planning is a minor extension of the path planning task described above. The nominal performance deterioration is conveyed to the search algorithm by removing unachievable maneuvers and trims from the possible action list i.e. if the vehicle can no longer make an aggressive turn, the unachievable trims and maneuvers leading to those trims are removed an cannot become part of the solution trim and maneuver sequence. A representative reconfiguration result is presented in Figure 4.

III. 2D Waypoint Path-Planning

The path planning specifications considered here for 2D waypoint path-planning require that the final state of the vehicle meet a goal state specified by the two-dimensional ground position $(North, East)$, heading (χ), and turn rate $(\dot{\chi})$. Thus the search space is four-dimensional and the discrete size of each dimension is determined by the range and resolution of the respective state. The following subsection addresses the issue of search space discretization.

Possible vehicle actions at each step of the search are contained in a library of trims and maneuvers. Trims are steady-state vehicle conditions which, in this case, are defined by their turn rate $(\dot{\chi})$. Maneuvers are transitions between trims. For a given node in the search space the possible actions are any maneuver originating in the trim state of the node or to stay in the trim state of the node. When a node is expanded all nodes which are reached from this set of maneuvers or maintaining the trim are added to the current search.

For simplicity, a straight-line heuristic is used to compute the minimum time to the goal. The heuristic function is then:

$$h(n) = \frac{X_g(n)}{V_t}, \tag{2}$$

where V_t is the airspeed (constant for this path planner), and $X_g(n)$ is the straight line distance from the vehicle $North, East$ position to the goal $North, East$ position. There is much room for improvement in this

heuristic since it does not take heading or turn rate into account. One possible heuristic which includes heading and turn rate constraints is the Dubins'[1] path mentioned the introduction.

A. Selection of Grid Spacing

Discretization of the search space requires selection of the discrete spacing of each dimension. The current search dimensions are 2D position (*North*,*East*), heading (χ), and current trim turn rate ($\dot{\chi}$). As mentioned previously, the upper limit of grid spacing is based on terminal error tolerance and the requirement that each action taken from one node results in encountering another node, the latter is referred to as the *successor requirement*.

The possible actions from a particular node include all maneuvers connected to the trim state of that node as well as maintaining the trim itself. In order to satisfy the successor requirement, *North* and *East* position grid spacing must be selected concurrently with selecting a trim-expansion time step to ensure that the time step is long enough to lead the search into another grid space. However, care must be taken to prevent the search from skipping over nodes in order to prevent the possibility of jumping past the goal node.

The $North, East$ spacing should be chosen according to the final state error tolerance with consideration of minimizing the total grid size for reduced computational burden. Heading is unrestricted so actions may be initiated from any heading, thus the *East* and *North* spacing should be chosen equal, this spacing is denoted x. If the trim is traversing the grid space directly in the North or East direction than the trim displacement should be exactly x. However if the trim is traversing diagonally the trim expansion displacement should be $x\sqrt{2}$, refer to Figure 3. To handle this discrepancy the trim expansion duration steps from one duration to a longer duration if it is required to satisfy the successor requirement.

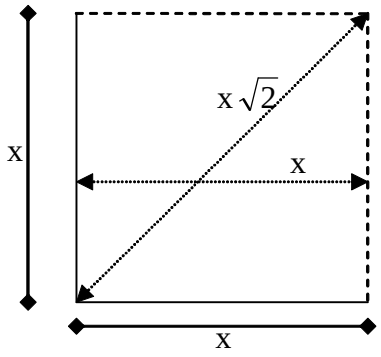

Figure 3. Trim Distance Across Grid

Finally, the spacing for the heading dimension must be determined. Assuming the *North* and *East* spacing has been chosen correctly, it is not necessary to consider the successor requirement when selecting the heading spacing because it is already satisfied by the position change. The heading spacing may be chosen according to the specification of final heading error tolerance as well as consideration of how much heading variation through a given node results in a significant change in path cost.

B. Results

The first result presented here illustrates the reconfiguration capability of this approach. The task, summarized in Table 1, requires the path to start and end at 90 deg heading, wings-level trim, and also change *East* position by 4000 ft and *North* position by 13,000 ft. The available turn rates for the nominal path plan are [-7.5, -4, -2, -1, -0.5, -0.1, 0, 0.1, 0.5, 1, 2, 4, 7.5] deg/sec while the reconfigured path has only up to ±2 deg/sec turn rates available. A summary of the path durations as well as required computation time is given in Table 2.

