AIAA-2003-6559

AUTONOMOUS VEHICLE TECHNOLOGIES FOR SMALL FIXED WING UAVS

Derek Kingston[1], *Randal Beard[1], Timothy McLain[2], Michael Larsen[3], Wei Ren[1]

[1]Electrical and Computer Engineering
[2]Mechanical Engineering
Brigham Young University,
Provo, UT 84602

[3]Information Systems Laboratory, Inc.
San Diego, CA 92121

Abstract

Autonomous unmanned air vehicle flight control systems require robust path generation to account for terrain obstructions, weather, and moving threats such as radar, jammers, and unfriendly aircraft. In this paper, we outline a feasible, hierarchal approach for real-time motion planning of small autonomous fixed-wing UAVs. The approach divides the trajectory generation into four tasks: waypoint path planning, dynamic trajectory smoothing, trajectory tracking, and low-level autopilot compensation. The waypoint path planner determines the vehicle's route without regard for the dynamic constraints of the vehicle. This results in a significant reduction in the path search space, enabling the generation of complicated paths that account for pop-up and dynamically moving threats. Kinematic constraints are satisfied using a trajectory smoother which has the same kinematic structure as the physical vehicle. The third step of the approach uses a novel tracking algorithm to generate a feasible state trajectory that can be followed by a standard autopilot. Monte-Carlo simulations were done to analyze the performance and feasibility of the approach and determine real-time computation requirements. A planar version of the algorithm has also been implemented and tested in a low-cost micro-controller. The paper describes a custom UAV built to test the algorithms.

1 Introduction

The increasing power of computational resources makes possible the development of autonomous flight control systems which are capable of dealing with the complex task of path planning in dynamic and uncertain environments. Unmanned aerial vehicles (UAVs) require robust, real-time path generation to account for terrain obstructions, weather, and moving threats such as radar, jammers, and unfriendly aircraft. Such path planning algorithms and route navigation aids are needed to accomplish envisioned future UAV missions.[1]

There are two general approaches to trajectory generation: interpolation of a trajectory database and formulation of the trajectory as the solution to an optimal control problem. Methods which query and interpolate trajectory databases fall into several categories: probabilistic roadmaps,[2] lazy probabilistic roadmaps,[3] and rapidly-exploring trees.[4,5] Probabilistic roadmaps are path-planning algorithms which consist of off-line building of a graph of uniformly spaced randomly selected configurations called milestones. A recent extension[6] of the probabilistic roadmap approach uses Lyapunov function scheduling to deal with system dynamics in an environment with moving obstacles. For aerospace systems with complex high-dimensional dynamics, this motion planning approach is based on a quantization of system dynamics into a library of feasible trajectory primitives.[7]

Trajectory generation via numerical solution of optimal control problems[8–11] is computationally intensive and requires recently developed techniques from geometric nonlinear control theory for feasible implementation. These techniques are based on finding a differentially flat output for the system.[12–14] From such an output and its derivatives, the complete differential behavior of the system can by reconstructed. In Ref. 15, 16, a flat output is used to find a lower dimensional space in which trajec-

*Corresponding author, email:beard@ee.byu.edu

tory curves are generated using B-splines and sequential quadratic programming. In a similar vein, Refs. 17, 18 transform the nonlinear optimization problem to a linear one using feedback linearization, which requires finding a flat output, making it possible to convert constrained dynamic optimization problems into unconstrained ones. In Ref. 19, the differential flatness property was used to develop an iterative approach to finding a feasible solution which satisfies terminal path constraints using an H_∞ estimator.

The path planner described in this paper uses a modified Voronoi diagram[20] to generate possible paths. The Voronoi diagram is then searched via Eppstein's k-best paths algorithm.[21] Similar path planners have been previously reported in Refs. 22–24. The basic idea is to plan a polygonal path through a set of threats using a Voronoi algorithm in connection with an A* or Dijkstra algorithm. The polygonal paths are then made flyable by a trajectory smoother that dynamically smooths through the corners of the paths such that the curvature of the smoothed path is flyable by the UAV. The described algorithm works well when the minimum turning radius is small compared to the path links.

An overview of the system architecture is set forth in Section 2. Sections 3 describes the small UAVs used at BYU and the associated on-board computing and sensor hardware. Sections 4 and 5 explain our approach to the autopilot design, and the associated ground station. The trajectory tracker, trajectory smoother, and path planner are described in Sections 6, 7, and 8, respectively.

2 System Architecture

The architecture set forth in this paper is built around a systematic division of the problem into five distinct hierarchical layers: path planning, trajectory smoothing, trajectory tracking, autopilot, and the UAV. In this paper, paths refer to a series of waypoints which are *not* time-stamped, while trajectories will refer to time-stamped curves which specify the desired inertial location of the UAV at a specified time. Figure 1 shows a schematic of the architecture. At the top level is a path planner (PP). It is assumed that the PP knows the location of the UAV, the target, and the location of a set of threats. The PP generates a path

$$\mathcal{P} = \{v, \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N\}\},$$

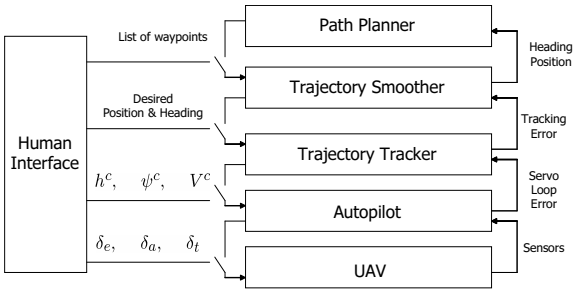

Figure 1: Proposed system architecture.

where $v \in [v_{\min}, v_{\max}]$ is a feasible velocity and $\{\mathbf{w}_i\}$ is a series of waypoints which define straight-line segments along which the UAV attempts to fly. Note that at the PP level, the path is compactly represented by $\mathcal{P}$. Higher level decision making algorithms reason and make decisions according to the data represented in $\mathcal{P}$.

