Title
Controlling Small Fixed Wing UAVs to Optimize Image Quality from On-Board Cameras

Author
Jackson, Stephen Phillip

Publication Date
2011

Peer reviewed|Thesis/dissertation

Controlling Small Fixed Wing UAVs to Optimize Image Quality from On-Board Cameras

by

Stephen Phillip Jackson

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor J. Karl Hedrick, Chair
Professor Raja Sengupta
Professor Francesco Borrelli

Fall 2011

Controlling Small Fixed Wing UAVs to Optimize Image Quality from On-Board Cameras

Abstract

Controlling Small Fixed Wing UAVs to Optimize Image Quality from On-Board Cameras

by

Stephen Phillip Jackson

Doctor of Philosophy in Mechanical Engineering

University of California, Berkeley

Professor J. Karl Hedrick, Chair

Small UAVs have shown great promise as tools for collecting aerial imagery both quickly and cheaply. Furthermore, using a team of small UAVs, as opposed to one large UAV, has shown promise as being a cheaper, faster and more robust method for collecting image data over a large area. Unfortunately, the autonomy of small UAVs has not yet reached the point where they can be relied upon to collect good aerial imagery without human intervention, or supervision. The work presented here intends to increase the level of autonomy of small UAVs so that they can independently, and reliably collect quality aerial imagery.

The main contribution of this paper is a novel approach to controlling small fixed wing UAVs that optimizes the quality of the images captured by cameras on board the aircraft. This main contribution is built on three minor contributions: a kinodynamic motion model for small fixed wing UAVs, an iterative Gaussian sampling strategy for rapidly exploring random trees, and a receding horizon, nonlinear model predictive controller for controlling a UAV's sensor footprint.

The kinodynamic motion model is built on the traditional unicycle model of an aircraft. In order to create dynamically feasible paths, the kinodynamic motion model augments the kinetic unicycle model by adding a first order estimate of the aircraft's roll dynamics. Experimental data is presented that not only validates this novel kinodynamic motion model, but also shows a 25% improvement over the traditional unicycle model.

A novel Gaussian biased sampling strategy is presented for building a rapidly exploring random tree that quickly iterates to a near optimal path. This novel sampling strategy does not require a method for calculating the nearest node to a point, which means that it runs much faster than the traditional RRT algorithm, but it still results in a Gaussian distribution of nodes. Furthermore, because it uses the kinodynamic motion model, the near optimal path it generates is, by definition, dynamically feasible.

A nonlinear model predictive controller is presented to control the non-minimum phase problem of tracking a target on the ground from a UAV with a fixed camera. It is shown that this novel controller is probabilistically guaranteed to asymptotically converge to the

path that minimizes the cross-track error of the UAV's sensor footprint. In addition, for a minimum phased problem, it is shown that its tracking performance is on par with a sliding mode controller, which at least theoretically, is capable of achieving perfect tracking.

Finally, all three of these contributions are experimental validated by performing a variety of tracking tasks using the Berkeley Sig Rascal UAV.

To my parents, who have always encouraged me,

and my brother, who has always inspired me.

Contents

List of Figures

Acknowledgments

Being a student in the Vehicle Dynamics Lab at Berkeley has been an amazing experience. It has been a privilege to work with so many spirited, motivated, and intelligent people.

In particular, I want to thank Professor Hedrick for giving me the freedom, confidence and resources to pursue my personal interests, and Professor Sengupta for his encouragement, advice and enthusiasm. I also want to thank my labmates Xiao Xiao, Jack Tisdale, Jared Garvey, Ben Lavis, Zu Kim, Will Grossman, Chris Richardson, Sam Vroomen, and especially Mike Morimoto, Mark Godwin, and Brandon Basso for their invaluable support in class and at Camp Roberts; and my roommates, Paul Cripe, Pascal Martin, and Conor O'Brien who have been more like brothers than roommates. Lastly, I want to thank my parents Jerry and Sheba, and my actual brother Wesley, who have never hesitated to give me their absolute confidence and support, and from time to time, palm fronds.

Chapter 1

Introduction

Small UAVs have shown great promise as tools for collecting aerial imagery both quickly and cheaply. But their limited autonomy has made it difficult to capitalize on their potential as a large sensor network made up of cheap individual agents. Without greater autonomy, the human supervision required to manage large networks of UAVs will grow linearly as the number of agents increases. In particular, the amount of aerial imagery a team of UAV collects becomes unmanageable very quickly. The first step in solving that problem is ensuring that the imagery a UAVs does collect is atleast on target, and of usable quality.

This work presents a novel approach to controlling the sensor footprint of a UAV such that it optimizes the quality of the images the UAV collects. The approach is composed of three innovations. Firstly, an innovation on the traditional unicycle model that improves it's accuracy by adding kinodynamic constraints. Secondly, a sampling strategy for a rapidly exploring random tree that quickly converges to a near optimal path by creating a Gaussian distribution of nodes around the 'best' path. And lastly a receding horizon, nonlinear model predictive controller that leverages the other two innovations to solve the non-minimum phase problem of controlling a UAV's sensor footprint.

These three innovations come together to create a tracking controller that is capable of reliably putting any target path within a UAV's sensor's field of view.

1.1 Motivation

The prevalence of Unmanned Aerial Vehicles (UAVs) has increased dramatically in the last decade [19]. Globaly, over $5bn was spent on UAVs in 2009, and it is expected that another $70bn will be spent on UAVs in the next decade[12]. Although large UAVs like the Predator and Global Hawk have received a lot of attention in the media, the majority of UAVs are small enough to be hand launched, and carry little more than a simple camera and video transmitter[1].

The plurality of small UAVs has lead to an explosion of research in recent years that

aims to exploit the efficiency and scalability of these UAVs by creating large sensor networks composed of intelligent autonomous agents[45][42]. Most of the work in this area assumes that a UAV's *sensor footprint*, the projection of a UAV's sensor's FOV onto the ground, encompasses the ground within a given radius around the aircraft. That assumption implicitly assumes that the cameras on the UAV are equipped with gimbals that can either keep the camera pointed directly down as the UAV rolls, or can point the camera at any location within that given radius. That assumption is often necessary because it significantly reduces the complexity of the control problem. Either the UAV is only required to stay in close proximity to its target, while the gimbal actually tracks it, or, the UAV is assumed to have a camera always pointing perpendicular to the ground, effectively reducing the tracking problem to two dimensions.

Unfortunately, these assumptions become less valid as the size of the UAV is reduced. Small UAVs have very strict power and payload constraints. Small UAVs may simply not have the payload capacity to carry a gimballed camera. If they do, the limited power available would reduce both the speed and accuracy of the gimbal, both of which are necessary for good tracking performance.

In reality, small UAVs generally have fixed cameras, which results in relatively poor sensor on target tracking performance. Solutions so far have included, flying wings level over a target, orbiting a target at a constant radius, and using cameras with a large field of view. Predictably, these solutions do not work well unless the target is a static point or a straight path. Likewise, because the sensor on the UAV has to have a larger field of view in order to keep the target within the image, the resolution of the target is lower than it could be, and is therefore significantly less useful.

In order for small UAVs to autonomously gather aerial imagery, the must be capable of controlling their sensor footprint to reliably track arbitrary targets.

1.2 Previous Work

For several years, the Center for Collaborative Control of Unmanned Vehicles has been working on the problem of using vision detection to track locally linear ground structures using small fixed wing UAVs with fixed downward looking cameras. Initially, the strategy was to use visual servoing to align a straight road vertically in the center of the field of view of the camera [16][14]. The next approach was to plan a path from the aircraft to the target path using a sline curve, and to apply backstepping to generate a control input[37][38]. That controller was used to track a straight path, and then a slightly curving canal. It was assumed that the canal was locally linear since in any given image frame it appeared to be straight. Again the goal of the controller was to position the plane directly over the canal so that in stable flight, the plane would be relatively level and the canal would appear nearly in the center of the image.

Fundamentally, the same controller was also used to track a naturally winding river.

Though the controller tracked the relatively straight section of the river fairly well, it consistently over shot the river around sharp bends. Moreover, even when the UAV tracked the river exactly as intended around each bend, the camera would lose sight of the river due to the necessarily large bank angle of the aircraft [36]. In particular, this experiment showed the need for a controller that can minimized the cross-track error of a UAV's sensor footprint, and not just the cross-track error of the UAV. Unfortunately, finding a feasible aircraft path that keeps an arbitrary target path within a UAV's sensor's field of view is an NP hard problem.

A significant amount of work has been done to try and solve this type of NP-hard path planning problem[26][35][15]. Likewise, there has been significant work devoted to controlling UAVs to follow desired aircraft trajectories[5][28]. Based on the literature, finding a feasible path using a rapidly exploring random tree seemed like the most natural approach. Moreover, solving the path planning problem fast enough and reliably enough to apply backstepping, seemed like the best method for control.

1.3 Experimental Platforms

Experimental verification is an important part of developing any controller. Developing real time controllers often requires the use of model reduction techniques that reduce the number of states of the plant to a model and controller that can run in real time. Model reduction makes the assumption that some states have such little impact on the output of the plant, that they can be neglected from the model of the plant. Experimentation using high fidelity models, or even better, the actual plant, is the only way to validate those assumptions. The Center for the Collaborative Control of Unmanned Vehicles (C3UV) has access to both high fidelity simulations, and a small fleet of UAVs.

Every UAV is composed of three main elements: an aircraft, an autopilot and a payload. C3UV's fleet is comprised of four types of aircraft two mid sized airplanes, the 110 Sig Rascal and the MLB Bat IV, and two smaller flying wings, the Berkeley Wing and the Zagi-XS.

Each aircraft carries its own autopilot and a variety of payloads. The autopilot controls the surfaces of the aircraft (e.g. throttle, ailerons) in order to maintain a target altitude, airspeed, and bank angle. The payload consists of one or more sensors, and may include a small computer and additional wireless transmitter's and receivers. The payload may also be adjusted to include addition power either in the form of more batteries, or more fuel.

Different degrees of autonomy obviously require more or less additional computing. Feed forward flight paths require very little on board computation. Examples of this type of behavior would include a mission to survey a plot of land. A flight path can be loaded into the UAV before take off, and images from the plane could either be stored on the aircraft or transmitted wirelessly to the ground. Sensing for motion can greatly increase the autonomy of a UAV, but requires significantly more computing power, which in some cases must reside on the aircraft.

UAV's may require on board computing for several reasons. If the computations are required for feedback, the latency of a wireless connection to an off board computer could cause the feedback loop to perform poorly, or possibly make it unstable. Similarly, the amount of data being sent may exceed the bandwidth of the wireless channel. Lastly, wireless connections are susceptible to greater noise than wired connections, which can increases latency, decreases bandwidth, and corrupt data.

Computer vision requires both a lot of data and many computations. Since each UAV at C3UV has a computer on board the aircraft, they are all capable of running fast vision-in-the-loop control algorithms. As a result, C3UV's fleet of UAVs operates with an unprecedented level of autonomy by implementing sensing for motion algorithms at the tasking, searching and tracking levels.

1.3.1 Piccolo Autopilot

The Piccolo is a commercial off the shelf autopilot produced by Cloud Cap Technologies[2]. It is approximately $24in^3$ and weights just over 7oz, see figure 1.1. It incorporates GPS, pitot and static pressure sensors, and data from a three axis gyroscope and accelerometer. The software is capable of stabilizing a variety of aircraft including both rotary and fixed wing UAVs. The low level controllers are capable of tracking reference velocities, altitudes, yaw angles and bank angles. The mid level controllers are capable of tracking paths in the air defined by a series of waypoints. Each waypoint is defined by a latitude, longitude and altitude. In addition to it's controllers, the piccolo also runs an observer, which estimates the mean wind velocity.

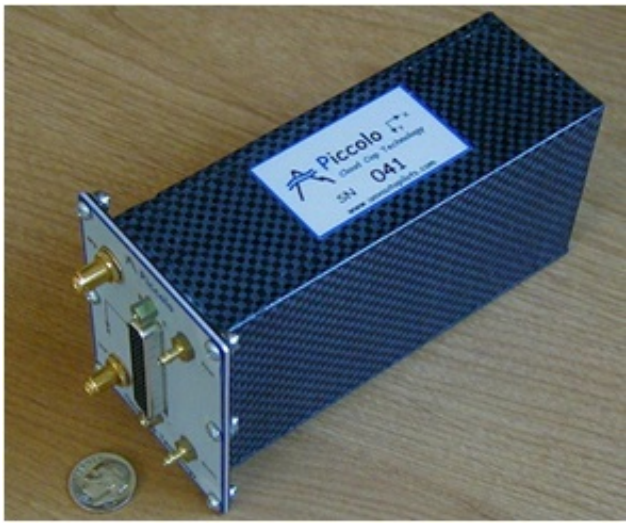

Figure 1.1: The Piccolo autopilot.

The Piccolo communicates with Cloud Cap's ground station over a proprietary, spread spectrum 900 MHz channel (see fig). The Cloud Cap ground station is capable of communicating with multiple Piccolo's, but it can only facilitate manual control of one UAV at a time. It can, however, connect to a computer running Cloud Cap's Command Center software, which gives the ground station operator control of every UAV at once. The Command Center has access to every autopilot parameter on each Piccolo, and is the primary method of communication with the Piccolo autopilot.

Figure 1.2: The Cloudcap groundstation.

In addition to the ground station, the computer on board the UAV is also capable of communicating with the Piccolo autopilot via two serial port connections. Using an SDK provided by Cloud Cap, the on board computer is able to access the same parameters as the Command Center, which, in effect, gives the computer on board the aircraft absolute control. For safety, the on board computer includes code that prevents it from writing to the Piccolo whenever it is targeting waypoint 99. Waypoint 99 is therefore a terminal state of the autonomous system. As a precaution, before each flight, a safe location is chosen for the waypoint 99 so that in the event of total loss of communication, the UAV would go to a safe location to loiter.

1.3.2 Arduino Autopilot

The Ardupilot is and open source autopilot being developed by the DIYdrones community, and it is very small, which figure 1.3 shows. At this point, the controllers on the

Ardupilot are significantly less sophisticated than those on the Piccolo, but they are cable of stabilizing most fixed wing aircraft. The Ardupilot is a fraction of the weight and size of the Piccolo, and orders of magnitude less expensive, which makes it an ideal choice for inexpensive, light weight UAVs[3].

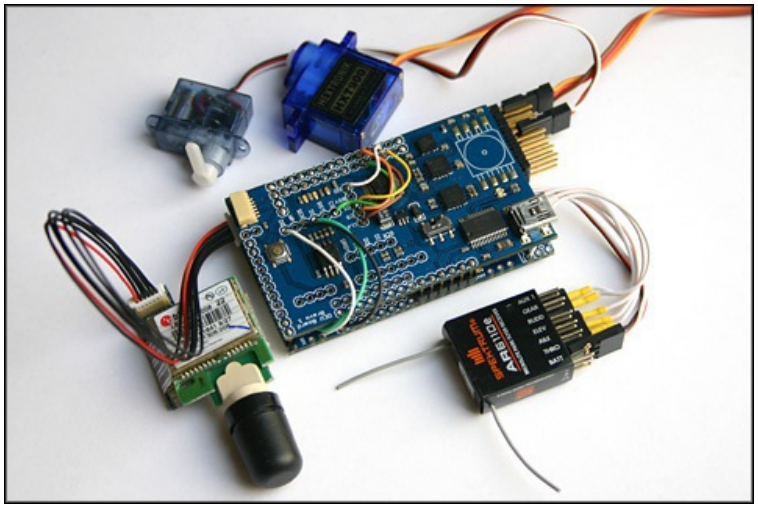

Figure 1.3: The Ardupilot open-source, open-hardware autopilot.

The Ardupilot hardware is a variation of the Arduino Mega micro controller, with a shield that incorporates GPS,a 6 axis gyroscope and accelerometer, and the optional to add a pitot and static pressure sensors, along with a 3 axis magnetometer. The Ardupilot also has an available serial port, which can be used to relay data between the autopilot and an on board computer.

QGroundControl is an open source UAV ground station project that is compatible with several UAV autopilots including the Ardupilot. The Ardupilot uses an Xbee commercial off the shelf radio to communicate with the ground. The Xbee radio can send and receive data from a computer on the ground, but manual control of the aircraft is handled by a standard R/C transmitter and receiver.

1.3.3 HIL Simulation

Hardware-in-the-loop (HIL) simulations are an important part of experimental flight tests. HIL simulations use as much of the actual flight hardware as possible, and the highest fidelity aircraft simulator available, which inputs the surface commands, and outputs fake sensor telemetry. HIL simulation is the last step before any experimental flight, and ensures that the experimental code will perform as expected on the actual hardware that is going to be flown.

Cloud Cap provides an extremely high fidelity simulator that uses a parametric model to simulate any particular UAV in both steady and turbulence wind. The Ardupilot uses a custom built interface to an open source flight simulator called FlightGear. FlightGear provides a descent simulation of the mid level dynamics, but does not have enough fidelity to simulate low level dynamics.

Vision-in-the-loop simulations can be done by using virtual graphics to simulating cameras. This can be done with a variety of programs including FlightGear, and Google Earth. The premise is the same in both cases. In order to simulate the image that would be generated by a camera on board the aircraft, the position and attitude data must be ported from the aircraft simulator to the graphic simulator. That data can then be used to render an image based on the UAV's telemetry. That image can then be captured and ported to vision processing component of the autopilot.

In many cases, HIL simulations can have high enough fidelity that there is little difference between the simulated and actual flight data. The differences are only apparent when evaluating the results of low level controllers, which can vary drastically depending on the amount of wind in real life.

1.3.4 Sig Rascal

The Sig Rascal is heavily modified R/C airframe with a 110in wingspan, and a 32cc, two stroke engine gasoline engine, figure 1.4. It has a maximum gross weight of 27lbs and a maximum flight time of 1.5 hours. It is constructed primarily of balsa wood with a Monokote skin, but it's mid section has been torn out and retrofitted with carbon fiber in order to accommodate its payload.

Figure 1.4: The Sig Rascal Berkeley UAV.