Table 1. 2D Planning Initial and Goal State

State	Start	Goal
χ	90°	90°
$\dot{\chi}$	0 deg/sec	0 deg/sec
$North$	2750 ft	6750 ft
$East$	2000 ft	15000 ft

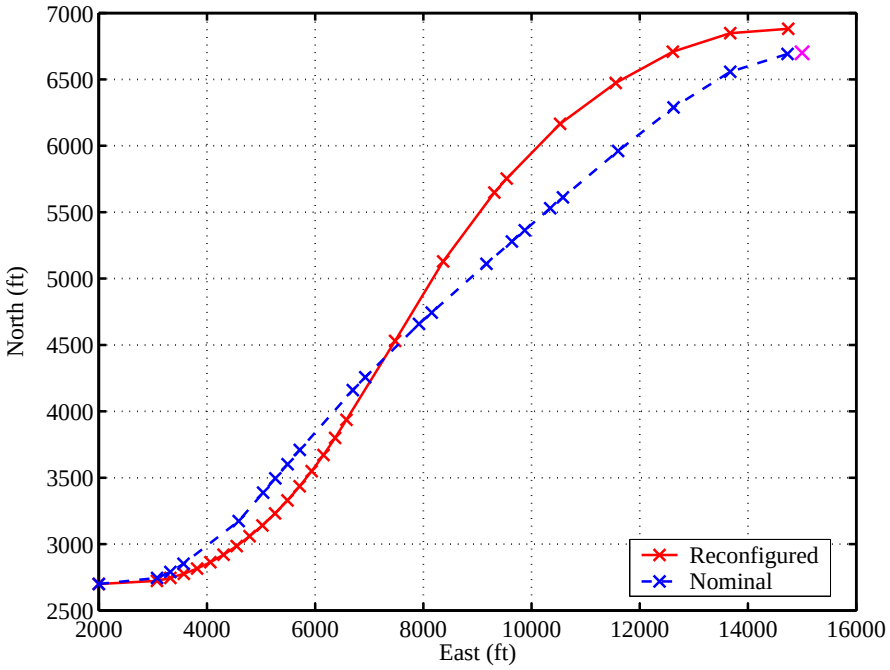

Figure 4. Reconfigured Plan Ground Track

Figures 5 and 6 compare the path planning results from the sub-optimal modification with the optimal A* algorithm. Task specification and available trims and maneuvers are the same as in the reconfiguration result. Suboptimal A* computation time savings as well as path duration costs are summarized in Table 2. In both the nominal and reconfigured case the path duration increased only slightly, 1.9% and 0.7% respectively whereas the search time was drastically reduced, -96.7% and -41.7% respectively.

	Nominal			Reconfigured		
	Optimal	Suboptimal	%Change	Optimal	Suboptimal	%Change
Path Duration (sec)	53.93	54.96	+1.9	54.89	55.28	+0.7
Search Time (sec)	1685	56	-96.7	84	49	-41.7