The Trajectory Smoother (TS) transforms, in real-time, the waypoint trajectory into a time parameterized curve which defines the desired inertial position of the UAV at each time instant. The output of the TS is the desired trajectory $\mathbf{z}^d(t) = (z_x^d(t), z_y^d(t))^T$ at time t.

The Trajectory Tracker (TT) transforms $\mathbf{z}^d(t)$ into desired velocity command V^d, altitude command h^d, and heading command ψ^d. The autopilot receives these commands and controls the elevator, δ_e, and aileron, δ_a, deflections and the throttle setting δ_t.

Recognizing that it will be useful for human operators to interact with the UAV at different autonomy levels, careful attention has been given to the human interface. As shown in Figure 1, the human can interact with the UAV at the stick-and-throttle level, the autopilot command level, the time-parameterized trajectory level, or at the waypoint path planning level.

3 UAV Hardware

The BYU MAGICC lab currently operates a fleet of four small, low cost, fixed-wing UAVs, one of which is shown in Figure 2. Specifications for two of the airframes are given in Table 3. Our UAVs are constructed from EPP foam and are extremely durable, having survived multiple crashes. The airframe is an in-house design patterned after ZAGI gliders, which are popular dogfight planes in the RC hobby community.[25] The UAVs are powered by an

Figure 2: BYU UAV MAGICC II.

electric motor in a push propeller configuration and are hand launched and belly landed. The plane is actuated by two elevons. Fixed wing tips provide vertical stabilization.

	MAGICC I	MAGICC II
Wingspan	60 in.	38 in.
Payload	32 oz.	8 oz.
Flight time	15 min.	30 min.
Cruise Speed	30 mph	35 mph
Max Speed	45 mph	65 mph
Min Speed	15 mph	15 mph

Table 1: Specifications for BYU UAVs.

In addition to the airframe, we have designed the autopilot board shown in Figure 3. The CPU on the autopilot is a 29 MHz Rabbit microcontroller with 512K Flash and 512K RAM. The autopilot has four servo channels, two 16 channel, 12 bit analog-to-digital converters, four serial ports, and five analog inputs. On-board sensors include three-axis rate gyros with a range of 300 degrees per second, three axis accelerometers with range of two g's, an absolute pressure sensor capable of measuring altitude to within two feet, a differential pressure sensor capable of measuring airspeed to within 0.36 feet per second, and a standard GPS receiver. The autopilot package weighs 2.25 ounces including the GPS antenna. The size of the autopilot is roughly 3.5 inches by 2 inches.

Communication between the airplane and the ground station is accomplished via a low-cost 900 MHz wireless modem.

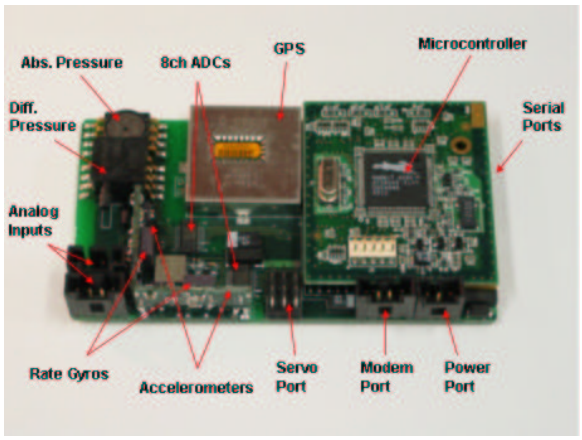

Figure 3: BYU autopilot hardware.

4 Autopilot Design

In this section we briefly describe the autopilot design as well as the design of a three state Kalman filter to produce an attitude estimate. Following standard procedures,[26–28] we have assumed that the longitudinal and lateral dynamics of the UAV are decoupled and have designed longitudinal and lateral autopilots independently. As shown in Figure 4, the inputs to the longitudinal autopilot are commanded altitude, h^c and commanded velocity, V^c. The outputs are the elevator deflection, δ_e, and the throttle command, δ_t. The Altitude Hold autopilot converts altitude error into a commanded pitch angle θ^c. The Pitch Attitude Hold autopilot converts pitch attitude error into a commanded pitch rate q^c. The Pitch Rate Hold autopilot converts pitch rate error to elevator command. The Velocity Hold autopilot converts velocity error to throttle command.

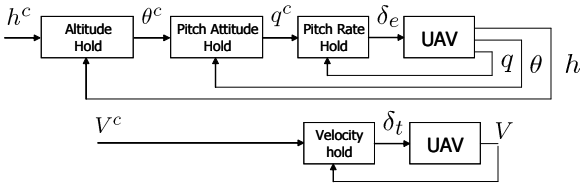

Figure 4: Autopilot for longitudinal motion.

