2004-03-11

Implementation Issues of Real-Time Trajectory Generation on Small UAVs

Derek B. Kingston
Brigham Young University - Provo

IMPLEMENTATION ISSUES OF REAL-TIME TRAJECTORY GENERATION ON SMALL UAVS

by

Derek Bastian Kingston

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

Brigham Young University

April 2004

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Derek Bastian Kingston

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Randal W. Beard, Chair
Date	Timothy W. McLain
Date	James K. Archibald

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Derek Bastian Kingston in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Randal W. Beard
Chair, Graduate Committee

Accepted for the Department

Michael A. Jensen
Graduate Coordinator

Accepted for the College

Douglas M. Chabries
Dean, College of Engineering and Technology

ABSTRACT

IMPLEMENTATION ISSUES OF REAL-TIME TRAJECTORY GENERATION ON SMALL UAVS

Derek Bastian Kingston

Department of Electrical and Computer Engineering

Master of Science

The transition from a mathematical algorithm to a physical hardware implementation is non-trivial. This thesis discusses the issues involved in the transition from the theory of real-time trajectory generation all the way through a hardware experiment. Documentation of the validation process as well as modifications to the existing theory as a result of hardware testing are treated at length. The results of hardware experimentation show that trajectory generation can be done in real-time in a manner facilitating coordination of multiple small UAVs.

ACKNOWLEDGMENTS

I wish to express sincere appreciation to Professor Randy Beard for his guidance throughout the implementation and writing of this thesis. In addition, many others in the MAGICC lab have helped make this work possible including Reed Christiansen, Joshua Hintze, Wei Ren, Matt Blake, Walt Johnson, and David Hubbard. I also owe a debt of gratitude to my wife, Sarah, for her support and patience.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Research is fundamentally iterative in nature: it starts with a foundation of accepted knowledge and takes a step forward [1]. Hardware implementation and experimentation play an important role in this step forward. Often researchers concentrate on the theoretical aspects of a problem, but hardware implementation adds dimension and insight that theory and simulation lack [2]. Perhaps, the most productive style of research is when theory, simulation, and hardware implementation are blended to give a full understanding of the problem at hand.

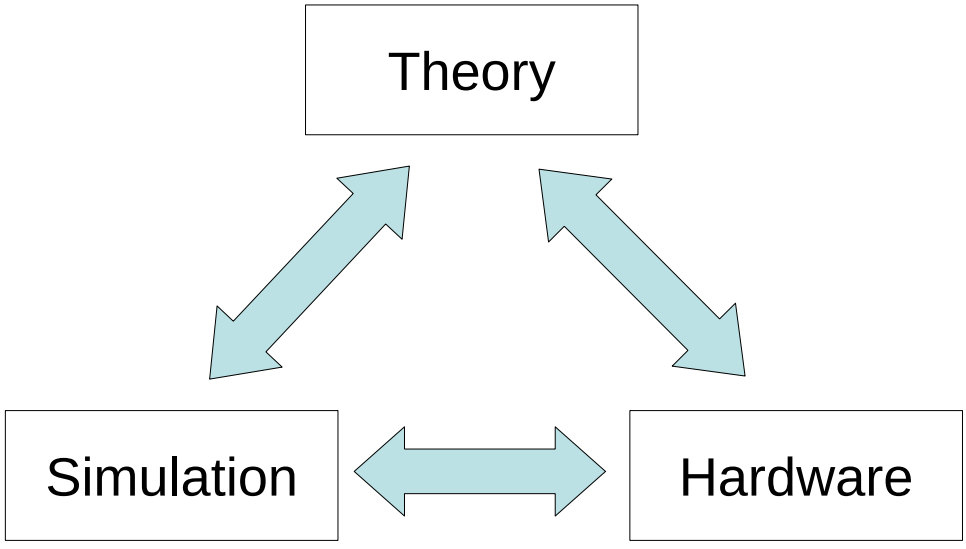

Figure 1.1: Design Cycle

Hardware experimentation exposes many aspects of the problem that are simply overlooked or not modelled well [3]. Many problems involve constraints such as: communication, timing, computation, size, power, cost, dynamics, saturation, delay, etc. that make the optimal solution impossible to calculate. In these situations, hardware implementation may be the only way to assess how well specific solutions perform with regard to these constraints.

Validation of algorithms and theory by hardware implementation lends credibility to the theories and can, at the same time, give the researcher an idea as to the best avenues to pursue to improve the performance of the system [2, 4].

An important topic of research is the cooperation and coordination of multiple Unmanned Air Vehicles (UAVs). To fully appreciate the complexity and constraints of such a problem, a hardware testbed is necessary [5]. In addition to experimentation with new theories of coordination, validation of existing theories can also be performed. An interesting aspect of UAV cooperation is the coordinated timing scenario. The capability for each individual UAV to be able to transform a straight-line path into a path that satisfies the constraints of flight dynamics while at the same time preserving path length (and hence timing) greatly simplifies the timing problem. The operation of "smoothing" a straight-line path into a feasible trajectory in an optimal manner has been presented in [6]. The purpose of this thesis is to show how theory, simulation, and hardware implementation are combined to develop real-time trajectory smoothing for small UAVs.

The development and realization of a hardware platform allows for the validation of the assumptions made in [6]. It also provides a way to adjust the theory to compensate for unanticipated errors in the modelling and setup of the initial problem. Note that with a tested, reliable trajectory smoothing algorithm, a greater level of abstraction (as well as a reliable platform) is available for those interested in the coordinated flight problems. Obviously, the closer the testbed is to the target platform, the more useful the results will be. Testbeds can range from simple mechanical models in a wind tunnel to mobile robots emulating UAVs to actual flying airplanes (like the UAVs being flown at BYU).

As attested in [1, 2, 3, 4, 5] and [7], hardware experiments (and research in general) lead to more areas of research. This thesis is really just an extension of the research into coordination and cooperation of multiple vehicles. Hopefully, it will lead to and guide the direction of further research in this area.

1.1 Problem Statement

The approach to coordination of multiple UAVs revolves around the architecture in Figure 1.2. The layered design allows for higher levels to be replaced or

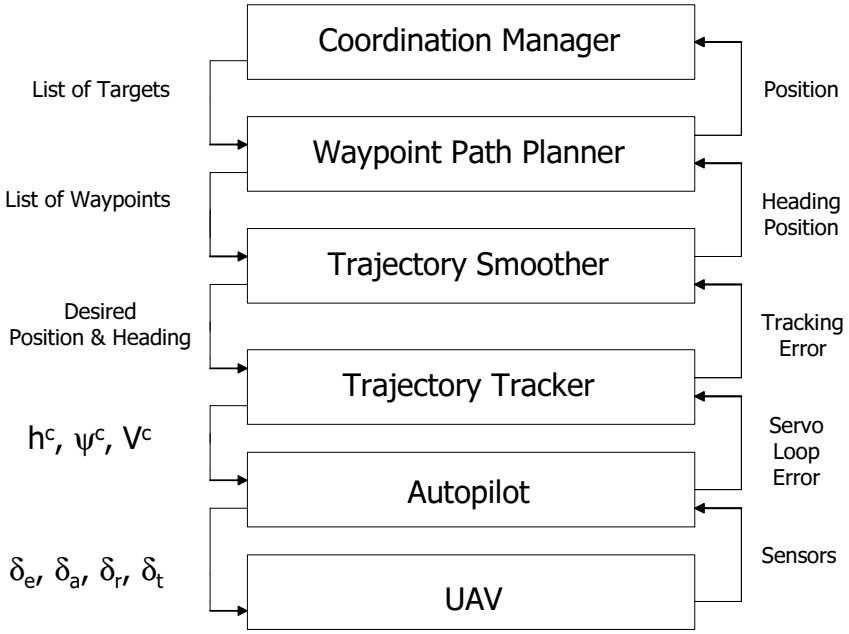

Figure 1.2: System Architecture

modified without affecting the underlying pieces. However, different low-level implementations will affect the performance of higher levels. The fundamental approach is to test different path planning and coordination algorithms without regard to the dynamics of the UAV.

One of the critical blocks in the design architecture is the Trajectory Smoother (also referred to as a trajectory generator since it generates a smoothed path). The Trajectory Smoother allows straight-line paths to be transformed into dynamically feasible trajectories. The level of abstraction afforded by correct implementation of trajectory generation permits a much simpler approach to path planning and coordination – namely, paths are straight-line segments and timing is calculated by summing the lengths of the segments and dividing by velocity. This thesis will focus primarily on the implementation of the Trajectory Smoother in hardware. The interface of the Trajectory Smoother to the other levels in the design architecture generated a great deal of additional research, which will also be addressed. For example, a lack of rigorous solutions of attitude estimation for small UAVs necessitated the development of robust attitude and position estimation to provide feedback to the Trajectory Tracker.

This thesis is organized as follows. Chapter 2 explains the mathematical approaches to path planning and trajectory generation. The transition from algorithm to real-time C code is addressed in Chapter 3. The validation of the real-time code in simulation and resulting modifications are in Chapter 4. Chapter 5 covers the move from simulation to a hardware platform. Attitude and position estimation for small UAVs is addressed in Chapter 6. Hardware results of real-time trajectory generation are presented in Chapter 7 along with conclusions and recommendations for future work.

Chapter 2

Mathematical Underpinnings

This chapter develops the theory behind two critical parts of the design architecture: path planning and trajectory smoothing.

2.1 Trajectory Generation

In developing flight plans, it is often useful to implement a path planning routine that plans only straight-line segments. In this way, the search space is significantly reduced from the standard set of feasible paths that can be realized on a UAV. In partitioning the problem into tractable sub-problems, it is useful to consider a Trajectory Smoother. The purpose of the Trajectory Smoother is to generate a trajectory that follows as closely as possible to the straight-line path, but at the same time guarantees that the path is feasible. Anderson proved that this problem has an optimal solution in [6]. An overview of his solution is presented in this chapter. For complete details see [6, 8] and [9]. The rest of this work is focused on implementing this algorithm in hardware to facilitate coordination of UAVs.

The goal of trajectory generation is to take a set of straight-line path segments and generate a dynamically feasible trajectory. A "dynamically feasible trajectory" is a time-stamped set of waypoints that satisfy the dynamics of UAV motion. A typical model of UAV dynamics is the 12 state model developed in [10]. While this model is very accurate, it imposes too many constraints to the problem of feasible trajectory generation. Therefore, to make the problem tractable, a six state model (when the

UAV has heading, altitude, and velocity hold autopilots) is used

$$\begin{aligned} \dot{z}_x &= v\cos(\psi) \\ \dot{z}_y &= v\sin(\psi) \\ \dot{\psi} &= \alpha_\psi(\psi^c - \psi) \\ \dot{v} &= \alpha_v(v^c - v) \\ \ddot{h} &= -\alpha_{\dot{h}}\dot{h} + \alpha_h(h^c - h) \end{aligned} \tag{2.1}$$

where ψ^c, v^c, and h^c are the commanded heading, velocity, and altitude to the autopilot and α_* are positive constants [11]. This model captures the critical dynamics of the system [11, 12].

A further simplification is to assume that the path will be traversed at constant velocity (v^c) and constant altitude. The UAV dynamics then reduce to

$$\begin{aligned} \dot{z}_x &= v^c\cos(\psi) \\ \dot{z}_y &= v^c\sin(\psi) \\ \dot{\psi} &= \alpha_\psi(\psi^c - \psi) \end{aligned} \tag{2.2}$$

where v^c is the constant velocity along the path and ψ^c is the heading command to the autopilot.

Using the dynamics given by (2.2) as a guide, we choose to give the trajectory generator a similar form. This choice ensures dynamic feasibility of the trajectory. To be precise, let the trajectory generator have the following form

$$\begin{aligned} \dot{x}_r &= v^c\cos(\psi_r) \\ \dot{y}_r &= v^c\sin(\psi_r) \\ \dot{\psi}_r &= u \end{aligned} \tag{2.3}$$

where (x_r, y_r) is the inertial position of the trajectory, ψ_r is its heading, v^c is the commanded linear speed along the path, and u is the commanded heading rate. The dynamics of the UAV impose input constraints of the form

$$\begin{aligned} 0 < v_{min} \leq & v^c \leq v_{max} \text{ and} \\ -\omega_{max} \leq & u \leq \omega_{max}. \end{aligned} \tag{2.4}$$

The constraint on v^c in Equation (2.4) is due to the fact that fixed-wing aircraft must travel at some non-zero velocity to maintain lift, while at the same time, the thrust of any given actuator is limited in the amount of force it can exert which limits the maximum velocity. The input u is the turning rate of the aircraft and is constrained due to the saturation of the roll angle by the autopilot.

With the velocity fixed at v^c, the minimum turn radius is defined as $R = v^c/\omega_{\max}$. 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, shown in Figure 2.1.

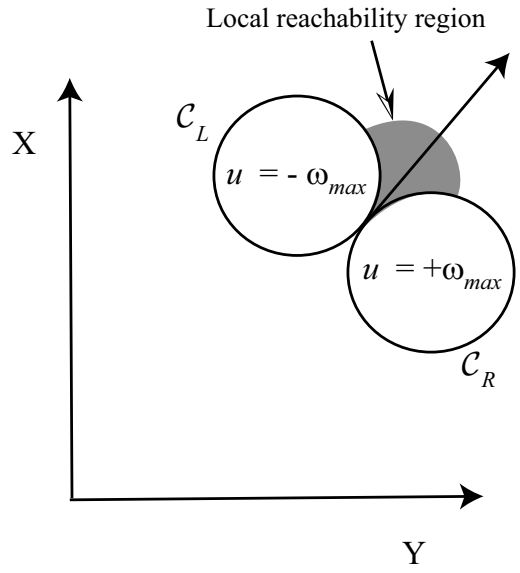

Figure 2.1: Local reachability region of the Trajectory Smoother.

With this in mind, it seems natural that for a 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 [6] that this is the case. By postulating that u 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 2.1 *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, as shown in Figure 2.2.*

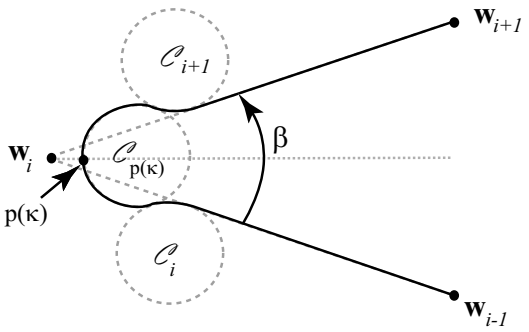

Figure 2.2: A dynamically feasible κ-trajectory.

A κ-trajectory always passes through the point $\boldsymbol{p}(\kappa)$, therefore, 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 [6], thus simplifying coordination problems. In any case, u satisfies Equation (2.4) which, in turn, guarantees that the trajectory is dynamically feasible.

The careful reader will realize that a provably feasible reference trajectory does not necessarily provide a simple way to follow or track that trajectory. Other issues not dealt with in the theoretical development include understanding the validity of the assumptions made in constructing the reference trajectory (i.e. the lower order

model), real-time capability, and performance on actual hardware. These topics will be addressed in the remaining chapters.

2.2 Path Planning

With the Trajectory Smoother generating feasible trajectories out of straight-line paths, the path planner can be constructed to use well-known path planning algorithms. It is hard to overstate how valuable this is. Constrained path planning is often much more difficult and computationally intense than is planning straight-line paths and smoothing afterward.

Our path planning technique centers around the construction and search of a Voronoi graph [13]. The Voronoi graph provides a method for creating straight-line paths through a field of point threats. A prime advantage of the Voronoi graph is the computational speed with which the graph can be created and searched (see Section 3.3).

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 2.3 shows an example of a Voronoi graph constructed from point threats.

Finding good paths through the Voronoi graph requires the definition of a cost metric associated with travelling along each edge. In our work, two metrics have been employed: path length and threat exposure [14]. 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 prevent safe, but meandering paths from being chosen.

Once a metric is defined, the graph is searched using an Eppstein search [15] which has the ability to return k best paths through the graph. Once k best paths are

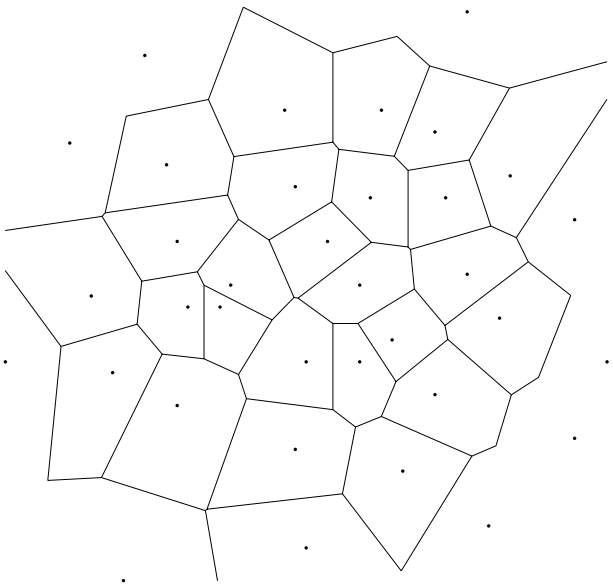

Figure 2.3: Voronoi graph with point threats

known, a coordination agent can decide 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 to the Trajectory Smoother which smooths through the path, taking into account the dynamics and constraints of the vehicle.

An advantage to having the graph creation step and the graph search step separated is that a graph can be constructed once for a threat field common to many agents and then searched individually for each agent. Inserting an agent and its associated target position is done by simply considering the start and target positions of the agents as point threats for the Voronoi graph construction. Once the graph has been computed, the start and target positions are connected to the graph by adding graph edges from those positions to the nodes making up their associated container cells. Significant computation is saved by computing the graph only once for the entire team, especially for large numbers of threats.

Chapter 3

From Mathematics to Executable C Code

This chapter contains a presentation of the procedure followed to convert the mathematical algorithms of path planning and trajectory generation to a collection of C code that implements those algorithms. Changes that were made to correct or enhance the existing theory are also documented in this chapter.

3.1 Polygon Regions During Path Planning

The Voronoi approach to path planning was chosen primarily for its computational speed. However, there are many drawbacks in the way it constructs a graph. The most restrictive aspect of Voronoi path planning is the inability to specify inaccessible regions such as "no fly" zones. This isn't much of a drawback in simulation, but is a severe shortcoming when it comes to hardware testing, e.g. the robot or UAV has strict boundaries which the path planning must not violate. This is typical of the issues dealt with when performing hardware experiments. To compensate for this, the implementation of Voronoi path planning in C code was designed to account for arbitrary polygon regions.