Table 2. Suboptimal Comparison

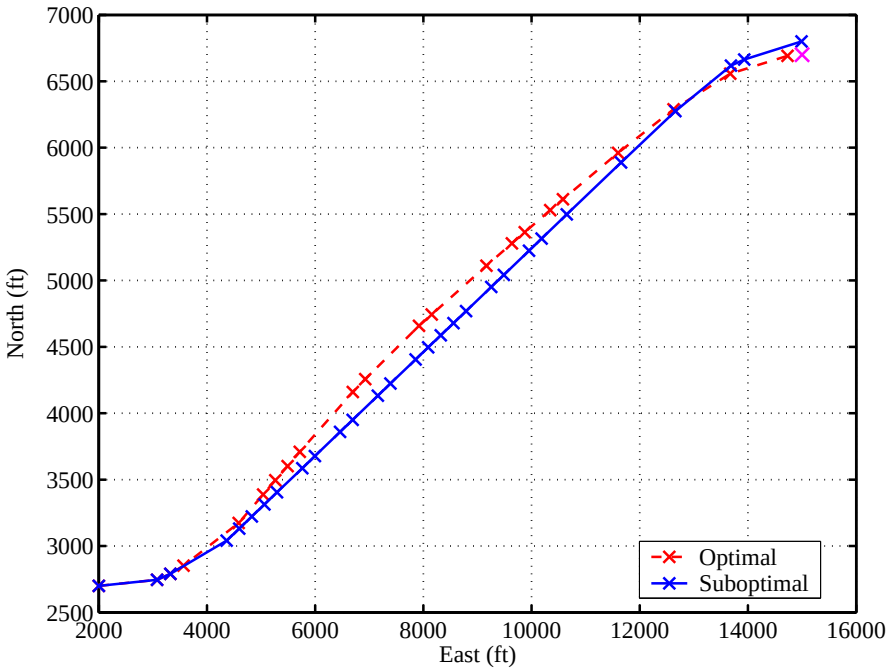

Figure 5. Nominal Case

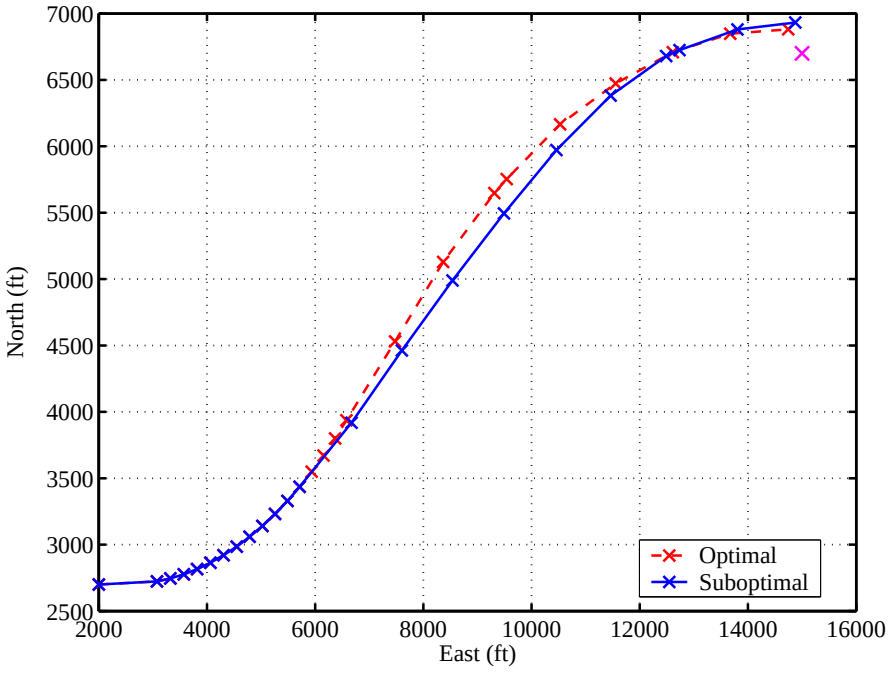

Figure 6. Reconfiguration Case

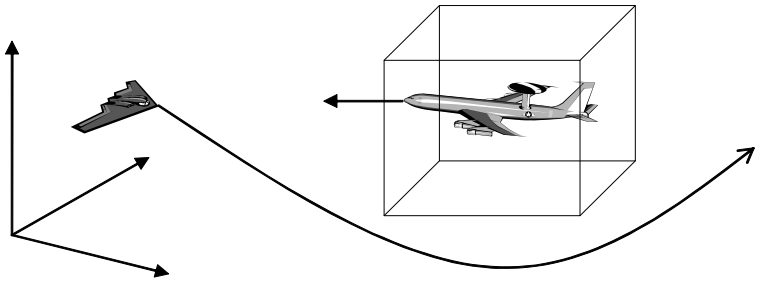

Figure 7. Obstacle Avoidance Scenario

IV. 3D Obstacle Avoidance Path-Planning

This section presents the application of the agile path planning algorithm developed in this paper to obstacle avoidance in 3D. Obstacle avoidance is significantly different from the waypoint planning application in the previous section because paths must necessarily be more agile for obstacle avoidance, and can also use relatively more vertical motion. This motivates the need for planning in 3D.