The lateral autopilot is shown in Figure 5. The input command to the lateral autopilot is the commanded heading, ψ^c. The output is the aileron command δ_a. The Heading Hold autopilot converts heading error to roll attitude command, ϕ^c. The Roll Attitude Hold autopilot converts roll angle error to roll rate command, p^c. The Roll Rate Hold autopilot converts the roll rate error to aileron com-

mand, δ_a. Each autopilot mode is realized with a PID controller augmented with output saturation and integrator anti-windup.

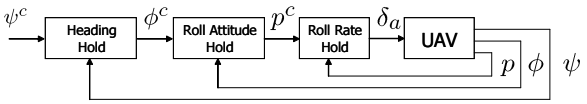

Figure 5: Autopilot for lateral motion.

Unfortunately, roll and pitch angles are not directly measurable with low-cost sensors. In addition, heading angle is measured with GPS at very low data rates. To compensate, the autopilot has been augmented with an Extended Kalman Filter (EKF) to estimate roll, pitch, and heading angle. The EKF uses rate gyro information to do the time update and the accelerometers in combination with the airspeed sensor to do the measurement update.

The EKF time update equations are as follows:

$$\dot{\phi} = p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \quad (1)$$

$$\dot{\theta} = q\cos\phi - r\sin\phi \quad (2)$$

$$\dot{\psi} = q\sin\phi\sec\theta + r\cos\phi\sec\theta \quad (3)$$

where p, q, and r are roll, pitch, and yaw rates as measured by on-board rate gyros. A simple Euler approximation is used to convert to a discrete update equation.

Part of the complication of estimating roll and pitch angles is the lack of a good sensor to measure them directly. However, roll and pitch angles can be approximated using accelerometer measurements. The basic idea is to use an estimate of the direction of the gravity vector to extract roll and pitch angles. This can be done from the following set of equations:

$$\dot{u} = -g\sin\theta + A_x + vr - wq \quad (4)$$

$$\dot{v} = g\sin\phi\cos\theta + A_y - wr + wp \quad (5)$$

$$\dot{w} = g\cos\phi\cos\theta + A_z + uq - vp \quad (6)$$

where A_* are the accelerometer measurements, and u, v, w are the body velocities of the UAV. Because u, v, and w are not measured, some simplifying assumptions need to be made. First, we assume that $\dot{u}$, $\dot{v}$, and $\dot{w}$ are zero. This is true over short periods of time as well as in steady-state flight. The next assumption is to notice that in steady-state flight (coordinated turn or level flight) $w \approx 0$. The last assumption is that u and v are some unknown factor of airspeed (V_p). Through analysis of flight data on our UAVs the following have been determined to be decent measurements of roll and pitch:

$$\theta_m = \sin^{-1}\left(\frac{A_x - 0.2V_pq}{g}\right), \quad (7)$$

$$\phi_m = \sin^{-1}\left(\frac{-A_y + 0.4V_pr}{g\cos\theta_m}\right). \quad (8)$$

These measurements, along with a GPS heading measurement, are used to do the EKF measurement update. Figure 6 shows the filter output (blue) versus the measured values (green). This was generated from real data gathered from the airplane.

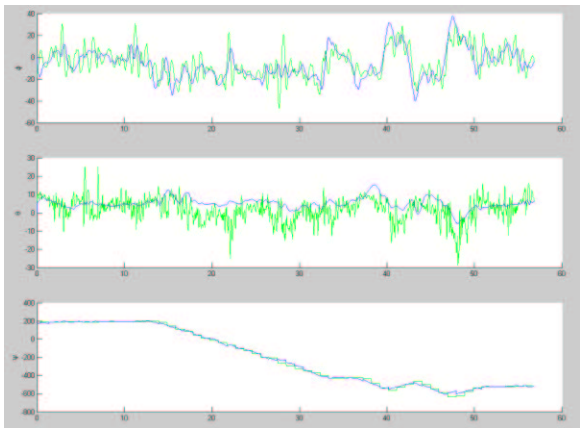

Figure 6: EKF (blue) vs measured (green) of roll, pitch, and heading

5 Ground Station

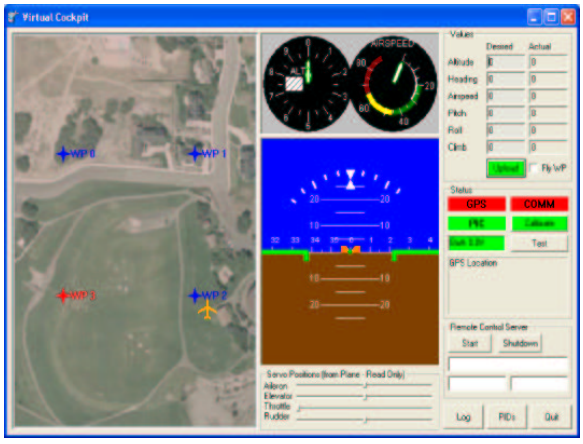

Figure 7: Virtual Cockpit Interface

Essential to development, debugging, and visualization is the Ground Station. This software package

allows easy interface to everything on the UAV; from raw analog-to-digital sensor readings to the current PID values in the control loops. Every second, status packets are sent to the Ground Station from the UAV over a 900 MHz wireless link to indicate the state of the airplane and its controllers. This allows for real-time plotting of position (map and waypoints), altitude, airspeed, etc. It also allows the user to be aware of GPS status, battery voltage, and communication link status.