In our work, construction of a Voronoi graph for obstacles modelled 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. When the polygons are specified, a corresponding center and radius that bound the polygon are computed. Each edge in the graph is checked to see if it intersects the bounding circle and if so,

the edge is checked for intersection with all edges of the polygon. Those graph edges that intersect a polygon are removed from the graph. Figure 3.1 shows the initial polygon based graph and the final graph after pruning. Note that graph edges that

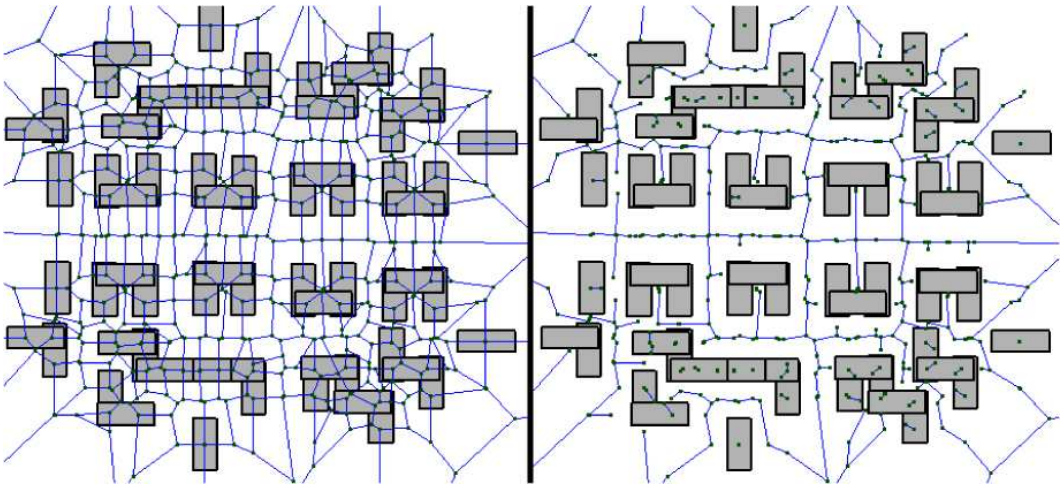

Figure 3.1: Voronoi graph before and after pruning with polygon threats

are completely contained within polygons are not removed (since they never intersect a polygon edge), however, the Eppstein search will never consider these segments because there does not exist any path segment connecting the graph edges on the inside of polygons with the graph edges on the outside.

In implementing Voronoi path planning, we started with Voronoi graph code written by Steve Fortune [16] and Eppstein search code written by Victor Jimenez and Andres Marzal [17]. An interface to the Voronoi graph construction was built to send polygon vertices as point threats. The resulting graph is pruned to take out the graph edges that intersect polygon regions. The set of feasible graph edges are assigned a cost based on length and proximity to threat vertices and then searched with the Eppstein search for the best path.

The test case for debugging used a fairly rich environment with many randomly spaced threats. When other cases were tested, it was found that the Voronoi graph

method can be poorly connected for some configurations of threats. When the threat field had few threats or was highly symmetric the Voronoi method failed to give sensible paths. It was found that, on average, three threats are needed to give a well connected Voronoi graph. When a threat field produces a Voronoi graph that does not allow a path from start position to the target, an additional method of planning is needed.

We modified the path planner to re-plan with a "brute force" method whenever the Voronoi method failed. This additional planner simply assigns the corners of squares that enclose the bounding circle of the polygons as nodes in a graph. Each node is linked to every other node and to the start and target positions. This new graph is pruned and searched in the same manner as the Voronoi graph. This method is computationally inefficient, but for small numbers of threats, it can find short, feasible paths in real-time. A comparison of the Voronoi method and the "brute force" method is shown in Figure 3.2. The lefthand side of Figure 3.2 shows the Voronoi

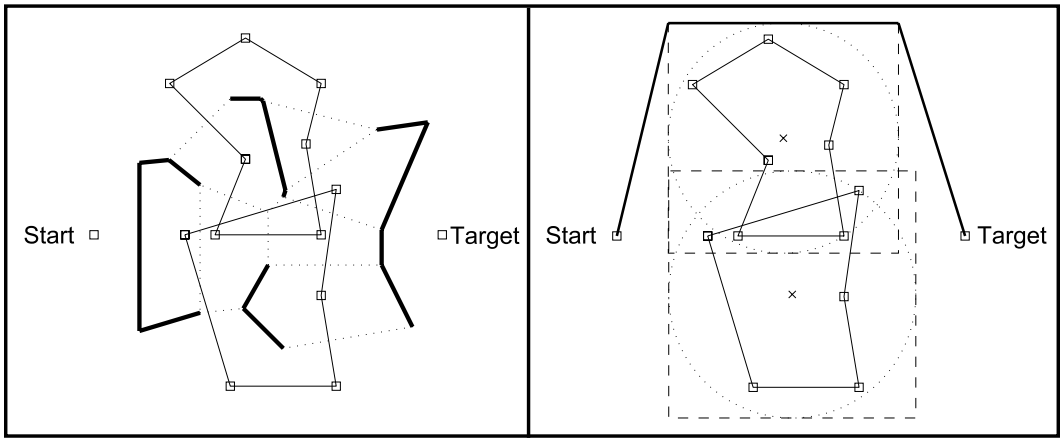

Figure 3.2: Path planning for a Voronoi failure.

diagram which was constructed using the vertices of the polygon obstacles and the start and target positions. After the pruning operation, only the bold lines are left in the graph, making a connection from the start to the target position unrealizable.

The righthand side of Figure 3.2 shows the same set of obstacles with their respective bounding circles. The corners of the squares that enclose the bounding circles are linked together into a graph which is searched to produce the bold path from the start to the target position.

3.2 Visualization of Trajectory Generation

In addition to proving the optimality of his trajectory generation strategy, Anderson also provided base MATLAB code for implementation [6]. Porting this base code to C required a numerical integration routine. Since a real-time application is the target, we chose a fixed step fourth order Runge-Kutta solver [18]. To assist in debugging and also for visualization, a graphical frontend was added (Figure 3.3).

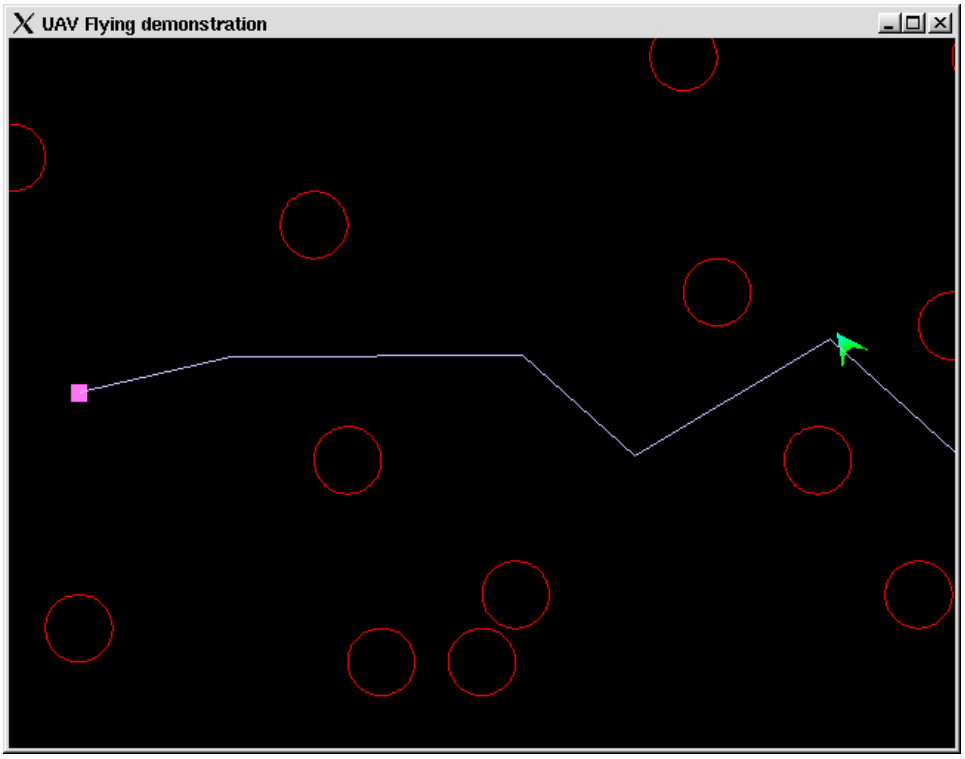

Figure 3.3: Visualization of Trajectory Generation.

3.2.1 Robust Circle Intersections

The choice of using a fixed step integration routine facilitates real-time computation, but the resulting discretization causes re-evaluation of some aspects of trajectory generation. Most notably, determining how the reference trajectory executes turns becomes more difficult.

The Trajectory Smoother can be thought of as a state machine that drives Equation (2.3). The state depends on the section of the turn being performed and state transitions occur when the reference trajectory intersects the circle or line defining the turn. The process for performing a turn is shown in Figure 3.4. The constraints

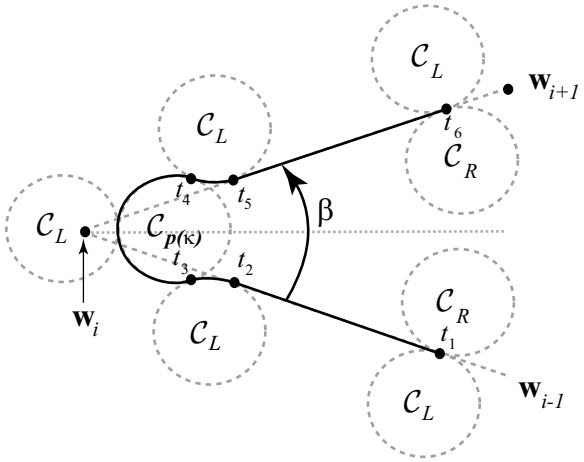

Figure 3.4: Graphical depiction of the selection of u.

given in Equation (2.4) are manifest graphically by circles of minimum turn radius on both sides of the reference UAV and are denoted by $\mathcal{C}_R$ and $\mathcal{C}_L$. At times t_3 and t_4, the right turning circle $\mathcal{C}_R$ is assumed to be directly over $\mathcal{C}_{\mathbf{p}(\kappa)}$ and so is not shown at those times.

A typical κ-turn proceeds in the following manner. At time t_1 the Trajectory Smoother is tracking the waypoint segment $\overline{\mathbf{w}_{i-1}\mathbf{w}_i}$. When the left turning circle $\mathcal{C}_L$ intersects $\mathcal{C}_{\mathbf{p}(\kappa)}$ at time t_2, u is set to $-\omega_{\max}$. The left turning constraint is followed

until the right turning circle $\mathcal{C}_R$ corresponds exactly with $\mathcal{C}_{\mathbf{p}(\kappa)}$ at time t_3. The Trajectory Smoother input u is then set to $+\omega_{\max}$ and the right turning constraint is followed until the left turning constraint $\mathcal{C}_L$ intersects the waypoint segment $\overline{\mathbf{w}_i\mathbf{w}_{i+1}}$ at time t_4. The Trajectory Smoother input u is again set to $-\omega_{\max}$ until it reaches the waypoint segment at time t_5 where u is set to zero [9].

Due to the discrete implementation of the Trajectory Smoother, switching times can be difficult to detect. This is illustrated by the following scenario. Suppose that the Trajectory Smoother is tracking the straight-line segment $\overline{\mathbf{w}_{i-1}\mathbf{w}_i}$ and anticipating the intersection of $\mathcal{C}_L$ with $\mathcal{C}_{\mathbf{p}(\kappa)}$ in order to detect the switching time t_2 as in Figure 3.5. The circle $\mathcal{C}_L$ may not intersect the circle $\mathcal{C}_{\mathbf{p}(\kappa)}$ exactly at the sample times. Therefore, we need a robust method for detecting circle-circle intersections.

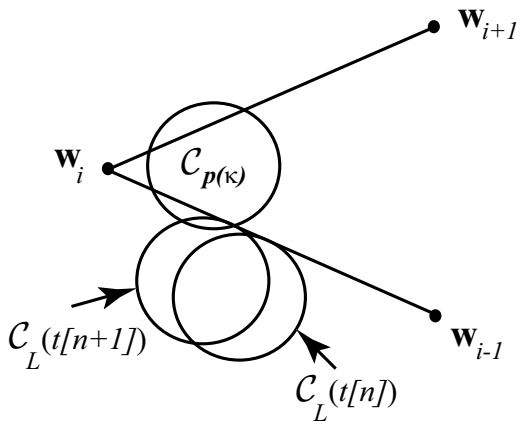

Figure 3.5: Effect of fixed sample-rate on switching-time detection.

A simple method for detecting circle-circle intersections is to monitor the distance between circle centers. When this value is less than $2R$ then the circles obviously intersect. However, there are cases when this method will fail to detect the intersection. When the reference trajectory is not tracking the line precisely or when $\kappa \approx 1$, the distance between circle centers can be greater than $2R$ both before and after the intersection as in Figure 3.6.

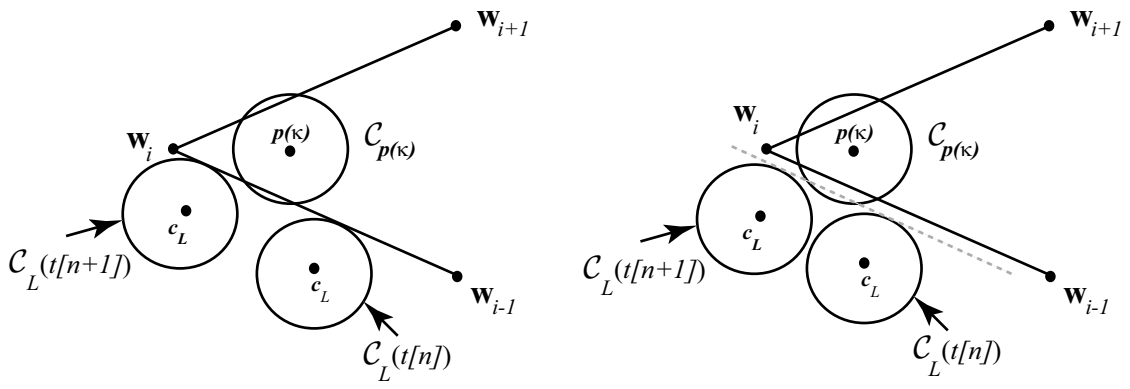

Figure 3.6: Cases when a simple distance test will fail.

To ensure that the UAV will correctly execute turns despite an imprecise discrete implementation, switching times will be detected by dividing space into two distinct regions and then checking for the crossover from one region to another. Specifically, let $\mathcal{S}_1$ be the region containing waypoint $\mathbf{w}_{i-1}$ and $\mathcal{S}_2$ the region containing waypoint $\mathbf{w}_i$. The line dividing $\mathcal{S}_1$ from $\mathcal{S}_2$ is perpendicular to the path segment $\overline{\mathbf{w}_{i-1}\mathbf{w}_i}$ and passes through $\mathbf{p}(\kappa)$ as shown in Figure 3.7. The usual check for circle

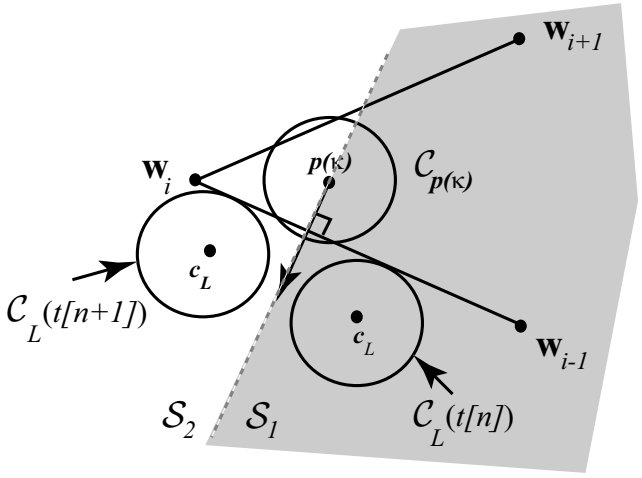

Figure 3.7: Robust circle intersection method.

intersection is used, however, when the center of $\mathcal{C}_L$ has crossed into region $\mathcal{S}_2$ and a circle intersection has not been detected, then the intersection is regarded as missed and the state is updated as if t_3 has been reached. Note that the state progression of the system skips the state associated with switching time t_2 in this case. Normally, when t_2 has been detected, the reference UAV turns opposite the direction of the main turn until t_3 is detected, but if t_2 has been missed, then the UAV must turn towards the segment $\overline{\mathbf{w}_i\mathbf{w}_{i+1}}$ to minimize the error incurred by missing t_2.

Using similar ideas, all switching times can be made robust to discrete implementation [9]. When switching time t_5 is detected, the reference UAV will not be precisely aligned with the segment $\overline{\mathbf{w}_i\mathbf{w}_{i+1}}$. Anderson foresaw this in his initial formulation and presented a tracking algorithm to pull the reference back onto the segment [6]. A modified version of this tracking algorithm is proven to satisfy the constraints of the UAV in [9].

3.3 Real-Time Analysis

With C code implementations of path planning and trajectory generation algorithms, it is possible to assess the real-time nature of these algorithms.

Definition 3.1 *Let t_c be the computation time for one iteration of the algorithm and N be the maximum number of times per second required for the algorithm to be computed. An algorithm is defined as* real-time *if and only if $Nt_c \leq 1$.*

An interesting aspect to Definition 3.1 is the choice of N. In a path planning scenario, N will be determined by the maximum allowable time for planning a path. If a new path is to be planned every time the environment has changed, then the computation time for planning a path must not exceed the time between updates of the environment. For example, if the environment is checked for changes at a rate of 10 Hz, then the path planner execution time must not exceed 100 ms. Essentially all path planners scale in complexity with regard to the number of threats in the scene or the size of the environment considered, therefore, a particular path planning algorithm can only be guaranteed to operate in real-time for up to M threats where the computation for M threats equals the time between environment updates.

In considering the real-time capability of the Trajectory Smoother, we note that the computation time for an iteration is fixed, regardless of changes in the scene: an iteration in the Trajectory Smoother consists of circle-line and circle-circle intersections and numerical integration – input changes simply reset the state. With t_c known, we can easily solve for N. If the rate of the control loop that depends on the reference trajectory is less than N, then we can state that the Trajectory Smoother operates in real-time.

To determine the computation time associated with the path planning and trajectory smoothing algorithms, Monte-Carlo simulations were performed. The path planning algorithm was simulated on randomly generated scenes to estimate how t_c changes as a function of the number of threats. Each threat was simulated as an asymmetrical 8 sided polygon. Figure 3.8 shows the data points gathered through Monte-Carlo simulation and the second order curve fit of that data. This data suggests

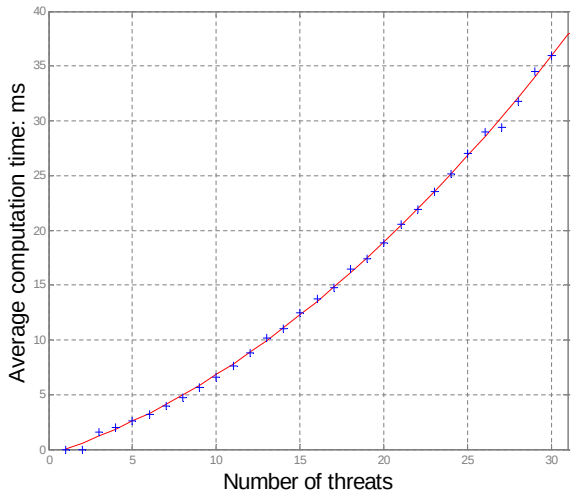

Figure 3.8: Monte-Carlo results for estimating t_c as a function of threat number.

that for an environment update rate of 40 ms (25 Hz), the path planner will operate in real-time for threat fields with less than 30 threats. By extrapolation of the data,

threat fields of up to 194 threats could be dealt with in real-time for an environment update rate of 1 Hz. Clearly, for low rates of new threat information (on the order of 1 Hz), paths through considerable size threat fields can be planned in real-time. By recalling that the computation time is dominated by the number of threats, similar results hold for teams of agents – that is, path planning can be achieved in real-time for teams of agents.