Because of its single cylinder and oversize engine, a custom engine mount was built to

isolate the engine's vibrations, shown in figure 1.5. In principle, the mount attaches from the aircraft to the back side of the engine along it's center of gravity such that the vibrations are mostly around the axis of the mount. The mount is also attached along the sides of the engine to give resistance to the moments about the engine. The mounts on the side of the engine have a relatively low spring constant, which isolates the aircraft from a significant portion of the engine's vibrations.

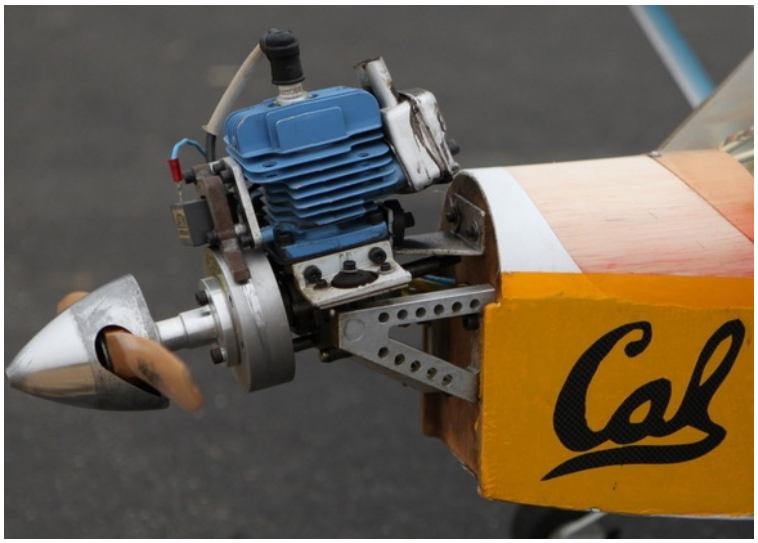

Figure 1.5: A custom vibration isolating engine mount.

The Sig Rascal Fleet is designed to have modular payloads. While the power train and autopilot are permanently mounted to the airframe, the payload is mounted to a removable tray which also serves as the belly of the aircraft. This payload tray is independent of the aircraft such that a variety of payload can be loaded into any one aircraft, and so a given payload could be load in to any one of the Sig Rascals.

Payload

The Sig Rascal's payload is highly reconfigurable, but usually carries a PC104 form factor computer and a variety of antenna's and sensors, see figure 1.6. The PC104 computer is powered by a dual Core Pentium processor, and has a 802.11g wireless modem. The modem is connected through a 1 watt amplifier to an omni directional antenna mounted to the bottom. The PC104, camera's and gimbal are mounted to the aircraft and payload tray with vibration isolating mounts. All of the electrical connections between the aircraft (power, serial connections to the autopilot, video signal) are bundled together into a 32 pin umbilical cable.

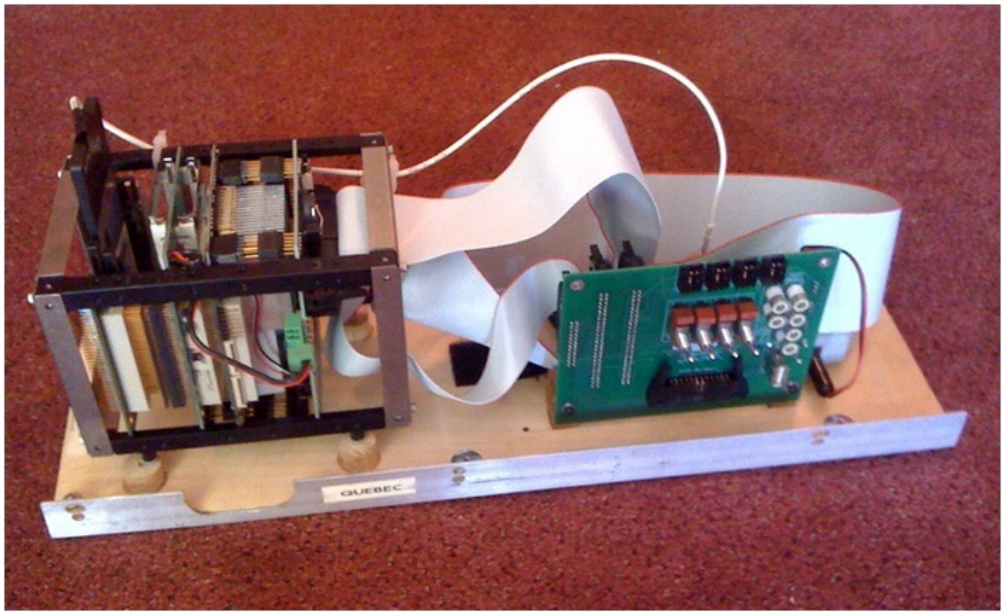

Figure 1.6: The Sig Rascal experimental payload module.

1.3.5 MLB Bat IV

The Bat IV is a much larger airframe that was designed in conjunction with a commercial UAV manufacturer, MLB. Figure 1.7 shows a picture of this UAV. The aircraft has nearly a 4m wing span and a maximum gross weight of 70lb. The aircraft is propelled by a two cylinder, pusher engine. The aircraft carries, nearly four gallons of fuel, and has a flight time of 8 hours. The electronics are powered by an alternator attached to the engine, but the Bat IV also carries an uninterruptible power supply that can be plugged into shore power while it is on the ground.

Figure 1.7: MLB's BAT IV UAV.

The aircraft is divided into several compartments. There is a cavity in the nose of the aircraft that can carry a gimbal. The forward part of the fuselage houses all of the aircraft specific electronics, including the autopilot. the bottom length of the fuselage is removable and exposes the payload compartment.

Payload

The payload compartment of the Bat IV was designed in succession to the Sig Rascal, and was built to be even more reconfigurable. A metal frame hangs down from the main fuselage, which allows any number of components to be easily added and removed. At the front of the frame, there are ballast plates to stabilize the aircraft. Typically, the Bat IV carries two PC104 computers, an amplifier and an antenna, and a wireless router that networks all of the components, .

1.3.6 Berkeley Zagi

The Zagi is a six foot foam flying wing, which can be seen in figure 1.8. It was built by C3UV as a light weight inexpensive alternative to the Sig Rascal and Bat IV. It is propelled by a 300 watt electric, pusher motor, and has a flight time of about 30 minutes. The gross weight is about 8 pounds, which includes a 2 pound payload.

Figure 1.8: C3UV's six foot zagi UAV.

Payload

The Zagi payload is consumed mostly by a small media pc. In addition to the computer, it can also carry any light weight low power sensor, which would typically be a camera. The computer on board the aircraft has a built in frame grabber, and an Ethernet port, which allows the Zagi to carry both analogue and digital cameras.

1.4 Outline of Dissertation and Contributions

The work presented in each chapter build on contributions made in the previous chapter. The first contribution, is a novel kinodynamic unicycle model. The chapter introducing that model presents the theory behind modelling fixed wing aircraft as unicycles. It then goes on to show why augmenting the unicycle model, with a first order approximation of the aircraft's roll dynamics, is the next logical step to increasing the accuracy of the model's estimate of the a UAV's sensor footprint. Finally, it presents, experimental data that validates both models, and shows the improvement of the kinodynamic unicycle model over the traditional model.

The next chapter goes on to present a motion planner that uses the kinodynamic unicycle model to create feasible paths that keeps targets paths within a UAV's sensor's field of view. The chapter reviews a number of path planning approaches that haven been applied to UAVs, and presents the original rapidly exploring random tree algorithm in detail. Then it goes on to present a novel iterative Gaussian sampling strategy for an RRT algorithm that rapidly converges to a nearly optimal path.

The final theoretical chapter, presents to model predictive controller that uses both the kinodynamic model, and the iterative Gaussian Sampling RRT inorder to solve the nonminumim phase problem of tracking a target path with a UAV's sensor footprint. The chapter explains the theoretical challanges, and presents several controllers which have previously been applied to try and approximate a solution to the problem. The performance of each of those controllers is discussed and then compared to the novel NMPC controller.

Lastly, the contributions of this paper are validated by applying them to several real world tracking problems. The final chapter discusses the performance of the NMPC tracking controller based on the experimental data that was collected from tracking straight paths, curved paths, disjoint paths, points and trajectories.

Chapter 2

Fixed Wing UAV Dynamics

A great deal of research regarding small fixed wing UAVs has been built on the assumption that a kinematic unicycle model approximates the dynamics of small fixed wing UAVs. Despite the fact that the model is used so often, very little research has been published to explicitly validate that model with flight results[4]. This is particularly interesting in the situations when the kinematic unicycle model is used to plan paths for UAVs, which by definition result in dynamically infeasible paths. In order to generate feasible paths for a UAV, the UAV model must be based on the forces effecting the UAV, rather than the simplified kinematic constraints. This chapter presents a novel kinodynamic motion model for small fixed wing UAVs along with experimental flight data that validate both the kinematic and kinodynamic unicycle models.

In any control system, it is important to choose an appropriate model that describes the dynamics of plant being controlled. Since the state of a UAV's sensor footprint is determined by both its position and attitude, it is important to have a model of both. The kinematic unicycle equations do not model the roll dynamics of the UAV at all, therefore it is an inappropriate model for controlling a UAV's sensor footprint. Augmenting the unicycle model with first order roll dynamics creates a kinodynamic model of a UAV that can be used to control the sensor footprint. And, in addition, the kinodynamic model can also be used to generate feasible aircraft paths.

2.1 Fixed Wing Aircraft Dynamics

By convention, the body fixed coordinates of an aircraft are oriented such that the x-axis runs parallel to the fuselage, the y-axis runs parallel to the right wing, and the z-axis points down. As figure 2.1 shows, the attitude of the aircraft is parametrized by the angles (ψ, θ, ϕ) with respect the world coordinated frame: North, East, and Down.

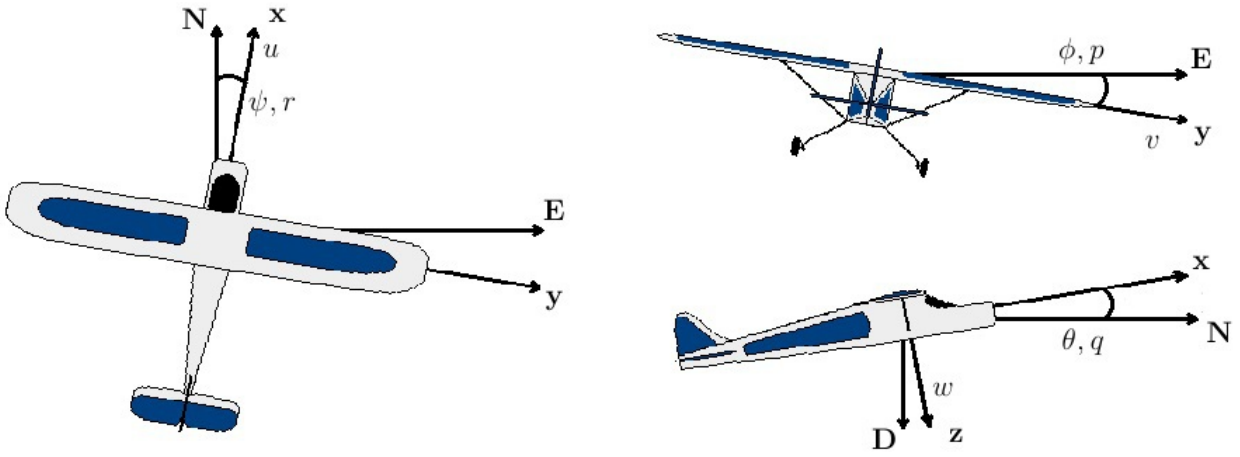

Figure 2.1: Conventional nomenclature for a fixed wing aircraft.

The velocity of the aircraft with respect to the body fixed coordinates is given by the vector (u, v, w) and relates to the world reference frame by the following equations.

$$\begin{bmatrix} \dot{x}_N \\ \dot{y}_E \\ \dot{z}_D \end{bmatrix} = T_\psi T_\theta T_\phi \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.1}$$

Where matrices T_ψ, T_θ, and T_ϕ are the rotation matrices of the aircraft angles ψ, θ, and ϕ). Similarly, the body fixed angular rates, p, q, and r relate to the derivatives of the angle T_ψ, T_θ, and T_ϕ be the following equations.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta\sin\phi & \sin\theta\cos\phi \\ 0 & \cos\theta\cos\phi & -\cos\theta\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.2}$$

Given these conventions, then the dynamics of a fixed wing aircraft can be represented as the following equations.

$$\begin{aligned} F_x &= m(\dot{u} + qw - rv) + mg\sin\theta + T \\ F_y &= m(\dot{v} + ru - pw) - mg\cos\theta\sin\phi \\ F_z &= m(\dot{w} + pv - qu) - mg\cos\theta\cos\phi \end{aligned} \tag{2.3}$$

$$\begin{aligned} M_x &= I_x\dot{(p)} + I_{xy}\dot{r} + (I_z - I_y)qr + I_{xz}qp \\ M_y &= I_y\dot{q} + (I_x - I_z)pr + I_{xz}(r^2 - p^2) \\ Mz &= I_z\dot{r} + I_{xz}\dot{p} + (I_y - I_x)qp - I_{xz}qr \end{aligned} \tag{2.4}$$

Where, m is the mass of the vehicle, T is the thrust of the engine, and F_x, F_y and F_z are aerodynamic forces on the aircraft.

2.2 Modelling an Fixed Wing Aircraft as a Unicycle

For most mid to high level controllers, it is prudent to reduce the complexity of aircraft's dynamics, and is some cases, remove the dynamics all together. For research topics that are only concerned with the position of the UAV, and only vaguely concerned about how the aircraft gets from one position to another, the kinematically derived unicycle equations are a good approximation of the UAV's dynamics. Those equations are as follows.

$$\begin{aligned} \dot{x} &= V\cos\psi \\ \dot{y} &= V\sin\psi \\ \dot{\psi} &= u \end{aligned} \tag{2.5}$$

The unicycle model assumes that the aircraft has nearly zero pitch and roll, $|\theta| << 1$ and $|\phi| << 1$. There for the horizontal velocity of the aircraft, V, which is normally described as,

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = V \begin{bmatrix} \cos\theta \\ \sin\theta\sin\phi \\ \sin\theta\cos\phi \end{bmatrix} \tag{2.6}$$

becomes simply $u = V$.

Furthermore, simplifying 2.1 leads to the following equation.

$$\begin{bmatrix} \dot{x}_N \\ \dot{y}_E \\ \dot{z}_D \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix} \tag{2.7}$$

Which, represents the first two equations in the unicycle model.

The final term of the unicycle model comes from the assumption that the dynamics of the yaw rate are fast enough, that for the desire fidelity, they can be modelled as instant, allowing the yaw rate to be set arbitrarily. In many situations, that is a valid assumption. Controlling the position of a UAV's sensor footprint, which is a function of the yaw rate dynamics, requires a higher order order model.

2.3 Augmenting the Unicycle Model to Include Roll Dynamics

Conventional aircraft turn by banking, which tilts the lift force, F_z, giving it a component in the radial direction. In addition, most aircraft try to reduce F_y. the lateral forces, to zero. A *coordinated turn* is one that involves zero lateral forces. In other words, for a coordinated turn, the radial component of the lift force is equal to the centripetal force causing the aircraft to turn. Under normal circumstances, a fixed wing aircraft will almost always make coordinated turns

Assuming that $F_y = 0$ and $\theta << 1$, then equation 2.3 becomes

$$mrV = mg\cos\theta\sin\phi \tag{2.8}$$

The yaw rate $\dot{\psi}$ can be resolved into the body axes by the following equations

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \dot{\psi} \begin{bmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{bmatrix} \tag{2.9}$$

By substituting r from equation 2.9 into equation 2.8 and simplifying,

$$mV\dot{\psi}\cos\theta\cos\phi = mg\cos\theta\sin\phi$$

$$\tan\phi = \frac{V\dot{\psi}}{g} \tag{2.10}$$

the equation becomes what is known as the *coordinated turn assumption*, which relates yaw rate to bank angle.

Equation 2.10 shows how banking the aircraft rotates the the force vector F_z, which causes the aircraft to turn at the rate $\dot{\psi}$. Likewise, achieving a desired turn-rate requires the aircraft to have a specific bank angle. The kinematic unicycle model assumes that $\dot{\psi}$ can be set arbitrarily, which in turn implies that the bank angle of the aircraft can be set arbitrarily. Obviously, that is not true. The desired turn-rate of the aircraft can only be achieved according to the settling time of the roll dynamics. Taking the derivative of equation 2.10 shows that the dynamics of the turn-rate are a function of the roll rate.

$$\ddot{\psi} = \frac{g}{V\cos^2\phi}\dot{\phi} \tag{2.11}$$

Approximating the roll dynamics of the UAV by a first order system with a time constant τ results in the following equation.

$$\dot{\phi} = \frac{1}{\tau}(\phi_d - \phi) \tag{2.12}$$

Substituting equation 2.12 into equation 2.11 and assuming that $\phi << 1$ results in the following equation.

$$\ddot{\psi} = \frac{g}{V\tau}(\phi_d - \phi) \tag{2.13}$$

Substituting equation eq:coordinatedturn into equation 2.13 gives an equation the second order turnrate dynamics of the UAV.

$$\ddot{\psi} = \frac{g}{V\tau}\left(\tan^{-1}\left(\frac{Vu}{g}\right) - \tan^{-1}\left(\frac{V\dot{\psi}}{g}\right)\right) \tag{2.14}$$

Recognizing that for small ν, $\tan(\nu) \approx \nu$, then turnrate dynamics symplify to the following.

$$\ddot{\psi} = \frac{1}{\tau}(u - \dot{\psi}) \tag{2.15}$$

Equation 2.15 describes the turn-rate dynamics of a UAV according to the forces that are acting on the UAV. Incorporating those dynamics into the unicycle model leads to the following augmented, or kinodynamic unicycle model.

$$\begin{aligned} \dot{x} &= V\cos\psi \\ \dot{y} &= V\sin\psi \\ \ddot{\psi} &= \frac{1}{\tau}(u - \dot{\psi}) \end{aligned} \tag{2.16}$$

Qualitatively, the kinodynamic unicycle equations approximate the rotational inertia of the aircraft, which acts like a first order filter with respect to the commanded bank-angle, or turn-rate. Figure 2.2 exemplifies the effect that the roll dynamics have on the UAV's sensor footprint.