The Ground Station was designed primarily to help tune the autopilot. To this end, a user-configurable data-logging tool was added. The Data-logger commands the autopilot to store the state of the airplane for a specified period of time. When the log is completed, it is transmitted back to the Ground Station for viewing. This is especially helpful to see what the UAV was planning and commanding during maneuvers. Essentially everything the autopilot keeps track of can be data-logged, allowing the user to reconstruct a flight as the autopilot viewed it. This capability has been used to debug the autopilot, build Kalman filters, and develop waypoint navigation capability.

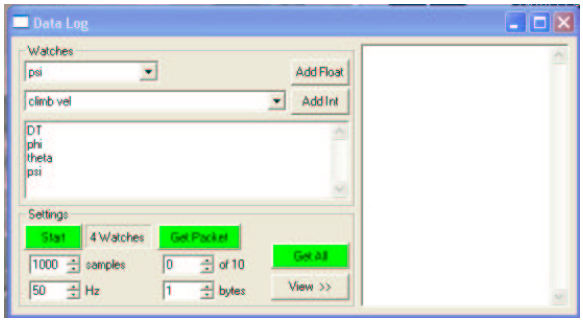

Figure 8: User-Configurable Data Log Interface

Because the autopilot control system is made up of PID loops, tuning the PID values is very important to the performance of the UAV. To facilitate this, a real-time PID tuning and graphing utility is integrated into the Ground Station. It allows the user to request and set PID values on any loop while also providing the capability to command steps, ramps, and square waves to the different loops. These commands are plotted next to the performance of the UAV in real-time. Using the graphical information, the user can easily adjust values to tune the loops to desired specifications. This autopilot has been flown on a number of different platforms, each requiring the PID loops to be re-tuned. Using the Ground Station has accelerated the tuning process, so that the autopilot can be tuned for a new UAV less than 15 minutes.

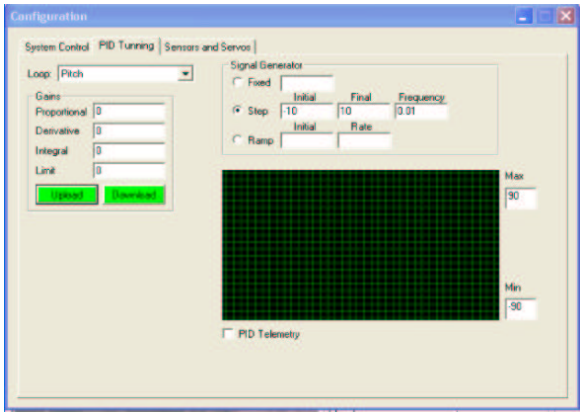

Figure 9: Ground-station: PID Tuning.

To enable research geared toward designing robust and intuitive user interfaces, the Ground Station allows other platforms to connect to the UAV via a simple Application Program Interface (API). This allows any device to become the interface to the UAV. Our UAV has been controlled through a voice activated headset and also through a hand-held PDA.

6 Trajectory Tracker

This section gives a brief overview of our trajectory tracker. A complete description is contained in Refs. 29,30. We will assume that the UAV/autopilot combination is adequately modeled by the kinematic equations

$$\begin{aligned} \dot{x} &= v^c \cos(\psi) \\ \dot{y} &= v^c \sin(\psi) \\ \dot{\psi} &= \omega^c, \end{aligned} \tag{9}$$

where (x, y) is the inertial position of the UAV, ψ is its heading, v^c is the commanded linear speed, and ω^c is the commanded heading rate. The dynamics of the UAV impose input constraints of the form

$$\begin{aligned} 0 < v_{min} \leq & v^c \leq v_{max} \\ -\omega_{max} \leq & \omega^c \leq \omega_{max}. \end{aligned} \tag{10}$$

As we will describe in the next section, the trajectory generator produces a reference trajectory that satisfies

$$\begin{aligned} \dot{x}_r &= v_r \cos(\psi_r) \\ \dot{y}_r &= v_r \sin(\psi_r) \\ \dot{\psi}_r &= \omega_r \end{aligned} \tag{11}$$

under the constraints that v_r and ω_r are piecewise continuous and satisfy the constraints

$$\begin{aligned} v_{min} + \epsilon_v \leq & v_r \leq v_{max} - \epsilon_v \\ -\omega_{max} + \epsilon_\omega \leq & \omega_r \leq \omega_{max} - \epsilon_\omega, \end{aligned} \tag{12}$$

where ϵ_v and ϵ_ω are positive control parameters.

The trajectory tracking problem is complicated by the nonholonomic nature of Equations (9) and the input constraint on the commanded speed v^c.