Similar Monte-Carlo simulations were performed to evaluate the execution time for an iteration of the Trajectory Smoother. On a 1.8 GHz Intel Pentium 4 processor, one iteration of the Trajectory Smoother took on average 36 μ-seconds. At this speed, the Trajectory Smoother could run at 25 KHz - well above the dynamic range of typical UAVs. Moving toward embedded systems, we found that one iteration of the Trajectory Smoother required a maximum of 47 milli-seconds on a Rabbit Microprocessor running at 29 MHz. The low computational demand allows the Trajectory Smoother to be run in real-time at approximately 20 Hz on an embedded system on-board the UAV.

Chapter 4

Simulation and Validation

This chapter contains the results of simulation and validation of the Trajectory Smoother on different mathematical models of a UAV. Also included is a presentation of the discovery of and solutions to problems uncovered through simulation.

4.1 Validation

To validate the correct operation of the Trajectory Smoother, many different scenarios were implemented and the resulting trajectories observed. For most scenarios, this simulation setup showed satisfactory results; however, anomalous behavior was exhibited when path segments were connected with extreme angles between successive segments, i.e. when shallow or sharp turns were required.

In [6], Anderson provides a method for determining the time extremal path under 3 different conditions: minimum path length, equal path length, and passing directly over the waypoint. These types of turns are illustrated in Figure 4.1. By parameterizing the distance from the waypoint and the closest point that the UAV passes by κ, these three cases are equivalent to constraining $\kappa = 1$ for the minimum path length, $\kappa = 0$ for passing directly over the waypoint, and κ found by a bisection search for equal path length trajectories. Now we consider the relationship between κ and the angle between successive path segments β

$$\kappa = \frac{d \sin\left(\frac{\beta}{2}\right)}{R\left(1 - \sin\left(\frac{\beta}{2}\right)\right)} \tag{4.1}$$

where d is the distance from the waypoint to the closest point that the UAV passes and R is the minimum radius turn allowed by the dynamics of the UAV.

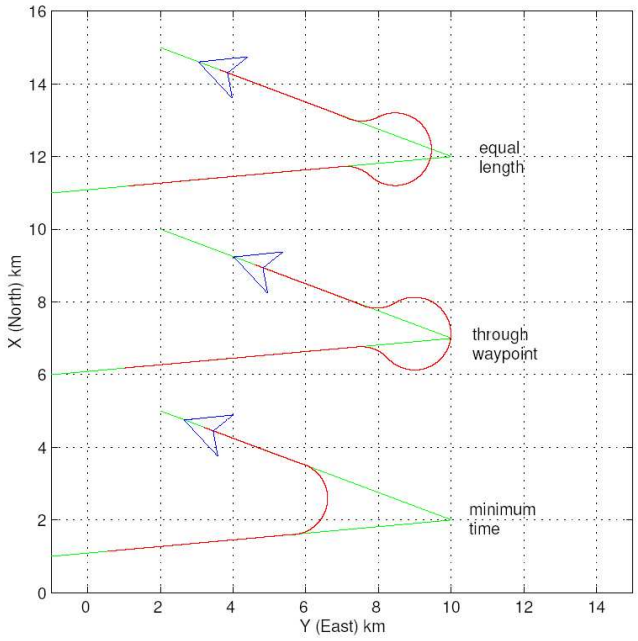

Figure 4.1: Sample κ turns.

4.1.1 Shallow Turn Compensation

Equation (4.1) shows that κ can be found for any desired distance d from the waypoint to the closest passing point. Due to the $\sin\left(\frac{\beta}{2}\right)$ term, however, κ will be strongly influenced by the angle β. In fact, as $\beta \to \pi$ (corresponding to no turn at all), $\kappa \to 0$ for *any* distance d. This matches intuition because for two segments joined into a straight line with no angle between them, the straight-line path is equivalent to the path that passes directly over the waypoint, the minimum length path, and the equal distance path. Strange behavior is exhibited when β is very close to π because the switching points for successive turns are so close that the discrete implementation is too course to distinguish the state of the trajectory in the turn.

To address the "shallow turn" condition, a threshold is placed on the angle β. For $\beta \geq \beta_c$, κ is simply set to zero which causes the trajectory to continue in linear motion without turning. Instead of computing the turn switching points and explicitly performing a turn, the Trajectory Smoother checks for when the reference trajectory

comes to within a quarter radius distance from the middle waypoint. When this condition is met, the Trajectory Smoother switches state, assigning the next segment as the current segment and relying on the local tracking routine to bring the trajectory back to the straight-line path. For shallow turns, the difference in path length from the theoretical path and the actual path using this method is negligible.

4.1.2 Sharp Turn Compensation

The anomalous behavior for sharp turns is also due to the influence of β on Equation (4.1). Note that as $\beta \to 0$ (corresponding to a U-turn), $\kappa \to 0$. This result is nonsensical – clearly, for a U-turn there exists an equal-length path which doesn't pass though the waypoint. The discrepancy is due to the parametrization of κ. For a U-turn, $\kappa \in (0, 1]$ corresponds to $d = \infty$, which forces all other distances to be parameterized by $\kappa = 0$. This can be visualized by imagining two segments that form a very sharp turn with an arc connecting these two in a minimum-distance configuration (Figure 4.2). The sharper the turn, the further out the arc will be pushed to allow a fixed radius circle to connect the two segments.

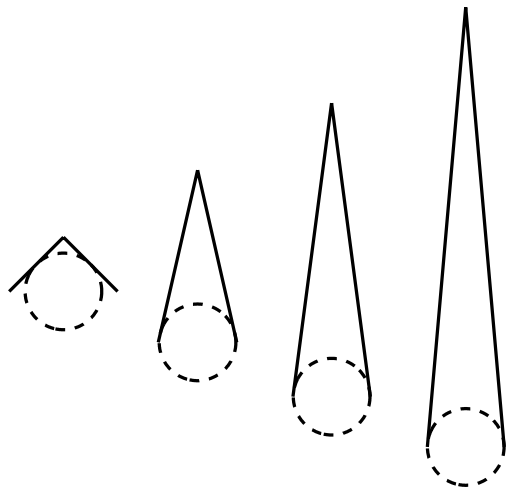

Figure 4.2: Sharp turn parametrization problem.

To correct for sharp turns, the form for an equal path length U-turn is computed. Figure 4.3 shows the geometry to calculate the distance D from the U-turn switchback point to the center of the turn-switching circle. To perform a perfect

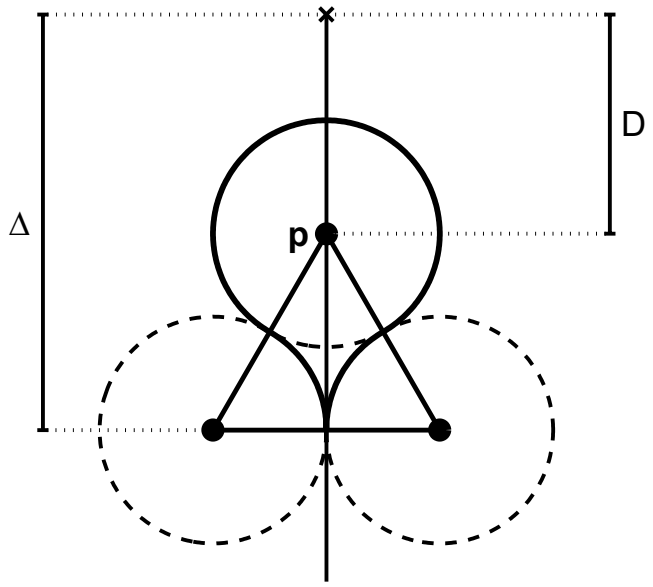

Figure 4.3: U-turn equal path length switching points.

U-turn, three circles of minimum turn radius R are arranged so that the centers of the circles form an equilateral triangle. The arclength for a given angle θ is given by

$$L(\theta) = \theta R. \tag{4.2}$$

The total distance traveled along the edges of the circles, L_{total}, is then

$$L_{\text{total}} = L\left(2\pi - \frac{\pi}{3}\right) + L\left(\frac{\pi}{3}\right) + L\left(\frac{\pi}{3}\right) = \frac{7\pi}{3}R \tag{4.3}$$

where the first term is the path length along the top circle and the last two terms are the distances on the edges of the lower circles. To ensure that the U-turn path has path length equal to the straight line path, L_{total} must equal the distance from the start of the turn to the waypoint and back (2Δ). Solving for Δ yields

$$\Delta = \frac{L_{\text{total}}}{2} = \frac{7\pi}{6}R. \tag{4.4}$$

The distance to the center of the turning circle, D, is then simply the height of the triangle subtracted from Δ, where it is noted that each side of the triangle is $2R$ in length. Using the Pythagorean Theorem to calculate the height of the triangle, the distance D is found to be

$$D = \left(\frac{7\pi}{6} - \sqrt{3}\right) R. \tag{4.5}$$

Unless a $\kappa = 0$ turn is desired, sharp turns are made equivalent to U-turns in the equal length path case, i.e. the turn switching circle is set to exactly D from the waypoint about which the sharp turn is to be performed. Similar to the shallow turn case, a threshold on β signals the use of this method rather than the conventional minimum distance or equal path length methods.

4.1.3 Robustness to Input Changes

Instability in some systems can be attributed to rapid changes in input. In this section, we investigate how input changes to the Trajectory Smoother affect the generated trajectory. In particular, we want to guarantee that the reference trajectory will *always* satisfy the constraints of the UAV which, in turn, gives the autopilot an input signal that can be realized in a stable manner.

Definition 4.1 *Let a trajectory r be a series of time-stamped waypoints for all $t \in [0, \infty)$ and let $r(t)$ be the position of the trajectory at time t and $r[t_1, t_2)$ be the set of waypoints in r for $t \in [t_1, t_2)$. Then, given an ordered set of trajectories, $R = \{r_1, r_2, \ldots, r_N\}$, with cooresponding times along trajectories, $T = \{t_1, t_2, \ldots, t_N\}$, the resulting* **sequence** *of trajectories, $S\left[0, \sum_{i=1}^{N} t_i\right)$ is*

$$S = \begin{cases} r_1[0, t_1) & 0 \leq t < t_1 \\ r_2[0, t_2) & t_1 \leq t < t_1 + t_2 \\ \vdots & \vdots \\ r_N[0, t_N) & \sum_{i=1}^{N-1} t_i \leq t < \sum_{i=1}^{N} t_i \end{cases}. \tag{4.6}$$

Theorem 4.2 *Given that a set of trajectories R exists in which each element of R, r_i, satisfies the constraints (2.4) for $t \in [0, t_i)$, then the resulting sequence of trajectories, S, will also satisfy the constraints (2.4) provided that $r_2(0) = r_1(t_1)$, $r_3(0) = r_2(t_2)$, $\ldots$, $r_N(0) = r_{N-1}(t_{N-1})$.*

The proof of Theorem 4.2 follows from considering that if the constraints (2.4) are satisfied for *all* segment trajectories composing S and each segment trajectory of S is connected to the next without separation in time, then at no time will S experience a condition which will violate the constraints (2.4).

The Trajectory Smoother has been implemented so that changes in input waypoints simply generate a new κ-trajectory that begins at the last position of the previous κ-trajectory. Since all κ trajectories satisfy the constraints (2.4) and each successive κ-trajectory starts where the previous one left off, it follows from Theorem 4.2 that the Trajectory Smoother will satisfy the constraints (2.4), regardless of input changes.

To illustrate that the Trajectory Smoother will generate a trajectory that satisfies constraints (2.4), a simulation was developed where the input path segments were rapidly and randomly changed. Figure 4.4 shows the reference trajectory and the times when the input changed (indicated by the crosses). As can be seen, the

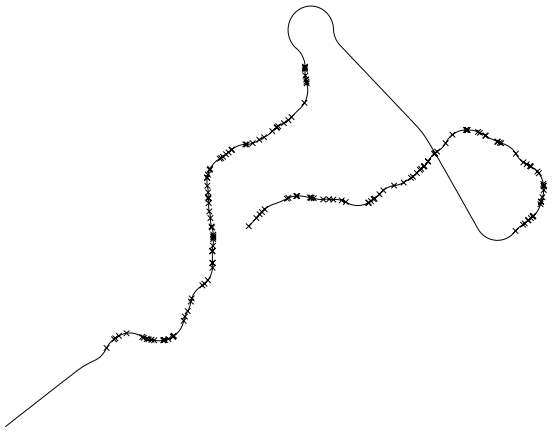

Figure 4.4: The Trajectory Smoother is robust to rapid input changes.

Trajectory Smoother is robust to rapid input changes. Although no sensible behavior is exhibited (due to an insensible input), the resulting trajectory is smooth and allows for stable operation of the autopilot.

4.2 Simulation With a 6 State Model

The Trajectory Smoother was developed under the assumption that an autopiloted UAV could be modelled with the six state model given in Equation (2.1). The next step to evaluate the correctness of the Trajectory Smoother is to input the reference trajectory generated by the Trajectory Smoother to a UAV with dynamics given by Equation (2.1).

4.2.1 The Trajectory Tracker

Before an analysis of the Trajectory Smoother could be performed for a UAV modelled by Equation (2.1), an important piece of the design architecture had to be developed – the Tracker. The purpose of the Tracker is to command the UAV in such a way that the difference between the actual position of the UAV and the reference position from the Trajectory Smoother is driven to zero.

Using knowledge of the structure of the Trajectory Smoother coupled with the assumption that an autopiloted UAV behaves as in Equation (2.1), a tracker was proposed that guarantees convergence in finite time while at the same time satisfying the velocity and heading rate constraints of the UAV [19, 20]. The constraints were modified to allow the Tracker some control authority to bring the UAV back to the reference trajectory. Thus the Trajectory Smoother satisfies the contraints given in Equation (2.4) and the Tracker satisfies the constraints

$$\begin{aligned} 0 < v_{\min} - \epsilon_{v_{\min}} \leq \quad & v \quad \leq v_{\max} + \epsilon_{v_{\max}} \\ -\omega_{max} - \epsilon_{\omega_{\max}} \leq \quad & \omega \quad \leq \omega_{max} + \epsilon_{\omega_{\max}} \end{aligned} \tag{4.7}$$

where ϵ_* are the expansion of the constraints on the reference trajectory.

The Tracker uses feedback of velocity, heading, and position along with the output of the Trajectory Smoother to generate desired heading and velocity commands for the autopilot. The Tracker is implemented by [19, 20]:

$$
\begin{aligned}
x_0 &= \psi^r - \psi \\
x_1 &= -\sin(\psi)(X^r - X) + \cos(\psi)(Y^r - Y) \\
x_2 &= -\cos(\psi)(X^r - X) - \sin(\psi)(Y^r - Y) \\
\overline{x_0} &= x_0 + \frac{x_1}{\sqrt{1 + {x_1}^2 + {x_2}^2}} \\
u_0 &= -\text{sat}\left(m_0\overline{x_0}, \epsilon_{\omega_{\max}}, -\epsilon_{\omega_{\max}}\right) \\
\lceil u_1 \rceil &= (v_{\max} + \epsilon_{v_{\max}}) - v^r \cos(x_0) \\
\lfloor u_1 \rfloor &= (v_{\min} - \epsilon_{v_{\min}}) - v^r \cos(x_0) \\
u_1 &= \text{sat}\left(-m_1 x_2, \lceil u_1 \rceil, \lfloor u_1 \rfloor\right) \\
\omega &= \omega^r - u_0 \\
\psi^c &= \psi + \frac{\omega}{\alpha_\psi} \\
v^c &= v^r \cos(x_0) + u_1
\end{aligned}
\tag{4.8}
$$

where X^r, Y^r, v^r, ψ^r, and ω^r are the North position, East position, velocity, heading, and heading rate of the reference trajectory, and X, Y, v, and ψ are the actual North position, East position, velocity, and heading of the UAV, and where α_ψ is defined in Equation (2.1), *sat* is the saturation function, m_0 and m_1 are positive tuning parameters, and ψ^c and v^c are the heading and velocity commands to the autopilot.

An appealing aspect to the Tracker is its simplicity. Referring to [19, 20] and Equation (4.8), the Tracker is essentially a change of variables and an appropriate saturation constraint. This facilitates the real-time nature of the overall solution – the computation required to track the reference trajectory is trivial when compared with the computation required to generate that trajectory.

Equation (4.8) shows the dependence of the Tracker on knowledge of the underlying model that was used for design. Note that to compute a commanded heading, ψ^c, a commanded heading rate was first calculated and then divided by α_ψ. The parameter α_ψ characterizes the heading response of an autopiloted UAV as in Equa-

tion (2.1). As will be shown in Section 4.3, the Tracker can be very sensitive to modelling errors in α_ψ.

4.2.2 Simulation Results for a 6 State Model

Unsurprisingly, simulation of the Trajectory Smoother and Tracker with a UAV modelled as in Equation (2.1) shows precise performance. This is to be expected because the reference trajectory and the associated tracking algorithm were developed with all constraints considered and with the assumption that UAV dynamics are approximated by Equation (2.1). A much more interesting simulation scenario is when the UAV is modelled as the full 12 state model, which is addressed in Section 4.3.

4.3 Simulation With a 12 State Model

This section investigates the assumption that a 6-state model can be used to design a reference trajectory for a 12-state aircraft. This is done by simulating a precise model of a UAV with an associated autopilot.

As presented in [10], aircraft dynamics can be modelled as

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = \text{DCM}^T \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{4.9}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \text{DCM} \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \frac{1}{m}\mathbf{F} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{4.10}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)\sec(\theta) & \cos(\phi)\sec(\theta) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4.11}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \mathbf{J}^{-1} \left(\mathbf{M} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \mathbf{J} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \right) \tag{4.12}$$

where the states of the aircraft are

$$\begin{aligned} X &= \text{distance North} \\ Y &= \text{distance East} \\ -Z &= \text{altitude} \\ u &= \text{velocity out the nose} \\ v &= \text{velocity out the right wing} \\ w &= \text{velocity out the belly} \\ \phi &= \text{roll angle} \\ \theta &= \text{pitch angle} \\ \psi &= \text{yaw angle} \\ p &= \text{roll rate} \\ q &= \text{pitch rate} \\ r &= \text{yaw rate} \end{aligned} \tag{4.13}$$

and DCM is the Direction Cosine Matrix which rotates from the inertial frame to the body frame (a function of ϕ, θ, and ψ); m is the mass of the UAV; $\mathbf{J}$ is the inertia tensor expressed in the body frame; $\mathbf{F}$ is a vector of translational forces acting on the UAV in the inertial frame; $\mathbf{M}$ is a vector of torques in the inertial frame; and g is the gravitational constant.

