# Static and Dynamic Obstacle Avoidance for Miniature Air Vehicles

Jeffery Brian Saunders
*Brigham Young University - Provo*, saunders.jeff@gmail.com

Brandon Call

Andrew Curtis

Randal W. Beard
*Brigham Young University - Provo*, beard@byu.edu

Timothy W. McLain
*Brigham Young University - Provo*, mclain@byu.edu

# Static and Dynamic Obstacle Avoidance in Miniature Air Vehicles

Jeffery B. Saunders, Brandon Call, Andrew Curtis, Randal W. Beard[*]
Timothy W. McLain
Brigham Young University, Provo, UT 84602

**Small unmanned air vehicles are limited in sensor weight and power such that detection and avoidance of unknown obstacles during flight is difficult. This paper presents a low power low weight method of detection using a laser range finder. In addition, a rapidly-exploring random tree algorithm to generate waypoint paths around obstacles known a priori is presented, and a dynamic geometric algorithm to generate paths around detected obstacles is derived. The algorithms are demonstrated in simulation and in flight tests on a fixed-wing miniature air vehicle (MAV).**

**Index Words - Obstacle avoidance, waypoint path planning, rapidly exploring random tree, unmanned air vehicles, miniature air vehicles.**

## I. Introduction

An important problem in the design of fixed-wing miniature air vehicles (MAVs) is obstacle avoidance. There are two general classes of obstacles: a priori known obstacles (e.g. from a terrain map) for which a trajectory can be generated before flight, and pop-up obstacles encountered during flight. The objective of this paper is to present reliable computer algorithms that can address both cases in real time. In this paper we focus on fixed wing miniature air vehicles (MAV) with wingspans less than 48 inches. Traditional laser range finders or radars are too heavy to be mounted on a MAV; a low power, light-weight solution is required.

Previous work in UAV motion planning includes the randomized path planner approach of Frazzoli, Dahleh, and Feron.[6] This approach produces four versions of an algorithm that vary in complexity and completeness. The results comply with the dynamics of fixed-wing aircraft and compute in real time, but the computation time can be too high for a MAV that needs to quickly avoid a pop-up obstacle. The algorithms proposed in Ref. 6 assume high quality sensors that can find the dimensions of all obstacles This assumption is not feasible for MAVs.

Minquez and Montano[11] developed an object avoidance algorithm for ground robots using the geometry of the obstacles. Using a laser range finder, they detect the objects around the robot and use the results to find feasibility regions through which to plan a path. Unfortunately, the algorithm requires a high quality laser range finder and computational resources that are not currently available on MAVs.

An aircraft avoidance method was formulated by Raghunathan [13] that used the known trajectories of other vehicles to prevent collision between the aircraft. This method requires that the trajectories of the obstacles are known a priori.

In this paper we propose a solution that uses a light-weight, low-cost laser ranger and a computationally efficient algorithm to detect and avoid obstacles. Our approach uses a priori known terrain maps to generate an initial waypoint path through the known obstacles, and a reactive planner that responds to pop-up obstacles detected by the laser ranger. The a priori planner plans waypoint paths by exploring a terrain map using a Rapidly-Exploring Random Tree (RRT).[10] The MAV tracks the planned trajectory using standard trajectory tracking techniques.

The reactive planner is used to dynamically avoid obstacles. We assume that a simple single point laser ranger that operates at as slow as two hertz is mounted on the MAV. Using only the returned distances, the MAV can avoid detected obstacles in its path while minimizing the deviation from the planned trajectory. The algorithm operates in real-time and is demonstrated in flight tests.

The organization of the paper is as follows. Section II describes the system architecture including our basic simulation and hardware configurations. Section III describes the a priori path planner and Section IV details the dynamic obstacle avoidance algorithm. A description of the virtual world used for simulation as well as the virtual

[*]Corresponding author: beard@ee.byu.edu

sensors is given in Section A and an analysis of the optimal values for the inner loop parameters is in Section V. Finally, Sections VI and VII give the simulation results and flight test results respectively.

## II. System Architecture

As shown in figure 1, the system architecture consists of five basic parts: the Virtual Cockpit, autopilot, laser ranger, simulator, and virtual sensors. The Virtual Cockpit is located on the ground station and sends waypoints to the MAV together with other commands through a 900 MHz RF link. Its main objective is to monitor and collect telemetry from the MAV. The advantage of the virtual cockpit is that it can connect to either an autopilot or a simulator with an identical interface, which makes simulation similar to flight tests. Simulation data can be collected and analyzed in the same way as flight data.

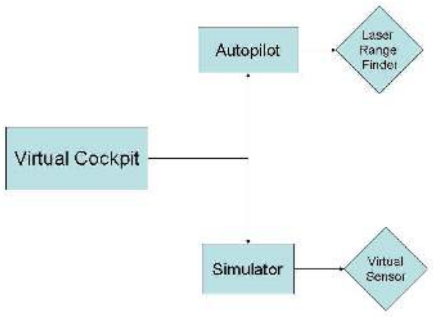
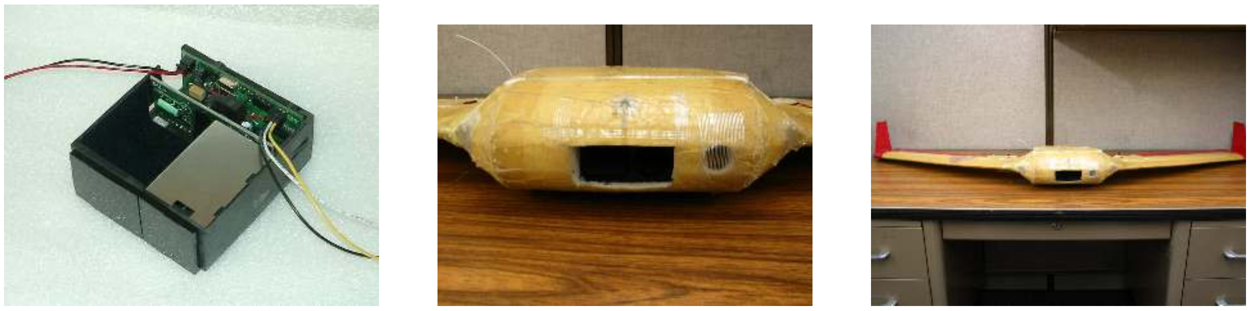


**Figure 1. MAV system architecture used for simulation and flight. The Virtual Cockpit monitors flight. The autopilot is located on the aircraft and the simulator is located on a computer through a network connection.**

The Autopilot is located on the MAV. It processes the control loops of the MAV and monitors its telemetry and position. The obstacle avoidance reactive planner is run on the autopilot. The autopilot connects to the Virtual Cockpit through the RF link and sends telemetry and position data to the ground station. The progress of the MAV can be monitored from the ground from the Virtual Cockpit. The autopilot runs the reactive planner obstacle avoidance algorithms and queries the laser ranger. In this work we have used the Kestrel autopilot[1] which was developed at BYU.