The aim of the obstacle avoidance application is to construct a time-optimal 3D path that guides the UAV around an obstacle and returns it to within a corridor of its original path at its original trim. This allows the UAV to return to whatever task it was performing. It is assumed that the position of the obstacle is known, and that the obstacle is stationary. The case of a moving obstacle can be easily incorporated, but is beyond the scope if this paper. The use of a goal corridor is computationally beneficial because providing A* a space in which to terminate, rather than point (or a single grid quare) significantly speeds up the search time. Therefore, sizing the goal region requires consideration of both available computation time as well as reasonable/safe deviations from the original, unobstructed path, the latter limits the upper limit of the refinement while the former limits the lower limit.

For the obstacle avoidance problem, automaton trims are defined as constant turn-rate ($\dot{\chi}$) and flight-path angle (γ) trajectories, and their combinations. An example trim library is given in Table IV; for simplicity all trims are at the same airspeed ($V = 500 ft/sec$). A fully-connected maneuver library is used, consisting of 72 maneuvers ($\#trims * (\#trims - 1)$).

Table 3. Vehicle Trims States

Trim	$\dot{\chi}$ (deg/sec)	γ (deg)
1	-6.5	-5.0
2	0.0	-5.0
3	6.5	-5.0
4	-6.5	0.0
5	0.0	0.0
6	6.5	0.0
7	-6.5	7.5
8	0.0	7.5
9	6.5	7.5

A. Goal Region Definition

As previously described, goal region for obstacle avoidance was selected to be a "corridor" around the original path of the vehicle. An overhead view of an example goal corridor is given in Figure 8, which shows that the goal region is defined by a line segment and two rays. N_1 and N_2 are made parallel to the initial heading while N_3 is parallel to a line formed by connecting obstacle corner one and corner two. Rays N_1 and N_2 ensure that the solution path will terminate in the direction in which the UAV was originally heading, and also specify a lateral tolerance; N_3 ensures the vehicle moves past the obstacle. While not shown in this diagram, there are similar boundaries for altitude, and terminal heading, whereas the terminal trim is required to be the same as the initial trim. Thus a solution path is required to (a) terminate in a 3D corridor, (b) terminate within a specified tolerance of the initial heading, (c) provide the correct terminal trim, and (d) lie entirely outside the obstacle region. Table 4 gives a example set parameters defining the goal region; note that the terminal flight-path angle and turn-rate specify the terminal trim.

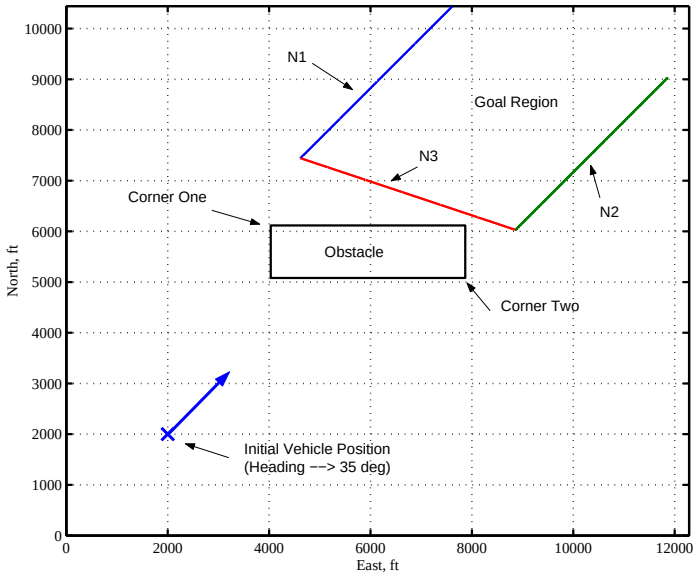

Figure 8. Overhead View of Goal Region

Table 4. Goal Region Subspace

State	Final Value
γ	prescribed by final trim
χ	+/- 10° of initial
$\dot{\chi}$	prescribed by final trim
North / East	+/- 2000 ft of initial path
Alt	+/- 500 ft of initial

B. Cuboid Obstacle Definition

The obstacle is represented by a cuboid in the search space because of its geometric simplicity. Additionally, the edges of the cuboid are parallel to their respective axes in the NED reference frame. Given this construction, the cuboid can be completely defined by specifying two opposite corners. The obstacle itself is at the core of the cuboid, and the goal of the A* search is to plan a path that does not path through this region. However since the paths are constructed by sequencing discrete trajectory segments, it may "cut

corners" of the obstacle cuboid. Thus the obstacle must be surrounded by a "discrete-path buffer" to ensure that the solution truly lies outside the obstacle cuboid.