The forces and torques acting on the aircraft can be approximated using knowledge of the aerodynamic properties of a given aircraft. Once the forces and torques are characterized, an autopilot can be developed. In practice, for small UAVs, identification of the aerodynamic coefficients can be costly or difficult to obtain. For this reason, autopilots are developed using multiple PID loops that are tuned until acceptable performance is realized. When the aerodynamic coefficients are well known or have been calculated or estimated, an autopilot can be implemented with a Linear Quadratic Regulator (LQR). The method used to develop the autopilot is not important to the Trajectory Smoother or Tracker. The critical dependency is whether or not the autopilot can reduce the dynamics of the aircraft to the 6 state model in Equation (2.1).

Both PID and LQR autopilots have been developed in simulation and both show similar performance. Figure 4.5 shows typical responses of an autopiloted UAV to steps in velocity, heading, and altitude. The roll angle demonstrates that for large steps in heading, roll angle is adequately approximated by a first order system, but heading response is not. The physical interpretation is that the aircraft must have a

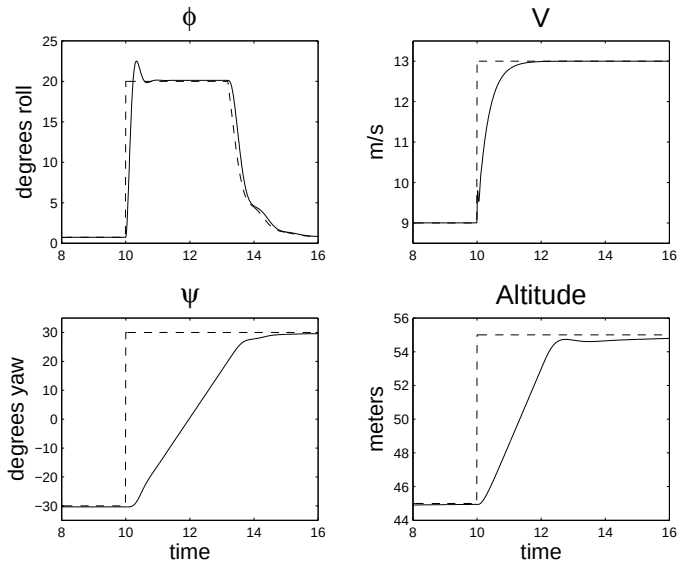

Figure 4.5: Autopilot response to heading, velocity, and altitude steps.

roll angle to initiate a change in heading, but strict limits apply to how large a roll angle can be sustained. For large steps in heading, the roll angle is saturated and heading can only change at the rate allowed by the maximum roll angle. How then can the 6 state model of an autopiloted UAV be correct if the heading response is clearly not first order? Under what conditions is the 6 state model accurate, if any?

In the same vein as linear approximations to non-linear systems or Riemann sums to approximate integrals, we simply choose to let heading response be modelled as first order, but only considered accurate for *small* changes in heading. Therefore, the 6 state model is only "valid" when the commanded changes in heading allow heading response to be accurately characterized as first order. In other words, for an autopiloted UAV to be validly modelled with a 6 state model, the input must be guaranteed not to command heading changes that violate the first order assumption. The notion of having heading changes be small for accurate modelling is similar to step size for numeric integration – a result is generated, but accuracy is sacrificed as the step size increases. To label a model as "valid" necessitates that the accuracy be within some allowable tolerance to the actual response.

With this in mind, the full 12 state model of an aircraft augmented with an LQR autopilot was simulated. Figure 4.6 shows the reference and actual trajectory of a simulated UAV as well as the distance of the UAV from the desired trajectory over a period of 3 minutes. The maximum error observed is less than 1.2 meters and most of the time the error is less than 10 cm. This is considered very accurate, therefore, the assumption that an autopiloted UAV can be modelled with Equation (2.1) appears to be a good assumption.

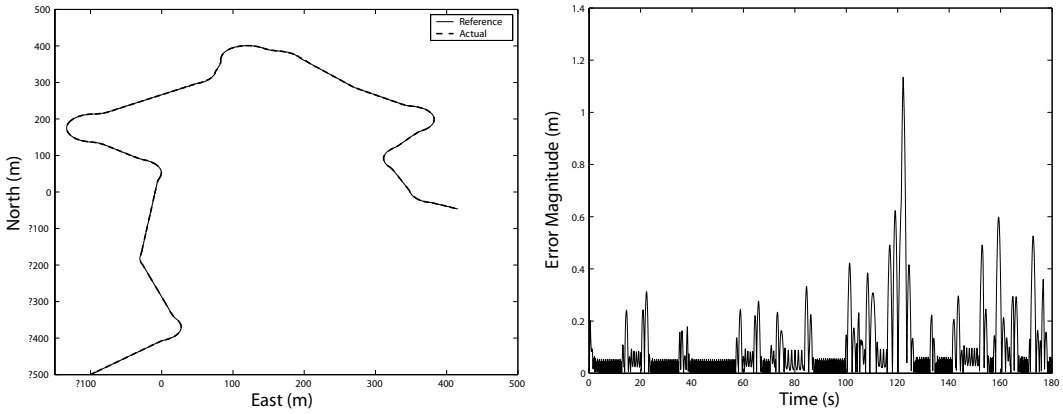

Figure 4.6: Actual and reference trajectories (left) and Tracking error (right).

Empirically, the use of the 6 state model to design the Trajectory Smoother and Tracker is warranted. However, the Tracker was observed to be sensitive to accurate knowledge of the rise time of the heading response. This is due to the fact that the Tracker depends on knowing the rise time of the heading angle to perform the necessary calculation to obtain actual commanded heading. Typically, rise time is directly measured by analyzing the system response to a step. Since the first order assumption is accurate only for small changes, calculating the rise time is difficult in this case. Rise time, therefore, becomes a parameter that must be adjusted until the Tracker performs well. It can be difficult to choose a rise time in heading that gives the best accuracy for a given sample rate and aircraft configuration.

4.4 Robot Platform Implementation

Two-wheeled mobile robots have very similar dynamics to autopiloted UAVs. For an autopiloted UAV flying a constant altitude, the dynamics reduce to

$$\begin{aligned} \dot{z}_x &= v\cos(\psi) \\ \dot{z}_y &= v\sin(\psi) \\ \dot{\psi} &= \alpha_\psi(\psi^c - \psi) \\ \dot{v} &= \alpha_v(v^c - v) \end{aligned} \tag{4.14}$$

with constraints

$$\begin{aligned} 0 < v_{min} \leq & v \leq v_{max} \text{ and} \\ -\omega_{max} \leq & \dot{\psi} \leq \omega_{max}. \end{aligned} \tag{4.15}$$

The only difference between these dynamics and constraints and the two-wheeled mobile robot case is the robots velocity constraint

$$-v_{max} \leq v \leq v_{max}. \tag{4.16}$$

By simply adding a minimum velocity saturation to the velocity controller on the robot, a two-wheeled robot can mimic the dynamics of an autopiloted UAV.

An experiment was constructed to test the Trajectory Smoother and Tracker on a robot platform. Two-wheeled robots configured with minimum velocity constraints and equipped with low-level heading and velocity controllers were provided with commands generated by the Tracker. An overhead camera provided position and heading feedback at 30 Hz and wheel encoders allowed velocity feedback. These same sensing capabilities exist on UAVs in the form of GPS and airspeed, respectively. The Tracker was easy to tune in the robot case because a robot has true first order response in heading and so rise time was easily measured.

Figure 4.7 shows the result of the robot experiment. The error from the actual position of the robot and the desired potion given by the reference trajectory is less than 10 cm. We conclude from this that the Trajectory Smoother and Tracker are accurate for a platform with similar dynamics to autopiloted UAVs.

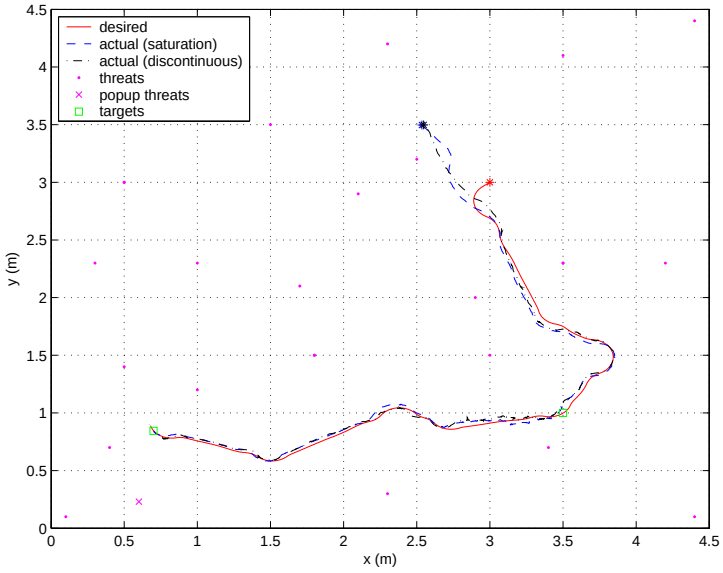

Figure 4.7: Robot experiment results.

4.5 Summary of Simulation and Validation

This chapter has introduced the special case handling needed for correct operation of the Trajectory Smoother in a discrete implementation. Specifically, the Trajectory Smoother was shown to be robust to input changes, and support for extreme turns was developed. Simulations were performed to ensure correct operation of the Trajectory Smoother and to validate assumptions made in the design of the Trajectory Smoother and Tracker. Using a precise model of aircraft dynamics, the assumption that an autopiloted UAV can be modelled as in Equation (2.1) was validated. It was shown, however, that α_ψ must be tuned well to accurately use the Tracker. A robot platform experiment showed that the Trajectory Smoother and Tracker operate in real-time with hardware similar to UAVs.

Chapter 5

The Transition to Hardware

This chapter will address the main issues involved in the transition from a simulation environment to hardware implementation as well as the hardware configurations that allow testing and operation of the Trajectory Smoother and Tracker.

5.1 Hardware Specific Considerations

The main goal for most engineering projects is performance on actual hardware. Of course, theory, simulation, and testing allow ideas to be validated and explored, but hardware implementation provides the truest test of the usability of a particular engineering solution. The most common obstacles to the transition from a successful simulation to real hardware are aspects of discretization, communication restraints, and delay.

5.1.1 Discretization

Most physical control systems today are sampled discretely and processed digitally. Often continuous methods that work well in simulation transition smoothly to the discrete world when the sample rate is fast enough. However, a smooth transition is not by any means guaranteed. If possible, any continuous algorithm that will be implemented digitally should be transitioned as early as possible to discrete space. This will allow the designer to understand the subtleties early in the design process.

For the most part, the dominant discretization issues have been addressed in Chapter 4. Changes were made to the initial construction of the reference trajectory (circle intersection methods and extreme turn compensation) to solve problems that

were primarily a result of discretization. It is also interesting to note that the first attempt to develop a tracker (of the adaptive strain) was discarded due to limitations arising from discretization. The current implementations of the Trajectory Smoother and Tracker work well at update rates of approximately 20 Hz.

5.1.2 Communication Constraints

Communication is not instantaneous, robust, or unlimited in any hardware system. Careful design is needed to minimize the frequency and size of data that must be communicated. If large amounts of critical data must be communicated, hardware should be selected that supports this, but a re-design may be the best solution. Heavy reliance on the communication channel must also be protected by default sensible behavior when the communication channel is lost. Failure to do so can be costly and dangerous.

5.1.3 Delay

The effect of delay, or hysteresis, in a system is perhaps the most overlooked aspect in the design process. All systems will have some delay – even computation time can be a significant source of delay. For many systems, the delay is small enough to be ignored, but for many others it must be explicitly addressed. Hysteresis in sensors and communication delays are the most common source of delay in an overall system.

One of the most important sensors for UAV navigation is GPS. We use the output of GPS as a position, heading, and ground speed sensor. Unfortunately, there is approximately $1/10^{\text{th}}$ second delay from the GPS receiver due to onboard calculation. Often, heading and velocity measurements from GPS are delayed up to 1.1 seconds. This delay can significantly affect the Tracker which depends on feedback of position and velocity to generate the next set of commands to the autopilot. To explicitly address the delay from the GPS receiver, an estimate of position and heading is generated using an Extended Kalman Filter. Chapter 6 addresses issues of real-time filtering in the presence of delayed sensor information.

Another source of delay in the system is the delay due to the communication from the Ground Station to the UAV. The Ground Station provides an interface to the operator allowing multiple levels of interaction. It communicates to the UAV over a low-cost wireless modem. This transmission link can lead to delay and bandwidth constraints due to slow baud rates. If critical control software runs on the Ground Station using sensor information gathered on the UAV, the delay from sensor output to received command can be significant. For example, if GPS positions are used to generate a new command from the Ground Station, then the total delay is the sum of the delay to (1) sample the GPS, (2) send the sample to the Ground Station, (3) process the information on the Ground Station, (4) send the new command back to the UAV, and finally, (5) implement that command. Any significant delay in the communication channel can cause major problems. The solution to dealing with this delay is to make sure that the information communicated over the wireless link is not timing critical. Often, such a bottleneck in the system will strongly influence how the overall system should be designed.

5.2 Short Path Segments

A specific hardware implementation challenge to the Trajectory Smoother is the handling of short path segments. This known weakness was first addressed by Anderson in [6]. The issue is that for a path to be dynamically feasible, two turns cannot overlap. In simulation, the path planner adjusts the path to remove most of the short segments using a simple threshold on segment length. In hardware, however, the path planning is done by human operators who may command short path segments. To address human interface issues as well as to fix the adjustment made by the path planner (it didn't work in all situations), an algorithm for extending a path to ensure no overlapping turns was developed.

This path extension algorithm transforms any path into a path with segments long enough to allow all turns to be completed before another is required. The algorithm "pushes" path segments out through the given waypoints until a length is reached that satisfies the needed non-overlapping condition. The extension involves

knowledge of the discretization of the system to ensure that turns do not overlap even in the presence of wide sample times. Figure 5.1 shows how a path is extended to allow the Trajectory Smoother to generate sensible trajectories.

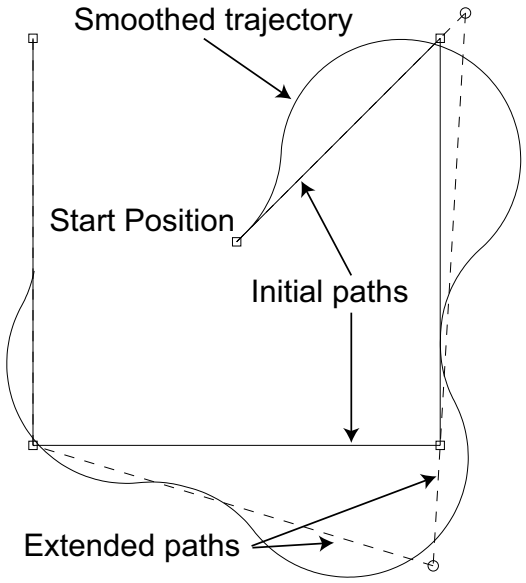

Figure 5.1: An extended path.

5.3 Tracking Tradeoffs

The Tracker has been shown to work well in UAV simulation and in a mobile robot scenario. As mentioned in Section 4.3, however, it can be difficult to tune for UAVs. The tuning process in simulation consisted of changing α_ψ until good tracking was observed. In hardware, the process is much more complicated. The presence of wind makes it difficult to know if tracking error is caused by wind or by an improperly tuned Tracker. In addition, GPS only updates at a rate of 1 Hz which is too slow for the tracker, even in simulation. Approximately a 20 Hz update rate is needed for the Tracker to perform well. Note that the robot experiment in Section 4.4 had an update rate of 30 Hz. To be able to use the Tracker in hardware, an Inertial

Navigation System (INS) is needed to estimate position at a 20 Hz rate (presented in Chapter 6) as well as careful tuning on a windless day.

An alternative is to use a simple "follow-the-rabbit" tracker. The idea here is to continuously update a point on the reference trajectory just in front of the UAV. The autopilot commands a heading to face the point and a velocity to stay a specified distance behind it. In effect, the moving point (the rabbit) leads the UAV along the desired trajectory. The advantage to this method is insensitivity to update rate. By allowing the Trajectory Smoother to generate the trajectory ahead of the UAV, the "rabbit" point is easily obtained. When GPS is received, the current point from the Trajectory Smoother is sent and the autopilot continuously controls the UAV to a heading and velocity. We chose to have the Trajectory Smoother lead the UAV by 2 seconds. This allowed a 1 Hz update to be sufficient for good tracking. There is no guarantee of convergence for this particular tracker and at 1 Hz the control is loose, but it does allow adequate tracking of the reference trajectory.

5.4 Software Configuration for Hardware Implementation

The development of the Trajectory Smoother has essentially been done in parallel with the development of the autopilot and physical aircraft. The volatility associated with design and testing has made it difficult to merge the two projects. Two phases of implementation exist to build up to complete hardware operation of the Trajectory Smoother. The first phase is the simplest possible setup to allow for rudimentary operation and testing of the system. In this phase, the Trajectory Smoother runs on the Ground Station at 20 Hz. When the GPS receiver updates the position of the UAV, the Ground Station is notified and the reference "rabbit" position is transmitted back in response. Autopilot routines control the UAV to the heading needed to point to the "rabbit" position and the velocity commanded by the Trajectory Smoother. This phase can be enhanced when an INS is operational onboard the UAV. In that case, communication bandwidth will be dedicated to the output of the Trajectory Smoother allowing the autopilot to control to a changing "rabbit" point at 20 Hz. Hopefully, this will allow tight control and tracking.

The second phase of hardware implementation relies on the availability of processing power onboard the UAV. The current software designs of the autopilot and Trajectory Smoother do not allow a real-time implementation when running together on the autopilot processor. In the future, the amount of calculation required to generate the reference trajectory should be reduced to enable the autopilot, Trajectory Smoother, and Tracker to run onboard the UAV. In this scenario, only user waypoints will be transmitted from the Ground Station to the UAV. With the communication link only issuing changes in desired waypoints, the communication constraints due to delay and bandwidth disappear. The reduction on the reliance of the communication link will allow multiple UAVs to operate from one Ground Station. It is worthy to note that the software has been designed to facilitate this transition. The handshaking and communication that will exist between UAV and Ground Station already take place inside the Ground Station software, i.e. the software is already separate to allow parts to be moved up to the UAV when more processing power is available.

5.5 Hardware Results

Numerous simulations and robot experiments showed that the Trajectory Smoother could be a viable solution to real-time trajectory generation for UAVs. The true test of the Trajectory Smoother is a test on actual UAV hardware.