The laser ranger is a small low power device (approximately 2 Watts) that can be mounted on a MAV as shown in figure 2. It has a range of 400 meters on non-reflective surfaces and runs at 10 Hz.[2] It was mounted to detect obstacles directly in front of the MAV. In hardware we did not use a scanning algorithm for the laser. However, in simulation we assumed a laser ranger with two degrees of freedom and use the scanning algorithm described in section A.



(a) Laser ranger from Opti-Logic (b) Laser ranger mounted on a MAV (c) MAV with Laser Ranger

**Figure 2. Photographs show Opti-Logic's laser ranger mounted on a MAV ready to fly the reactive planner obstacle avoidance algorithm.**

The simulator provides a method of testing obstacle avoidance without flying the MAV. The simulator is set up to run autopilot code in a simulated flying environment. It takes into account the dynamics of the MAV as well as environmental conditions such as wind. The result is a realistic simulation that closely models the real world. The simulator Aviones was written and developed at BYU and can be downloaded free of charge at http://sourceforge.net.

The simulator contains virtual sensors, i.e., sensors that mimic the response of real sensors on the MAV. In this case a laser ranger is one of the virtual sensors. The autopilot requests distance estimates and the simulator uses the virtual sensors to find the distance to objects in the simulated environment.

## III. A Priori Path Planner

### A. Rapidly-Exploring Random Trees

The task of the a priori planner is to find a traversable path from a starting position $x_{start}$ to a goal position $x_{goal}$ through a terrain, or configuration space $\mathbb{C}$. To solve this problem for MAVs in urban terrain, we need a planner in three dimensions that can handle nonholonomic vehicle constraints and tight turning requirements. In,[9] Lavalle introduced Rapidly-Exploring Random Trees (RRTs) as a potential solution to such path planning problems. Important advantages of this method over other techniques are discussed in.[8]

The basic RRT algorithm is as follows:

1. Pick a random state $x_{rand}$ in $\mathbb{C}$.
2. Using a metric $\rho$, determine the node $x_{near}$ in the tree that is nearest $x_{rand}$.
3. Apply a control input $u$ to move the graph toward $x_{rand}$ an incremental distance.
4. If there are no collisions along this segment, add this new node $x_{extend}$ to the tree.
5. Repeat until you have reached $x_{goal}$.

This structure is the basis of the algorithm, but in,[8,10] several variations are mentioned. We have implemented a version of the RRT algorithm to plan 3D waypoint paths for MAVs, and have successfully used the algorithm in both simulation and flight tests.

### B. Modified RRT Algorithm

The RRT algorithm that we implemented differs in some key ways from the basic algorithm listed above. The standard RRT algorithm produces a time-parameterized set of control inputs to move from $x_{start}$ to $x_{goal}$. The validity of this result depends on the accuracy of the state-space model being used. In real-life situations with the MAVs, we encounter sensor inaccuracies, wind, and other unmodeled factors. These are some key disadvantages of using an open-loop path planner. We have extended RRTs to be used in closed-loop path planning. Planned paths are represented as waypoints and the paths are tracked by an autopilot using trajectory smoothing.[3,4,7] Knowing the feedback control policy used by the autopilot, and therefore the expected cross-track error at points along the path, we can plan valid paths without specifying the exact control inputs.

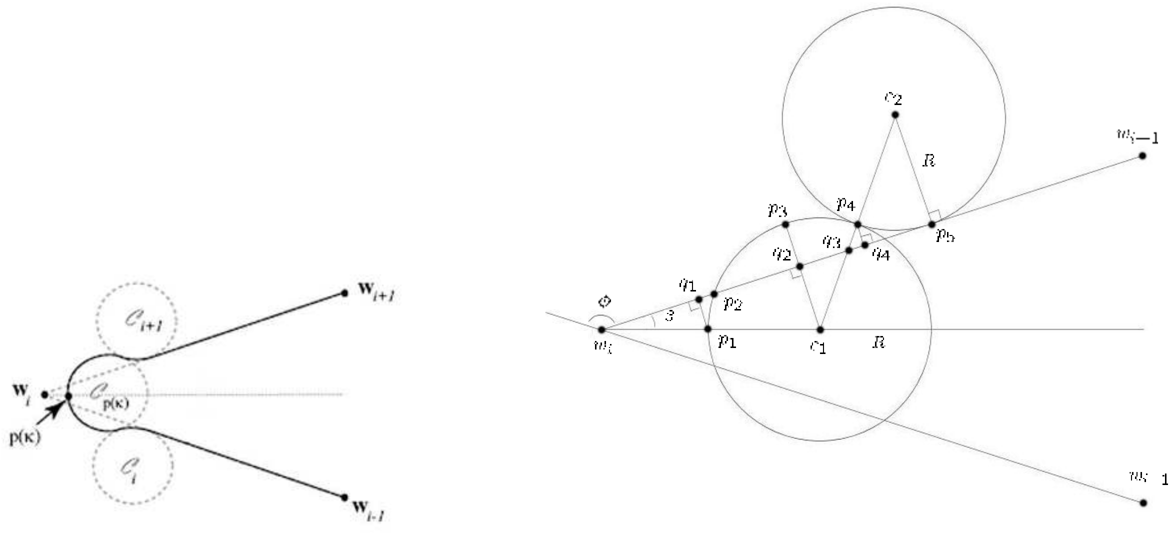
The steps of the modified algorithm are as follows:

1. Pick $x_{rand}$ in $\mathbb{R}^3$. With small probability, set $x_{rand} = x_{goal}$ to pull the graph toward the goal.
2. Determine the node $x_{near}$ in the tree that is nearest $x_{rand}$. The metric $\rho$ is defined as the closest point in Cartesian space, with constraints on the 2-D turn angle and the longitudinal slope to the new point.
3. Move an incremental distance from $x_{near}$ toward $x_{rand}$, resulting in a new state $x_{extend}$.
4. Search along the segment $x_{near}$ to $x_{extend}$ for collisions with the terrain.
5. If no collision is found, add the new node ($x_{extend}$) and the edge ($x_{near}$ to $x_{extend}$) to the tree.
6. Test to see if $x_{extend}$ can be connected directly to $x_{goal}$. If so, add the valid path from $x_{start}$ to $x_{goal}$. Go to step 1.