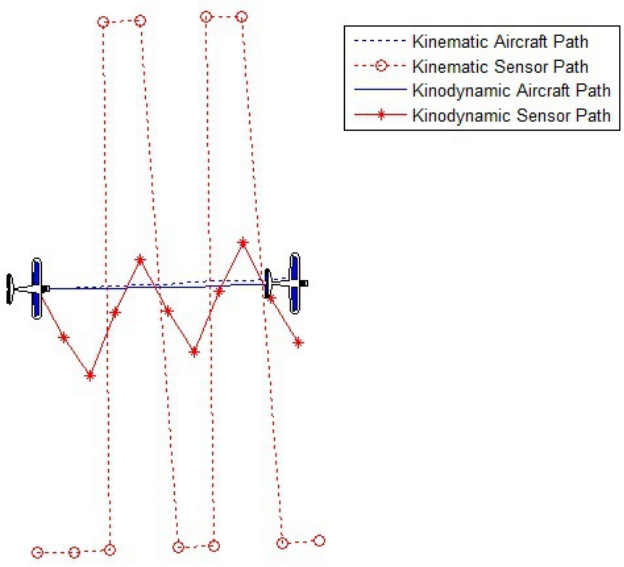

Figure 2.2: A comparison of the sensor paths computed by the kinematic and kinodynamic aircraft models.

Figure 2.2 shows the simulated aircraft and sensor path for a control sequence of alternating left and right turns using both the kinematic and kinodynamic unicycle models.

2.4 Modelling Wind in the Unicycle Context

The unicycle and augmented unicycle equations are based on the aerodynamic forces that effect the aircraft. For instance, the velocity of the aircraft, V is not the velocity of the aircraft with respect to the ground, it is the *true airspeed*, or rather, the velocity of the aircraft with respect to the *air-frame*. A full discussion of UAV dynamics with respect the airframe can be found here [7].

The air-frame moves with respect to the ground-frame according to the mean wind velocity. Consider a stationary point in the wind frame, $\mathbf{x}_w = 0$. That points position in the ground frame will move with a velocity equal to the mean wind. Therefore that same point in the ground frame is given by the following equation.

$$\mathbf{x}_g = \mathbf{x}_0 + \mathbf{W}t \tag{2.17}$$

Assuming $\mathbf{x}_0 = 0$, it follows that

$$\mathbf{x}_g - \mathbf{W}t = \mathbf{x}_w \tag{2.18}$$

and that,

$$\dot{\mathbf{x}}_g - \mathbf{W} = \dot{\mathbf{x}}_w \tag{2.19}$$

In other words, the transformation between the wind-frame and the ground-frame is given by the following equation.

$$\dot{\mathbf{x}}_g = \dot{\mathbf{x}}_w + \mathbf{W} \tag{2.20}$$

Though the dynamics of a fixed wing UAV are easier to derive in the wind-frame, it is usually more useful to know the position of the UAV with respect to the ground frame. Applying transformation 2.20 to the unicycle model gives the following model a UAV with respect to the ground-frame.

$$\begin{aligned} \dot{x} &= V\cos\psi + W_x \\ \dot{y} &= V\sin\psi + W_y \\ \dot{\psi} &= u \end{aligned}$$

And similarly, the kinodynamic unicycle model with respect to the ground frame is given by the following.

$$\begin{aligned} \dot{x} &= V\cos\psi \\ \dot{y} &= V\sin\psi \\ \ddot{\psi} &= \frac{1}{\tau}(u - \dot{\psi}) \end{aligned}$$

2.5 Model Validation Against Flight Results

The following section presents data from experimental flight connected at Camp Roberts, CA, using the UC Berkeley Sig Rascal UAV[23]. The data is composed of several flights taken over the course of a few weeks. The selection of flight paths attempts to explore several different flight regimes, which includes straight lines, orbits, figure eights, and sinusoids.

2.5.1 Evaluating the Coordinated Turn Assumption

The assumption that the UAV makes coordinated turns is crucial to the controllers ability to control the UAV's sensor footprint by commanding different turn-rates. This assumption is easily examined by applying the coordinated turn assuption from equation 2.10 to the

flight data. The roll error in figure 2.3 is given by the following equation.

$$e_{roll} = \phi - \tan^{-1}\left(\frac{V\dot{\psi}}{g}\right) \tag{2.21}$$

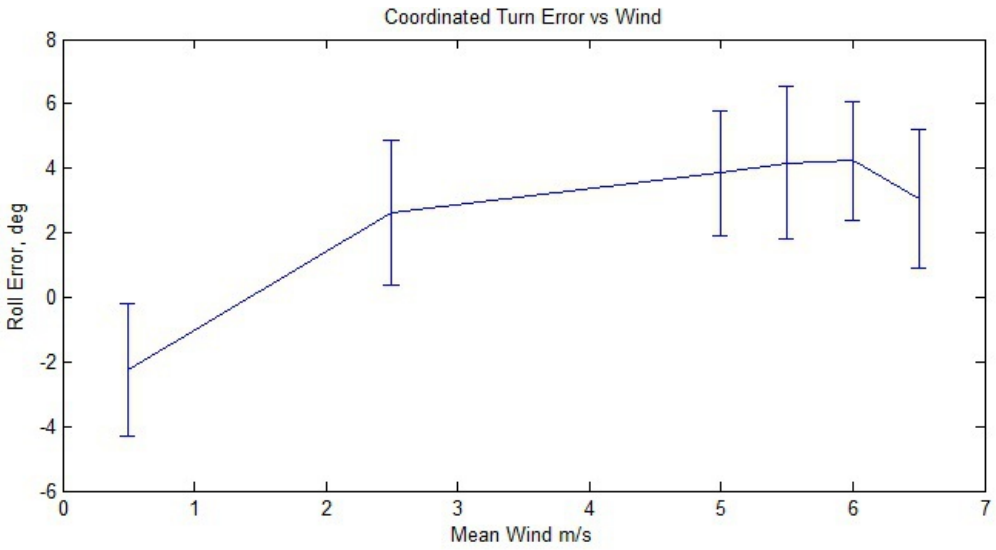

Figure 2.3: Flight data comparing the coordinated turn assumption versus wind.

The flight data shows that in low wind conditions it's likely that the expected bank angle will only vary a from degrees from aircraft's actual bank angle. However, as the mean speed of the wind increases, the deviation could be a significant source of error.

2.5.2 Evaluating the Zero Pitch assumption

Another simplification to the aircraft model, is that the pitch of the aircraft remains nominally zero. Figure 2.4 shows an aggregation of flight data comparing the pitch of the aircraft to the mean wind speed.

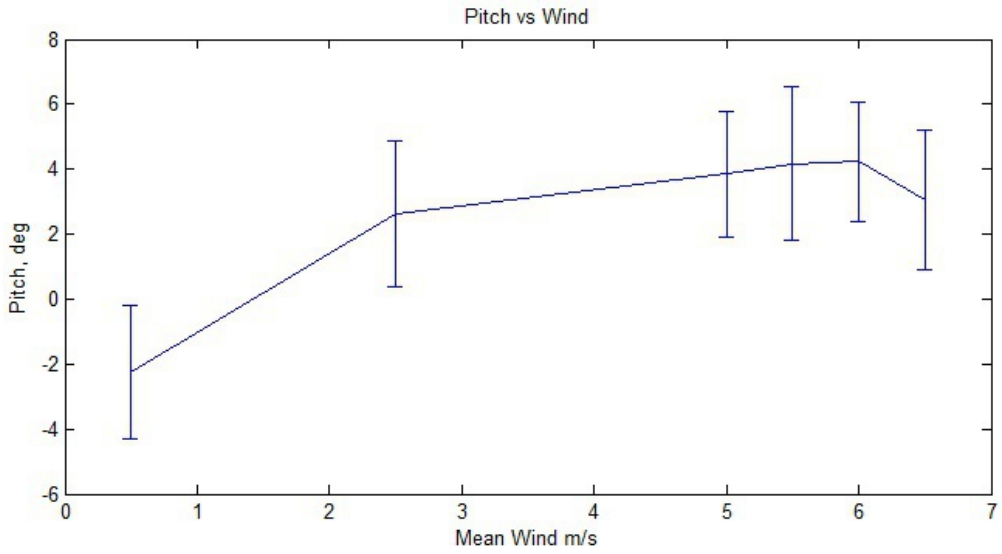

Figure 2.4: Flight data comparing the zero pitch assumption versus wind.

Again, the flight data shows that the pitch remains nominally zero in low wind conditions, but deviates as the wind increases.

2.5.3 Evaluating the Unicycle Models

The results show that the augmented, kinodynamic unicycle model, reduces model error by about 25% under nominal wind conditions. The model error is measured by the rate of divergence of the predicted aircraft and sensor paths from the actual aircraft and sensor paths. The rate of divergence was calculated by sampling the flight data at many points along the path and then integrating over both unicycle models using the control inputs that were recorded from the flight. Figure 2.5 shows an example of the sample paths generated by integrating over the kinematic unicycle model.

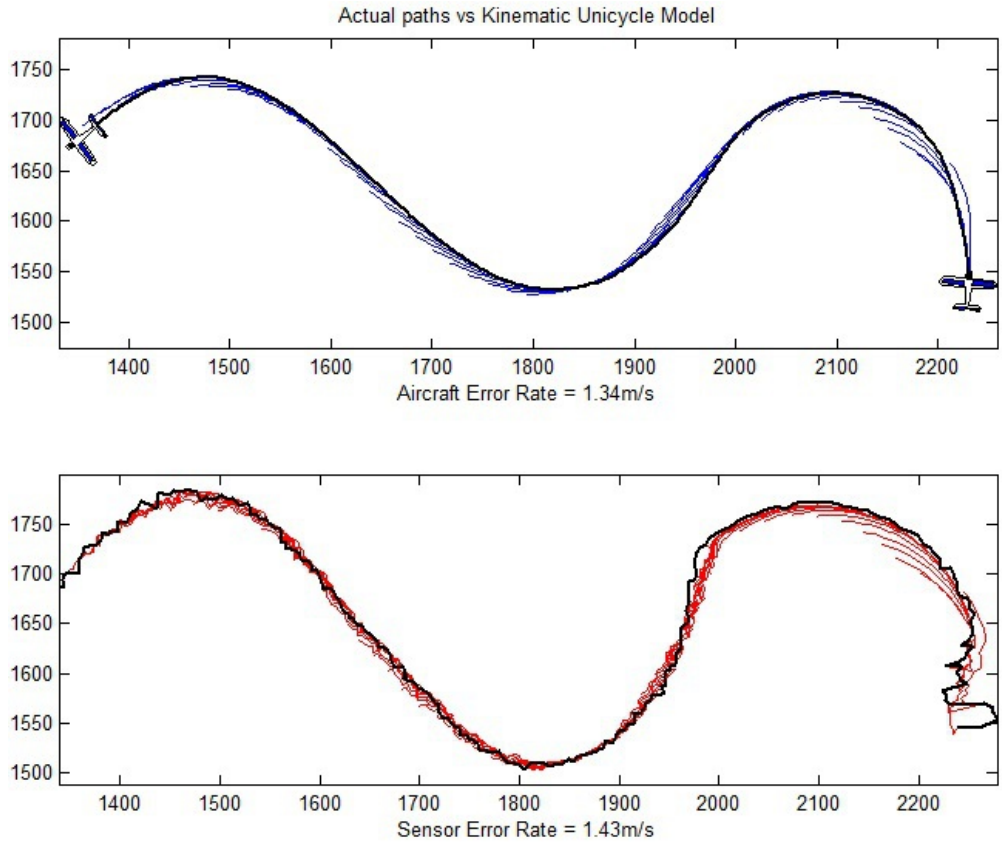

Figure 2.5: Modelling error of the aircraft (top) and sensor footprint (bottom) produced by the kinematic unicycle model.

The top plot in figure 2.5 show the actual path of the aircraft along with a series of simulated aircraft paths, which were generated by the unicycle model. The bottom plot shows a similar set of simulated sensor paths, where the sensor path is represented by the center of the UAVs sensor footprint. The divergence rate is calculated by dividing the cross-track error by the simulation time. Figure 2.6 shows the same plot, but for the kinodynamic unicycle model.

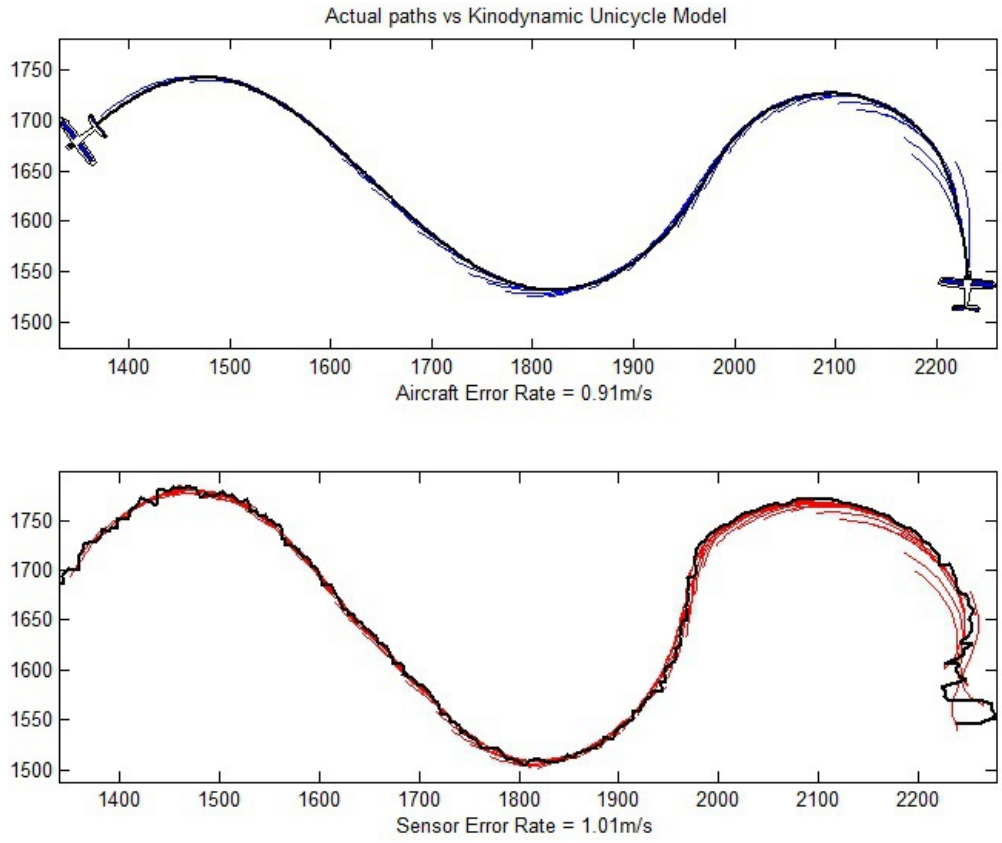

Figure 2.6: Modelling error of the aircraft (top) and sensor footprint (bottom) produced by the kinodynamic unicycle model.

In this example, compared to the kinematic model, kinodynamic model reduces the rate of divergence of the sensor path by nearly 30%, from 1.43m/s to 1.01m/s. In either case, both models seem to have high enough fidelity to be used as a model for a local controller. The divergence rate can me normalized into a dimensionless parameter by dividing by the velocity of the vehicle, which in both flights was about 22m/s. In which case, the error associated with the kinodynamic controller would be about 5%. This error metric represents the lateral error with respect to the longitudinal distance the UAV has travel. Though this data seems to validate both unicycle models, it is important to recognize that this particular data is from a flight with very little wind.

As the mean velocity of the wind increases, the improvement of the kinodynamic model over the kinematic model decreases. Presumably, as the wind speed increases, the primary source of error becomes the wind disturbance, which is not well modelled in either unicycle models. Figure 2.7 shows how the improvement of kinodynamic model decreases as the wind speed increases.

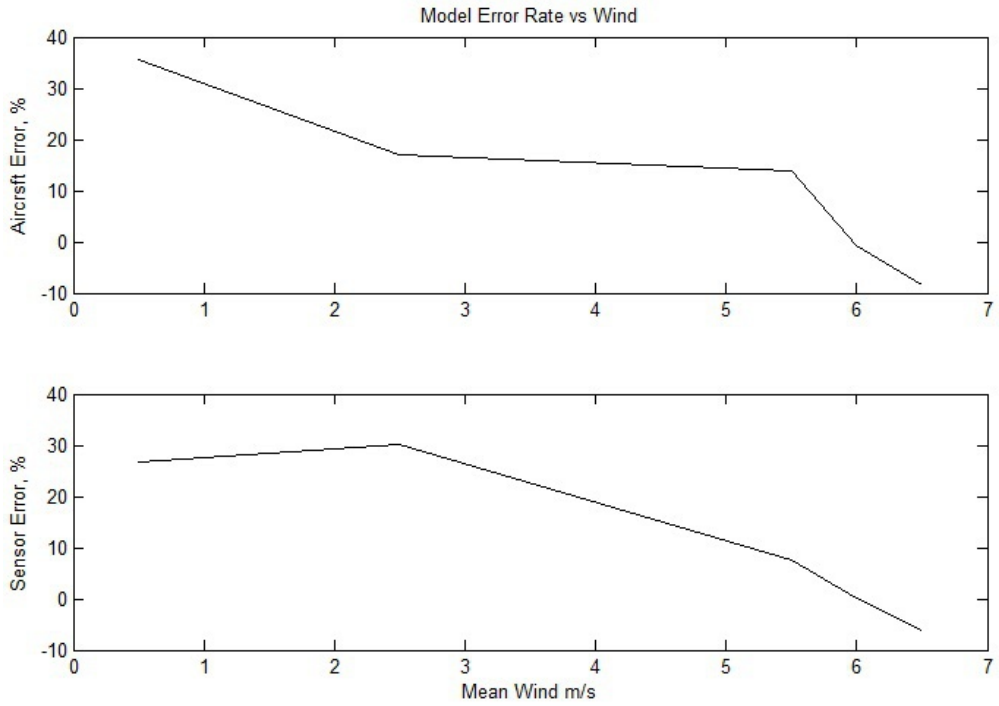

Figure 2.7: The reduction in model error by the kinodynamic unicycle deceases as the mean velocity of the wind increases.