The airframe on which the hardware experiments take place is a 48-inch-wingspan ZAGI glider [21] controlled by an autopilot board developed at BYU [22]. The Trajectory Smoother operates on the Ground Station with the follow-the-rabbit tracker configuration described in Sections 5.4 and 5.3. Figure 5.2 shows GPS data gathered from flight along with the desired trajectory. As can be seen, the flight of the UAV follows the shape and, roughly, the position of the reference trajectory. The discrepancy between the actual path and desired path is due primarily to the presence of wind and a slow update rate. The fact that the UAV path is near the reference trajectory over the entire flight suggests that coordination of UAVs is feasible using the Trajectory Smoother.

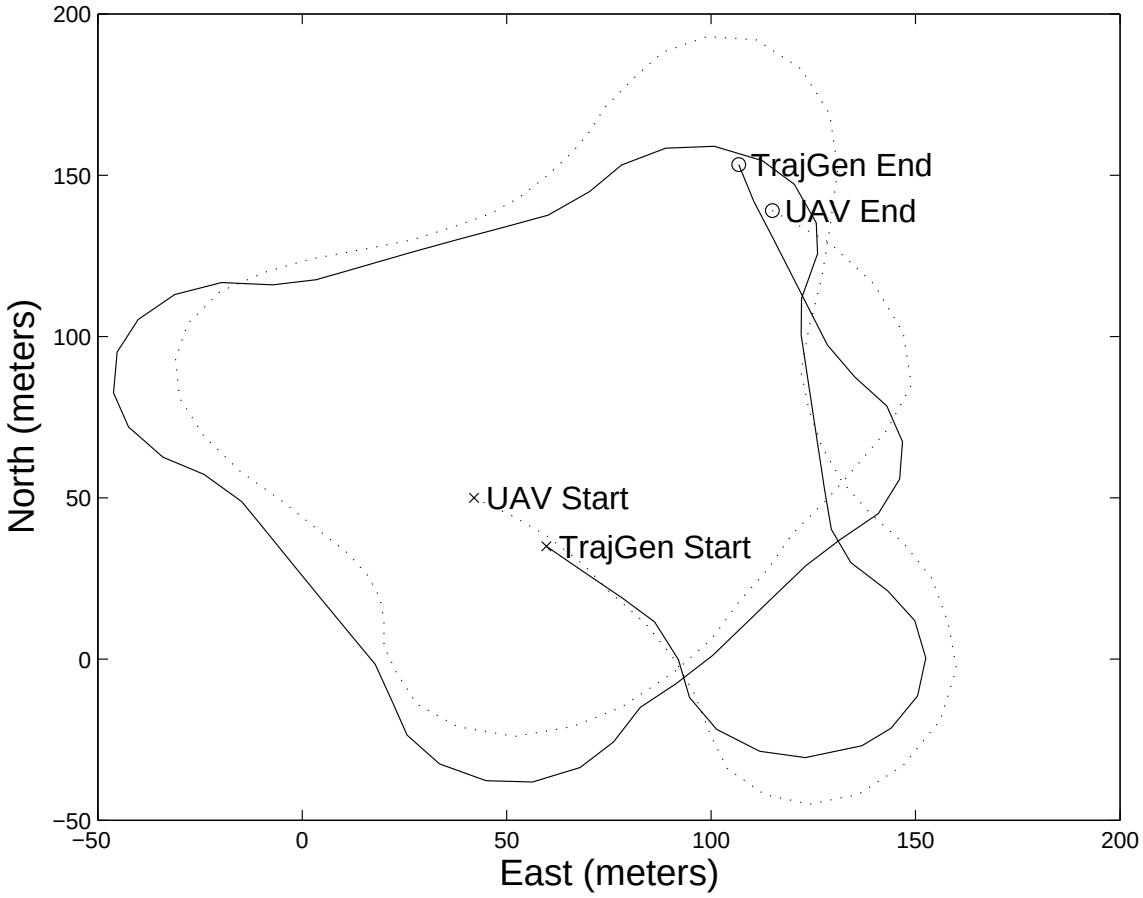

Figure 5.2: Results of hardware implementation

Chapter 6

Real-Time Filtering for Accurate Position Feedback

6.1 Introduction

To allow for better trajectory tracking, an estimate of position between GPS updates is needed. This will allow the Tracker to run at a rate fast enough to track the reference trajectory with little error. In order to produce an accurate position estimate, an accurate attitude estimate is also needed. This chapter develops attitude and position estimates for small UAVs (like those flown at BYU) where the payload and power requirements are extreme. Hardware issues of latency and noise are also addressed.

Recent interest in design and flight of small UAVs and Micro Air Vehicles (MAV) has prompted research into control and navigation of such vehicles. The potential for small inexpensive air vehicles is vast: reconnaissance, surveillance, search and rescue, remote sensing (nuclear, biological, chemical), traffic monitoring, natural disaster damage assessment, etc [23, 24]. A necessary part of these missions is accurate navigation of the vehicle. Typically, MAVs and small UAVs are very difficult to fly without a trained pilot [24]. To automate the stabilization and navigation of these vehicles, suitable estimation and control schemes are needed. This chapter will develop attitude and position estimation filters that use instruments that are small enough to fit on a small UAV (on the order of 50 cm wingspan).

Small UAVs and MAVs have constraints that prohibit traditional solutions to the attitude and position estimation problems. Most notably, accurate navigation-grade gyros are simply too large to be flown on these small aircraft. Power constraints

require that low-power embedded microprocessors be used, which can restrict the complexity of algorithms that can be implemented in real-time. One of the contributions of this chapter is to show that adequate estimation can be achieved even with these constraints. This solution can also be used as a back-up to more accurate estimation on larger aircraft.

This chapter will present a two-stage solution to attitude and position estimation. The first stage is attitude estimation. Many attitude estimation schemes are available. This chapter will present a straight-forward solution that operates in real-time and satisfies the weight and power constraints of a small UAV. In addition, the attitude estimation scheme will explicitly deal with latency using a unique distributed-in-time formulation. Attitude will be directly available for state feedback, and more importantly, as an input to the second-stage navigation filter. It is notable that with a cascaded filter approach (i.e. the output of the first stage filter feeds the second stage), *any* unbiased attitude estimation scheme can be used at the first stage, including vision-based [24] or infrared-based [25] schemes.

A formulation for an Attitude Heading Reference System (AHRS) will be presented in Section 6.2 with simulation results shown in Section 6.3. Section 6.4 contains the Inertial Navigation System (INS) formulation. Section 6.5 presents simulation results of the cascaded INS, and Section 6.7 offers conclusions.

6.2 Real-Time Distributed-in-time AHRS

Three primary concerns with attitude estimation are: (1) determining a way to mathematically represent attitude, (2) effectively using sensors to measure attitude, and (3) using the dynamics of the rigid body to reduce the measurement noise or provide an estimate of attitude between measurements. Initialization and latency compensation for real-time systems also needs to be addressed.

6.2.1 Attitude Representations

There are many different mathematical representations of attitude. The most general attitude parametrization is the Direction Cosine Matrix (DCM) [26]. This real

orthogonal matrix maps vectors from the reference frame to the body frame. However, because it requires nine parameters (six of which are dependent), it is desirable to find a representation with the fewest number of parameters. Euler angles represent the parametrization with the fewest elements (ϕ, θ, ψ). The DCM is generated from Euler angles by forming rotation matrices about a single axis and then multiplying these matrices together. Euler angles have clear intuitive meaning, but exhibit a singularity at certain angles. To overcome this drawback, the unit quaternion representation can be used.

Unit quaternions are defined by the relationship [26]

$$\boldsymbol{q} = [q_0 \ q_1 \ q_2 \ q_3]^T, \ \|\boldsymbol{q}\| = 1 \tag{6.1}$$

and the DCM associated with $\boldsymbol{q}$ is

$$\mathbf{D}(\boldsymbol{q}) = \begin{bmatrix} 1-2({q_2}^2+{q_3}^2) & 2(q_1q_2+q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2-q_0q_3) & 1-2({q_1}^2+{q_3}^2) & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3-q_0q_1) & 1-2({q_1}^2+{q_2}^2) \end{bmatrix}. \tag{6.2}$$

Most real-time attitude solutions use a quaternion parametrization of attitude because of the computational simplicity (no trigonometric calculations required) and the avoidance of singularities [26].

6.2.2 Attitude Measurements

With a good mathematical representation of attitude, the focus becomes effectively measuring attitude. A number of different solutions to the attitude measurement problem have been proposed and implemented. One popular method is to use the carrier phase of GPS signals [27, 28]. This involves at least three GPS antennas with a known geometry. Once the phase ambiguity is resolved, phase differences between antennas can be calculated and a good estimate of attitude is made. The solution improves as the baseline between antennas increases, making implementation on small UAVs or MAVs difficult. Gebre-Egziabher et al [29] proposed an ultra short baseline solution (approx 36 cm baseline), but even this is too large and heavy for small UAVs and MAVs [24, 30]. Other promising methods for small UAVs and

MAVs use vector measurements of the magnetic and gravitational fields and then solve a set of nonlinear equations using optimization methods to come up with an attitude measurement [31, 32]. A unique approach is to use the signal-to-noise ratio on the GPS antenna to measure attitude [33]. Because the end goal of estimating attitude is usually to control or stabilize the aircraft (i.e. use the estimate as state feedback), Akella et al [34] formulated a feedback law that skips the estimation step and regulates the attitude with only gyro and inclinometer measurements.

Perhaps the most straight-forward way to measure attitude is to use the accelerometers to give a measurement of roll (ϕ) and pitch (θ) with GPS velocity used to calculate heading (ψ). Complications arise, however, due to the aircraft acceleration in the reference frame. This means that the accelerometers will not measure the gravity vector, rather they will measure the aircraft apparent gravity ($\boldsymbol{g} - \boldsymbol{a}_{\text{aircraft}}$). By augmenting accelerometer measurements with GPS calculated accelerations, a good estimate of roll and pitch can be made [31, 35] [1]. In addition, the use of GPS to calculate heading will only work if the aircraft has a velocity from which to calculate heading, therefore, the AHRS presented in this chapter will only be valid for fixed-wing aircraft flying at velocities high enough to generate a GPS heading angle. This limitation can be overcome with the addition of a magnetic compass, but for aircraft where GPS heading is reliable, the extra hardware and complexity may not be desirable.

It should be noted that these attitude measurement schemes do not necessarily produce a measured quaternion vector. To determine the error between the estimated attitude and the measured attitude, one can formulate the estimation algorithm to create an estimated vector in the same frame as it is being measured (i.e. rotate the known reference magnetic field vector by the current estimate of attitude and then take the difference from the magnetometer reading [36]). One could also transform the measurements into a quaternion and then take the difference (although care must be taken to make sense of the quaternion error [31, 36]). For measurements of Euler angles, it is simple to transform the estimated quaternion attitude to Euler

[1] More on this can be found in section 6.2.4.

representation and then take the difference. To transform between quaternion and Euler representations, the following relationships are used: quaternion→Euler and Euler→quaternion [37]:

$$\text{eul}(\boldsymbol{q}) = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \tan^{-1}\left(\dfrac{2(q_2q_3 + q_0q_1)}{1 - 2({q_1}^2 + {q_2}^2)}\right) \\ \sin^{-1}(-2(q_1q_3 - q_0q_2)) \\ \tan^{-1}\left(\dfrac{2(q_1q_2 + q_0q_3)}{1 - 2({q_2}^2 + {q_3}^2)}\right) \end{bmatrix}, \tag{6.3}$$

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) - \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) - \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \end{bmatrix}. \tag{6.4}$$

6.2.3 Attitude Filtering

Once a mathematical structure is in place and a set of measurements available, a filter to combine the measurements with the dynamics of the vehicle can be formulated. Many filter formulations use an Extended Kalman Filter (EKF), but complementary filters and Unscented Kalman Filters (UKF) [36] are also used. In this thesis we will use an EKF.

The attitude dynamics of a rigid body can be described by

$$\dot{\boldsymbol{q}} = \boldsymbol{\Omega}(\boldsymbol{\omega})\boldsymbol{q} \tag{6.5}$$

where

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \frac{1}{2}\begin{bmatrix} 0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & \omega_3 & -\omega_2 \\ \omega_2 & -\omega_3 & 0 & \omega_1 \\ \omega_3 & \omega_2 & -\omega_1 & 0 \end{bmatrix} \tag{6.6}$$

and ω_1 is the roll rate, ω_2 the pitch rate, and ω_3 the yaw rate. The unit norm constraint of the quaternion must also be satisfied. Using discrete approximations (we use a simple Euler approximation) will necessitate normalization of the quaternion every time the quaternion is updated.

To avoid the need to know the inertia tensor of the vehicle (needed to compute angular accelerations) we use rate gyro measurements to update the quaternion estimate. While this introduces noise, it allows an implementation that is general for a large range of vehicles. Small UAVs and MAVs have very small payload capacity which necessitates the use of small MEMS devices to measure angular rates. These devices drift over time. Because the estimate of $\boldsymbol{q}$ is effectively the integral of the rate gyros (6.5), some compensation for drift will be needed to give a reliable estimate. For this reason the state vector is given by

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{b} \end{bmatrix} \tag{6.7}$$

where $\boldsymbol{q}$ is the quaternion estimate of attitude and $\boldsymbol{b}$ is the estimate of bias on the rate gyros. The drift on gyros can be modelled as a random walk, so the bias estimate is simply $\dot{\boldsymbol{b}} = \boldsymbol{0}$. The system is described by

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\omega}) \\ \boldsymbol{y} &= \boldsymbol{h}(\boldsymbol{x}) \end{aligned} \tag{6.8}$$

where

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\omega}) = \begin{bmatrix} \boldsymbol{\Omega}(\boldsymbol{\omega} - \boldsymbol{b})\boldsymbol{q} \\ \boldsymbol{0} \end{bmatrix}, \quad \boldsymbol{h}(\boldsymbol{x}) = \mathrm{eul}(\boldsymbol{q}) \tag{6.9}$$

and $\boldsymbol{\omega}$ is the vector of angular rates measured by the gyros.

To finish the formulation of the EKF for the state $\boldsymbol{x}$ as defined above, the Jacobian of the dynamics and the measurements are needed. These matrices are the linearizations about the current estimate for the state update and the measurement, respectively. They are defined as

$$\mathbf{A}_k = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_k,\ \boldsymbol{\omega}=\boldsymbol{\omega}_k} \qquad \mathbf{C}_k = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_k} \tag{6.10}$$

where $\hat{\boldsymbol{x}}_k$ is the best estimate of the state at time k and $\boldsymbol{\omega}_k$ is the vector of measured angular rates at time k.

Formulation of $\mathbf{A}_k$

This section shows the explicit expression for $\mathbf{A}_k$ as introduced in Equation (6.10). Recall that $\boldsymbol{q}$ is the quaternion representation of UAV attitude, $\boldsymbol{b}$ is the vector of estimated biases on the rate gyros, and $\boldsymbol{\omega}$ is the vector of measured angular rates evaluated at time k. $\mathbf{A}_k$ is given by

$$\mathbf{A}_k = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{0}^{3\times 4} & \mathbf{0}^{3\times 3} \end{bmatrix} \tag{6.11}$$

where

$$\mathbf{A}_{11} = \frac{\partial}{\partial \boldsymbol{q}} \Big\{ \boldsymbol{\Omega}(\boldsymbol{\omega} - \boldsymbol{b})\boldsymbol{q} \Big\} = \boldsymbol{\Omega}(\boldsymbol{\omega} - \boldsymbol{b})$$

and

$$\mathbf{A}_{12} = \frac{\partial}{\partial \boldsymbol{b}} \Big\{ \boldsymbol{\Omega}(\boldsymbol{\omega} - \boldsymbol{b})\boldsymbol{q} \Big\} = \frac{1}{2} \begin{bmatrix} q_1 & q_2 & q_3 \\ -q_0 & q_3 & -q_2 \\ -q_3 & -q_0 & q_1 \\ q_2 & -q_1 & -q_0 \end{bmatrix}.$$

Formulation of $\mathbf{C}_k$

This section shows the explicit expression for $\mathbf{C}_k$ as introduced in Equation (6.10). Recall that $\boldsymbol{q}$ is the estimated quaternion attitude at time k. Note that the bias terms in the state are not related to the measurements, so the partial derivative with respect to $\boldsymbol{b}$ is zero. $\mathbf{C}_k$ is given by

$$\mathbf{C}_k = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial \phi}{\partial \boldsymbol{q}} & \mathbf{0}^{3\times 1} \\ \frac{\partial \theta}{\partial \boldsymbol{q}} & \mathbf{0}^{3\times 1} \\ \frac{\partial \psi}{\partial \boldsymbol{q}} & \mathbf{0}^{3\times 1} \end{bmatrix} \tag{6.12}$$

where

$$\frac{\partial \phi}{\partial \boldsymbol{q}} = \frac{2c}{a^2 + c^2} \begin{bmatrix} q_1 \\ q_0 + 2q_1 a \\ q_3 + 2q_2 a \\ q_2 \end{bmatrix}^T \tag{6.13}$$

$$a = 2(q_0 q_1 + q_2 q_3), \quad c = 1 - 2({q_1}^2 + {q_2}^2)$$

and

$$\frac{\partial \theta}{\partial \boldsymbol{q}} = \frac{2}{\sqrt{1 - \gamma^2}} \begin{bmatrix} q_2 \\ -q_3 \\ q_0 \\ -q_1 \end{bmatrix}^T \tag{6.14}$$

$$\gamma = -2(q_1 q_3 - q_0 q_2)$$

and

$$\frac{\partial \psi}{\partial \boldsymbol{q}} = \frac{2\beta}{\alpha^2 + \beta^2} \begin{bmatrix} q_3 \\ q_2 \\ q_1 + 2q_2 \alpha \\ q_0 + 2q_3 \alpha \end{bmatrix}^T \tag{6.15}$$

$$\alpha = 2(q_1 q_2 + q_0 q_3), \quad \beta = 1 - 2({q_2}^2 + {q_3}^2).$$

6.2.4 Initialization

Initialization of the EKF involves producing an estimate of the state ($\hat{\boldsymbol{x}}_0$) along with the error covariance of that estimate ($\mathbf{P}_0$). To simplify the error covariance estimate, it is assumed that $\mathbf{P}_0$ is diagonal. To get a good estimate of the initial bias $\hat{\boldsymbol{b}}_0$, a simple measurement of the gyros on the aircraft before takeoff is made. The confidence of the initial estimate is very high (the aircraft is not moving so the measurements on the gyros must be the bias), so the lower 3 diagonal elements of $\mathbf{P}_0$ are set to very small numbers (e.g. $1e^{-6}$).

To initialize $\hat{\boldsymbol{q}}_0$ a measurement of attitude is made. For an application where the measurement is made using GPS measurements and requires a GPS velocity, this

must be done in the air (i.e. heading is only measured when the vehicle is moving). With a magnetic compass onboard, $\hat{\boldsymbol{q}}_0$ can be initialized before takeoff.