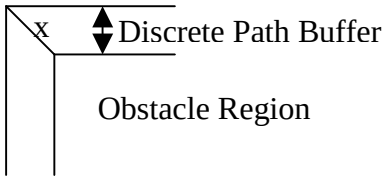

Figure 9. Buffer Zone

Construction of the discrete path buffer requires consideration of the motion primitives included in the library. For trim segments, where the path is a line or arc segment, interference checking can be performed at an arbitrarily high frequency. However, since computation requirements increase with this frequency, a rate of 1Hz was selected for this work.

Due to their nature, maneuvers cannot be represented by a simple arc or line segment. Therefore, interference checking for maneuver segments is only possible at those points at which intermediate path information is stored in the library. An additional complication is that the maneuvers are generally of different duration. To facilitate interference checking during maneuvers, intermediate path information for each maneuver is included in the maneuver library at the rate of 1Hz. Thus interference checking during maneuvers may only be performed once every second, as well as at its start and end points.

Checking path progression every second at an airspeed of 500 ft/sec implies that interference checking is performed along 500 ft lengths. However, these 500 ft segments between interference checks may still pass through the obstacle region unbeknownst to the planning algorithm. Thus the buffer must be sized such that the greatest possible path penetration into it will not cross into the obstacle cuboid. Figure 10 illustrates buffer penetration, x, in the horizontal plane. Here, θ is the angle between the chord and the edge of the buffer, s is the path length over which no interference checking is performed. The maximum buffer penetration, x, occurs when the chord is at a 45 degree angle with the edge of the buffer. Given the longest unchecked path length s, the penetration distance as a function of the arc radius can be determined using the geometric construction shown in Figure 11. Here, R is the radius of curvature, O is the center of curvature, c is the chord length, d is the distance from the center of curvature to the corner of the buffer, h is the distance from the corner of the buffer to the chord, and z is the distance from the chord to the arc. Additionally, x can be expressed as $x = h + z$.

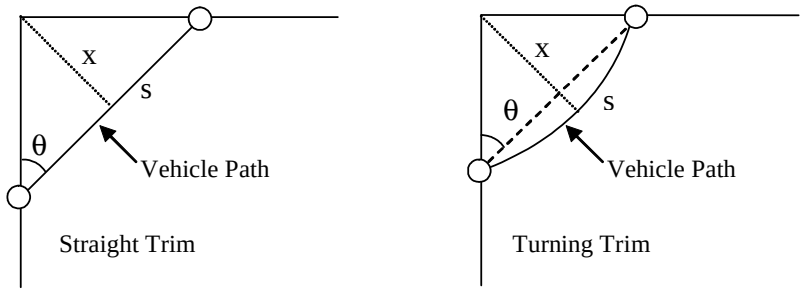

Figure 10. Horizontal-Plane Penetration

The following geometric relations are used to determine maximum possible penetration as a function of trim

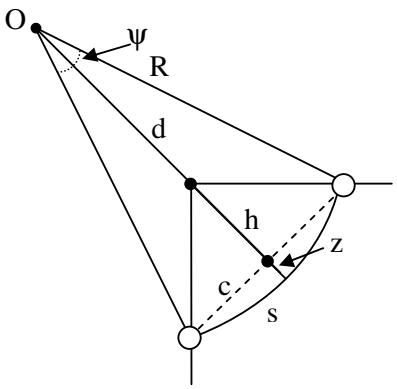

Figure 11. Geometric Construction

turn radius, $x(R)$.

$$s = R\psi \tag{3}$$

$$c = 2R\sin\left(\frac{\psi}{2}\right) \tag{4}$$

$$h = \frac{c}{2} \tag{5}$$

$$d + h = R\cos\left(\frac{\psi}{2}\right) \tag{6}$$

$$z = R - (d + h) \tag{7}$$

$$x = z + h \tag{8}$$

$$x = R - (d + h) + h \tag{9}$$

Combining the above expressions yields

$$x(R) = R - R\cos\left(\frac{s}{2R}\right) + R\sin\left(\frac{s}{2R}\right) \tag{10}$$