The kinodynamic model performs well in low wind conditions, but the performance drops off shortly after about 5m/s, which, for reference, is about 25% of UAV's true airspeed. Figure 2.9 shows the divergence of the kinodynamic model with respect to the mean wind speed.

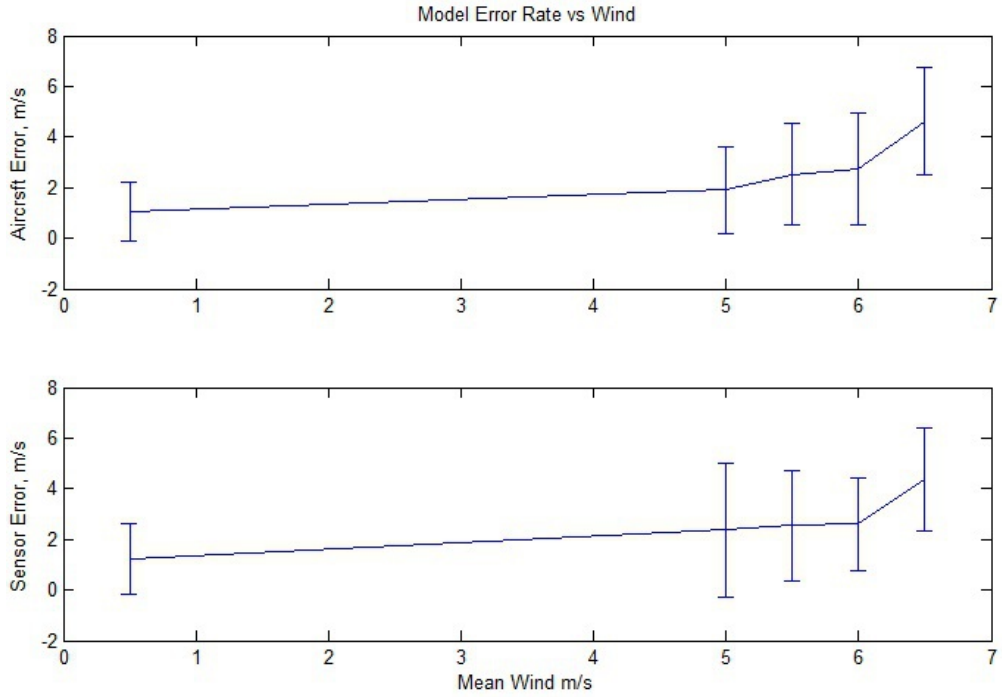

Figure 2.8: Divergence rate of the kinodynamic unicycle model with respect to the mean wind speed.

Another important factor to consider, is that the unicycle models do not have access to the actual wind's velocity, they only have access to an estimate of wind's mean velocity, which is generated by the Piccolo autopilot. The Piccolo's estimate of the wind's mean velocity changes through out each flight, and upon inspection is correlated to the attitude and heading of the aircraft. Assuming that the actual mean velocity of the wind remains constant during each flight, the average of the piccolo's estimates may be closer to the actual mean velocity. Figure 2.9 plots the rate of divergence of the kinodynamic model versus the error of the wind estimates with respect to the average mean estimate.

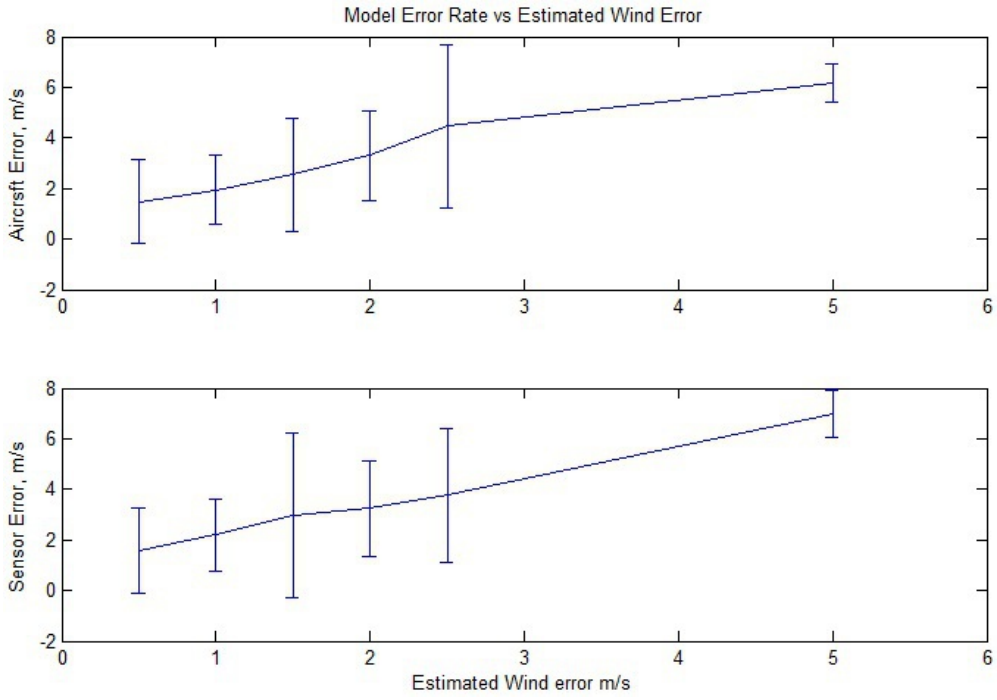

Figure 2.9: Divergence rate of the kinodynamic unicycle model with respect to the error of the wind estimate.

Figure 2.9 shows that the kinodynamic unicycle model, and similarly the kinematic unicycle model, are particularly sensitive to poor wind estimates. But, assuming that the wind is less than 5m/s and reasonably well estimated, the data also shows that both unicycle models preformed reasonable well, though the kinodynamic unicycle model performed a little bit better.

2.6 Conclusions

This chapter presented a novel kinodynamic unicycle model, and used actual flight data to evaluated its accuracy. In addition, it showed that when there is minimal wind, augmenting the unicycle model with roll dynamics improves the accuracy of the predicted sensor footprint by about 25%. The experimental data also shows that in minimal wind conditions, both unicycle models adequately reproduce the flight trajectory of an actual fixed wing aircraft.

In addition, this chapter also evaluated several assumption, which were made in order to reduce the complexity of the fixed wing UAV dynamics. Both the coordinated turn assumption, and zero pitch assumption, turned out to be accurate when the mean wind speed was small, and both assumptions became less accurate as the wind increased. In fact,

by every metric, the performance of both unicycle models degraded rapidly as the mean wind speed increased, which motivates the need for a better model of the effect wind disturbances.

Future work may explore how the attitude of the aircraft correlates to disturbances caused by the wind. The UAV's profile with respect to the wind varies greatly depending on the UAV's heading and roll angle. A greater profile would result in greater drag on the aircraft, which means that the wind would have a greater effect on the aircraft. Neither unicycle model considers the aircraft's drag profile when accounting for the wind. Improving the wind model associated with these unicycle models is probably the best way to improve their overall performance.

Chapter 3

On-board Motion Planning for Small UAVs

This chapter presents novel iterateive Guassian sampling stratagy for building a rapidly exploring random tree that quicky converges to a near optimal path. Moreover, the sampling strategy eliminates the need for a nearest neighbour method, which is usually the most computationally expensive step of the RRT algorithm.

In addition to presenting the IGSRRT path planner, this chapter also describes several other path planners which are commonly applied to UAV motion planning problems. Each path planner is briefly described, and discussed with respect to its advantages and disadvantages as a UAV motion planner.

3.1 Introduction to the Motion Planning Problem

The goal of a motion planner is to create a series of set points, or functions that define a path for the vehicle or robot to follow[21]. If a path is also parametrized by time, then it is a trajectory. For example, a roadway could be considered a path, while following a vehicle along the runway would be a trajectory, since the location of the vehicle is defined not only by the road, but by it's position along the road, which is parametrized by time. Figure 3.1 shows a typical vehicle motion planning problem. In this case, the small green circle represents the starting location for the vehicle, and the larger orange circle represents the desired end location, or goal configuration. The two red squares are obstacles in the environment.

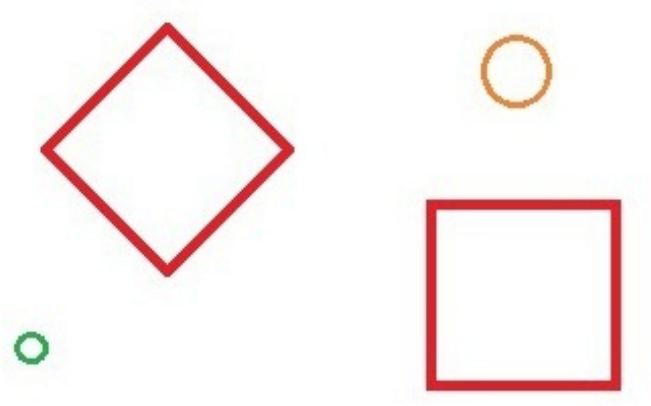

Figure 3.1: World space for a typical vehicle motion planning problem.

In motion planning problems, world space refers to the physical world that makes up the environment for the motion planning problem. The world space does not contain any information about the vehicle or robot that needs to move with in it. The configuration space (Cspace) on the other hand defines all of the possible positions and orientations a vehicle could occupy in the world space. Consequentially, the Cspace usually has a larger dimensionality than the world space. The number of parameters that are necessary to fully define an objects position, orientations, and configuration is know as the degrees of freedom (DOF) of the object. The dimensionality of the Cspace is equal to the vehicles DOF. In the case of a point mass, the Cspace and the world space equivalent. Figure 3.2 shows the configuration space for a dynamically unconstrained disk the size of the green starting configuration. The dark grey area is not part of the vehicle's Cspace, because it represents the configurations that would cause a collision with the obstacles. The free space is every part of the world space that is not occupied by obstacles.

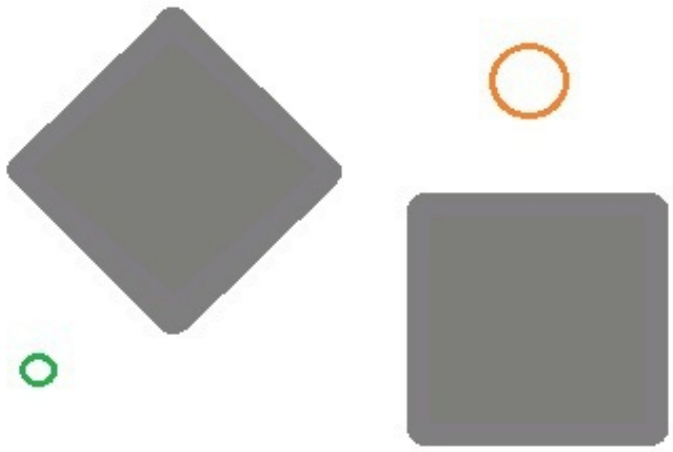

Figure 3.2: Configuration space for the previous environment, if the vehicle was a disk the size of the green starting configuration, with no dynamic constraints.

A path that connects two points in the Cspace without colliding with any obstacles is known as a feasible path. The complexity of finding a feasible path is a function of the number of parameters necessary to define the obstacles in the environment, and the number of DOF and the dynamic constraints of the vehicle. If the time required to solve the problem scales as a polynomial with respect to those parameters, then it is said to be P-complex, which are generally efficiently solvable. If the time required to verify a solution to the problem scales as a polynomial with respect to those parameters then it is said to be NP-complex. Since NP problems are only efficiently verified, solving an NP problem may require verifying many potential solutions within a decision tree, which may take a very long time to compute. If a problem can be shown to be as hard to solve as any NP problem, then it is said to be NP-hard.

Motion planning algorithms are classified by their completeness. A *complete* algorithm is one that always returns a solution if it exists, and terminates if no solution exists. To be *complete*, it must satisfy both of those conditions such that it either finds a solution, or proves that none exists. A path planning algorithm that only satisfies the first condition, meaning it always returns a solution when one exists, but does not terminate when there is no solution, is known as *exact*. Complete algorithms offer nice guarantees, but are often slow, and even intractable for many problems. Heuristic algorithms, on the other hand, can quickly return solutions that are close to the complete solution, but they are not guaranteed to do so. Despite their lack of guarantees, heuristic algorithms are often used in robotics in order to meet the time constraints imposed by dynamic constraints.

There are two weaker forms of completeness: *resolution completeness* and *probabilistic completeness*. Both of these types of completeness deal with the limitations of numerical

algorithms. *Resolution completeness* states that because of discretization of the Cspace, the algorithm is not complete, but that algorithm asymptomatic approaches completeness as the resolution of the discretization increases. Likewise, for algorithms that rely on random sampling, *probabilistic completeness* states that as the number of samples increase, the chance of finding a solution asymptomatic approaches one.

3.1.1 Typical Motion Planning Considerations

Motion planning comes in many forms, but in each case, the purpose is the same: to provide an n-dimensional list of reference points or functions that can be referenced by a lower level tracking controller. The performance of the combined motion planner and tracking controller is highly dependent on the degree of coordination between them. Choosing feasible set points reduces the transient effects that result as the tracking controller changes from one reference to another. For example, a box pattern, like the green dashed path shown in figure 3.3, is not a feasible path for an air plane because a plane cannot instantaneously change its direction of travel at each corner. The outer blue path and inner red path, show two prototypical transient behaviours.

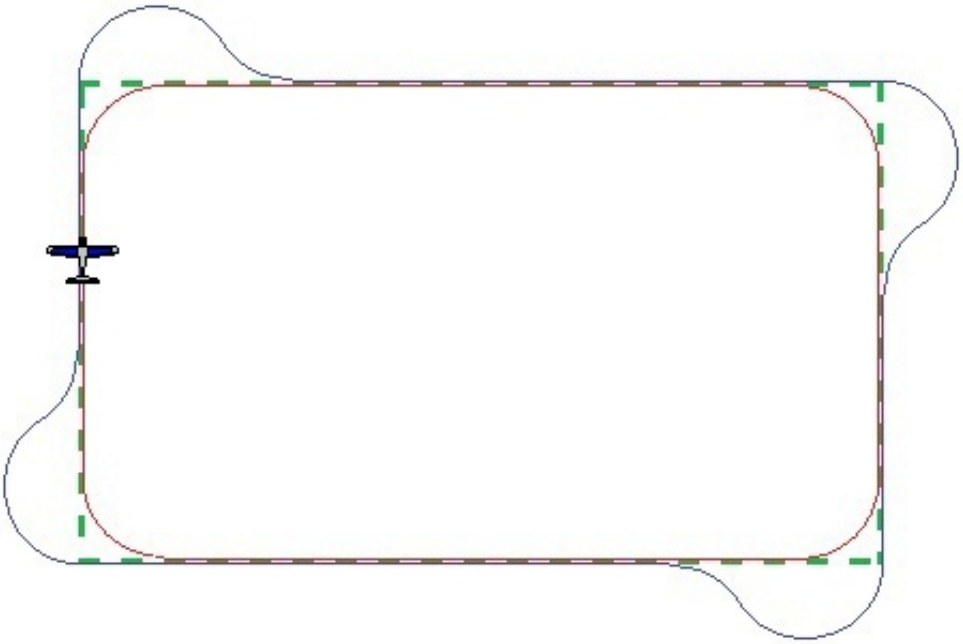

Figure 3.3: Prototypical examples of a fixed wing aircraft tracking a box patter.

The outer blue path is the result of the aircraft tracking each line segment of the box pattern to its full extent before switching to the next line segment. The inner red path is

an example of a controller that begins tracking the next line segment before reaching the end of the current line segment. In all cases, the dynamics of a fixed wing aircraft, prevent the tracking controller from having zero cross-track error along the full length of each line segment.

In some cases, the transient behaviour, like that shown in figure 3.3, can be inconsequential. For instance, if UAV sampling the air quality around a power plant followed the outer blue path instead of the green box, it would have very little effect on the data it collected. On the other hand, if a UAV was flying low around a city block and the green box represented the center line of city streets, flying either the blue or red path could cause the UAV to crash into a building.

Less sophisticated path planning algorithms may result in poorer tracking performance, but they require less computing power, which means the UAV could carry a smaller payload, and have a longer flight time.

3.1.2 Motion planning for Small UAVs

Motion planning algorithms are used in a variety of robotic applications and each case has its own special considerations. Motion planning is used to plan paths for robotic end effectors, which often operate in three dimensions, and may have several degrees of freedom for each articulated joint. Likewise, ground vehicles are usually constrained to travel on a two dimensional surface, which may be strictly passable or impassable at some points, or have a gradient defining how passable the surface is at every point [11]. On the other hand, other motion planning problems for ground vehicles are specifically about finding feasible paths over extremely rough terrain [24].

It follows then that path planning for fixed wing UAVs would present its own unique challanges. The most important constraint for any fixed wing aircraft is that it must maintain a minimum airspeed. The minimum velocity constrain on fixed wing aircraft is a non-holonomic constraint, which greatly increases the complexity of the motion planning problem. In addition, it also imposes a constraint on the time it takes to solve the motion planning problem. Since a fixed wing aircraft cannot stop, it must be able to find a feasible solution quickly, before it runs into any of the constraints of the configuration space.

For small fixed wing UAVs, despite the fact that they are free to fly in three dimensions, it is often desirable to find feasible paths within a single plane at constant altitude. Although small UAVs are capable of changing their altitude, doing so is an inefficient use of fuel, and cannot usually done with very high bandwidth.