## C. Turns

The most significant kinematic constraint on the MAVs is the inability to instantaneously change direction. In an urban environment, we would of course want to be able to turn sharply. Since this is not possible, we must understand the true limitations on the MAV and plan accordingly. Once we can quantify how close the MAV will stay to the waypoint path, we can use this information in the RRT path planner.

Important research on trajectory generation is found in.[3,4,7] Kingston shows in[7] that due to control limits, the MAV has a limited local reachability region, bordered by minimum radius circles. In,[3] Anderson proved that a time-optimal trajectory along a waypoint path will be a sequence of straight-line path segments combined with arcs along these minimum radius circles, properly placed near the vertex. Anderson showed that a class of trajectories called $\kappa$-trajectories are time-extremal.



(a) Waypoint path smoothing with $\kappa$-trajectories (from[7]).

(b) Definitions of segments.

**Figure 3. $\kappa$-trajectories.**

**Definition III.1** *A $\kappa$-trajectory is defined as the trajectory that is constructed by following the line segment $\overline{w_{i-1}w_i}$ until intersecting $C_i$, which is followed until $C_{p(\kappa)}$ is intersected, which is followed until intersecting $C_{i+1}$, which is followed until the line segment $\overline{w_iw_{i+1}}$ is intersected, as shown in Figure 3(a).*

These $\kappa$-trajectories always pass over $p(\kappa)$. Depending on the situation, different values of $\kappa$ can be chosen. $\kappa = 0$ will guarantee a flight over the waypoint. $\kappa = 1$ will guarantee a minimum-length path by cutting each corner. Anderson also shows in[3] that $\kappa$ can be chosen to guarantee that the trajectory will be the same length as the waypoint path.

Knowing the control law the MAV is operating under and the trajectory it will follow, we can plan a waypoint path that can be tracked while avoiding collisions with the terrain. Referring to Figure 3(b), we can visualize the MAV moving along the segments from $w_{i-1}$ to $w_i$ to $w_{i+1}$. The 2-dimensional turn angle at $w_i$ is $\phi$. Our problem is figuring out how far we can expect the MAV to be from the straight-line path at any given point along the trajectory. Considering the straight line segment $\overline{w_iw_{i+1}}$, the optimal trajectory starts at $p_1$ ($p(\kappa)$), follows circle 1 through $p_2$ and $p_3$ to $p_4$, switches to circle 2 and follows it to $p_5$, then follows $\overline{w_iw_{i+1}}$ toward $w_{i+1}$. Due to symmetry, we can restrict attention to one half of the path. We divide $\overline{w_iw_{i+1}}$ into smaller segments, and in each segment we know the maximum perpendicular distance the trajectory will be from $\overline{w_iw_{i+1}}$ (both inside and outside the path). The segments representing these maximum distances are listed in Table 1.

From[4] we know that

$$\overline{w_ip_1} = \kappa R\left(\frac{1}{\sin(\frac{180-\phi}{2})} - 1\right) = \kappa R\left(\frac{1}{\cos(\frac{\phi}{2})} - 1\right). \tag{1}$$

Using this information and basic geometry we can determine the remaining distances listed in Table 1. The distance information is used in the RRT algorithm to determine collisions with the terrain. Paths that present a collision threat

| Segment on Path | Maximum Inside Distance | Maximum Outside Distance |
|---|---|---|
| $\overline{w_i q_1}$ | 0 | 0 |
| $\overline{q_1 p_2}$ | $\overline{p_1 q_1}$ | 0 |
| $\overline{p_2 q_4}$ | 0 | $\overline{p_3 q_2}$ |
| $\overline{q_4 p_5}$ | 0 | $\overline{p_4 q_4}$ |

**Table 1. Distances from Path to Trajectory**

are eliminated from the tree.

### D. Additional Features

We must briefly mention some other features of the a priori path planner that are not directly related to RRTs. Path smoothing is one of these features. Though it is not part of the RRT, it can improve the output and reduce the number of waypoints the MAV must traverse. Nodes in the path planned by the RRT planner are tested to see if they are truly necessary or if they can be skipped, with a direct edge from a previous node to a future node. Because of the incremental and random growth of the RRT, resultant paths will not be straight for long distances, even in long corridors where a straight path will do. The path smoothing step eliminates these extraneous nodes, which are important in growing the tree, but which can be later deleted.

The RRT path planner also allows for three different altitude modes: constant climbs, constant altitudes, and constant heights above ground. The RRTs have solved the general problem in 3 dimensions, and from that solution we can work on more specific problems. The standard mode is where the MAV can change altitude, irrespective of the terrain. This is effective in urban terrain flying, where it may be more efficient to climb and fly over low buildings instead of planning a longer path around them. Incremental collision testing for this mode is done along a constant climbing path (within the maximum slope limits) from $x_{near}$ to $x_{extend}$. Sometimes we want to fly at a constant altitude, and this is easily accomplished by mapping $x_{rand}$ to the constant altitude plane. We can also run the planner to plan paths at a constant height above ground. Here, the mapping is done from $x_{rand}$ to a state at the same $x$ and $y$ position, but at the designated height above the terrain. Incremental testing in this case must test between the two points to make sure there are no sharp altitude changes that the MAV sensors would not be able to quickly compensate for. In this mode, the resultant paths are over relatively smooth segments of the terrain.

## IV. Reactive Path Planner

The reactive path planner consists of three parts. The first is the laser ranger scanning algorithm. We note here that the scanning algorithm is used only in simulation. Our hardware laser ranger is mounted in a fixed position and cannot scan. Due to scanning limitations, it is critical that the laser ranger scan only in the most needed areas, which include the current trajectory of the MAV and any obstacles near the MAV that could present problems during a trajectory change. Second, the reactive planner tracks waypoint paths. Whether these paths are from the a priori path planner or the obstacle avoidance algorithm, the MAV needs to track each path quickly and accurately. Third is the obstacle avoidance path planner which considers all of the objects detected and plans waypoint paths around them.

The laser ranger is assumed to be mounted on a servo (in simulation) with adjustable azimuth with ranges to approximately 60 degrees in either direction. An elevation degree of freedom is also assumed. These two assumptions allow the MAV to scan directly ahead as well as any areas it might need to maneuver into. Third we assume that the laser ranger runs at 2-3 hertz. This allows it to run at low power and be small enough to mount on a MAV. Our last assumption is the existence of available memory to store laser ranger data.