Attitude Measurements with Low-Cost GPS

Measurements of ϕ and θ are made by measuring the acceleration in the body frame and relating it to a reference acceleration vector in the Earth Centered Earth Fixed (ECEF) frame (usually the gravity vector $\boldsymbol{g} = [0\ 0\ g]^T$, where g is the gravitational constant). [2] The reference vector is constructed such that when the reference vector is measured in the body frame, this will correspond to zero roll and zero pitch angles.

To relate the measured acceleration in the body frame to the reference vector in the ECEF frame, a mathematical relationship between the two should be formalized. The rotation matrix from the ECEF frame to the body frame is

$$\mathcal{C}_{\text{ECEF}\rightarrow\text{body}} = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ \sin(\theta)\sin(\phi) & \cos(\phi) & \cos(\theta)\sin(\phi) \\ \sin(\theta)\cos(\phi) & -\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix} \tag{6.16}$$

and it follows that the transformation of acceleration in the ECEF frame to acceleration in the body frame is

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \mathcal{C}_{\text{ECEF}\rightarrow\text{body}} \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \tag{6.17}$$

where $\boldsymbol{a}$ is in the body frame and $\boldsymbol{r}$ is in the ECEF frame.

For a non-accelerating body, the reference acceleration vector will simply be the gravity vector $\boldsymbol{g} = [0\ 0\ g]^T$, where g is the gravitational constant, 9.81 m/s^2. The

[2] The reader should be aware that accelerometers measure the acceleration due to the forces *opposing* gravity; so if a measure of the gravity vector is desired and the accelerometers are configured to follow the convention of $+x$ out the nose, $+y$ out the right wing, and $+z$ out the belly of the aircraft, then the measurements will need to be negated to give a measurement of gravity. From a practical view, this means that the accelerometers are attached in such a way that, when stationary, a positive acceleration is measured in the direction of gravity.

structure of this reference vector gives rise to the following set of equations

$$a_x = -\sin(\theta)g \tag{6.18}$$

$$a_y = \cos(\theta)\sin(\phi)g \tag{6.19}$$

$$a_z = \cos(\theta)\cos(\phi)g \tag{6.20}$$

which can be easily solved for θ and ϕ in terms of the acceleration measured in the body frame as

$$\theta = -\sin^{-1}\left(\frac{a_x}{g}\right) \tag{6.21}$$

$$\phi = \tan^{-1}\left(\frac{a_y}{a_z}\right). \tag{6.22}$$

When the aircraft (and hence the body frame) is accelerating relative to the ECEF frame, the accelerometers will not measure the gravity vector, rather they will measure the aircraft apparent gravity ($\boldsymbol{g} - \boldsymbol{a}_{\text{aircraft}}$). The acceleration of the aircraft relative to the ECEF frame, $\boldsymbol{a}_{\text{aircraft}}$, is the second derivative of GPS measurements rotated by the heading ψ. The rotation by ψ is necessary to align the X and Y GPS measurements with the body X and Y axes. With knowledge of this acceleration, a new reference vector can be constructed and ϕ and θ can be solved for as follows:

$$\begin{aligned}
r_x &= -\left(\cos(\psi)a_{\text{GPS}_x} + \sin(\psi)a_{\text{GPS}_y}\right) \\
r_y &= -\left(-\sin(\psi)a_{\text{GPS}_x} + \cos(\psi)a_{\text{GPS}_y}\right) \\
r_z &= g - a_{\text{GPS}_z} \\
\sigma_\theta &= \frac{r_x a_x + r_z\sqrt{{r_x}^2 + {r_z}^2 - {a_x}^2}}{{r_x}^2 + {r_z}^2} \\
\theta &= \tan^{-1}\left(\frac{\sigma_\theta r_x - a_x}{\sigma_\theta r_z}\right) \qquad (6.23) \\
r_\theta &= r_x\sin(\theta) + r_z\cos(\theta) \\
\sigma_\phi &= \frac{r_y a_y + r_\theta\sqrt{{r_y}^2 + {r_\theta}^2 - {a_y}^2}}{{r_y}^2 + {r_\theta}^2} \\
\phi &= \tan^{-1}\left(\frac{-\sigma_\phi r_y + a_y}{\sigma_\phi r_\theta}\right). \qquad (6.24)
\end{aligned}$$

The procedure for obtaining an attitude measurement using a low-cost GPS receiver and three axis accelerometers is now addressed. While many GPS receivers

can be configured to output velocities as well as positions, it will be assumed that the GPS receiver outputs only position information at 1 Hz.

To make a measurement of attitude the following steps are followed:

1. Obtain three consecutive GPS position measurements,

2. Difference the GPS measurements to obtain two velocity measurements,

3. Average the velocity measurements to give average velocity over 2 seconds,

4. Calculate the heading ψ from velocity:

$$\psi = \tan^{-1}\left(\frac{\dot{Y}}{\dot{X}}\right), \tag{6.25}$$

5. Difference the GPS calculated velocities to obtain a GPS acceleration measurement $\rightarrow \boldsymbol{a}_{\text{GPS}}$,

6. Average the accelerometers over the same 2 seconds as the GPS velocity is calculated $\rightarrow \boldsymbol{a}$,

7. Calculate roll ϕ and pitch θ using the accelerometers and GPS acceleration rotated by ψ using Equations (6.23) and (6.24).

Once attitude has been measured, Equation (6.4) is used to transform the measured Euler angles to an initial estimate of $\hat{\boldsymbol{q}}_0$. The confidence in $\hat{\boldsymbol{q}}_0$ is only as high as the confidence in the measurement, therefore, the top four diagonal elements of $\mathbf{P}_0$ should reflect the uncertainty in the measurement. Since the quaternion representation of attitude combines elements of all Euler angles, it is typical to assign the same confidence value to all elements of the quaternion. Simulations used $\mathbf{P}_0 = \text{diag}([0.1\ 0.1\ 0.1\ 0.1\ 0\ 0\ 0])$.

6.2.5 Latency and Real-Time Computation

When measurements are made in the manner outlined in section 6.2.4, significant latency is introduced. To evaluate the latency of a measurement, it is assumed

that a GPS receiver outputs positions at a rate of 1 Hz with latency of approximately 1/10th of a second (due to calculation onboard the GPS module and communication from the GPS receiver to the microprocessor).

Referring to Figure 6.1, by the time the information is available to make a measurement of attitude, the measurement will be delayed by 1.1 seconds. Note that to make the measurement, accelerometer values need to be known from the time the first GPS position considered was received – this requires logging of accelerometer measurements over a 2.1 second interval.

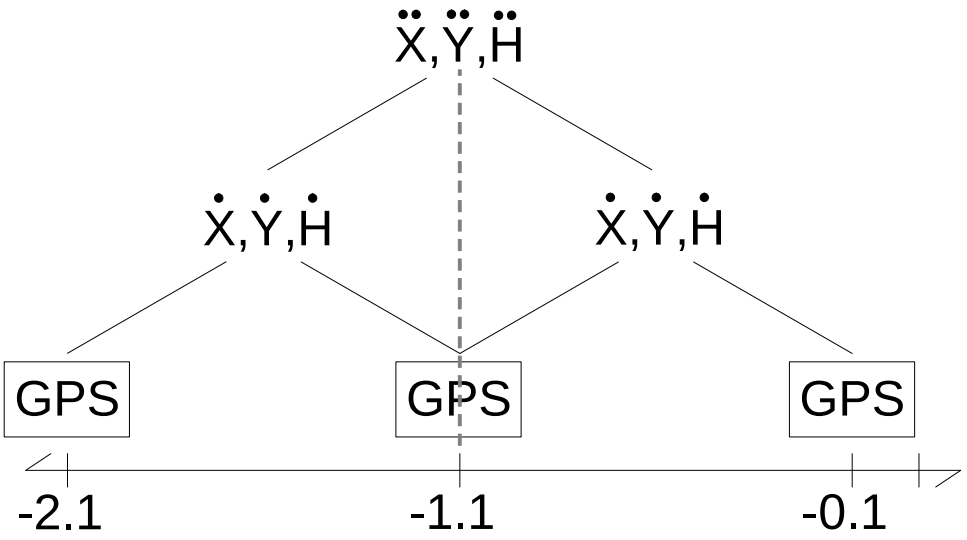

Figure 6.1: Latency from GPS acceleration calculation.

To compensate for latency, it is proposed that all sensor and state information be data logged over the 2.1 second period from which GPS acceleration will be calculated. Once a measurement is generated (with latency of 1.1 seconds), the state estimate corresponding to 1.1 seconds previous is updated. The updated estimate is then propagated using the stored sensor information up to the current time.

The propagation of the state from the measurement update up to the current time will dominate the computation time required to run the filter. For most systems,

there will not be enough computational ability to do this without significantly reducing the sample rate. To address latency while simultaneously allowing for a fast filter update rate, a distributed-in-time computation architecture is used. Figure 6.2 outlines the design of a distributed-in-time filter. The idea is to only call the time update

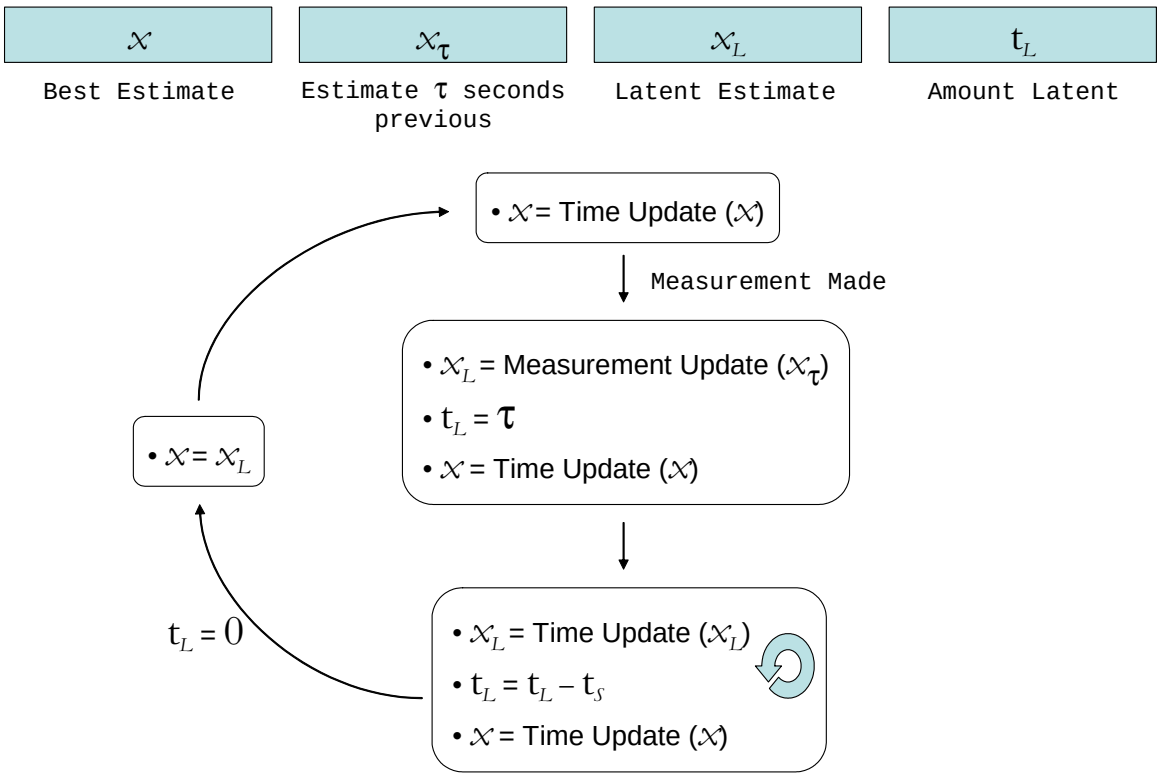

Figure 6.2: Distributed-in-time filter architecture.

routine often enough each time step to ensure real-time capability while also guaranteeing that the updated estimate reaches current-time before the next measurement is made. While the updated estimate is being propagated in time, a previous best estimate should also be propagated to make the best no-latency estimate available until the updated estimate has zero latency.

To illustrate the distributed-in-time architecture and to demonstrate real-time capability, an implementation of the AHRS has been designed and tested. The target

filter update rate is 30 Hz. An 8-bit microcontroller running at 14.7 MHz was used. On this platform, the time update takes 3.75 milli-seconds and the measurement update takes 9 milli-seconds. With these computational constraints and without a distributed-in-time architecture, the filter could not address latency without bringing the filter update rate down to below 15 Hz. To design a 30 Hz update rate, the computation dealing with latency is spread over multiple updates of the filter, with the computation for each step not to exceed 33 milli-seconds. Figure 6.3 shows how

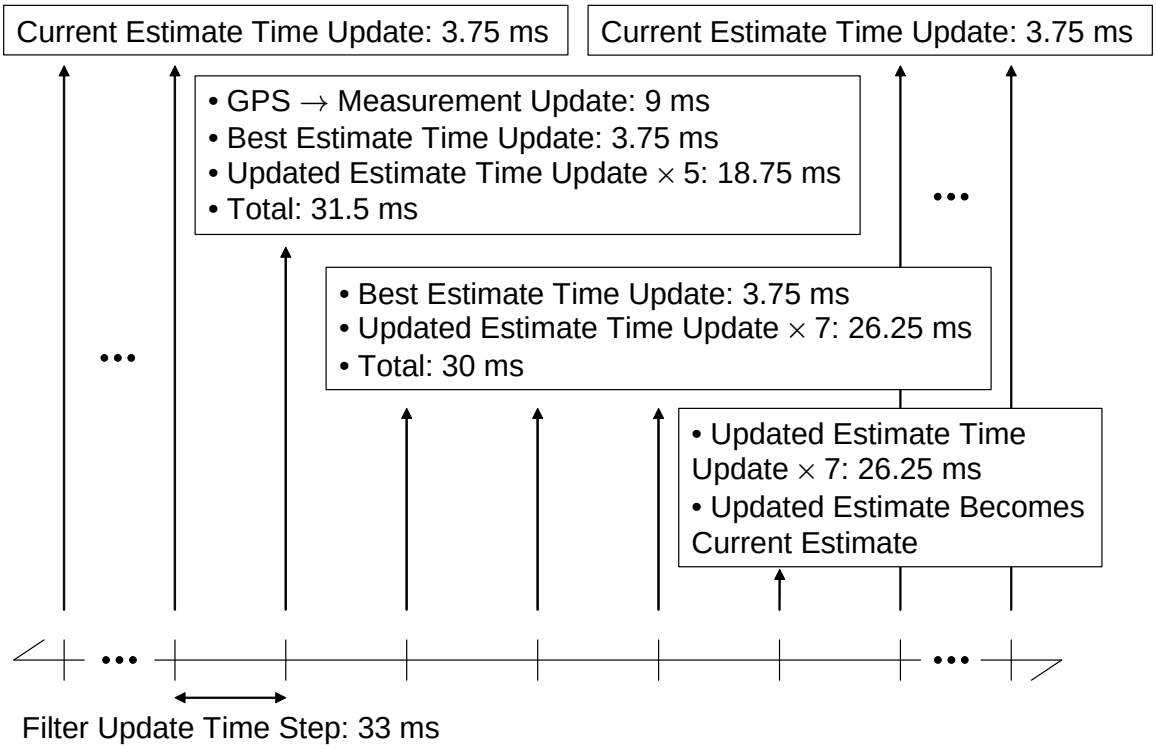

Figure 6.3: Latency compensation at 30 Hz.

the computation is distributed to ensure a no-latency estimate at 30 Hz. Note that the measurement updated estimate gets propagated 33 times (corresponding to the 1.1 seconds of latency) over 5 updates of the filter. A no-latency EKF with these

computational constraints could be designed to run at 78 Hz – the same rate that a normal EKF could run.

Theorem 6.1 *A no-latency filter can run at the same rate as its counterpart if the time required for a measurement update is greater than or equal to twice the time required for a time update and the latency in the measurement is less than twice the time between measurements.*

Proof: Let T be the sum of the time required for a time update and the time required for a measurement update. Then, when a measurement is made, T seconds of computation will be required. Therefore, a standard EKF could run at a rate no greater than $1/T$ Hz. If the latency associated with a measurement is less than twice the time between measurements and a latent estimate is time-updated twice every time step, then the latency will be zero by the time the next measurement is made. In order to propagate a latent estimate twice every time step and also propagate the best no-latency estimate, three time updates per time step are required. If the time required to perform three time updates is less than T (corresponding to the time for a measurement update being at least twice as large as the time for one time update), then the maximum computation time required at any given time step is still T, which means that the no-latency EKF will run at the same rate as the EKF that has latency.

Note that, in general, the time required to do a measurement update will be larger than the time required to perform a time update (due to the need to invert a matrix during the measurement update). Often, latent measurements have small latency compared to the time between measurements, so a no-latency EKF will, in many cases, be able to operate at the same rate as its counterpart.

The last issue to deal with in connection with the AHRS concerns measurements that are outside of the dynamic range of the aircraft. When a measurement is so corrupted by noise that it is not possibly a valid measurement, its inclusion in the filter update causes instability. When these extraneous measurements (due mainly to inaccuracies in the GPS acceleration calculation) are used in the filter, the bias estimates are skewed horribly until the next valid measurement. With the wrong bias

estimate, the attitude estimate is propagated incorrectly and the filter is completely inaccurate until the biases return to their true value. To overcome this, when a measurement is received, it is compared with the 3-σ bound (computed from the error covariance matrix). If the measurement is outside the limits, then a measurement update is not performed.

Summary

Initialize with:

$$\hat{\boldsymbol{x}}_0 = \mathcal{E}[\boldsymbol{x}_0] \quad \mathbf{P}_0 = \mathcal{E}[(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0)(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0)^T]$$

Time update:

$$\begin{aligned}
\hat{\boldsymbol{q}}_{k+1} &= \hat{\boldsymbol{q}}_k + T_s \boldsymbol{\Omega}(\boldsymbol{\omega}_k - \hat{\boldsymbol{b}}_k)\hat{\boldsymbol{q}}_k \\
\hat{\boldsymbol{q}}_{k+1} &= \frac{\hat{\boldsymbol{q}}_{k+1}}{\left\|\hat{\boldsymbol{q}}_{k+1}\right\|} \\
\hat{\boldsymbol{b}}_{k+1} &= \hat{\boldsymbol{b}}_k \\
\mathbf{P}_{k+1} &= \mathbf{P}_k + T_s(\mathbf{A}_k\mathbf{P}_k + \mathbf{P}_k\mathbf{A}_k^T + \mathbf{Q})
\end{aligned}$$

Measurement update:

Load $\hat{\boldsymbol{x}}_k$ and $\mathbf{P}_k$ from data log

$$\begin{aligned}
\mathbf{K}_k &= \mathbf{P}_k\mathbf{C}_k^T(\mathbf{C}_k\mathbf{P}_k\mathbf{C}_k^T + \mathbf{R})^{-1} \\
\hat{\boldsymbol{x}}_{k+1} &= \hat{\boldsymbol{x}}_k + \mathbf{K}_k(\boldsymbol{z}_k - \mathrm{eul}(\hat{\boldsymbol{q}}_k)) \\
\hat{\boldsymbol{q}}_{k+1} &= \frac{\hat{\boldsymbol{q}}_{k+1}}{\left\|\hat{\boldsymbol{q}}_{k+1}\right\|} \\
\mathbf{P}_{k+1} &= (\mathbf{I} - \mathbf{K}_k\mathbf{C}_k)\mathbf{P}_k
\end{aligned}$$

Use time update equations to propagate up to current time

$\boldsymbol{z}$ is the Euler angle measured attitude, $\mathbf{R}$ is the covariance of measurement noise, and $\mathbf{Q}$ is the covariance of process noise.