For a UAV to maintain its airspeed and increase its altitude, it must increase its overall energy, which is done by increasing the engine's rpms. Engines and propellers for small UAVs are usually selected to maximise flight time, which means that the two are selected together such that they operate most efficiently at the aircraft aerodynamic cruising speed. Increasing the rpms of the engine beyond its cruising means that the engine will not be using its fuel as efficiently as possible, which means that the UAV will have a shorter flight

duration. Although it is not necessary for a UAV to maintain a constant airspeed, the same argument holds for any case when the desired airspeed is any value that does not maintain the total energy of the aircraft.

For a UAV to maintain its airspeed and decrease its altitude, it must decrease its energy, but since small UAVs do not have controllable air brakes, it must rely on passive forces, like the aerodynamic drag on the airframe, to reduce its energy. Unfortunately, in every other circumstance (ie. fuel efficiency, maximum airspeed, maneuverability) it is advantageous to reduce, rather than increase, the drag forces on the aircraft. Similarly, the engine, which is typically chosen for efficiency rather than power, can only add energy to the system so fast, meaning that the aircraft can only gain altitude so quickly. Both of these saturation points inherently limit the bandwidth of any altitude controller for small UAVs.

Efficiency and saturated energy controls limit small UAVs to flying at a nearly constant altitude. Likewise, the same arguments can be made for maintaining a constant speed. The arguments above are about controlling the total energy of the aircraft, whether it be potential energy or kinetic energy, therefore it is also advantageous for small UAVs to maintain a constant cruising speed.

3.2 Geometric Path Planning

Geometric path planning algorithms differentiate themselves from other motion planning algorithms in that they do not account for the dynamic constraints of the vehicle. As it was stated before, ignoring the vehicle's dynamics will cause unwanted transient behaviour. If the transient behaviour is unimportant, then a model of the vehicle's environment, and the parameters of the motion planning problem, can can be reduced to the set of parameters that describes the physical geometry of the space surrounding the vehicle.

3.2.1 Skeleton Path Planning

Skeletal path planners reduce the configuration space of the vehicle to a network of one dimensional paths, which reduces the motion planning problem to a simple graph search problem. The skeletal motion planners are complete if they include every topologically feasible path in the Cspace.

Visibility Graphs

Visibility graphs connect the distinct features of every obstacle to the distinct features every other obstacle in the Cspace, and to the start and goal configurations. Connections that would cause collisions are omitted. Figure 3.4 shows the visibility graph for a point mass where the obstacles are represented as polytopes, and the vertices of each polytopes are used as the features in the visibility graph.

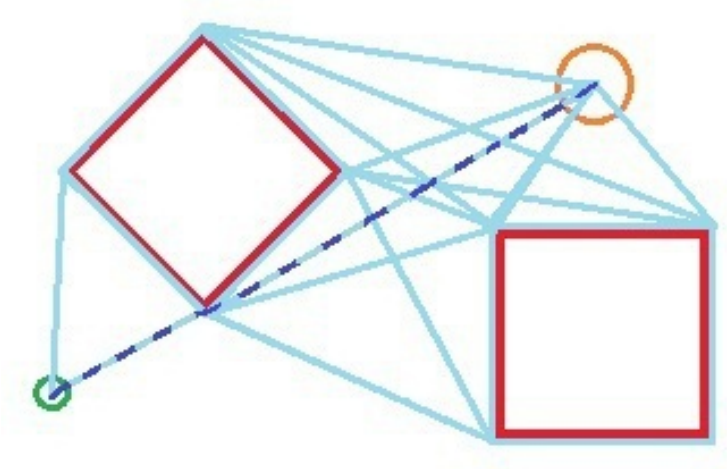

Figure 3.4: Visibility graph for a point mass. The dark, dashed line shows the shortest feasible path.

Once the visibility graph has been generated, each edge of the graph can be given a weight, and any number of graph search algorithms can be used to determine the best feasible path through the Cspace. Assuming the features are chosen well, visibility graphs can be used to find the shortest feasible euclidean path between the initial state and the goal state.

Visibility graphs will find the shortest path between two configurations, but it does that by skirting the edge of any obstacle that impedes the direct path between the starting configuration and the goal configuration. Because the path the visibility path generates does not account for any dynamic constraints or the potential of any cross track error that may exist while tracing the path, actually track the path generated by a visibility graph is likely to cause a collision, but it does serve as a lower bound to any path planning problem. One way to directly use the path generated by the visibility graph is to artificially increase the size of the obstacles in the configuration space such that the resulting path leave a buffer around each obstacle.

UAVs may use visibility graphs when the goal of the UAV is to get from one point to another as quickly, or most efficiently as possible. If the obstacles, which could be buildings or no fly zones, are sparsely populated, then a large buffer could be added around each obstacle, and the UAVs would be able to safely track the paths generated by the visibility graph. For receding horizon controllers, the visibility graph could be used to generate an estimate of the cost-to-go from any vertex on the graph.

Voronoi Graphs

Voronoi graphs are another form of skeletal decomposition, and like visibility graphs, they map features in the environment to a network of one dimensional paths. Unlike visibility

graphs, Voronoi graphs do not attempt to find the shortest euclidean distance to the goal, instead they create a network of paths that maximizes the distance from every obstacle. Figure 3.5 shows a Voronoi graph of a constrained environment.

As with the visibility graphs, each edge of the Voronoi graph can be assigned a weight, and any graph search algorithm can be used to find the best path. The bold dashed line in 3.5 shows the shortest path from the starting configuration to the goal configuration. In this case, the vehicle was constrained to move to one of the vertices of it's Voronoi cell.

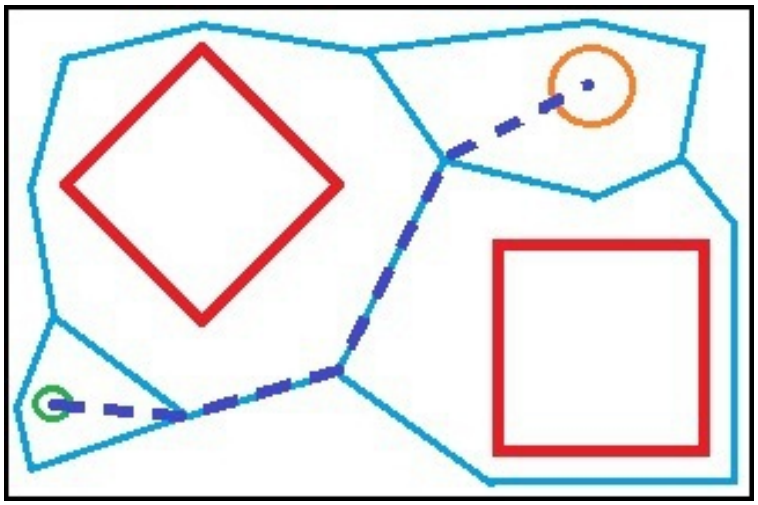

Figure 3.5: Generalized Voronoi between two obstacles in a constrained space.

There are several methods for building Voronoi graphs. If the obstacles in the environment can be represented as points, then the Voronoi graph is simple a Voronoi diagram. Voronoi diagrams can be computed by finding the hyperplane equidistant from two points that is orthogonal to the vector connecting the two points.

UAVs may use Voronoi graphs to plan paths around adversarial agents, such as surface-to-air missiles sites or radar towers. Voronoi graphs are also useful around closely spaced obstacles, for instance city buildings, where it is advantageous to keep as large a buffer as possible between the vehicle and any possible collision.

3.2.2 Potential Field

Potential field path planning algorithms provide fast way of computing an optimized path from a starting configuration to a goal configuration, but they are not guaranteed to find feasible solutions even when they exist. Potential fields are created by artificially creating a repulsive force between the vehicle and obstacles and and attractive force towards the goal. Potential fields are often explained as a statically charged particle moving through a field

of similar charged particles towards a single particle of the opposite charge. The degree to which each obstacle is 'charge' can be weighted in order to plan paths the preferentially avoid some obstacles over others. An example of a potential field is shown in figure 3.6. In this example, for simplicity, each obstacle is represented as a point, but potential fields can easily represent any shape. Paths through the potential field can be found by apply any gradient descent algorithm.

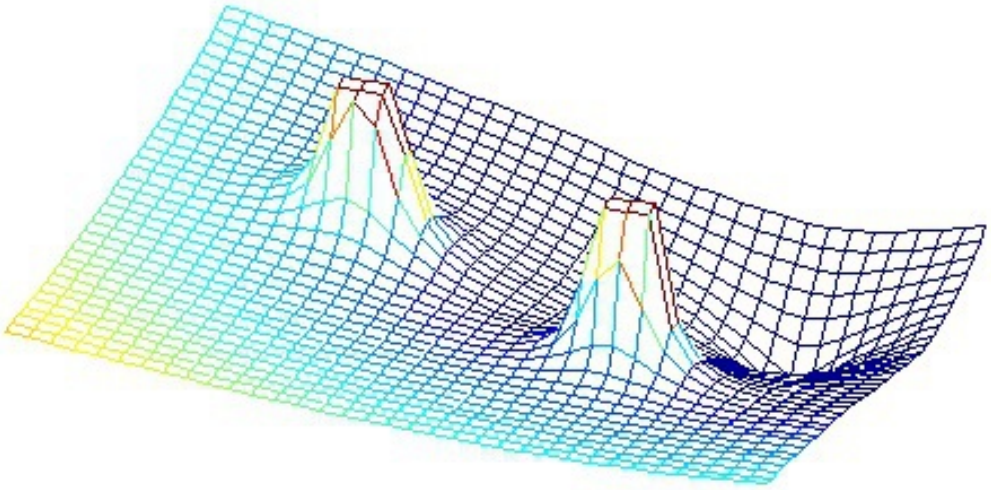

Figure 3.6: A potential field for two point obstacles and a goal state. The potential between each feature is inversely proportional to the distance from the vehicle.

Besides being very fast to compute, potential fields can be used to find paths around obstacles, or adversarial agents, without having to apply hard constraints. Unfortunately, in many situations potential graphs will have local minimums that prevent a simple gradient decent path planner from reaching the goal configuration. An example of a local minimum is shown in figure 3.7.

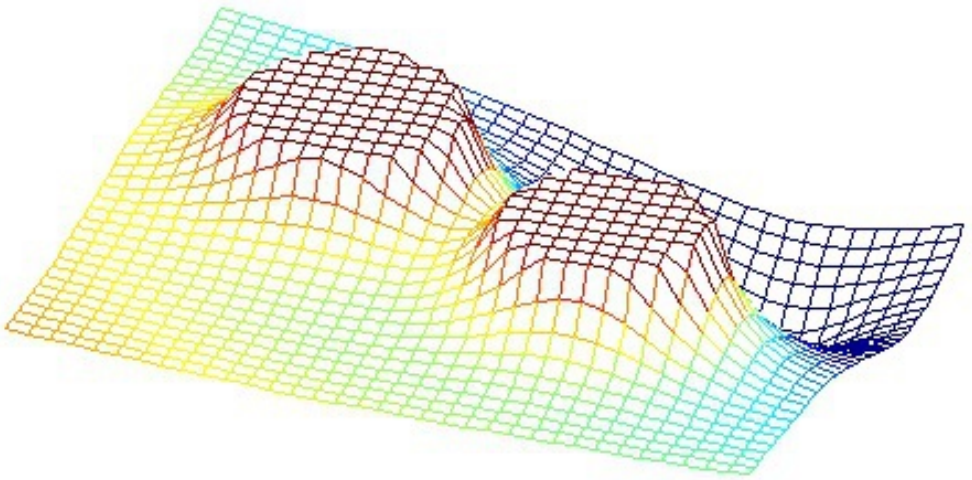

Figure 3.7: A potential field with a local minimum created between to obstacles. A simple gradient descent path planner would fail to find a feasible solution

Small UAVs may use potential field path planners because they require very little computation and can be solved very quickly. Simple range/bearing sensors can be used to calculate the resultant 'force' that that obstacle imposes on the vehicle. Potential field path planners may also be used as a local path planner between a series of way points that were chosen by a higher level path planner. Such heretical path planning schemes avoid the local minimum problems that arise from using the potential field method alone.

3.2.3 Cell Decomposition

In some cases it is useful to partition the environment into discrete cells. Doing so is called Cell Decomposition. Figure 3.8 shows how the example world is decomposed into cells. To find a feasible path, series of connected cells must be found from the starting cell to the goal cell. The path between the two points is the represented as a series of discrete configurations such that each configuration represents one cell.

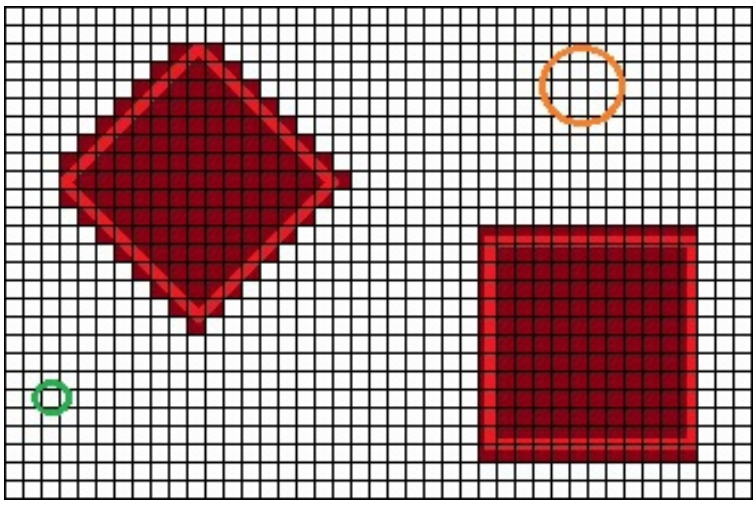

Figure 3.8: Cell decomposition of the configuration space.

If any part of the cell is inhabited by an obstacle, then the entire cell must be omitted from the Cspace of the vehicle. Finding a feasible path from the start to the goal can be done many ways, some of which will be discuses later, but an exhaustive search of the grid will always result in a feasible path if it exist, or otherwise prove that no path exist, but only if the resolution of the grid is fine enough. In other words, cell decomposition has *resolution completeness*. Cell decomposition is an important step in many numerical path planners.

3.3 Optimal Path Planning

When the goal of the path planning algorithm is not just to find a feasible path from start to goal configuration, but to find, in some sense, the best path, then is is necessary to run some type of optimization algorithm. Except for relatively simple Cspaces complete optimal path planning algorithms are very costly to solve. For complex environments, if solutions must be found quickly, then it is often necessary to use heuristic algorithms that approximates the optimal solution. In any case, the goal of an optimal path planner is to minimize some cost, or maximize some reward, that accumulates along the path. In the case of a UAV, it might be to minimize the chance of detection by advisories, or to maximize the chance of detecting targets[51].

3.3.1 A* Search

A* search is an optimal tree search algorithm with resolution completeness. The first step of an A* path planning algorithm, is to apply cell decomposition to the Cspace. Then

using an admissible heuristic, the algorithm estimates the *cost-to-go* from each cell, meaning it estimates the cost to reach the goal configuration from the current configuration. The cost-to-go from cell n is $h(n)$. Any heuristic that does not *over estimate* the actual cost-to-go is admissible. In other words, $h(n)$ is a lower bound on the cost of reaching the goal state from cell n. As the tree search progresses it calculates the actual cost, $g(n)$ of reaching cell n. Therefore, the estimated cost of any branch, or path, is given by equation 3.1.

$$f(n) = g(n) + h(n) \tag{3.1}$$

The tree search continues by expanding from which ever cell has the lowest estimated path cost, $f(n)$ until the goal is reached, and no other cell has a lower estimated cost. The speed and resolution completeness of A* path planning makes it a good choice for rough optimal path planning for small UAVs. Unfortunatly, since the paths generated by A* search are not dynamically constrained, the cell size must be large enough that the UAV can track from one cell to another despite its dynamic constraints.

3.3.2 Random Sampling

Many path planning problems have large enough dimensionality and complexity that it would be computationally impossible to search though a cell decomposition that has a high enough resolution to be complete. So, rather than systematically partitioning the Cspace in to regularly spaced nodes, nodes are created by randomly selecting configurations from the Cspace. Then as before, a tree search algorithm finds the path that is closest to the optimal path.

Probabilistic Roadmaps

A probabilistic roadmap is created by randomly sampling the Cspace n times, in order to create n nodes. Those nodes are then each connected to there nearest feasible neighbour, which results in one or more trees. More than one tree results when two trees cannot find a feasible path between any of their nodes. In that case, more samples of the Cspace can be taken in the space between the two trees in an attempt to find a feasible path between them. An example of a probabilistic roadmap is shown in figure 3.9.

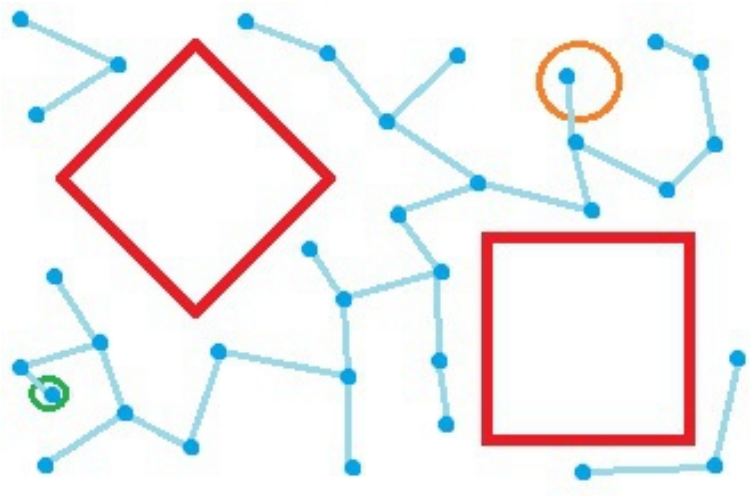

Figure 3.9: Cell decomposition of the configuration space.

Probabilistic roadmaps are very efficient at finding feasible paths in high dimensional, complex environments[6][26]. Though they are probabilisticly complete, they are not good at finding paths through narrow passages. Connecting the graph through a narrow passage requires a node to be randomly sampled near the inlet to the passage, which without special consideration, has a very low probability of being sampled, just as any other particular point has a low probability of being sampled.