### A. Laser Ranger

There are two algorithms that run in turn to gather information on obstacle locations. The first algorithm is to scan in the area of the MAV's expected trajectory. This provides the most pertinent obstacle information. The second algorithm scans the dimensions of obstacles already partially detected in an attempt to provide enough information to avoid them.

The first algorithm scans the expected trajectory of the MAV. This provides the first and most important information about what obstacles need to be avoided. We derived in[5] equations for determining the position of a fixed wing MAV $\sigma$ seconds in the future using the first order equations of motion:

$$\dot{N} = V \cos \psi \tag{2}$$
$$\dot{E} = V \sin \psi \tag{3}$$
$$\dot{\psi} = \frac{g}{V} \tan \phi \tag{4}$$
$$\dot{V} = \alpha_V (V^c - V) \tag{5}$$
$$\dot{\phi} = \alpha_\phi (\phi^c - \phi), \tag{6}$$

where $N$ and $E$ is the expected position of the MAV, $g$ is the gravitational constant, and $\psi$,$\phi$, and $V$ are the heading, roll angle, and airspeed of the MAV. Assuming a constant roll angle, equations (2-6) can be integrated to obtain the position of the MAV, $\sigma$ seconds from the current time $t$:

$$\hat{r}(\sigma; t) = \begin{pmatrix} N \\ E \end{pmatrix} + \frac{V^2}{g \tan \phi} \begin{pmatrix} -\cos(\psi + \frac{\sigma g \tan \phi}{V} + \cos \psi \\ \sin(\psi + \frac{\sigma g \tan \phi}{V} - \sin \psi \end{pmatrix}. \tag{7}$$

If initial points have been obtained about an objects location, the second algorithm can now begin to scan the dimensions of the object. We must assume there are two sets of data being stored. First, all the points of scanned obstacles. Second, the algorithm must record all of the areas already scanned where no obstacle was found. The latter is to prevent scanning in the same area twice. For simplification, we will refer to both sets of data as points. Since we want to disregard any points behind the MAV, we must remove them from consideration.

The remaining points are possible obstacles that need to be scanned. Assume there is a parameter called $triangle\_width$ that will be discussed later. Loop through the points and remove any point that has two other points that are within a Euclidian distance of $triangle\_width/2$. If a point is surrounded by two other points within $triangle\_width/2$ distance away, the obstacle avoidance algorithm assumes it is impossible to fly between them, therefore scanning from that middle point, does not provide additional information to the MAV. For example, in Figure 4, $p_2$ has a point to either side within $triangle\_width/2$ distance away. The obstacle avoidance algorithm will avoid the area between these three points. If the three points are all the information the MAV has about the object, the optimal direction to scan would be to the left of $p_1$ or to the right of $p_3$, depending on the current path of the MAV.

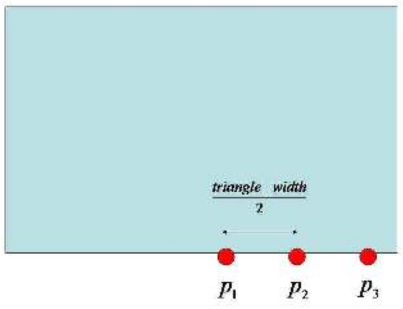


**Figure 4. Given that the red points have already been scanned, the MAV must choose which direction to scan. The best direction is to the left.**

From the remaining points, pick the closest obstacle point (not just a scanned point) to the MAV, that can be considered as the most important area to scan. If the point has one other point within $triangle\_width/2$ distance away, scan $triangle\_width/2$ in the opposite direction. If the point is by itself, then select the scan direction randomly with a bias toward the a priori planned trajectory.

Given the desired scan location, the last step is to calculate the commanded azimuth and elevation for the laser range finder. Given a point **v** that we want to scan and MAV position **u**, calculate the position of the point in vehicle coordinates as:

$$\begin{pmatrix} \tilde{v}_x \\ \tilde{v}_y \\ \tilde{v}_z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} \cos\psi & -\sin\psi & 0 & 0 \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & -u_x \\ 0 & 0 & 0 & -u_y \\ 0 & 0 & 0 & -u_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix}. \qquad (8)$$

The azimuth and elevation commands for the laser ranger are:

$$azimuth = \arctan(\tilde{v}_y/\tilde{v}_x) \qquad (9)$$

$$elevation = \arctan(\tilde{v}_z/\tilde{v}_y). \qquad (10)$$

## B. Reactive Planner

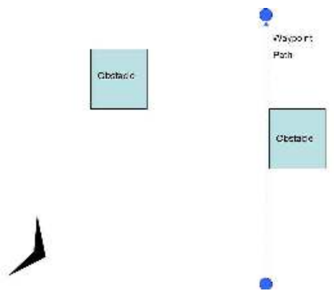
Consider the pop-up threat scenario shown in Figure 5. Figure 5(a) shows the starting position of the MAV as it enters a situation where obstacle avoidance is required. We assume the MAV has a forward ground velocity and is trying to track the given waypoint path. We also assume the laser ranger has detected enough points on objects to safely avoid the obstacle. The MAV detects the obstacles in figure 5(b). Notice that not all of the points are in front of the MAV. These points represent past obstacles that the laser ranger detected but are no longer relevant. These must be removed as in figure 5(c). To avoid the obstacles, we propose generating avoidance triangles around the obstacles and using the sides of the triangles as possible waypoint paths. Since the dimensions of the obstacles are unknown, they must be estimated by construction of a triangle around each point, as shown in Figure 5(d). Notice that some of the triangle sides intersect. If sides intersect, then they are not feasible paths around the obstacles. Figure 5(e) removes the intersecting sides. The remaining sides are all possible waypoint paths around the detected obstacles.

The MAV cannot always track each path generated before hitting an obstacle. In addition, some paths can be tracked faster than others. Using an estimation method described later, we can estimate at what point $\mathbf{r}$ the MAV can track each path. Using $\mathbf{r}$, we can determine whether the path can be tracked before hitting the obstacle. Infeasible paths can be removed as in figure 5(f). The remaining paths can be assigned a cost by using the distance from the MAV to $\mathbf{r}$, and the a path selected based on this cost. The final selected path is shown in figure 5(h)

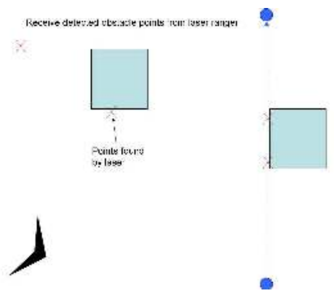
There are a few assumptions made in the reactive obstacle avoidance algorithm. First, we assume that the laser range finder has found enough points on pending obstacles so that the inner loop can plan a path around them. This implies that the inner loop and laser ranger scanning algorithm work together to provide sufficient data for obstacles. Second, we assume that the autopilot can quickly track desired headings. There is some room for error in this as will be seen below.