It should be noted that Eqn. (10) is only valid when the turn center lies outside the buffer. Figure 12 shows the maximum possible buffer penetration when checking at 1Hz, i.e., at 500 ft path lengths. Inspection of this figure shows that $x(R)$ is monotonically decreasing, and therefore the line trims have the least penetration whereas the turning trim with the smallest radius has the most. The maximum turn-rate of 6.5 deg/sec from Table 3 at 500 ft/sec of airspeed results in a turn radius is 4407 ft, which implies a maximum penetration depth of 257 ft. Therefore, the width of the discrete path buffer must be greater than 182 ft, $x_{\max}\cos(45\,\text{deg})$, to ensure that the planned path does not enter the obstacle cuboid.

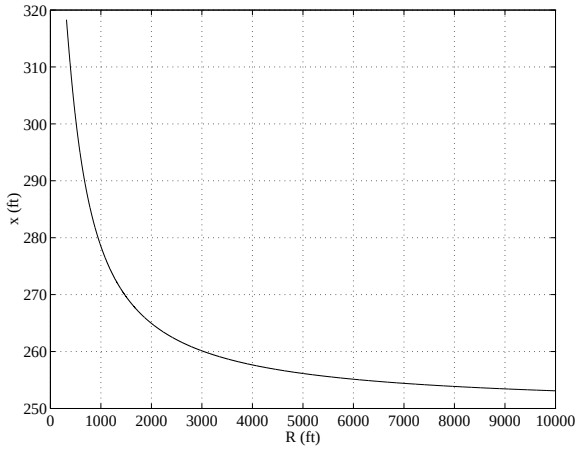

Figure 12. Penetration Trend

In addition to horizontal-plane penetration, vertical-plane penetration must also be considered. Fortunately, sizing the buffer vertically is straightforward since trim segments in the vertical plan are all line segments (no arcs). Using a maximum path length of 500 ft, and the maximum flight-path-angle magnitude of 7.5 degrees from Table 3, the altitude buffer is determined as $x_{long} = s\sin(|\gamma|_{\max}) = 65.3ft$.

C. 3D Obstacle Avoidance Results

1. Example One

In this example the UAV is cruising at 500 ft/sec at an altitude of 2750 ft with a heading of 45 deg when an obstacle is detected. Given these conditions, that path planner constructs a path that avoids the obstacle by diving underneath it, see Figures 13 and 14. The initial and final states of the path are given in Table 5.

Table 5. Example One Initial and Final States

State	Initial	Final
γ	0°	0°
χ	45.0°	45.32°
$\dot{\chi}$	0 deg/sec	0 deg/sec
$North\ /\ East$	2000 ft	13695 ft
Alt	2750 ft	2365 ft