Since node are sampled randomly from the Cspace, there is no consideration given for dynamic constraints, which means even just because a feasible path is found does not mean it can be tracked. Never the less some work has been done to extend probabilistic road maps to work with non-holonomic vehicles, but those approaches usually use a two stage approach that adjusts for constraints with a smoothing function that runs after the road map has been generated[48][49][39].

Rapidly Exploring Random Trees

Rapidly Exploring Random Trees (RRTs) are like probabilistic roadmaps in that they randomly sample the Cspace, but unlike probabilistic roadmaps they only sample nodes that are feasible given a vehicle's dynamic constraints. RRTs are an extension of probabilistic roadmaps and are specifically meant to handle path planning for robots with kinodynamic, non-holonomic, constraints, like UAVs[31][27][30].

RRTs are created by randomly sampling points in the configuration space, finding the nearest existing node to the randomly sampled configuration, then expanding the graph from the nearest node towards the sampled configuration, and finally creating a new node close to the sampled configuration. The new node is known to be dynamically feasible because of the

way the graph is expanded. The new node is found by integrating a dynamically constrained model over a fixed period of time, using a control action that is selected to 'steer' the vehicle towards the sampled configuration. For any dynamic model given by $\dot{x} = f(x, u)$ a RRT can be created by sampling the configuration space of the vehicle, x_{rand}, and expanding in that direction. Algorithm 1 details the original algorithm for generating an RRT [29].

Algorithm 1 Original algorithm for generating an RRT

```
1: procedure GENERATE_RRT(x_init, K, Δt)
2:     T.init(x_init);
3:     for k=1 to K do
4:         x_rand ← RANDOM_STATE();
5:         x_near ← NEAREST_NEIGHBOR(x_rand, T);  ▷ According to the metric ρ
6:         u ← SELECT_INPUT(x_rand, x_near);
7:         x_new ← NEW_STATE(x_near, u, Δt);
8:         T.add_vertex(x_new);
9:         T.add_edge(x_near, x_new, u);
10:    end for
11:    return T
12: end procedure
```

The basic algorythm that builds the RRT tree T is composed of four main functions. The function RANDOM_STATE() returns a randomly sampled configuration from the Cspace of the vehicle. The sample may be uniformly random or biased towards certain configurations. Previous work has shown that for holonomic models, selecting sample configurations biased towards large regions of the voronoi diagram created by the existing nodes in T, produces a RRT with uniformly distributed nodes in Cspace.

The next two functions, NEAREST_NEIGHBOR() and SELECT_INPUT(), are responsible for finding a path from the existing nodes in T to the randomly sampled configuration x_{rand}. These two steps are the most computational intensive part of the RRT algorithm. Finding the exact nearest node in T to x_{rand} requires calculating the exact path from every node in T to x_{rand}, which could be up to k times more difficult than the original problem. Likewise, once the nearest node is found calculating the control action u that will 'steer' x_{near} to x_{rand} is just as difficult again as the original problem. Consequently, simple heuristics, like those discussed in the geometric path planning section, must be used to estimate both functions in order for the algorithm to run efficiently.

The NEAREST_NEIGHBOR() function uses the metric ρ to calculate a scaler value that estimates which node in T is nearest to x_{rand}. The metric ρ could also account for other parameters besides just the distance required to travel from T to x_{rand}, such as fuel consumption, and control effort. However, in most cases, for efficiency, ρ is typical just the euclidean distance.

As discussed before, finding the control action that will bring a non-holonomic, kinodynamic model from one configuration state to another is an NP-hard problem, therefore SELECT_INPUT() will not return the exact control input that will drive the vehicle from x_{near} to x_{rand}, but rather an approximation that is *likely* to drive the vehicle close to x_{rand}. Many different heuristics can be used to approximate u.

NEW_STATE then takes the control u that is returned by SELECT_INPUT() and returns a new node x_{new}, which is added to the tree T. SELECT_INPUT() generates x_{new} by integrating the model $\dot{x} = f(x, u)$ over Δt. Calculating x_{new} by integrating over the model of the system ensures that regardless of the u returned by SELECT_INPUT(), the new node in the graph, x_{new}, will be dynamically feasible. This method of inherently accounting for dynamic constraints is what makes the RRT motion planner a good choice for non-holonomic, kinodynamically constrained sytems.

Obstacle Avoidance with RRTs

In addition to being well suited to planning paths subject to non-holonomic and kinodynamic constraints, RRT are also very good at planning paths around non-convex, and even time varying obstacles [20][34]. Their random nature precludes them from becoming stuck in local minimums like naive potential field algorithms. In the presents of obstacles, as the function NEW_STATE() integrates the model forward in time it also checks for any collisions along the way. If NEW_STATE() detects a collision, then the sampled configuration x_{rand} is rejected, and the loop returns to the beginning where a new sample is taken.

3.4 Planning UAV Paths that Optimize Image Quality

One of the main contributions of this thesis is a novel approach towards controlling small UAVS to optimize the quality of the image that is recorded by an on-board cameras. The first step in optimizing the quality of the image captured by the UAV is controlling the UAV to a state that puts the target in the image. Once the target is in the image, a number of other factors can the considered to improve the image. For instance, often the most important aspect of an image is the resolution of the target that the image has captured. Higher resolution images contain more information about the target. Other factors, such as optical distortion, glare, occlusions, and blur can all have a large impact on the quality of the image.

The path planner presented here attempts to solve the optimal image collection problem in two steps by first finding a set of feasible paths, and then selecting the optimal path from the set of feasible paths. The first step is to find a path that puts the target within the UAV's sensor's field of view, here forward referred to as the UAV's field of view. If the path planner samples many paths that put the target within the UAV's field of view, then from that set of feasible paths, the path planner returns the path with the least cost according to

a given cost function, which is chosen to optimize the quality of the image.

One aspect of the path planning problem that differs from most is that the goal is not to reach one particular state. If the goal state of this path planner was defined as any configuration that puts the UAV's field of view on the target, the path planning problem is not just to reach that state, but rather to persistantly stay within the bounds of that goal space.

The problem of keeping the target path within the sensors field of view is akin to restricting the UAVs free Cspace to those configurations that keep the target within the sensor's field of view. Consequently, finding a path that keep the sensor on the target, is reduced to the problem of finding a feasible path through the 'tunnel' that is created by restricting the Cspace. Though the actual 'tunnel' of free space is multi dimensional figure 3.10 attempts to depict one two dimensional slice.

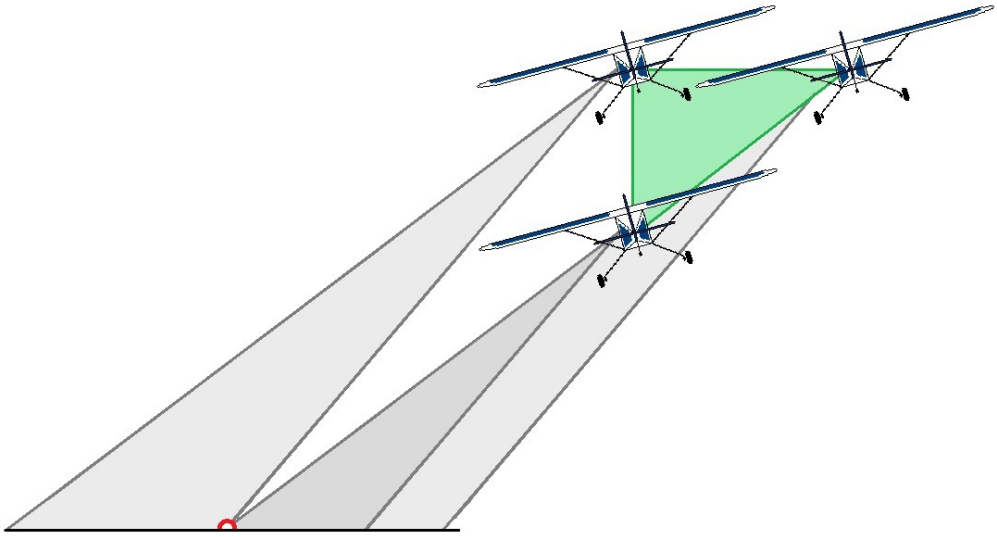

Figure 3.10: A two dimentional slice of the tunnel of free Cspace created by restricting the Cspace to configurations that keep the target within a sensor's feild of view.

In figure 3.10, the UAV's yaw, roll, and latitude have been fixed. The remaining free Cspace in the longitudinal and vertical directions is shown by the green triangle connecting the UAVs. In the actual problem presented here the UAV is restricted to a constant altitude, leaving it only four degrees of freedom (x, y, ψ, ϕ)

3.4.1 Relationship between the UAVs Lateral Displacement of its Sensor Footprint and its Turn Rate

For the small the UAVs presented here, the only control input is the UAVs desired turnrate. Commanding a desired turn rate is the only way to control the lateral motion of the UAV, which is necessary to steer the UAV towards any desired target. Unfortunately, as shown in equation 3.2 the turn rate of the aircraft also directly effects the aircraft's roll angle, which greatly effects the lateral displacement of the UAV's sensor footprint.

$$\phi = \tan^{-1} \frac{V\dot{\psi}}{g} \tag{3.2}$$

Where the lateral position, relative to the position of the UAV, of the sensor footprint is given by equation 3.3. Equation 3.3 assumes a body fixed downward looking field of view.

$$y = -z\tan\phi \tag{3.3}$$

By substituting equation 3.2 into equation 3.3, the resulting equiation 3.4 makes it apparent that the lateral displacement of the UAV's sensor footprint is inversely proportional to the UAV's turn-rate, which means that when the UAV steers towards the target, the field of view moves away from the target.

$$y = -\frac{zV}{g}\dot{\psi} \tag{3.4}$$

Since the UAV must sometimes steer towards the target path in order to eventually put the target within its field of view, it must at times move the sensor footprint away from the target, which means finding a feasible path is not always possible.

3.4.2 Approximating the Distance Between two UAV Configurations

A major requirement for just about any path planner, is the ability to approximate the distance between two independent configurations [22]. That distance function may be used in several ways including a metric to select the shortest path in a tree search, graph construction, or as a heuristic for the cost-to-go. In any case, the distance between two independent configurations is usually an approximation because finding the exact between two configurations would be just as hard as the original path planning problem. Often times the accuracy of the distance approximation can have a large effect on the efficiency of the planner.

In many cases, it is sufficient just to use the euclidean distance between the two states. Even for non-holonomic vehicles like fixed wing UAVs, this approximation is often valid, but a better approximation of the distance a UAV travels to get from one configuration to another is

given by the Dubin's distance [13]. The Dubin's distance between two points is the minimum distance between two states of a unicycle model. Every Dubin's path is composed of an initial maximum turn-rate turn, a final maximum turn-rate turn, and either a straight line or a third maximum turn-rate turn connecting the initial and final turns. Using the euclidean distance to approximating the distance between two UAV configuration is appropriate when the straight path of the Dubin's distance between those two configurations is much greater than both the initial and final turns. Another way of saying that, is that approximating the distance between two UAV configurations is appropriate when the two configurations are very far apart in the world space.

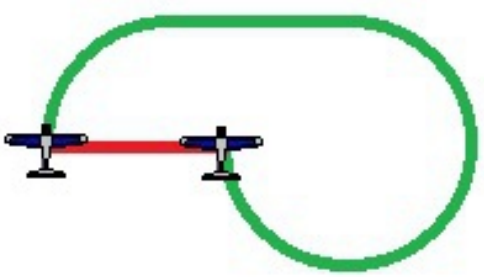

Figure 3.11: Dubin's distance compared to euclidean distance.

Figure 3.11 show an extreme case of how euclidean approximations can be very inaccurate over short distances. For probabilistic road maps and RRT, which use distance approximations to find the nearest nodes to random samples, bad distance functions, like this, make it hard to grow trees with a high density of nodes, which makes it hard to achieve probabilistic completeness. Because of their non-holonomic constraints, nodes which are very close together by euclidean distance, cannot typically be connected by a short feasible path. As a result the NEW_STATE() function in algorithm 1 does not generate a new node near the sampled node, which means that the RRT algorithm is incapable of generating a well distributed tree.

3.5 Solving the Field of View Path Planning Problem with an RRT

Finding a feasible path that keeps a target path within the field of view of a fixed camera on board a UAV is an NP-hard problem [20], but one that is well suited to the RRT framework. Unlike the original problem that RRTs were developed to address, this problem does not ask the RRT to find a feasible path through a complex environment in order to

reach a goal state, but rather to find any feasible path, or better yet a set of feasible paths that stays within a complex goal state for as long as possible.

Eventually this path planner will be re-described in a receding horizon context, but for now the only important aspect of that problem is that the distribution of the RRT is not just dependent on the current configuration of the UAV, but that the distribution of the RRT is also parametrized by time. Previous receding horizon RRT path planners have introduced such notions as *iterative planning* and *committed trajectories*, which address how the configuration space is continually sampled at every iteration of the planner[44]. Accordingly, this planner gives greater weight to the distribution of the states n steps along each path, as the n^{th} step will have the greatest effect on the optimal configuration of the UAV in the future.

While random sampling is what gives RRT the ability to find paths through complicated, non-convex Cspaces, sampling uniformly over the whole configuration space is often inefficient, and a number of algorithms have proposed more efficient sampling strategies. This planner build on the idea that it is more efficient to sample around known good configurations by using a guided Gaussian search algorithm.

Unlike most other UAV path planning problems, the solution to this problem is not actually an aircraft path, it is a sensor path for a camera fixed to a UAV. As it was stated before, the path of the sensor footprint is hard to control largely because it is inversely proportional to the turn rate of the UAV. It is also hard to plan because small changes in the UAV's roll angle have a drastic impact on the lateral position of the sensor footprint. Consequently, it is necessary to have a very dense sampling algorithm to account for configurations which are close together in the world space but are very different in the Cspace, due to different roll angles.

3.5.1 Generating an RRT with Iterative Gaussian Sampling

This planning algorithm addresses two main issues with the standard RRT algorithm, namely inefficient sampling and costly heuristic distance functions. The results presented here show that uniform random sampling in the control space produces a Gaussian distribution in the Cspace. As other approaches have shown, biased, Gaussian sampling around tight constraints can drastically improve performance[6]. The advantage of this sampling strategy is that it does not require the most computationally costly function of the RRT algorithm, namely the NEAREST_NEIGHBOUR() function.

It is well know that the NEAREST_NEIGHBOUR() function is the key bottle necks in the RRT algorithm [31]. This is especially true of path planning for UAVs, which as stated before must use more computationally costly distance metrics like Dubin's distance algorithms. Likewise, for relatively free configuration spaces that are free of non-convex obstacles, it is inefficient to sample uniformly across the Cspace. One common method it to bias the sample distribution around locations that are likely to generate good paths. By uniformly sampling in the control space and integrating every branch forward n steps into

the future, this algorithm produces a Gaussian distribution around an a priori suggested path. A key feature of this sampling strategy is that it produces a Gaussian distribution without having to run any NEAREST_NEIGHBOUR() function. Algorithm 2 details how to generating an RRT with Itereative Gaussian Sampling (IGS).

Algorithm 2 Generating an RRT with Itereative Gaussian Sampling

```
1: procedure GENERATE_RRT(x_init, u_init, K, Δt)
2:     u_best ← u_inti                          ▷ u = [u_1 ... u_n] is a sequence of control inputs
3:     J_best ← PATH_COST(x_init, u_init)
4:     for k=1 to K do
5:         u_rand ← RANDOM_CONTROL(u_best);
6:         J_rand ← PATH_COST(x_init, u_rand)
7:         if J_rand < J_best then
8:             u_best ← u_rand
9:         end if
10:    end for
11:    return u_best
12: end procedure
```

There are two main functions in algorithm 2. The first RANDOM_CONTROL() generates a vector of controls of length n that samples uniformly from the control space, which in this case is the set of desired turn-rate ranging from ψ_{min} to ψ_{max}. Ever path that the RRT algorithm generates is fully parametrized by the UAVs initial configuration $\mathbf{x}_{init}$, and the vector $\mathbf{u}$ that is integrated to generate it. Therefore it is unnecessary to store the entire distribution of nodes, as long as the vector U that generated the best path is stored. RANDOM_CONTROL() takes the previous best path as an input and uses it to generate random paths that are uniformly distributed around the previous best path.

The PATH_COST() function calculates the cost of each node along the path parametrized by $\mathbf{x}_{init}$ and $\mathbf{u}$. The PATH_COST() accounts for the bounds of the UAV's field of view by dramatically increasing the cost of paths that do not put the target within the UAV's field of view. The PATH_COST() function also calculates the cost of any other image quality parameters such as distance from the center of the sensor footprint, view angle, and distance from the target. The PATH_COST() uses a model of the UAV to integrate over the control sequence U to estimate the future cost of any potential path.

Algorithm 2 is extremely memory efficient and simple to implement. Since, generating a Gaussian distribution does not require referencing the current distribution of nodes, it is unnecessary to store the entire tree. Furthermore each path, including the best path, is completely parametrized by the vector of control inputs $\mathbf{u}$. The only cost associated with not storing the full graph, is slight increase in computation since some of the initial nodes, which may be close in the Cspace, are recomputed instead of referenced. On the the other hand, by not referencing, and growing the tree from just a few initial nodes, the resolution of

the graph just ahead of the UAV is very high, which drives the planner towards probabilistic completeness and the optimal solution.

3.5.2 Uniform vs Gaussian Distribution of Nodes

As was stated before, this discussion focuses on the distribution of nodes at the end of each path. Since this is a receding horizon path planner the Cspaces is actually sampled many times throughout the planning horizon of the vehicle, but only one instance of the single-query RRT is discussed here. Since the n^{th} node in the path has the greatest opportunity to explore the Cspace ahead of the vehicle, it has a large impact on the head of the UAV's path. Therefore the remainder of this discussion will focus on the distribution of the n^{th} node.