The steps to avoiding obstacles by generating intermediate waypoint paths make use of the limit cycle waypoint trajectory tracking method described in Ref. 12. The first step is to calculate the desired approach angle ($\psi_d$) to the current waypoint path which gives the heading the MAV needs to track to the obstacle avoidance algorithm. Using $\psi_d$, the MAV can always progress towards the waypoint path while still avoiding obstacles. The second step removes all points behind the MAV and farther than $1.25 * triangle\_length$ from the MAV. The reason for removing all points over a certain distance from the UAV is to prevent unneeded processing in deciding which path to follow. The third step is triangle generation. Given that the MAV cannot detect the exact size of the obstacle, the algorithm makes a buffer region around the point using a triangle. The triangle must be isosceles and the angle bisector of the odd angle must have the same heading as $\psi_d$ as shown in figure 6(a). Make a triangle for each point. The result will be multiple triangles that may overlap, but are all of equal size as in figure 6(b) Notice that $triangle\_width$ and $triangle\_length$ are two important dimensions of the triangles (figure 6(c)). $triangle\_length$ is the distance the MAV has to maneuver around the obstacle. If the MAV has a large turning radius, then $triangle\_length$ will have to be large as well. $triangle\_width$ is the buffer region between the obstacle point and the new waypoint path. As expected obstacle size grows, so will $triangle\_width$. There is further analysis on this in section V.

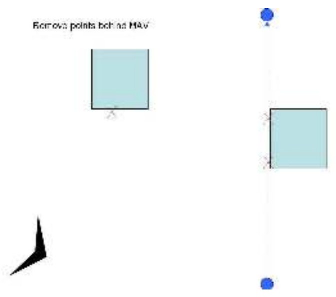
The next step is to remove intersecting sides. There will be at least two remaining sides after the intersection elimination. This is proven by the fact that each equivalent side is parallel. No matter how the triangles are arranged,



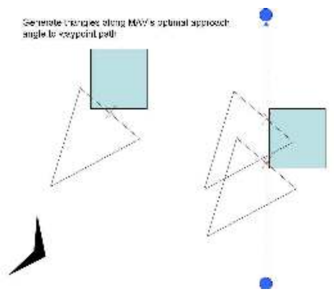
(a) MAV and waypoint path



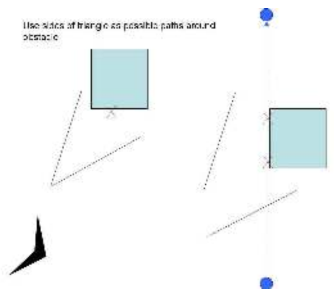
(b) Detected Obstacles



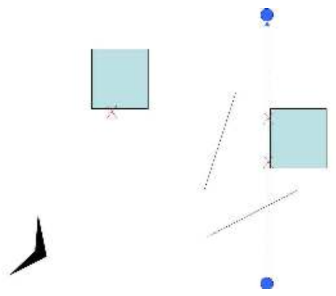
(c) Remove obstacles behind MAV

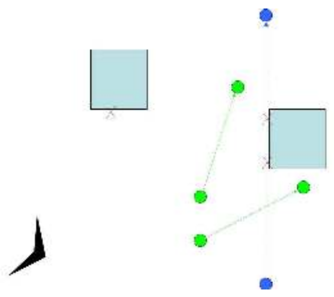


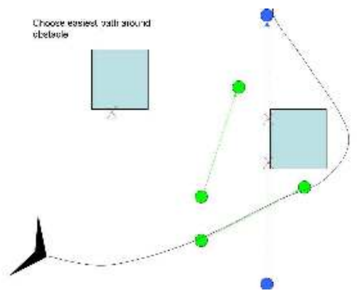
(d) Generated triangles



(e) Possible waypoint paths

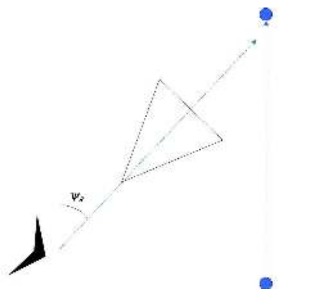

(f) Remove infeasible paths


(g) Two possible paths

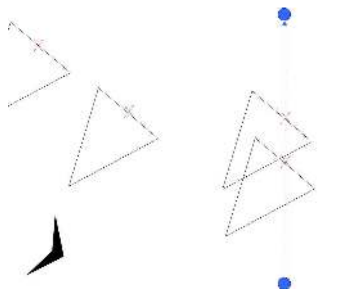


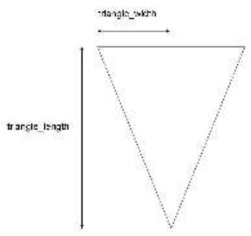
(h) Take path with smallest cost

**Figure 5. Reactive planner example. The MAV must detect and avoid obstacles while approaching a pre-planned waypoint path.**



(a) Optimal Approach Angle to waypoint path($\psi_d$).


(b) Generate triangles around points.



(c) Triangle dimensions.

**Figure 6. Triangle generation algorithm . Triangle sides are used as possible waypoint paths around obstacles.**

at least a set of non-parallel sides will remain. Because they do not intersect any other paths, and by our assumptions that we know all the points around an obstacle, then those paths also avoid obstacles.

There are two or more remaining triangle sides that can now be used as waypoint paths around the obstacle. The remaining task is to choose the best path. Each path is fairly close to the MAV and deviate equally from $\psi_d$. Unfortunately, there will still be some paths that cannot be tracked before colliding with an obstacle. We need a cost function that estimates how fast the MAV can track each path. A good cost function is simply the distance of the MAV to the first point on the waypoint path.