The first step in the design of the trajectory tracker is to transform the tracking errors to the UAV body frame as follows:

$$\begin{bmatrix} x_e \\ y_e \\ \psi_e \end{bmatrix} = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \psi_r - \psi \end{bmatrix}. \tag{13}$$

Accordingly, the tracking error model can be represented as

$$\begin{aligned} \dot{x}_e &= \omega^c y_e - v^c + v_r \cos(\psi_e) \\ \dot{y}_e &= -\omega^c x_e + v_r \sin(\psi_e) \\ \dot{\psi} &= \omega_r - \omega^c. \end{aligned} \tag{14}$$

Following Ref. 31, Eq. (14) can be simplified to

$$\begin{aligned} \dot{x}_0 &= u_0 \\ \dot{x}_1 &= (\omega_r - u_0) x_2 + v_r \sin(x_0) \\ \dot{x}_2 &= -(\omega_r - u_0) x_1 + u_1, \end{aligned} \tag{15}$$

where

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \psi_e \\ y_e \\ -x_e \end{bmatrix} \tag{16}$$

and

$$\begin{bmatrix} u_0 \\ u_1 \end{bmatrix} = \begin{bmatrix} \omega_r - \omega^c \\ v^c - v_r \cos(x_0) \end{bmatrix}.$$

The input constraints under the transformation become

$$\begin{aligned} -\epsilon_\omega \leq & u_0 \leq \epsilon_\omega \\ v_{min} - v_r \cos(x_0) \leq & u_1 \leq v_{max} - v_r \cos(x_0). \end{aligned} \tag{17}$$

Obviously, Eqs. (13) and (16) are invertible transformations, which means $(x_0, x_1, x_2) = (0, 0, 0)$ is equivalent to $(x_e, y_e, \psi_e) = (0, 0, 0)$ and $(x_r, y_r, \psi_r) = (x, y, \psi)$. Therefore, the original tracking control objective is converted to a stabilization objective, that is, our goal is to find feasible control inputs u_0 and u_1 to stabilize x_0, x_1, and x_2.

Note from Eq. (15) that when both x_0 and x_2 go to zero, that x_1 becomes uncontrollable. To avoid this situation we introduce another change of variables. Let

$$\bar{x}_0 = m x_0 + \frac{x_1}{\pi_1}, \tag{18}$$

where $m > 0$ and $\pi_1 \triangleq \sqrt{x_1^2 + x_2^2 + 1}$. Accordingly, $x_0 = \frac{\bar{x}_0}{m} - \frac{x_1}{m\pi_1}$. Obviously, $(\bar{x}_0, x_1, x_2) = (0, 0, 0)$ is equivalent to $(x_0, x_1, x_2) = (0, 0, 0)$. Therefore it is sufficient to find control inputs u_0 and u_1 to stabilize $\bar{x}_0$, x_1, and x_2. With the same input constraints (17), Eq. (15) can be rewritten as

$$\begin{aligned} \dot{\bar{x}}_0 &= (m - \frac{x_2}{\pi_1}) u_0 + \frac{x_2}{\pi_1} \omega_r \\ &\quad + \frac{1 + x_2^2}{\pi_1^3} v_r \sin\left(\frac{\bar{x}_0}{m} - \frac{x_1}{m\pi_1}\right) - \frac{x_1 x_2}{\pi_1^3} u_1 \\ \dot{x}_1 &= (\omega_r - u_0) x_2 + v_r \sin(\frac{\bar{x}_0}{m} - \frac{x_1}{m\pi_1}) \\ \dot{x}_2 &= -(\omega_r - u_0) x_1 + u_1. \end{aligned} \tag{19}$$

In Refs. 29, 30 we have shown that if

$$u_0 = \begin{cases} -\eta_0 \bar{x}_0, & |\eta_0 \bar{x}_0| \leq \epsilon_\omega \\ -\text{sign}(\bar{x}_0)\epsilon_\omega, & |\eta_0 \bar{x}_0| > \epsilon_\omega \end{cases} \tag{20}$$

$$u_1 = \begin{cases} \underline{v}, & -\eta_1 x_2 < \underline{v} \\ -\eta_1 x_2, & \underline{v} \leq -\eta_1 x_2 \leq \bar{v} \\ \bar{v}, & -\eta_1 x_2 > \bar{v} \end{cases}, \tag{21}$$

where $\underline{v} = v_{\min} - v_r \cos\left(\frac{\bar{x}_0}{m} - \frac{x_1}{m\pi_1}\right)$ and $\bar{v} = v_{\max} - v_r \cos\left(\frac{\bar{x}_0}{m} - \frac{x_1}{m\pi_1}\right)$, and η_0 and η_1 are sufficiently large (made precise in Refs. 29,30), then the tracking error goes to zero asymptotically.

Note the computational simplicity of Equations (20)-(21). We have found that the tracker can be efficiently implemented on the autopilot hardware discussed in Section 3.

Figure 10 shows a reference trajectory of the UAV in green and the actual UAV trajectory in blue. Figure 11 plots the tracking errors verses time and demonstrates the asymptotically stable characteristics of the trajectory tracker.

7 Trajectory Smoother

The Trajectory Smoother (TS) translates a straight-line path into a feasible trajectory for a UAV with velocity and heading rate constraints. Our particular implementation of trajectory smoothing also has some nice theoretical properties including time-optimal waypoint following.[32,33]

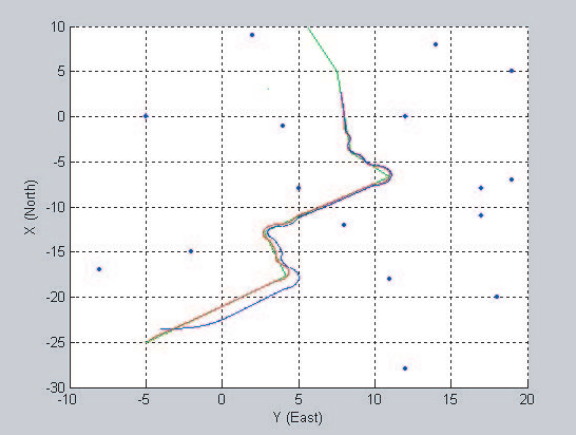

Figure 10: The simulation scenario: waypoint path (green), smoothed reference trajectory (red), and actual trajectory (blue).