Uniform Distribution of Nodes

Uniformly sampling the Cspace of a UAV is difficult mostly because of the challenge of finding a NEAREST_NEIGHBOUR() function that accurately chooses which node to expand in the direction of the sampled configuration. The uniform distribution used in the following simulations was constructed off-line with rejection sampling. A uniform distribution of paths in the world space is shown in figure 3.12.

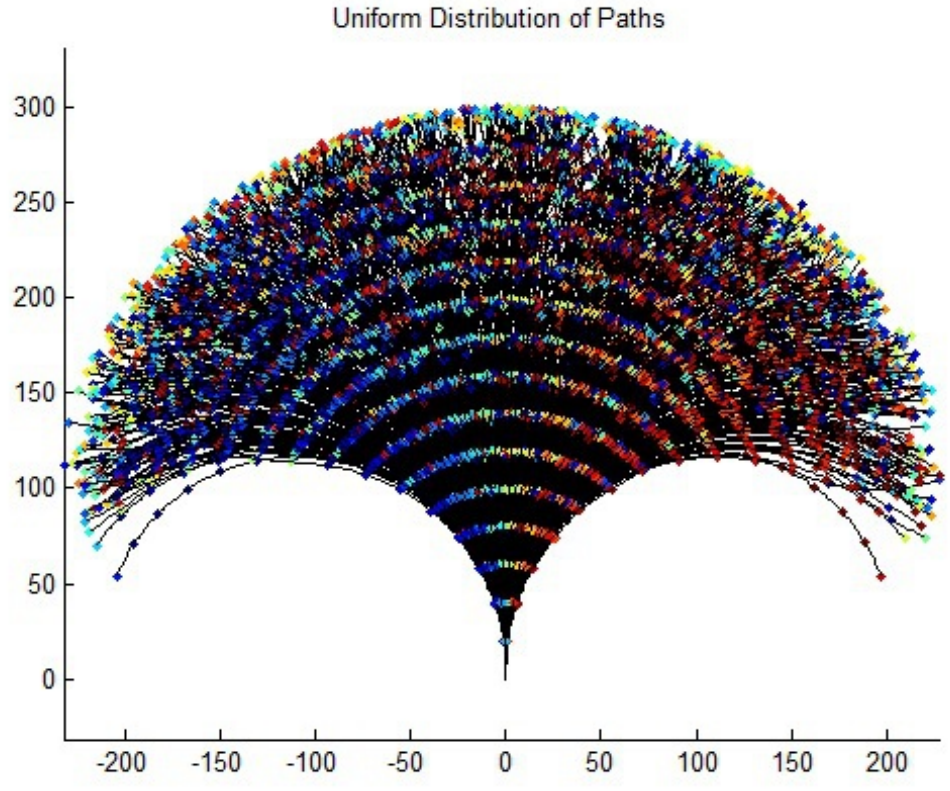

Figure 3.12: A uniform distribution of paths in the world space.

It is difficult to show the four degree of freedom Cspace in just two dimensions, but in figure 3.12 all four dimensions are represented. The x and y axes represent the UAV's latitude and longitude. The yaw angle of the UAV is equal to the tangent line of any point along the UAV's path. And finally, the roll angle of the UAV is shown at every node on a color scale ranging from blue to red. Even so, it is still difficult to see the uniformity of the distribution. Figure 3.13 attempts to simplify this picture even more by looking at only the n^{th} node along each path.

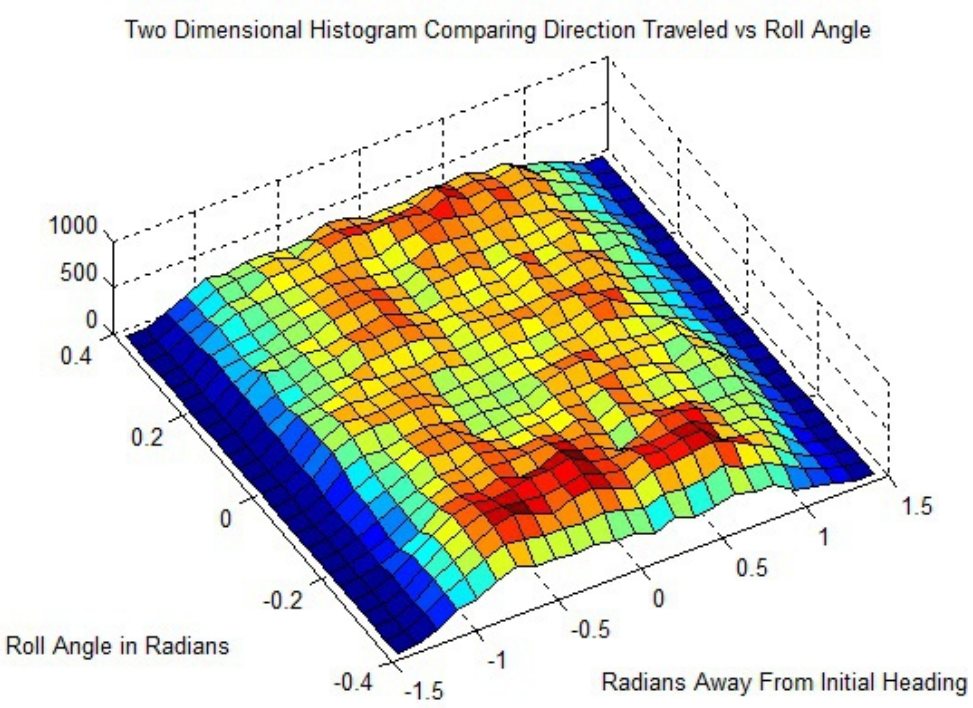

Figure 3.13: A near uniform distribution of nodes

In this two dimensional histogram, one axis is the roll angle of the UAV at the n^{th} node, while the second axis is the angle the UAV has travelled from the initial configuration. In the world space shown in figure 3.12, the path angle can be visualized as the angle between a y-axis, and the vector pointing from X_{init} to the last node of the path. The mathmatical representation of this metric is given by the equation 3.5.

$$\angle = \tan^{-1} \frac{x_n.y - x_{init}.y}{x_n.x - x_{init}.x} - x_{init}.\psi \tag{3.5}$$

The path angle tries to capture the displacement of the UAV in the world space, along with a rough approximation of the yaw angle of the aircraft. This metric only makes sense because of the short time horizon presented here. metric assumes that in order for the UAV to fly far to the left or right, the mean turn-rate, must have been to either the left or the right, which would result in a final yaw angle that is either biased to the left or right. Figure 3.13 shows that n^{th} nodes are approximately uniformly distributed for path angles between ± 1 radian.

Gaussian Distribution of Nodes

Similar to figure 3.12, figure 3.14 shows a Gaussian distribution on paths. As stated in algorithm 2 the Gaussian distribution lies around and initial guess of the best path, which, in this case, for the purposes of comparison to the uniform distribution is a straight line.

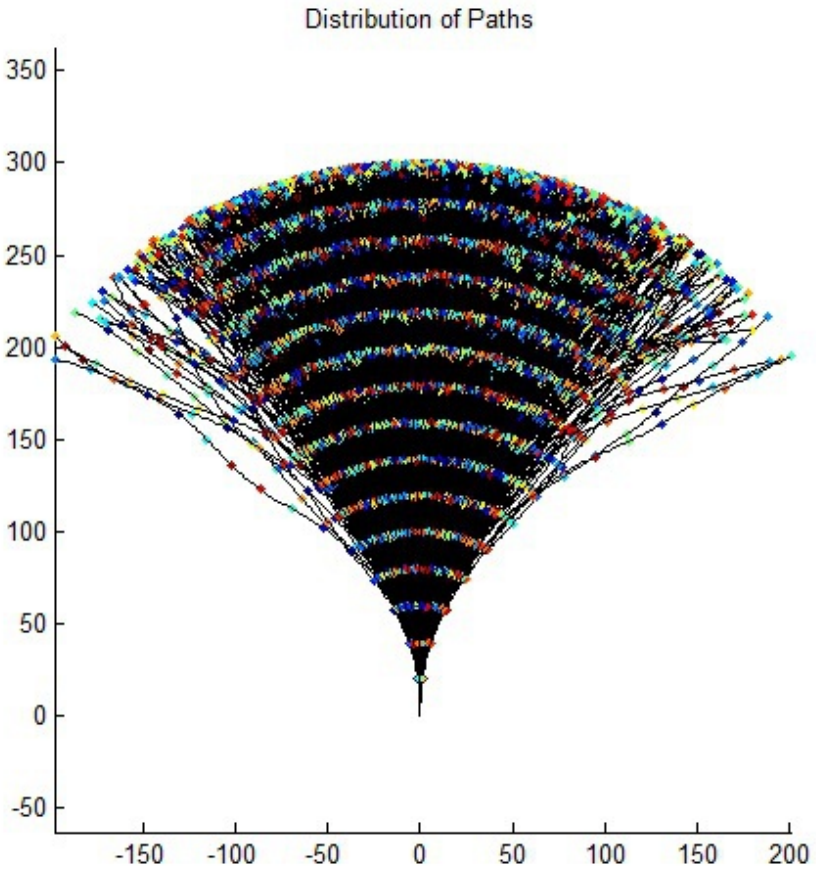

Figure 3.14: A Gaussian distribution of paths in the world space.

Figure 3.15 shows the same two dimensional histogram of roll angle vs path angle as figure 3.13. In this case, it is apparent that the roll angle of the n^{th} node is indeed uniform, which is to be expected since the desired turn rate, and by extension roll angle, for that node is selected uniformly from the control space.

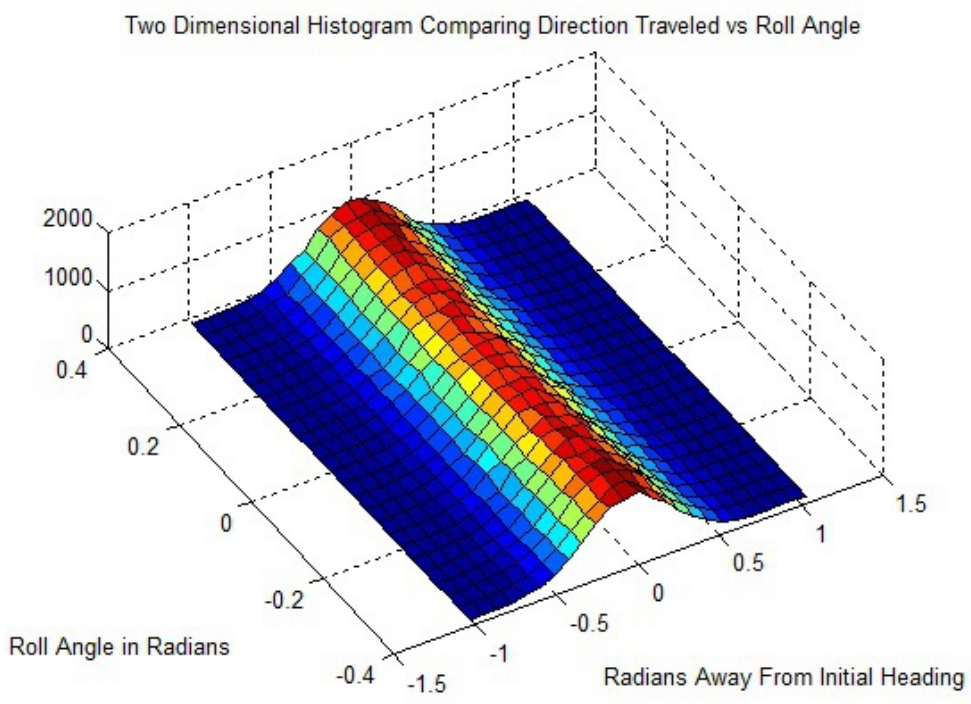

Figure 3.15: A Gaussian distribution of nodes

It also looks like the path angle is normally distributed around a mean of zero. The *Central Limit Theorem* supports this observation since the final lateral displacement of the UAV is subject to a sequence of independent identically distributed random variables, namely the uniformly sampled sequence of turn-rates. The histogram in figure 3.16 confirms this by fitting a Gaussian curve to the path angle data presented in figure 3.15.

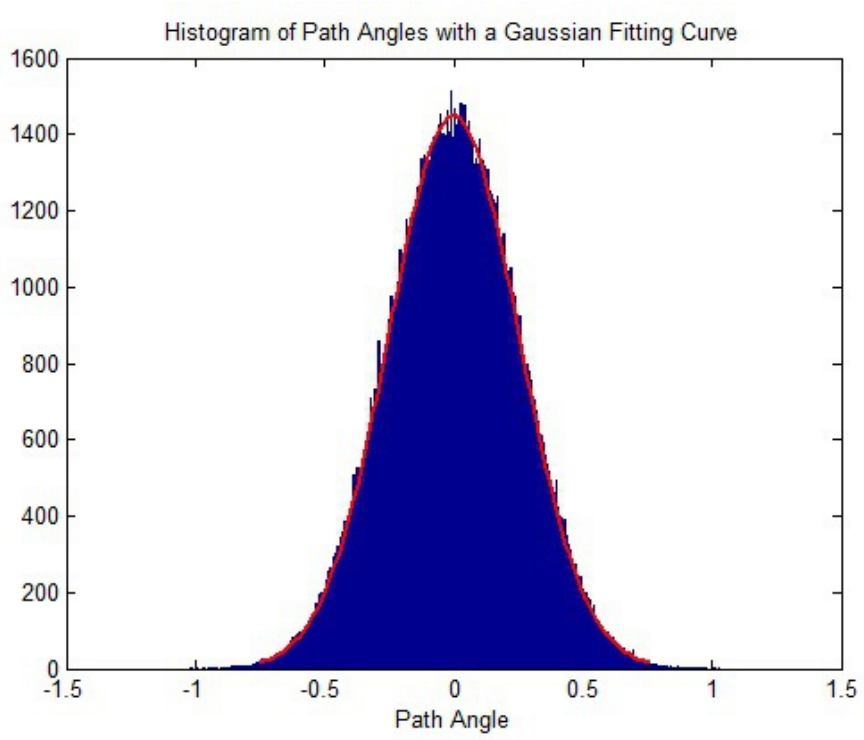

Figure 3.16: A Gaussian distribution of nodes

Uniform vs Gaussian Performance

There has been substantial work done to show that uniformly distributed RRTs are probabilist complete. Previous work has shown that an RRT with a uniform distribution of nodes will find a feasible path with a probababolity that exponentially approaches one[20]. This section aims to demonstrate by Monte Carlo simulation that Iterative Gaussian Sampling, as described in algorithm 2 can to achieve a significantly lower cost than uniform sampling.

The Monty Carlo simulations were conducted by first randomly generating a feasible target path in the configuration space. The target paths were generated uniformly across the Cspace. The target paths were generated by integrating a give control sequence over a standard period of time, so it is known that there is atleast one feasible solution. Then both the uniform RRT and Gaussian RRT attempt to find the UAV path that generates the lowest cost. The lowest cost path in this case is path that keeps the target path as close as possible to the center of the UAV's sensor footprint. Specifically, it is the mean squared cross-track error.

For a given path length n a number of simulations were run, each time with a different target path. The *Standard Error of the Mean* is used to qualify the results of the Monte

Carlo simulations. The Standard Error of the Mean is used to generate a confidence interval that bounds the actual average mean squared cross-track error of each path planner. Figure 3.17 shows how the standard error of the sample set decreases as the number of Monte Carlo simulations increase.

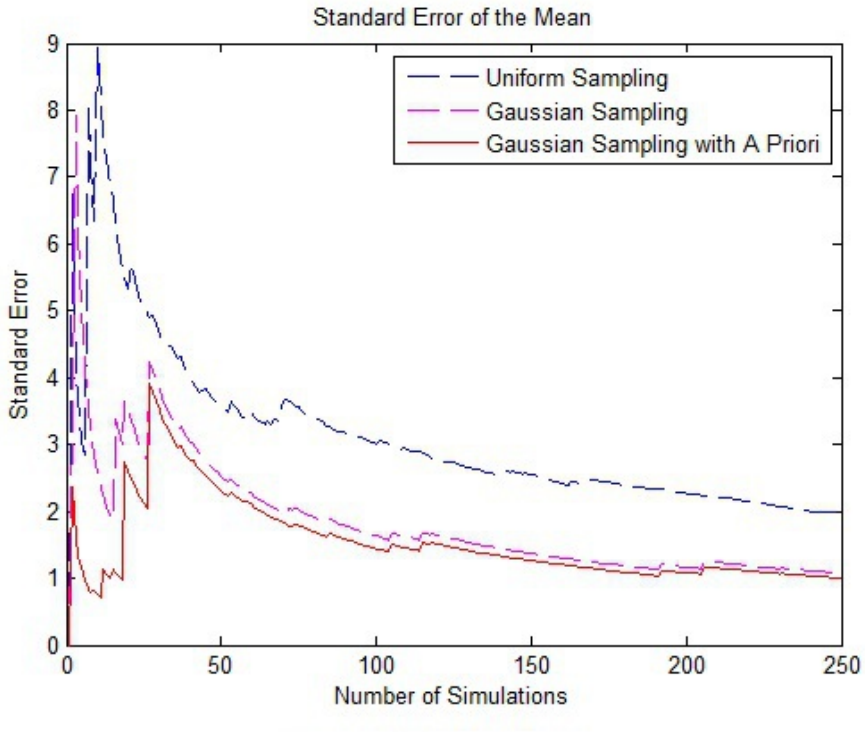

Figure 3.17: As the number of Monte Carlo simulations increase, the standard error decreases.

Using the Standard Error to qualify the Monte Carlo results means that the Monte Carlo estimates of the average mean squared cross-track error of each RRT algorithm can be bounded by an interval of 95% confidence. So given the number of simulations run, it can be said with 95% confidence that the true average mean squared cross-track error of each algorithm is with in a bounded interval of the sample average of mean squared cross-track errors.

Figure 3.18 shows the average mean cross-track error generated by both RRT algorithms. From this graph it is obvious that for the same number of sampled nodes, the Iterative Gaussian Sampling method generates paths that keep the target path several times closer to the center of the UAV's field of view far better than the uniform sampling method.