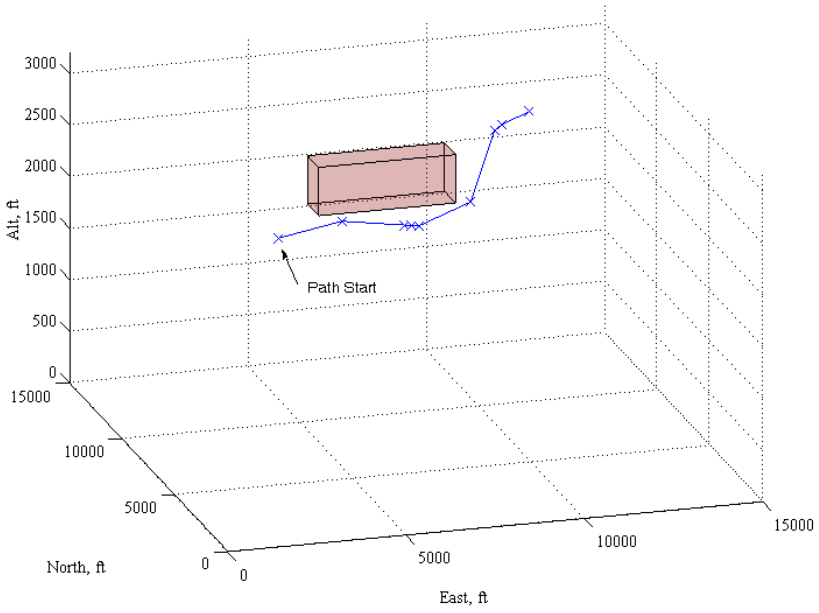

Figure 13. Example One – Three Dimensional Path

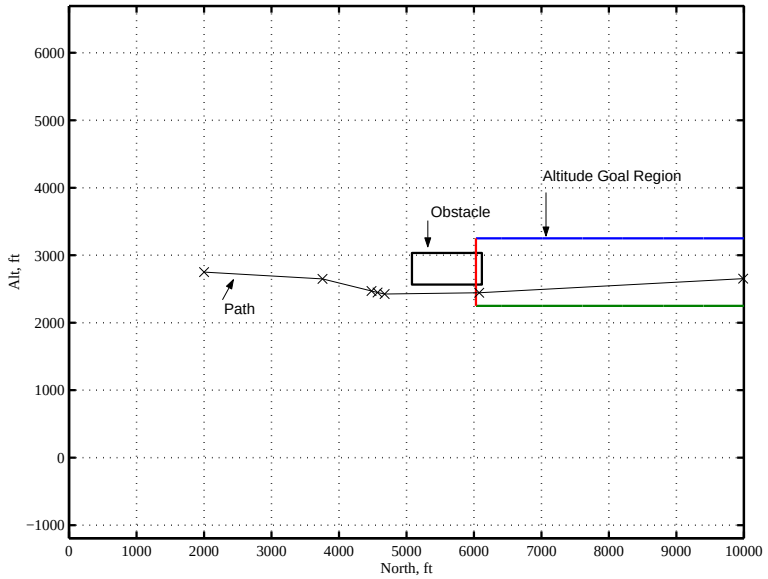

Figure 14. Example One – Overhead View

2. Example Two

This example is the same scenario as example one except that the size of the obstacle has been extended downward. In this case, the search algorithm finds a favorable path around the obstacle instead of under it. The three-dimensional path is presented in Figure 15, the ground track including the goal region is presented in Figure 16, and Table 6 contains the initial and final path states. As summarized in Table 7, both examples one and two require only 30 msec of computation time using a Pentium IV processor.

Table 6. Example Two Initial and Final States

State	Initial	Final
γ	0°	0°
χ	45.0°	38.69°
$\dot{\chi}$	0 deg/sec	0 deg/sec
North / East	2000 ft	13456 ft
Alt	2750 ft	3168 ft