There are two cases to consider to estimate the point by which the MAV can track the waypoint path. First, determine whether the MAV needs to turn left or right to track the path. This can be done by calculating the cross product of the position vector of the MAV by the waypoint being approached minus the MAV position vector (e.g. $\mathbf{u} \times (\mathbf{w_2} - \mathbf{u})$). If the cross product is positive, the MAV must turn left, otherwise right. Now create a circle to the left or right of the MAV as needed using the turning radius. The circle should be positioned to represent the turn of the MAV. Translate the circle and waypoint path such that the center of the circle is at the origin. This simplifies the calculations. We must find where the circle intersects the waypoint path. First take the equation of the waypoint path:

$$y_i = m(x_i - u_x) + u_y \tag{11}$$

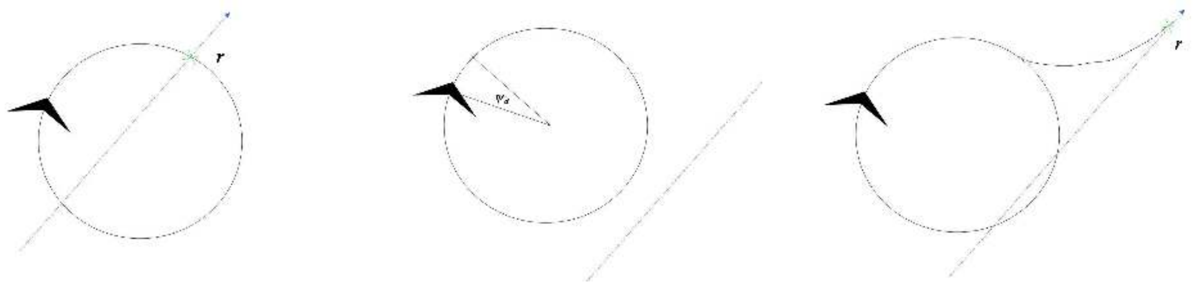
where $x_i$ and $y_i$ are the intersection points of the waypoint path and circle and $m$ is the slope. The equation of the circle is:

$$r_t^2 = x_i^2 + y_i^2. \tag{12}$$

Substituting and solving for $x_i$ we get

$$x_i = \frac{1}{2(1+m^2)}(2m^2 w_{2x} - 2m w_{2y} + 2\sqrt{-m^2 w_{2x}^2 + 2m w_{2x} w_{2y} - w_{2y}^2 + r_t + m^2 r_t}) \tag{13}$$

$y_i$ can be obtained by substituting $x_i$ back into equation (11). In the case that the square root is negative, the circle and waypoint path do not intersect and we know that we must use case two. If it is not negative, then we must find where $x_i$ and $y_i$ are on the circle. Translate the waypoint path such that it crosses through the center of the circle. Take the cross product of the waypoint path and $x_i$ and $y_i$. If the cross product is negative, use case two. Otherwise use case one. Case one consists of using $x_i$ and $y_i$ as the coordinate $\mathbf{r}$ which indicates the position at which the MAV will track the waypoint path as indicated in figure 7(a).



(a) Case 1. (b) $\psi_d$. (c) Case 2.

**Figure 7. Waypoint path interception.**

Case two projects the MAV position onto the waypoint path and adds an estimated distance. The estimated distance is calculated by finding $\psi_d$, the angle of the circle across the distance needed to fly before flying parallel to the waypoint path (figure 7(b)). That distance plus another two turning radii along the waypoint path from the projection are the estimated distance to $\mathbf{r}$, the point of tracking the waypoint path. This is illustrated in figure 7(c).

The cost function is the distance from the MAV to $\mathbf{r}$. The smallest cost is the path that can be tracked in the shortest distance. Choose the path with the smallest cost as the new waypoint path.

The reactive path planner is summarized by the following steps.

1. Calculate $\psi_d$.

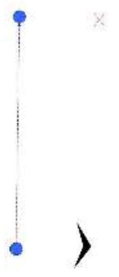
2. Remove all points behind the MAV and all points over $1.25 * triangle\_length$ distance away from the MAV.

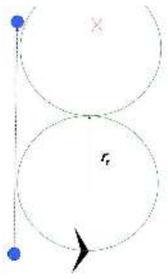
3. Generate triangles with dimensions $triangle\_width$ and $triangle\_length$ along the angle $\psi_d$.
4. Remove intersecting sides of triangles. The remaining sides are possible waypoint paths.
5. Calculate the estimated point of tracking for each of the waypoint paths ($\mathbf{r}$). The distance to this point is the cost for the corresponding waypoint path.
6. Choose the path with the smallest cost as the new waypoint path.

## V. Analysis

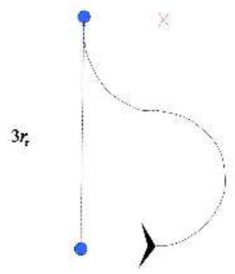
This section presents conditions under which the UAV is guarantees to track a generated path from the reactive planner. The constraints of a fixed wing air vehicle and limited processing efficiency severely limit finding feasible paths in all situations, but under certain conditions, successful avoidance can be guaranteed.

Consider figure 8(a). The red $X$ represents a detected obstacle and the blue dashed line is the desired waypoint path to track. The MAV is angled such that the obstacle is slightly in front, thus not filtering out the obstacle. To avoid the obstacle, the MAV must turn nearly 180 degrees toward the waypoint path and then turn back 90 degrees to begin tracking the path. This can be considered a worst case scenario. Any other scenario requires less distance to track the path.


(a) Worse case senerio



(b) Turning Radius



(c) Quickest path

**Figure 8. Analysis**

**Theorem V.1** *Given a MAV with turning radius $r_t$, parameter $triangle\_length \geq 3r_t$, and assuming (1) no wind, (2) sufficient points have been detected on a given obstacle, and (3) that the MAV is $triangle\_length$ distance away from the obstacle, the MAV can successfully track the waypoint path generated around that obstacle.*

*Proof:* Consider the situation where the MAV approaching a set of obstacles that requires it to track a path requiring at least a 90 degree heading change as depicted in figure 8(b). To track the path, the MAV must turn 180 degrees toward the path, and then turn another 90 degrees to track the path, as shown in figure 8(c). The MAV may cover up to $3r_t$ distance before tracking the path around the obstacle. The parameter $triangle\_length$ is the distance from an obstacle where a path begins. Therefore, if the UAV is at least $3r_t$ away from the obstacle, the MAV is guaranteed to avoid it. ■

## VI. Simulations Results

### A. Virtual World

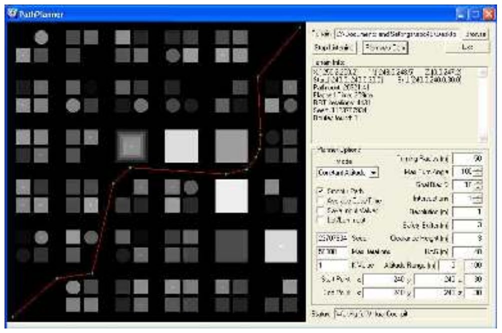
The goal of this section is to describe the simulation environment used in the paper. The simulator is the primary testing method for our architecture making it an important aspect in understanding the results. The simulator generates random cities through which the MAV can be flown. The parameters that can be changed are street width, building width, mean building height, and building height variance. The buildings created are placed on a flat terrain and can be detected by a simulated laser ranger. The buildings are rectangular and protrude vertically from the flat terrain surface.