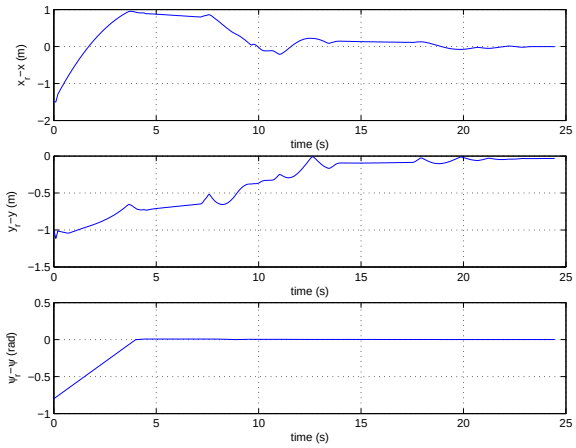

Figure 11: The trajectory tracking errors expressed in the inertial frame.

We start by assuming that an auto-piloted UAV is modeled by the kinematics equations given in Eq. (9), with the associated constraints given in Eq. (10).

The fundamental idea behind feasible, time-extremal trajectory generation is to impose on TS a mathematical structure similar to the kinematics of the UAV. The structure of the TS is given in Eq. (11) with constraints given by Eq. (12). To simplify notation, let $c = \omega_{\max} - \epsilon_\omega$.

With the velocity fixed at $\hat{v}$, the minimum turn radius is defined as $R = \hat{v}/c$. The idea of a minimum turn radius allows us to visualize the area of space that the UAV can reach in the next instant of time, i.e. the local reachability region.

With this in mind, it seems natural that for a

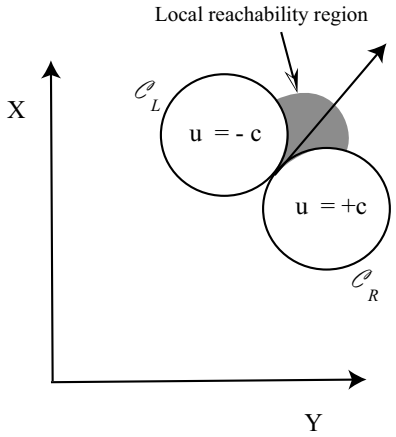

Figure 12: Local reachability region of the TS.

trajectory to be time-optimal, it will be a sequence of straight-line path segments combined with arcs along the minimum radius circles (i.e. along the edges of the local reachability region). In fact, Anderson proved in Ref. 34 that this is the case. By postulating that ω_r follows a bang-bang control strategy during transitions from one path segment to the next, he showed that a κ-trajectory is time-extremal, where a κ-trajectory is defined as follows.

Definition 7.1 *As shown in Figure 13, a κ-trajectory is defined as the trajectory that is constructed by following the line segment $\overline{\mathbf{w}_{i-1}\mathbf{w}_i}$ until intersecting $\mathcal{C}_i$, which is followed until $\mathcal{C}_{\mathbf{p}(\kappa)}$ is intersected, which is followed until intersecting $\mathcal{C}_{i+1}$ which is followed until the line segment $\overline{\mathbf{w}_i\mathbf{w}_{i+1}}$ is intersected.*

Note that different values of κ can be selected to satisfy different requirements. For example, κ can be chosen to guarantee that the UAV explicitly passes over each waypoint, or κ can be found by a simple bisection search to make the trajectory have the same length as the original straight-line path,[34] thus facilitating timing-critical trajectory generation problems.

The TS implements κ-trajectories to follow waypoint paths. In evaluating the real-time nature of the TS, we chose to require trajectories to have equal path length as the initial straight line paths. The computational complexity to find ω_r is dominated by finding circle-line and circle-circle intersections. Since the TS also propagates the state of the system in response to the input ω_r, Eq. (11) must be solved in real-time. This is done via a forth-order Runge-Kutta algorithm.[35] In this manner, the out-

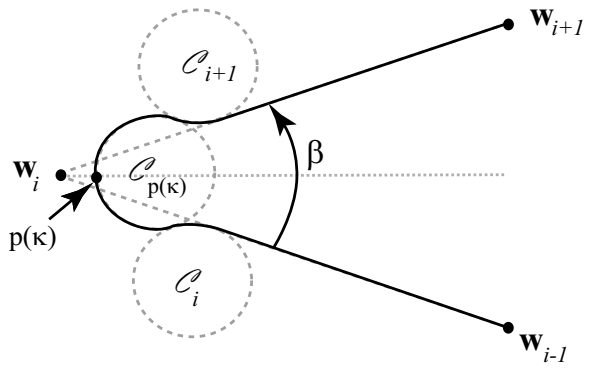

Figure 13: A dynamically feasible κ-trajectory.

put of the TS corresponds in time to the evolution of the UAV dynamics and ensures a time-optimal trajectory.