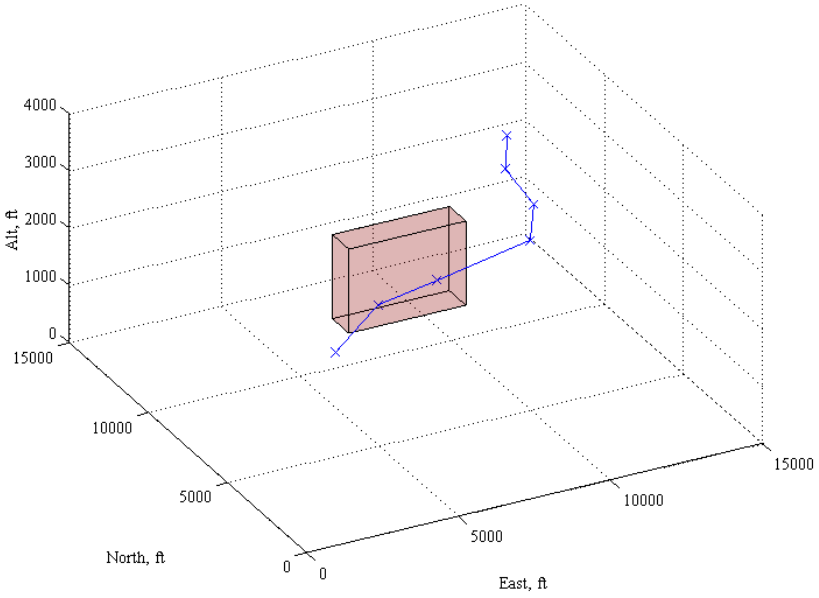

Figure 15. Example Two – Three Dimensional Path

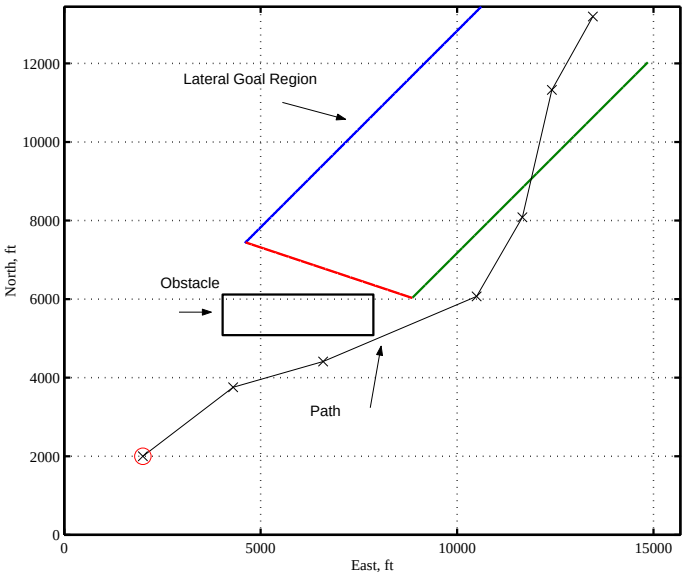

Figure 16. Example Two – Overhead View

Table 7. 3D Planning Summary

	Example One	Example Two
Plan Duration (sec)	35.19	33.08
Search Time (sec)	0.03	0.03

V. Conclusions

This paper presented a path planning approach for waypoint planning and obstacle avoidance using A* and motion-primitive libraries. A modified suboptimal A* search was used to rapidly assemble UAV motion primitives resulting in trajectories that explicitly account for the vehicle's flight envelope and nonlinear motion constraints. The efficacy of this approach as well as the ease of reconfigured planning has been illustrated via numerical example.

VI. Acknowledgements

This work was funded under AF PRDA 01-03-VAK *Intelligent Control of Unmanned Air Vehicles*, Mark Mears, Technical Monitor. A portion of the funding was provided by NAVAIR, Marc Steinberg, Technical Monitor. *Their support is gratefully appreciated.* The authors also acknowledge Eric Feron from MIT for his insight.

References

[1]Dubins, L. E., "On Curves of Minimal Length with a Constraint on Average Curvature and with Prescribed Initial and Terminal Positions and Tangents," *American Journal of Mathematics*, Vol. 79, 1957, pp. 497–516.

[2]Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *1985 IEEE International Conference on Robotics and Automation*, 1985, pp. 500–505.

[3]Janabi-Sharifi, F. and Vinke, D., "Integration of the Artificial Potential Field Approach with Simulated Annealing of Robot Path Planning," *Proceedings of the 1993 IEEE International Symposium on Intelligent Control*, 1993, pp. 536–541.

[4]McLain, T. W. and W., B. R., "Trajectory Planning for Coordinated Rendezvous of Unmanned Air Vehicles," *Proc. AIAA Guidance, Navigation, and Control Conference*, Denver, CO, Aug. 2000.

[5]Howlett, J. K., Goodrich, M. A., and McLain, T. W., "Learning Real-Time A* Path Planner for Sensing Closely-Spaced Targets From and Aircraft," *AIAA Guidance, Navigation, and Control Conference*, Austin, Texas, 2003.

[6]Richards, A. and How, J., "Aircraft Trajectory Planning with Collision Avoidance Using Mixed Integer Linear Programming," *Proceedings of the 2002 American Controls Conference*, Anchorage, AK, May 2002.

[7]Frazzoli, E., Dahleh, M. A., and Feron, E., "Real-Time Motion Planning for Agile Autonous Vehicles," *Journal of Guidance, Control, and Dynamics*, Vol. 25, No. 1, 2002, pp. 116–129.

[8]Frazzoli, E., Dahleh, M. A., and Feron, E., "A Hybrid Control Architecture for Aggressive Maneuvering of Autonomous Helicopters," *Proceedings of the 38th Conference on Decision & Control*, Phoenix, AZ, 1999, pp. 2471–2476.

[9]Russell, S. and Norvig, P., *Artificial Intelligence A Modern Approach*, Prentice Hall, 2003.