The physics engine of the simulator models aerodynamic forces, gravitational forces, forces due to control surfaces, forces due to thrust, and forces due to wind. The MAV aerodynamic coefficients have been tuned to match the airframes used in our hardware testbed.

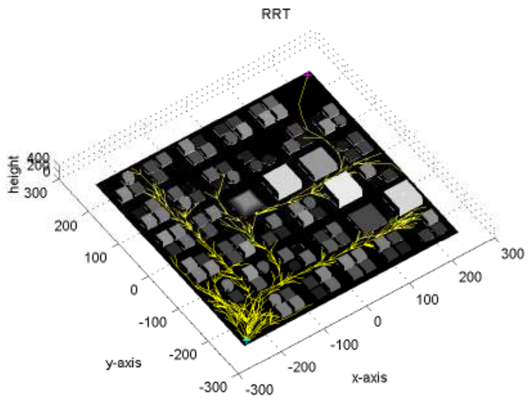
The simulator also emulates the Kestrel autopilot used at BYU.[1] It receives a .dll file of compiled Kestrel autopilot code and runs it at real time in an attempt to match the speed and timing of the MAV autopilot. The controls and sensors are run by the autopilot in the simulator in the sense that they are executed in hardware as algorithms. This method makes the simulation a close match to actual flight tests facilitating rapid prototyping.

## B. A Priori Planner

Both planning paths using RRTs and flying these paths in simulation have been successfully demonstrated. The RRT path planner was written in Visual C++ and experiments were conducted on a 2.4GHz Pentium 4 PC running Windows XP, with 512MB RAM. Figure 9(a) shows a random city that we generated for testing. Lighter buildings signify larger building heights. This city has 30m-wide streets. We can also see the options that can be input to the planner. One option that has a significant effect on the resultant path is the MAV turn radius. As turn radius decreases, path planning through narrow areas becomes easy. We must limit the turn radius to a realistic value. In simulation we use a 50m turn radius.



(a) A path through a city.



(b) MATLAB plot of the tree.

**Figure 9. The modified RRT algorithm is used to generate paths through synthetic urban terrain.**

We first test the planner in finding constant-altitude paths through this virtual city. Figure 9(b) shows a tree grown from one corner of the city to the opposite corner. The resultant path is then smoothed and can be seen in Figure 9(a). The average time it takes to plan such a path through this city is under .5s, but due to the randomness inherent in the algorithm, this time ranges from 15ms to 5s. As cities become more congested or the streets become narrower, computation time increases. In cities that don't have a full building on every block, the path planner can find paths very quickly, with street widths even down to 15m. The path planner explores the configuration space and tends to find the more open areas to turn in. In cities with tall buildings occupying up each block, the streets must be at least 30m wide to consistently find paths. When the path planning is allowed change altitude, the time to plan a path decreases because the MAV can fly over lower buildings. To fly through cities with narrow streets, this is often necessary.

It is reasonable to ask how effective this path planner would be in a realistic urban environment. Typical cities have streets ranging from 15m to 30m wide, with 20m streets being typical. Our testing showed that it would take a large grid of tall buildings occupying many contiguous blocks, with less than 30m between them, to prevent the path planner from finding a path. However, this situation is quite rare. While many cities have narrow streets, they also have open areas and some lower buildings over which the MAV can turn. Testing has shown that this planner will tend to find these areas rather than making several difficult turns.

In our simulator, we have successfully planned paths through different terrains (urban and others). We provide lateral control (following the waypoint path using the $\kappa$-trajectories) and longitudinal control (constant altitude, constant climb rate, or constant height above ground) to the simulated MAVs.

## C. Reactive Planner

Avoiding a single obstacle allows the algorithm parameters like $triangle\_width$ and $triangle\_length$ to be appropriately tuned. The parameters can be adjusted based on convergence speed to the desired waypoint, desired distance from obstacle during avoidance, and desired distance from obstacle to begin avoidance. In the simulation depicted in figure 10, the MAV was directed to fly on a waypoint path directly through a building. The building is represented as a solid gray box in the image with a width of 20 meters. The solid white line is the waypoint path. The dotted line is the actual path flown by the MAV. The parameters of the algorithm are:

$$\begin{aligned}\text{turning\_radius} &= 30\\ \text{triangle\_length} &= 3 \cdot \text{turning\_radius}\\ \text{triangle\_width} &= 35.\end{aligned}$$

The MAV used the virtual laser range finder running at three Hertz to detect the position of the obstacle and effectively avoided it.

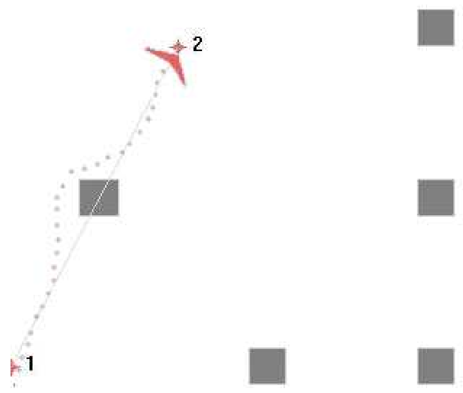


**Figure 10. Single obstacle simulation**

## VII. Flight Test Results

### A. A Priori Planner

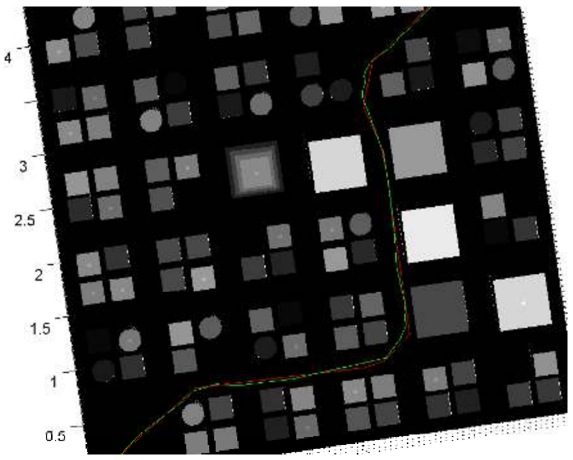
The RRT path planner was used to plan constant-altitude paths through a synthetic urban terrain with 30m-wide streets. The resultant waypoints were sent to a MAV in flight to test its ability to track the planed paths. Figure 11(a) superimposes the telemetry data on the synthetic terrain together with the planned waypoint path. From Figure 11(a) we see that the MAV was able to fly through this synthetic city without any collisions with the buildings. Similar results were obtained when the MAV flew different paths through the city. Although the MAV is not flying in wind conditions that accurately represent urban terrain, it was compensating for open space wind conditions, where wind speeds were roughly 30% of the airspeed.