6.3 AHRS Simulation Results

This section describes simulation results of the AHRS formulated in section 6.2. A full 6 degree of freedom model provides truth values. The simulated aircraft performed an 11 minute flight with multiple coordinated turn and climb maneuvers. Figure 6.4 shows the roll angle over the simulated flight (pitch and yaw angles show similar noise characteristics). The solid line is the true roll angle and the dashed line is the estimate from the AHRS.

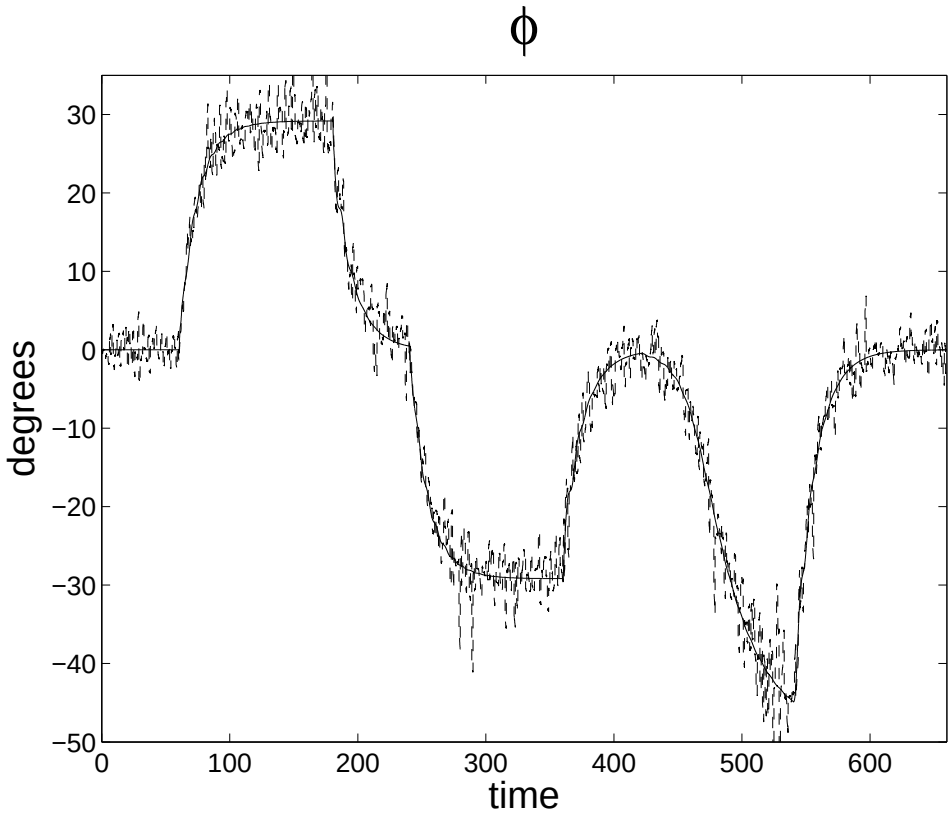

Figure 6.4: Roll angle over 11 minute flight.

Noise values on simulated sensors were selected based on manufacturer specifications and experimental data. Noise values were modelled as Gaussian $\mathcal{N}(0, \sigma^2)$, drift values were modelled as Random Walk (i.e. $\int \mathcal{N}(0, \sigma^2)$), and GPS was modelled

as a Gauss-Markov Process with delay of 1/10th of a second and a correlation time of 100 seconds. Table 6.1 show the standard deviation (σ) of each sensor.

Table 6.1: Simulated noise characteristics.

	σ
Accelerometers	2 mm/s^2
Accelerometer drift	0.5 mm/s^2
Gyros	0.03°/s
Gyro drift	3°/min
Altitude	2 m
GPS X,Y	5 m

The main goal of an AHRS for a small UAV or MAV is to provide real-time attitude information using small, cheap sensors. The AHRS satisfies this goal with attitude estimation standard deviation (σ) as given in Table 6.2.

Table 6.2: Standard Deviation of Estimate from Truth.

	σ
ϕ	3.25°
θ	2.33°
ψ	2.40°
p bias	0.96°
q bias	0.84°
r bias	0.75°

6.4 Cascaded INS Filter

The choice of cascading an AHRS filter with a separate INS filter was based primarily on the ease of design and computational efficiency. In a cascaded format, the AHRS and INS filters can be designed and tuned completely independent of each other. The relationship between the two filters can be described succinctly (and fairly accurately) by the input noise parameters to the INS filter. A cascaded structure also reduces computation. By removing the calculations that relate the attitude states to the inertial states, the cascade formulation has much less computational overhead than a full EKF. Even though the cascaded filter is sub-optimal [38, 39], the performance is comparable to the full EKF.

In other words, the cascade implementation allows for increased flexibility, is easier to implement and tune, and is dramatically less computationally expensive, with only a small performance loss.

6.4.1 INS Formulation

The general UAV equations of motion [10] for navigation are

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = \text{DCM}^T \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{6.26}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \text{DCM} \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \frac{1}{m}\boldsymbol{F} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{6.27}$$

where DCM is defined as in section 6.2.1; u, v, and w are the body velocities out the nose, the right wing, and the belly, respectively; X and Y are the inertial coordinates of the UAV in the Earth Centered Earth Fixed (ECEF) frame; $-Z$ is altitude; g is the gravitational constant; m is the mass of the UAV; $\boldsymbol{F}$ is the translational forces acting on the UAV; p, q, and r are the roll, pitch, and yaw rates, respectively.

A typical AHRS/INS filter will propagate the states in Equations (6.26) and (6.27) along with attitude and gyro bias states. This can be problematic because the

$\boldsymbol{F}/m$ term can be hard to characterize. Many implementations simply use accelerometers to estimate this term, but this is basically integrating the accelerometers twice to get position – any small drift in the accelerometers can cause problems. To avoid this and to break the filter into a cascaded format, the following assumptions will be made: (1) attitude is available from a preceding AHRS filter, (2) v and w are much less than u and close to zero (this is equivalent to assuming small angle of attack and very little sideslip), and (3) a measurement of altitude is available even when GPS lock has been lost (say from an absolute pressure sensor). With these assumptions in place, the cascaded INS filter takes the form given in Figure 6.5.

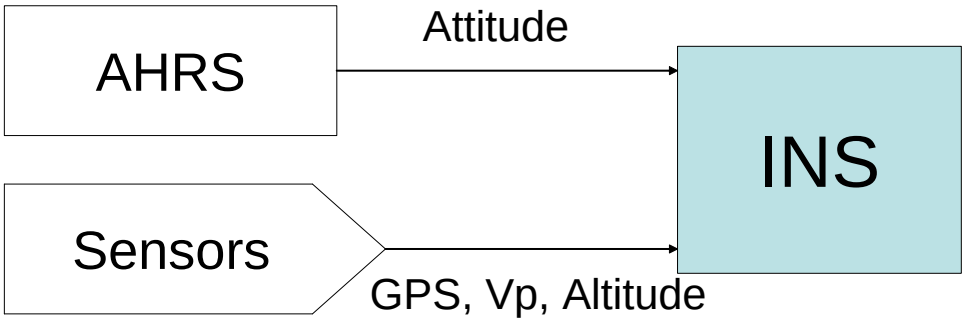

Figure 6.5: Diagram of Cascaded INS

The INS filter is summarized in the following table.

Inputs:

1. Attitude to form DCM
2. Airspeed (V_p)

Measurements:

1. GPS X and Y positions
2. Altitude

State:

$$\boldsymbol{x} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{6.28}$$

Time Update:

$$\hat{\boldsymbol{x}}_{k+1} = \hat{\boldsymbol{x}}_k + T_s \left(\text{DCM}^T \begin{bmatrix} V_p \\ 0 \\ 0 \end{bmatrix} \right) \tag{6.29}$$

$$\mathbf{P}_{k+1} = \mathbf{P}_k + T_s(\mathbf{G}_k \mathbf{Q} \mathbf{G}_k^T)$$

Measurement Update:

$$\begin{aligned} \mathbf{K}_k &= \mathbf{P}_k(\mathbf{P}_k + \mathbf{R})^{-1} \\ \hat{\boldsymbol{x}}_{k+1} &= \hat{\boldsymbol{x}}_k + \mathbf{K}_k(\boldsymbol{z}_k - \hat{\boldsymbol{x}}_k) \\ \mathbf{P}_{k+1} &= (\mathbf{I} - \mathbf{K}_k)\mathbf{P}_k \end{aligned}$$

z is the measured state (GPS and altitude), $\mathbf{R}$ is the covariance of measurement noise, and $\mathbf{Q}$ is the covariance of input noise. $\mathbf{G}_k$ is the partial derivative of the dynamics of the system with respect to each of the inputs.

In forming the INS, note that the inertial states are driven by the body velocities rotated by the attitude (Equation (6.26)). Treating attitude as the input to this filter allows us to form the DCM from the inputs, reducing the computation associated with calculating the statistical relationship between attitude changes and the resulting change in position. Using this same notion of reduction of complexity by replacing states, we can replace the vector $[u\ v\ w]^T$ by $[V_p\ 0\ 0]^T$ where V_p is the measured airspeed. In this way, we avoid the complex state update in Equation (6.27) as well as the problematic measurement of $[u\ v\ w]^T$ (this isn't actually too hard, but requires angle of attack and angle of sideslip sensors).

Note that u, v, and w are all inputs, with $u \approx V_p$ and v, $w \approx 0$. With noise characteristics defined for all attitude inputs as well as on u, v and w estimates, relationships from inputs to state changes can be related through $\mathbf{G}_k$. Also note that $\mathbf{A}_k$ is not present because state changes are driven completely by inputs.

Formulation of $\mathbf{G}_k$

$\mathbf{G}_k$ will depend on the attitude representation used to form the DCM. For a quaternion representation, $\boldsymbol{u} = [q_0\ q_1\ q_2\ q_3\ u\ v\ w]^T$ and the DCM (denoted $\mathbf{D}(\boldsymbol{q})$) is defined in Equation (6.2). For the state as defined in Equation (6.28), the system dynamics are

$$\boldsymbol{f}(\boldsymbol{u}) = \mathbf{D}\left(\boldsymbol{q}\right)^T \begin{bmatrix} u \\ v \\ w \end{bmatrix}. \tag{6.30}$$

Therefore

$$\mathbf{G}_k = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}=\boldsymbol{u}_k}, \tag{6.31}$$

where we set $u = V_p$ at time k and v, $w = 0$, and

$$\mathbf{G}_k = \begin{bmatrix} \mathbf{G}_1 & \mathbf{G}_2 \end{bmatrix} \tag{6.32}$$

where

$$\mathbf{G}_1 = \begin{bmatrix} 0 & 0 & -4V_pq_2 & -4V_pq_3 \\ 2V_pq_3 & 2V_pq_2 & 2V_pq_1 & 2V_pq_0 \\ -2V_pq_2 & 2V_pq_3 & -2V_pq_0 & 2V_pq_1 \end{bmatrix}$$

and

$$\mathbf{G}_2 = \mathbf{D}(\boldsymbol{q})^T.$$

The dimensions of $\mathbf{G}_k$ are 3×7, so $\mathbf{Q}$, if diagonal, will have the variance of the quaternion estimates as the top 4 elements, the variance of u (confidence that airspeed equals body forward speed) as the 5th element, and confidence in the assumptions that $v = 0$ and $w = 0$ as the 6th and 7th elements, respectively.

The advantages of an INS in this format are very low computational overhead and robustness even with loss of GPS or noisy inputs from attitude and airspeed.

6.5 INS Simulation Results

The INS presented in Section 6.4 was simulated in the same manner as the AHRS to determine accuracy of the INS filter. The INS was also programmed in C code for an 8-bit 14.7 MHz microprocessor. Real-time ability of the INS is demonstrated by the execution times of the INS on this platform: 2 milli-seconds for both time and measurement updates.

Two flight tests were performed; one where GPS was available for the full time and one where GPS information was ignored for the remainder of the flight after the filter had run for a short time. Figure 6.6 shows the filter results with the corresponding GPS measurements for the case when GPS connection is uninterrupted. A magnified portion of the data is shown to verify that the INS correctly smoothes through GPS measurements, providing an estimate of position at every instant of time. Figure 6.7 shows the case when GPS lock is lost halfway through the flight. As can be seen, even after 5 minutes of no GPS lock, the INS only strays a maximum of 22 meters from the true position. In both figures the solid line is the true position and the dashed line is the estimated postion from the INS.

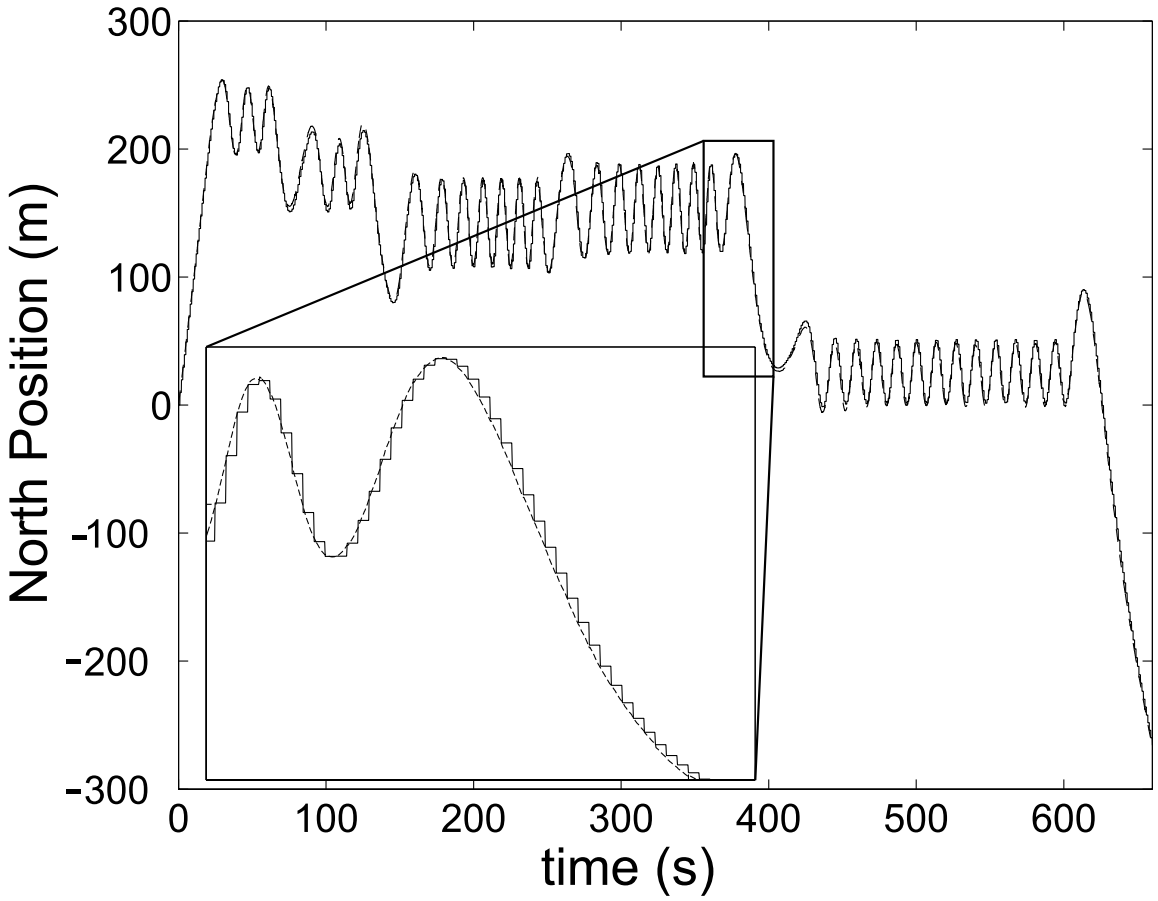

Figure 6.6: INS performance with no loss of GPS

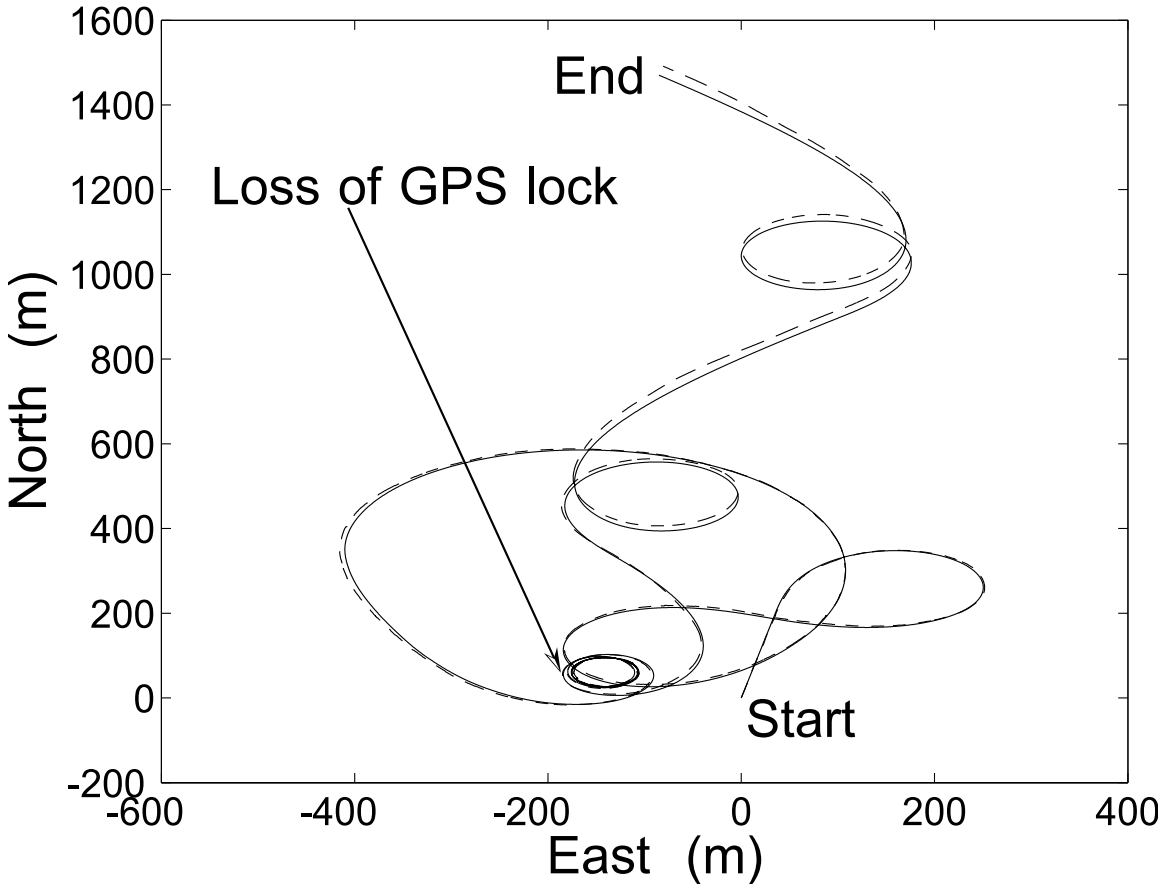

Figure 6.7: INS performance with loss of GPS

The performance of the INS is very good considering the quality of the inputs. Figure 6.8 shows the noise on the inputs to the INS. It is concluded that the INS is robust to both input noise and GPS loss.