Hardware implementation of the TS has shown the real-time capability of this approach. On a 1.8 GHz Intel Pentium 4 processor, one iteration of the TS took on average 36 μ-seconds. At this speed, the TS could run at 25 KHz - well above the dynamic range of typical UAVs. Moving toward embedded systems, we found that one iteration of the TS required a maximum of 47 milli-seconds on a Rabbit Microprocessor running at 29 MHz. The low computational demand allows the TS to be run in real-time at approximately 20 Hz on an embedded system onboard the UAVs described in Section 3.

8 Waypoint Path Planning

For many anticipated military and civil applications, the capability for a UAV to plan its route as it flies is important. Reconnaissance, exploration, and search and rescue missions all require the ability to respond to sensed information and to navigate on the fly.

In the flight control architecture shown in Figure 1, the coarsest level of route planning is carried out by the path planner (PP). Our waypoint planning technique centers around the construction and search of a Voronoi graph.[23] The Voronoi graph provides a method for creating waypoint paths through threats or obstacles in the airspace. A prime advantage of the Voronoi graph is the computational speed with which the graph can be created and searched.

In our work, we have modeled threats and obstacles in two different ways: as points to be avoided and as polygonal regions that cannot be entered. With threats specified as points, construction of the Voronoi graph is straightforward using existing algorithms. For an area with n point threats, the Voronoi graph will consist of n convex cells, each containing one threat. Every location within a cell is closer to its associated threat than to any other. By using threat locations to construct the graph, the resulting graph edges form a set of lines that are equidistant from the closest threats. In this way, the edges of the graph maximize the distance from the closest threats. Figure 14 shows an example of a Voronoi graph constructed from point threats.

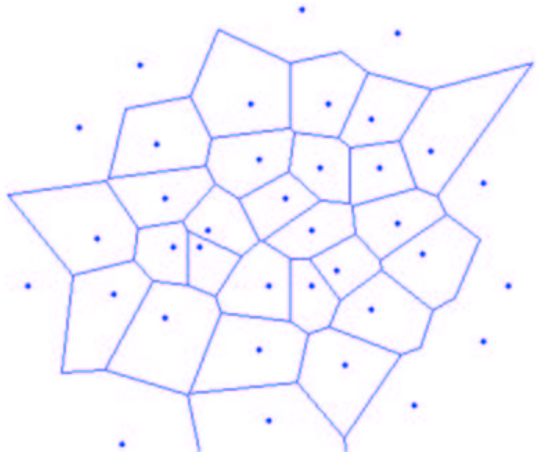

Figure 14: Voronoi graph with point threats

Construction of a Voronoi graph for obstacles modeled as polygons is an extension of the point-threat method. In this case, the graph is constructed using the vertices of the polygons that form the periphery of the obstacles. This initial graph will have numerous edges that cross through the obstacles. To eliminate these infeasible edges, a pruning operation is performed. Using a line intersection algorithm, those edges that intersect the edges of the polygon are detected and removed from the graph. Figure 15 shows the initial polygon based graph and the final graph after pruning.

Finding good paths through the Voronoi graph requires the definition of a cost metric associated with traveling along each edge. In our work, two metrics have been employed: path length and threat exposure. A weighted sum of these two costs provides a means for finding safe, but reasonably short paths. Although the highest priority is usually threat avoidance, the length of the path must be considered to

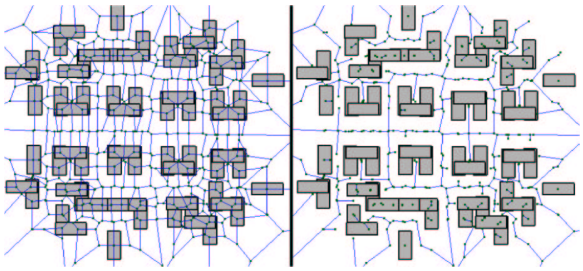

Figure 15: Voronoi graph before and after pruning with polygon threats

prevent safe, but meandering paths from being chosen.

Once a metric is defined, the graph is searched using an Eppstein search.[21] This is an computationally efficient search with the ability to return k best paths through the graph. Once k best paths are known, a coordination agent can choose which path to choose for each vehicle in the team (to ensure simultaneous arrival, for example). The points of this chosen path are passed on the the trajectory generator which smooths through the path, taking into account the dynamics and constraints of the vehicle.

9 Conclusion

This paper has described an approach to autonomous control for fixed-wing UAVs. In particular, we have described small UAV hardware, low-cost, light-weight autopilot technologies, and a computationally efficient approach to real-time trajectory generation.

Acknowledgements

This work was funded by AFOSR grants F49620-01-1-0091 and F49620-02-C-0094, and by DARPA grant NBCH1020013.

References

[1] *Uninhabited Air Vehicles: Enabling Science for Military Systems*. National Academy Press, 2000.

[2] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Proabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[3] R. Bohlin and L. Kavraki, "Path planning using lazy rpm," in *Proceedings of the IEEE International Conference on Robotics an Automation*, 1997.

[4] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," in *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, 1999.

[5] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Technical Report 98-11, Iowa State University, Ames,IA, Oct. 1998.

[6] E. Frazzoli, M. Dahleh, and E. Ferron, "Real-time motion planning for agile autonomous vehicles," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, (Denver, CO), August 2000. AIAA Paper No. AIAA-2000-4056.