We have also imported USGS terrain data (10m precision) into the RRT path planner to use in finding paths through real terrain. Using this data, we planned a path with GPS waypoints for the MAV to follow through a narrow canyon in Central Utah. The path planner quickly found a path free from collision with the terrain. The MAV successfully followed this path through the canyon while commanding a constant height above ground. Flight results are shown in Figure 11(b).

### B. Reactive Planner

A successful test of the reactive planner required at least one obstacle high enough to fly the MAV at without a need to change altitude low to the ground. This prevents the laser ranger from detecting points on the ground that might be mistakenly considered obstacles and allows for short altitude drops during turns.

We chose an isolated building on the BYU campus that is 50 meters high and 35 meters square. Other surrounding buildings rose to about 20 meters which separated the building as a single obstacle. We planned a path from the south side of the building to the north at an altitude of 40 meters. The MAV had no previous information on the location and dimensions of the building. The turning radius of the MAV is approximately $r_t = 30$ meters. We set


(a) Flight test data through synthetic urban terrain.

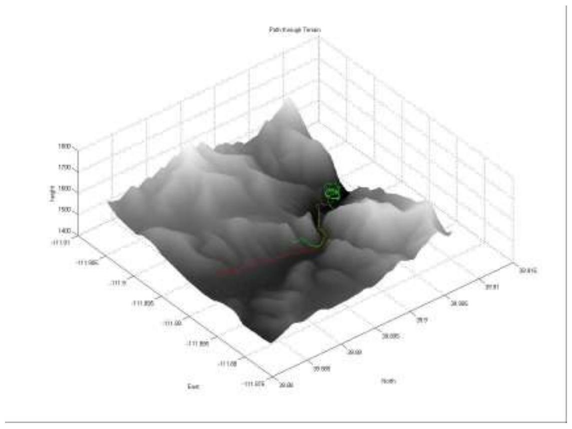

(b) Flight test data through canyon in southern Utah.

**Figure 11. Flight tests of the RRT algorithm though synthetic urban terrain and a canyon in southern Utah.**

$triangle\_length = 90$ meters as suggested by section V and $triangle\_width = 55$ meters to ensure that one detected point on the building generated a path sufficiently to the side of the building. A GPS telemetry plot of the results is shown in figure 12.

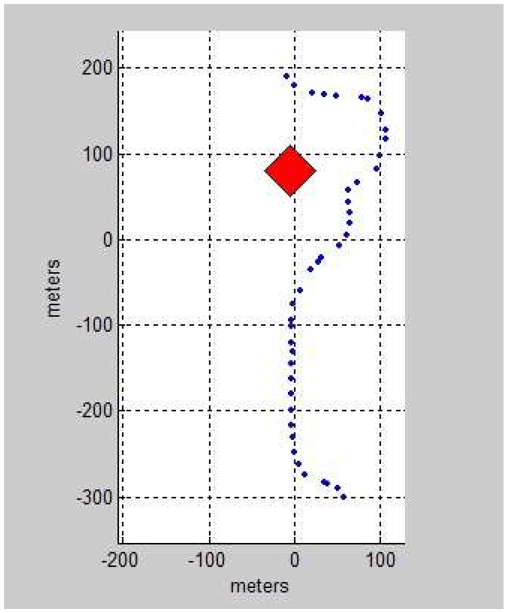


**Figure 12. Flight test results plot of a MAV avoiding a building with a planned path through the building.**

As the MAV approached the building, the laser ranger detected the building and calculated its position. When the MAV came within 90 meters of the building, the reactive planner generated a path around the building and the MAV began to track the path. Notice that as the MAV passed the building, it once again began to track the original waypoint path that went through the building. The MAV successfully avoided the building without human intervention.

## VIII. Conclusion

In this paper we have derived a two part solution to obstacle avoidance. The first part uses a modified RRT algorithm on an a priori terrain map to create a waypoint path. The second part uses a laser ranger to detect pop-up obstacles and generate new waypoint paths around them.

In flight we flew the a priori planner by generating a waypoint path using a synthetic city terrain map. We flew the generated waypoint path on a MAV. The MAV tracked the path well enough to avoid collision with buildings. The reactive planner was tested by planning a waypoint path through a building. The MAV successfully detected and avoided the building and continued to track the original waypoint path after avoidance.

The advantage of the two part algorithm is its computation time. The RRT algorithm can quickly compute waypoint paths on a ground station. The reactive planner can respond in real time to pop-up obstacles. The disadvantage is the suboptimal paths created by the a priori and reactive planners.

## Acknowledgments

This work was funded by AFOSR grant FA9550-04-C-0032.

## References

[1] http://procerusuav.com/.

[2] http://www.opti-logic.com/industrial_rangefinders.htm.

[3] E. Anderson. Extremal control and unmanned air vehicle trajectory generation. Master's thesis, Brigham Young University, April 2002.

[4] E. P. Anderson, R. W. Beard, and T. W. McLain. Real time dynamic trajectory smoothing for uninhabited aerial vehicles. *IEEE Transactions on Control Systems Technology*, 13(3):471–477, May 2005.

[5] Randal W. Beard, D.J. Lee, Morgan Quigley, Sarita Thakoor, and Steve Zornetzer. A new approach to observation of descent and landing of future mars mission using bioinspired technology innovations. *AIAA Journal of Aerospace Computing, Information, and Communication*, 2(1):65–91, January 2005.

[6] Emilio Frazzoli, Munther A. Dahleh, and Eric Feron. Real time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129, January-February 2002.

[7] D. Kingston. Implementation issues of real-time trajectory generation on small UAVs. Master's thesis, Brigham Young University, April 2004.

[8] J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 995–1001, San Francisco, CA, April 2000.

[9] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University, October 1998.

[10] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A. K. Peters, Wellesley, MA, 2001.

[11] Javier Minguez and Luis Montano. Nearness diagram (nd) navigation: Collision avoidance in troublesome scenarios. *Transactions on Robotics and Automation*, 20(1):45–59, February 2004.

[12] Derek Rich Nelson. Cooperative control of miniature air vehicles. Master's thesis, Brigham Young University, December 2005.

[13] Arvind U. Raghunathan, Vipin Gopal, Dharmashankar Subramanian, Lorenz T. Biegler, and Tariq Samad. Dynamic optimization strategies for three-dimensional conflict resolution of multiple aircraft. *Journal of Guidance, Control, and Dynamics*, 27(4):586–594, July–August 2004.