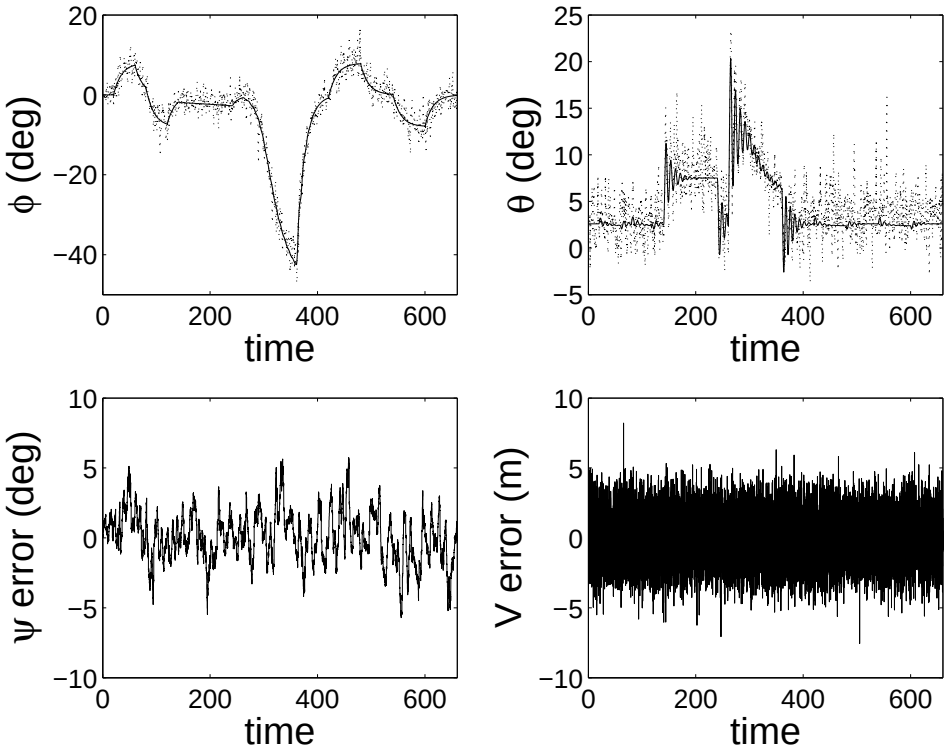

Figure 6.8: Input noise levels to INS

6.6 Preliminary Hardware Results

To verify the results obtained through simulation, a hardware experiment was performed. A suite of sensors identical to those used in BYUs low-cost autopilot was placed in a cradle at the end of a 2.5 meter boom. A motor drove the setup at variable speeds in a circle – effectively putting the sensors in a coordinated turn. Truth values for roll angle were obtained through a potentiometer attached to the cradle; pitch angle was assumed to be zero due to the mounting of the sensors in the cradle. Truth

values for inertial position and heading angle were obtained through integrating the encoder on the driving motor and using the geometry of the setup. As can be seen in Figure 6.9, the AHRS estimates roll and pitch angles to within 2 degrees. Figure 6.10 shows that the performance of the INS is also very accurate. It should be emphasized that these results were obtained with actual hardware sensors and in a manner that very closely approximates actual coordinated flight.

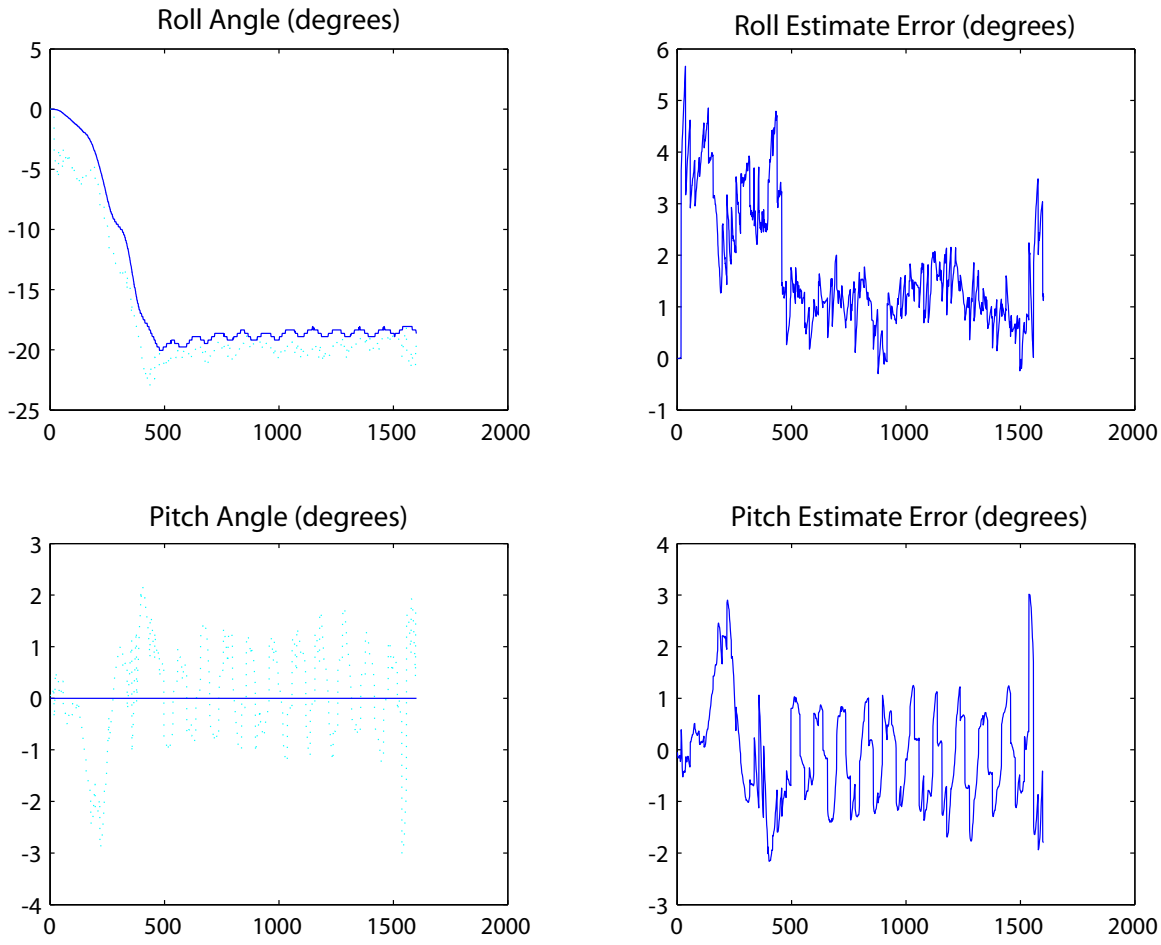

Figure 6.9: Attitude estimation during a preliminary hardware experiment

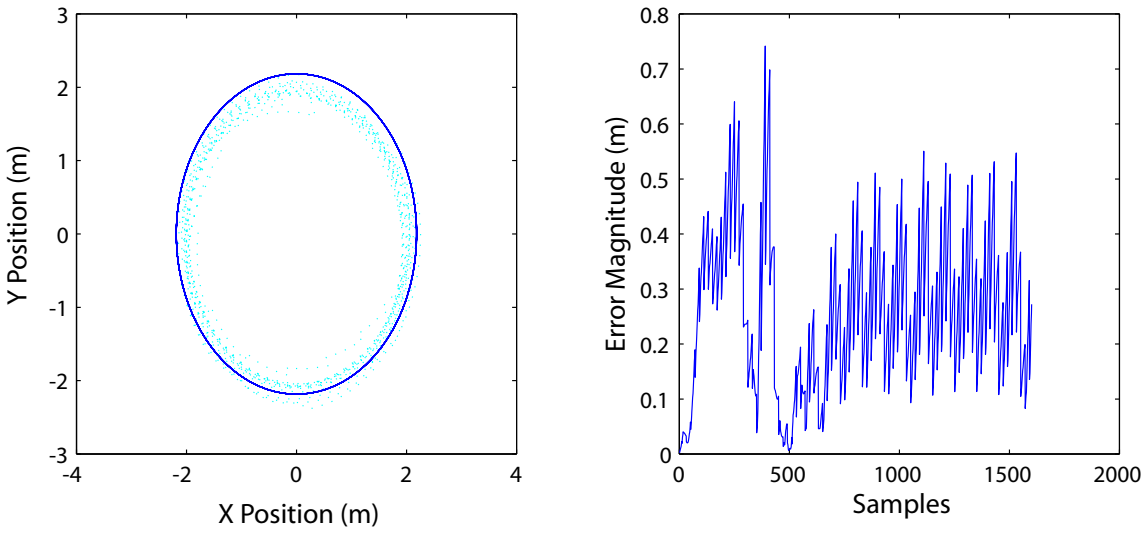

Figure 6.10: Position estimation during a preliminary hardware experiment

6.7 Conclusions

This chapter has demonstrated a real-time solution to attitude and position estimation for small UAVs and MAVs. A real-time AHRS using inexpensive, light-weight components has been developed that explicitly deals with latency and has been shown to be adequate for small UAV and MAV applications. A cascaded filter implementation of an INS has been demonstrated in simulation and preliminary hardware tests and has been shown to be robust to GPS loss and input noise.

For small UAV and MAV applications, payload is critical. For the AHRS and INS as outlined in this chapter only very light-weight, inexpensive, small sensors are needed. The total sensor payload including the GPS receiver is less than 0.45 oz. (12.7 grams). With future miniaturization, a complete sensor suite plus microprocessor could possibly be developed to be small enough to be flown on a MAV. A current hardware implementation at BYU weighs 2.2 oz. and has demonstrated autonomous flight on aircraft with wingspans as small as 21 in. With the filtering algorithms shown in this chapter, small UAVs and MAVs can effectively and efficiently estimate their attitude and position.

Chapter 7

Conclusions and Future Work

7.1 Future Work

The results from hardware experimentation are promising. Trajectories can be generated in real-time and followed in a manner that allows for cooperative control. At the same time, much improvement can be made. The designs of the Trajectory Smoother and Tracker do not explicitly address the effects of wind. In particular, feedback does not exist between the Tracker and the Trajectory Smoother. Modifying the Trajectory Smoother to slow down or speed up depending on the state of the Tracker should help account for wind and other disturbances.

A hardware implementation of the INS described in Chapter 6 will allow a faster update rate of the Tracker which will lead to tighter control. In addition, the Tracker proposed in [19, 20] can be tuned and implemented for better convergence.

Extending trajectory generation to three dimensions will allow more complex applications. This could be done using the same basic ideas of two dimensional trajectory generation, i.e. pull-up and push-down radii define transitions between vertical segments. A first step in three dimensional planning is to split the altitude dimension into multiple planes and plan in two dimensions with simple connections between planes.

7.2 Conclusions

This thesis has shown that trajectory generation can be done in real-time and will guide actual UAVs in a manner suitable for cooperative control. This claim is

supported by hardware experiments of trajectory generation and tracking on actual UAVs. Issues involved with transitioning from an algorithm to a hardware experiment were discussed at length. A method for estimating attitude and position was presented which will allow for better performance on actual hardware.

Bibliography

[1] D. S. Bernstein, "A student's guide to research," *IEEE Control Systems Magazine*, vol. 19, pp. 102–108, February 1999.

[2] D. S. Bernstein, "Four and a half control experiments and what I learned from them: A personal journey," in *Proceedings of the IEEE American Control Conference*, Albuquerque, NM, June 1997, pp. 2718–2725.

[3] C. R. Knospe and E. H. Maslen, "Meaningful control experiments," in *Proceedings of the IEEE American Control Conference*, Albuquerque, NM, June 1997, pp. 2703–2707.

[4] A. Alleyne, S. Brennan, B. Rasmussen, R. Zhang, and Y. Zhang, "Controls and experiments: Lessons learned," *IEEE Control Systems Magazine*, vol. 23, pp. 20–34, October 2003.

[5] J. M. Fowler and R. D'Andrea, "A formation flight experiment," *IEEE Control Systems Magazine*, vol. 23, pp. 35–43, October 2003.

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

[7] S. Yurkovich, Ü. Özgüner, and K. M. Passino, "Control system testbeds and toys: Serendipitous or suspect," in *Proceedings of the IEEE American Control Conference*, Albuquerque, NM, June 1997, pp. 2692–2696.

[8] 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.

[9] 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).

[10] J. Roskam, *Airplane Flight Dynamics and Automatic Flight Controls*, Design, Analysis and Research Corporation, Lawrence, KS, 2001.

[11] R. W. Beard, T. W. McLain, and M. A. Goodrich, "Coordinated target assignment and intercept for unmanned air vehicles," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Washington, DC, May 2002, pp. 2581–2586.

[12] A. W. Proud, M. Pachter, and J. J. D'Azzo, "Close formation flight control," in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, Portland, OR, August 1999, AIAA Paper No. 99-4207.

[13] P. R. 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.

[14] T. W. McLain and R. W. Beard, "Cooperative rendezvous of multiple unmanned air vehicles," in *AIAA Guidance, Navigation and Control Conference*, Denver, CO, August 2000, AIAA Paper No. 2000-4369.

[15] D. Eppstein, "Finding the k shortest paths," *SIAM Journal of Computing*, vol. 28, no. 2, pp. 652–673, 1998, http://www.ics.uci.edu/~eppstein/pubs/p-kpath.html.

[16] S. J. Fortune, "A sweepline algorithm for Voronoi diagrams," 1987, http://netlib.bell-labs.com/cm/cs/who/sjf/index.html.

[17] V. Jimenez and A. Marzal, "Eppstein's algorithm: A C++ implementation," 1999, http://terra.act.uji.es/REA.

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

[19] 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.

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

[21] Trick R/C, "Zagi thermal/handlaunch," http://zagi.com/html/Airplanes/zagi-thl.html.

[22] D. B. Kingston, R. W. Beard, T. W. McLain, M. Larsen, and W. Ren, "Autonomous vehicle technologies for small fixed wing UAVs," in *AIAA 2nd Unmanned Unlimited Systems, Technologies, and Operations–Aerospace, Land, and Sea Conference and Workshop & Exhibit*, San Diego, CA, September 2003, AIAA Paper No. 2003-6559.

[23] R. J. Fontana, E. A. Richley, A. J. Marzullo, L. C. Beard, R. W. T. Mulloy, and E. J. Knight, "An ultra wideband radar for micro air vehicle applications," in *Proceedings of the IEEE Conference on Ultra Wideband Systems and Technologies*, 2002, pp. 187–191.

[24] S. M. Ettinger, M. C. Nechyba, P. G. Ifju, and M. Waszak, "Vision-guided flight stability and control for micro air vehicles," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and System*, 2002, vol. 3, pp. 2134–2140.

[25] B. Taylor, C. Bil, S. Watkins, and G. Egan, "Horizon sensing attitude stabilisation: A VMC autopilot," in *International UAV Systems Conference*, Bristol, UK, 2003.

[26] J. R. Wertz, Ed., *Spacecraft Attitude Determination and Control*, D. Reidel Publishing Company, Dordrecht, Holland, 1978.

[27] H. M. Peng, Y. T. Chiang, F. R. Chang, and L. S. Wang, "Maximum-likelihood-base filtering for attitude determination via GPS carrier phase," in *Proceedings of the IEEE Position, Location, and Navigation Symposium*, San Diego, CA, March 2000, pp. 480–487.

[28] Y. T. Chiang, L. S. Wang, F. R. Chang, and H. M. Peng, "Constrained filtering method for attitude determination using GPS and gyro," in *Proceedings of the IEEE Radar, Sonar and Navigation*, October 2002, vol. 149, pp. 285–264.

[29] D. Gebre-Egziabher, R. C. Hayward, and J. D. Powell, "A low-cost GPS/inertial attitude heading reference system (AHRS) for general aviation applications," in *Proceedings of the IEEE Position Location and Navigation Symposium*, Palm Springs, CA, April 1998, pp. 518–525.

[30] B. Motazed, D. Vos, and M. Drela, "Aerodynamics and flight control design for hovering micro air vehicles," in *Proceedings of the IEEE American Control Conference*, Philadelphia, PA, June 1998, vol. 2, pp. 681–683.

[31] D. Gebre-Egziabher, G. H. Elkaim, J. D. Powell, and B. W. Parkinson, "A gyro-free quaternion-based attitude determination system suitable for implementation using low cost sensors," in *Proceedings of the IEEE Position, Location, and Navigation Symposium*, San Diego, CA, March 2000, pp. 185–192.

[32] J. L. Marins, X. Yun, E. R. Bachmann, R. B. McGhee, and M. J. Zyda, "An extended Kalman filter for quaternion-based orientation estimation using MARG sensors," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Maui, HI, October 2001, vol. 4, pp. 2003–2011.

[33] J. Madsen, "Obtaining 3-axis attitude solutions from GPS signal to noise ratio measurements," *AIAA Journal of Guidance, Control and Dynamics*, 2003, In review.

[34] M. R. Akella, J. T. Halbert, and G. R. Kotamraju, "Rigid body attitude control with inclinometer and low-cost gyro measurements," in *Elsevier Systems and Control Letters 49*, Santa Barbara, CA, Feburary 2003, pp. 151–159.

[35] J. Vaganay, M. J. Aldon, and A. Fournier, "Mobile robot attitude estimation by fusion of inertial data," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993, vol. 1, pp. 277–282.

[36] J. L. Crassidis and F. L. Markley, "Unscented filtering for spacecraft attitude estimation," *Journal of Guidance, Control, and Dynamics*, vol. 26, pp. 536–542, August 2003.

[37] MathWorks, "Euler angles to quaternions," http://www.mathworks.com/access/helpdesk/help/toolbox/aeroblks/euleranglestoquaternions.shtml.

[38] N. A. Carlson, "Federated square root filter for decentralized parallel processes," in *Proceedings of the IEEE National Aerospace and Electronics Conference*, Dayton, Ohio, May 1987, pp. 1448–1456.

[39] F. H. Schlee, N. F. Toda, M. A. Islam, and C. J. Standish, "Use of an external cascade Kalman filter to improve the performance of a global positioning system (GPS) inertial navigator," in *Proceedings of the IEEE Aerospace and Electronics Conference*, Dayton, OH, May 1988, pp. 345–350.