[7] E. Frazzoli, M. Dahleh, and E. Ferron, "Robust hybrid control for autonomous vehicle motion planning," Technical Report LIDS-P-2468, Massachusetts Institute of Technology, Cambridge, MA, Oct. 1999. submitted to the *IEEE Transactions on Automatic Control*.

[8] C. Hargraves and S. Paris, "Direct trajectory optimization using nonlinear programming and collocation," *AIAA J. Guidance and Control*, vol. 10, pp. 338–342, 1987.

[9] O. von Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals of Operation Research*, vol. 37, pp. 357–373, 1992.

[10] Y. Chen and J. Huang, "A new computational approach to solving a class of optimal control problems," *International Journal of Control*, vol. 58, no. 6, pp. 1361–1383, 1993.

[11] L. Singh and J. Fuller, "Trajectory generation for a UAV in urban terrain, using nonlinear MPC," in *Proceedings of the American Control Conference*, (Arlington, VA), 2001.

[12] A. Isidori, *Nonlinear Control Systems*. Communication and Control Engineering, New York, New York: Springer Verlag, 2nd ed., 1989.

[13] B. Charlet, J. Levine, and R. Marino, "On dynamic feedback linearization," *Systems and Control Letters*, vol. 13, pp. 143–151, 1989.

[14] M. Fliess, J. Levine, P. Martin, and P. Rouchon, "Flatness and defect of non-linear systems: introductory theory and examples," *International Journal of Control*, vol. 61, no. 6, pp. 1327–1360, 1995.

[15] M. Milam and R. M. K. Mushambi, "A new computational approach to real-time trajectory generation for constrained mechanical systems," in *Proceedings of the IEEE Conf. on Decision and Control*, (Sydney, Australia), pp. 845–851, December 2000.

[16] M. Milam, R. Franz, and R. Murray, "Real-time constrained trajectory generation applied to a flight

control experiment," in *Proceedings of the International Federation of Automatic Control Conference*, 2002.

[17] S. Agrawal and N. Faiz, "Optimization of a class of nonlineaer dynamic systems: new efficient method without Lagrange multipliers," *J. Optimization Theory and Applications*, vol. 97, no. 1, pp. 11–28, 1998.

[18] N. Faiz, S. K. Agrawal, and R. M. Murray, "Trajectory planning of differentially flat systems with dynamics and inequalities," *AIAA Journal of Guidance, Control and Dynamics*, vol. 24, pp. 219–227, March–April 2001.

[19] G. J. Toussaint, T. Basar, and F. Bullo, "Motion planning for nonlinear underactuated vehicles using H^∞ techniques," in *American Control Conference*, 2001.

[20] R. Sedgewick, *Algorithms*. Addison-Wesley, 2nd ed., 1988.

[21] D. Eppstein, "Finding the k shortest paths," *SIAM Journal of Computing*, vol. 28, no. 2, pp. 652–673, 1999.

[22] T. McLain and R. Beard, "Cooperative rendezvous of multiple unmanned air vehicles," in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, (Denver, CO), August 2000. Paper no. AIAA-2000-4369.

[23] P. Chandler, S. Rasumussen, and M. Pachter, "UAV cooperative path planning," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, (Denver, CO), August 2000. AIAA Paper No. AIAA-2000-4370.

[24] R. W. Beard, T. W. McLain, M. Goodrich, and E. P. Anderson, "Coordinated target assignment and intercept for unmanned air vehicles," *IEEE Transactions on Robotics and Automation*, vol. 18, pp. 911–922, December 2002.

[25] http://zagi.com.

[26] R. C. Nelson, *Flight Stability and Automatic Control*. Boston, Massachusetts: McGraw-Hill, 2nd ed., 1998.

[27] M. Rauw, *FDC 1.2 - A SIMULINK Toolbox for Flight Dynamics and Control Analysis*, February 1998. Available at http://www.mathworks.com/...

[28] J. Roskam, *Airplane Flight Dynamics and Automatic Flight Controls, Parts I & II*. Lawrence, Kansas: DARcorporation, 1998.

[29] W. Ren and R. W. Beard, "CLF-based tracking control for UAV kinematic models with saturation constraints," in *Proceedings of the IEEE Conference on Decision and Control*, 2003. (to appear).

[30] W. Ren and R. W. Beard, "Trajectory tracking for unmanned air vehicles with velocity and heading rate constraints," *IEEE Transactions on Control Systems Technology*, (in review).

[31] T.-C. Lee, K.-T. Song, C.-H. Lee, and C.-C. Teng, "Tracking control of unicycle-modeled mobile robots using a saturation feedback controller," *IEEE Transactions on Robotics and Automation*, vol. 9, pp. 305–318, March 2001.

[32] E. P. Anderson and R. W. Beard, "An algorithmic implementation of constrained extremal control for UAVs," in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, (Monterey, CA), August 2002. AIAA Paper No. 2002-4470.

[33] 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*, (in review).

[34] E. P. Anderson, "Constrained extremal trajectories and unmanned air vehicle trajectory generation," Master's thesis, Brigham Young University, Provo, Utah 84602, April 2002. http://www.ee.byu.edu/ee/robotics/publications/thesis/ErikAnderson.ps.

[35] R. L. Burden and J. D. Faires, *Numerical Analysis*. Boston: PWS-KENT Publishing Company, fourth ed., 1988.