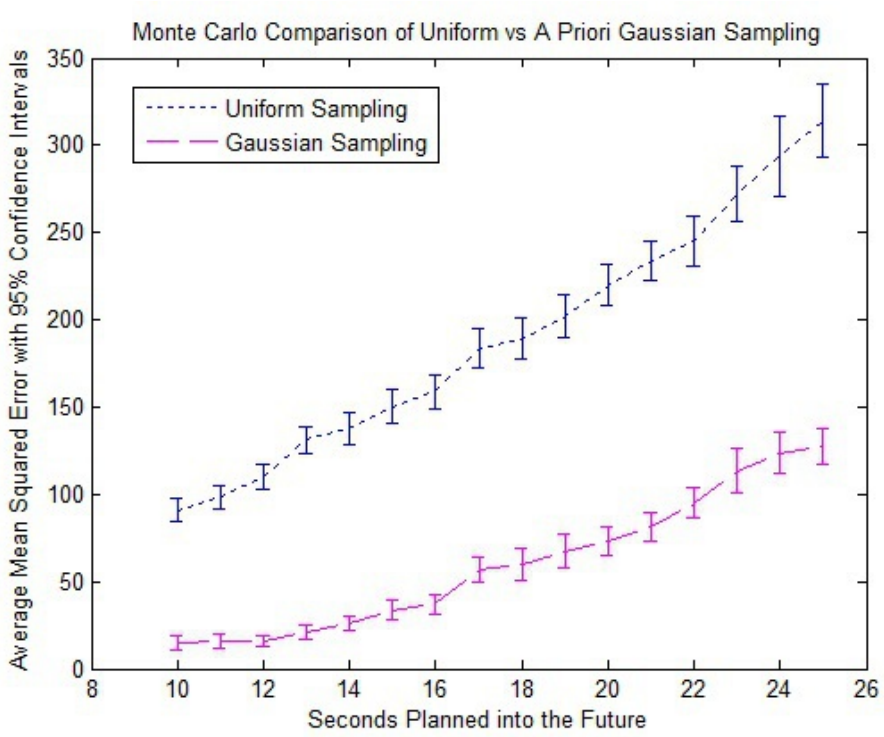

Figure 3.18: Results Average error of paths generated by uniformly distributed and normaly distributed RRTs based on Monte Carol simulations.

Figure 3.18 shows the performance of the iterative Gaussian sampling method with no a priori guess of the best path. With an a priori guess of the best a path, the average mean cross-track error is halved again (see figure 3.19)

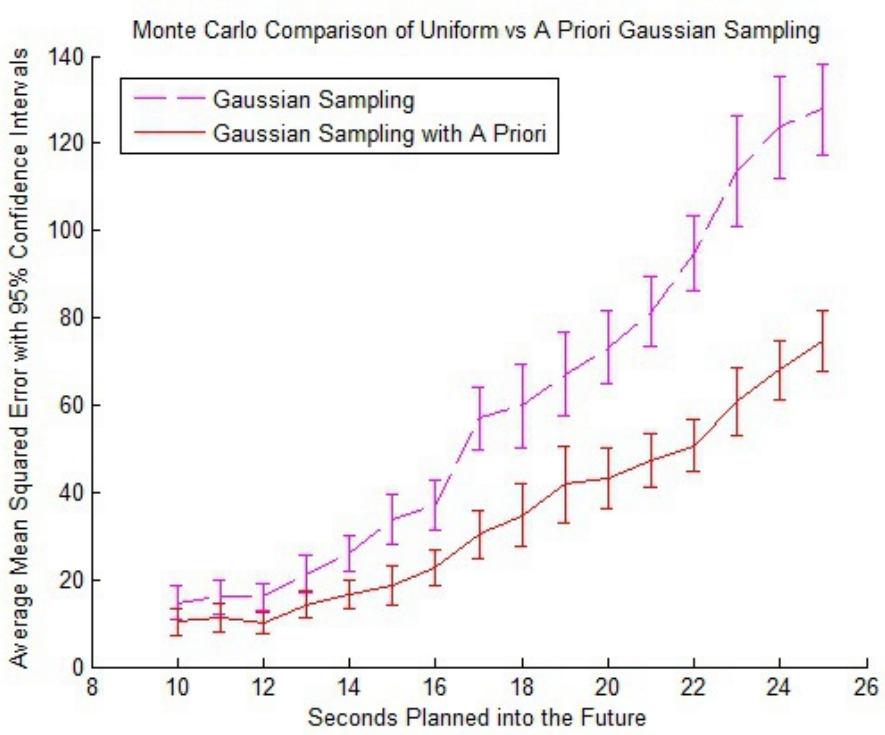

Figure 3.19: Improvement to the Iterative Gaussian Sampling RRT with a priori knowledge.

In figure 3.19 the IGS-RRT is given an a priori guess of where the best path may lie in the Cspace. Since, this is a receding horizon path planner, the IGS-RRT uses the previous path planning solution as a best guess for the next planning step. Since the RRT path planner produces a uniform distribution of nodes, it does not have an inherent method of using information from the previous iteration of planning. As a result, figure 3.20 shows that by using the information learned in the previous planning step the IGS-RRT is able to plan far more optimal paths than the traditional RRT algorithm.

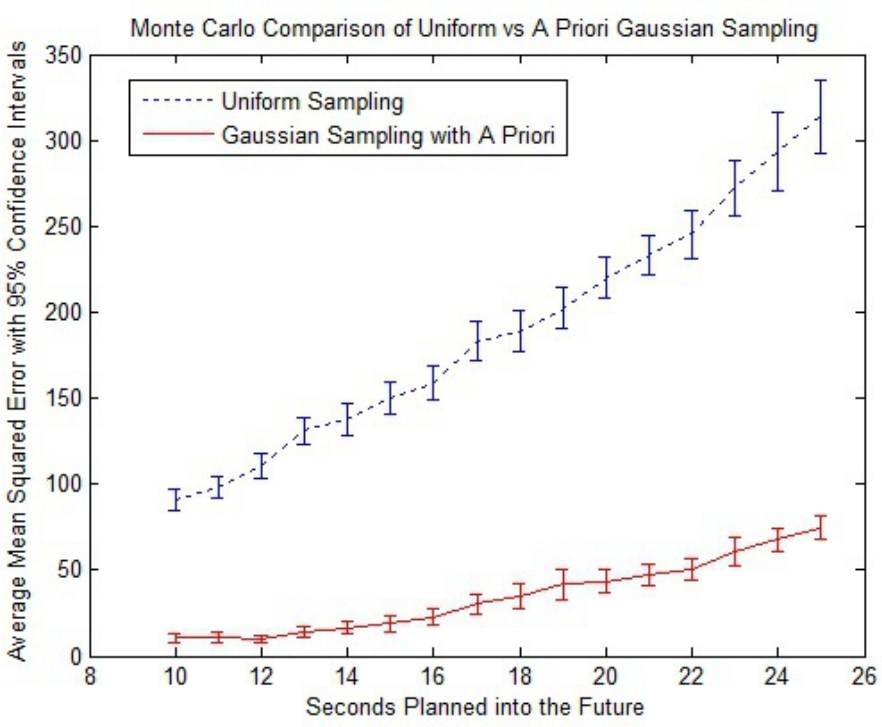

Figure 3.20: Improvement to the Iterative Gaussian Sampling RRT with a priori knowledge.

3.6 Conclusions

This chapter presented several common motion planners along with a novel iterative Gaussian sampling strategy for the rapidly exploring random tree algorithm. The relative advantages and disadvantages of each motion planner were discussed with respect to motion planning for small UAVs. In particular, the chapter discussed the difference between fast motion planning, and kinodynamic motion planning. Data from the Monte Carol simulations showed that, given the same number of samples, the IGS-RRT generated paths that cost, on average, nearly ten times less than paths generated by a uniform sampling strategy. The data also showed that the IGS-RRT algorithm was able to generate a Gaussian distribution of nodes without calling a 'nearest neighbour' function, which significantly reduces the computational complexity of the algorithm. Lastly, the IGS-RRT showed that by initializing the tree with the 'best path' from the last query, it essentially continues to iterate on the same tree, and returning solutions like an anytime algorithm.

Chapter 4

Turn-rate Controllers for Small UAVs

This chapter presents a novel nonliner model predictive controller for small UAVs that that solves the nonminimum phase problem of tracking a desired path with a UAV's sensor footprint. The NMPC receding horizon controller uses the kinodynamic model, and the iterative Gaussian sampling RRT algorithm discussed in the previous chapters to rapidly converge to a dynamically feasible, near optimal path. Then it uses the turn-rates that parametrizes the near optimal path to steer the UAV.

For comparison, a purely feedforward controller, a simple PID controller, and a spatial sliding mode controller are also presented. Although none of these controllers are capable of solving the nonminimum phase sensor tracking problem, a degenerate minimum phase tracking problem is considered so that their performance can be compared to the NMPC controller's performance.

Finally, hardware-in-the-loop data is presented to show that in the highest fidelity comparison, the NMPC controller is capable of achieving nominally the same tracking performance as the spatial sliding mode controller, for the degenerate minimum phase tracking problem.

4.1 Introduction

The goal of a typical UAV autopilot is to control the position of the UAV by either having it fly toward a specific point, or along a specific route. In contrast, the goal of the autopilot presented here is to control the position of a UAV's sensor footprint to track either a specific point or path. In this context, a UAV's sensor footprint is defined as the projection of the maximum field of view of any sensor, namely a camera, from the aircraft to the ground. For a UAV with a fixed camera, the sensor footprint of the UAV would simply be the camera's field of view projected onto the the ground. For a UAV with a gimballed camera, the sensor footprint of the UAV would be the union of all of the camera's sensor footprint's modulated across the gimbal's angles.

The goal of this controller is not to control just the position of the UAV, but it's attitude as well. What makes this controller unique is that the error it measures is the cross-track error of the sensor footprint rather than the tracking error of the UAV. The data presented here will attempt to show that directly controlling the error of the position of the sensor footprint rather than the position of UAV, improves the quality of the data collected, specifically images from a fixed camera.

4.1.1 Motivation

The goal of this controller is to increase the autonomous capabilities of small UAVs so that they are capable of reliably carrying out image and data collection with less supervision. The plurality of UAVs are small hand launched vehicles with minimal autonomy. They are many orders of magnitude cheaper than larger UAVs like the Predator and Global Hawk, but they have significantly less recognisances capabilities, most notably, UAVs like the Raven do not have a gimballed camera. The Predator and Global Hawk have gimbals that give them the capability of accurately placing their sensor footprint on just about any target within range regardless of their relative position and attitude to the target. The Raven on the other hand, must fly specific patterns around the target that will hopefully put its fixed sensor footprint on the target. At this point, their ability to reliably position their sensor footprint on a target is quite limited, but if that capability was improved, they would be capable of accomplishing the same missions as larger UAVs at a fraction of the cost.

4.1.2 Problem Statement

Consider a UAV with a body fixed downward looking camera. The position of the UAV is represented as (x, y) and the yaw and roll of the UAV are respectively ψ and ϕ. Figure 4.1 shows that the position of the center of the UAV's sensor footprint, (x_s, y_s), is simply the intersection of the body fixed z-axis with the ground plane. If the goal of the UAV is to keep its sensor footprint on a desired target, then the goal of the sensor footprint controller is to minimizes the error $e = (x_s, y_s) - f((x_s, y_s))$

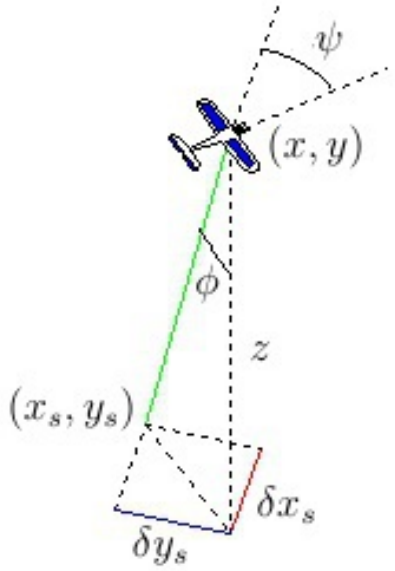

Figure 4.1: The geometry of the sensor footprint of a body fixed downward looking camera

Figure 4.1 shows the difference between the UAVs position and the position of the sensor footprint, which is:

$$\begin{aligned} \delta x_s &= z \tan\phi \sin\psi \\ \delta y_s &= -z \tan\phi \cos\psi \end{aligned} \tag{4.1}$$

Substituting the coordinated turn assumption,

$$\tan\phi = \frac{V\dot{\psi}}{g}$$

into difference equations 4.1 yeilds:

$$\begin{aligned} \delta x_s &= \frac{zV\dot{\psi}}{g} \sin\psi \\ \delta y_s &= -\frac{zV\dot{\psi}}{g} \cos\psi \end{aligned} \tag{4.2}$$

The equation for the center of the sensor footprint is therefore:

$$\begin{cases} x_s = x + C\dot{\psi}\sin\psi \\ y_s = y - C\dot{\psi}\cos\psi \end{cases} \tag{4.3}$$

Where $C = \frac{zV}{g}$.

4.1.3 The Control Challenge

The challenge of using a UAV's turnrate to control its sensor footprint is that the vehicle's lateral control is inversely proportional to the lateral displacement of its sensor footprint. In other words, inorder to control the sensor footprint to go to the left, it must first be commanded to the right. In a linear system, that type of behaviour would be indicative of a non-minimum phase system, meaning that the system has zeros in the right half plane. Since the idea of having zero's in the right half plane does not easily extend to non-linear systems, it is difficult to extend that concept to this problem, but looking at the system's zero-dynamics can give similar incite into the behaviour of the system[47].

The relative degree of a non-linear system is given by the number of times the output of the system must be differentiated before it becomes a function of the control input. For instance, if the output of a unicycle was the position of its sensor footprint, then the system would have a relative degree of zero, since the output is a direct function of the input.

$$e = y_s - f(x_s) = (y - Cu\cos\psi) - f(x + Cu\sin\psi) = g(\mathbf{x}, u) \tag{4.4}$$

Whenever the relative degree of a system is not equal to the order of the system some of the dynamics of the system will be 'unobservable'. Those unobservable dynamics are refered to as the *internal dynamics* of the system. If those internal dynamics are at least locally stable about a desired trajectory, then a number of feedback linearisation techniques can be used to control the system. If not, then perfect tracking is most likely not possible, and the goal of the controller should only be to minimize, if not bound, the tracking error[47].

The stability of the internal dynamics about the desired trajectory can be found by looking at the system's zero-dymanics. The *zero-dynamics* of a system describe the internal dynamics while the control is driving the error to zero.

Consider a UAV modeled as a kinematic unicycle, with a fixed downward looking camera, and a simple desired sensor path $y_s^d = f(x_s) = x_s$. Then the equations for its sensor footprint given by the equations 4.3, and the output error would be:

$$e = y_s - y_s^d = y_s - x_s \tag{4.5}$$

Then substituting equations 4.3 into equation 4.5 gives:

$$e = y_s - f(x_s) = (y - Cu\cos\psi) - (x + Cu\sin\psi) \tag{4.6}$$

Since the relative degree of the system is zero, then the error can be controlled directly

by the input. Using feedback linearisation, the controller that achieves perfect tracking is simply the solution to the equation:

$$e = y_s - f(x_s) = (y - Cu\cos\psi) - (x + Cu\sin\psi) = 0 \tag{4.7}$$

or

$$u = \frac{y - x}{C(\cos\psi + \sin\psi)} \tag{4.8}$$

Figure 4.2 shows the error that the feedback linearisation control law 4.8 can achieve. Though the error is, for the most part, reduced to zero, there is a large difference between the relative degree of the system and the actual third order model of the UAV, which means there are unobserved internal dynamics.

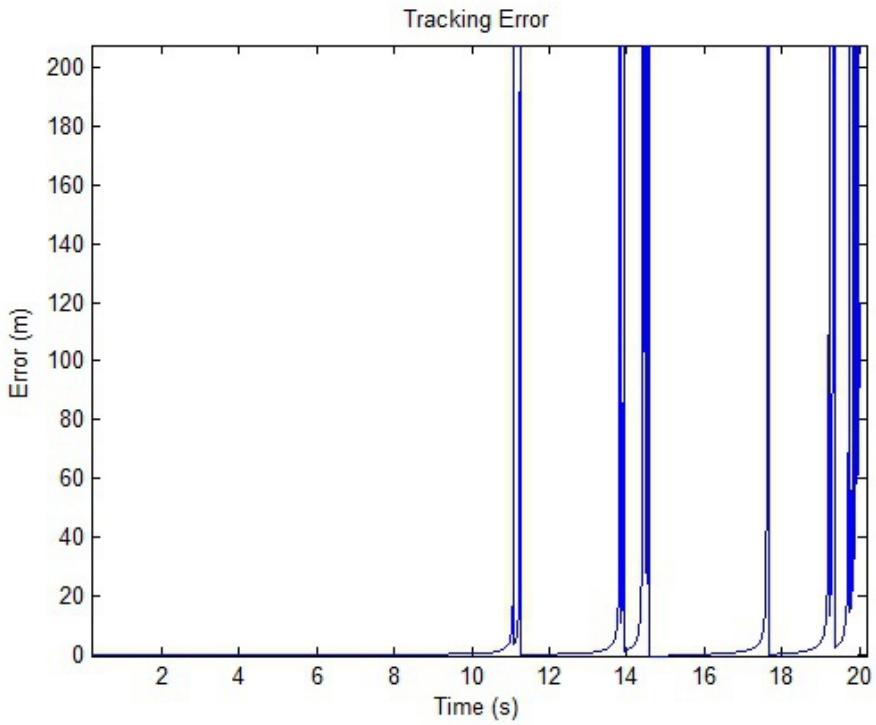

Figure 4.2: Error based on feedback linearisation.

If it were not for the numerical limitations of the simulation, the error would be uniformly zero, which means that the internal dynamics of this system are also the zero-dynamics. For the most part the error is controlled to zero, but the occasional near infinite spikes in error hint that zero-dynamics of the system may be unstable. Looking at the control input in figure 4.3 it is clear that the zero-dynamics of the system are indeed unstable.

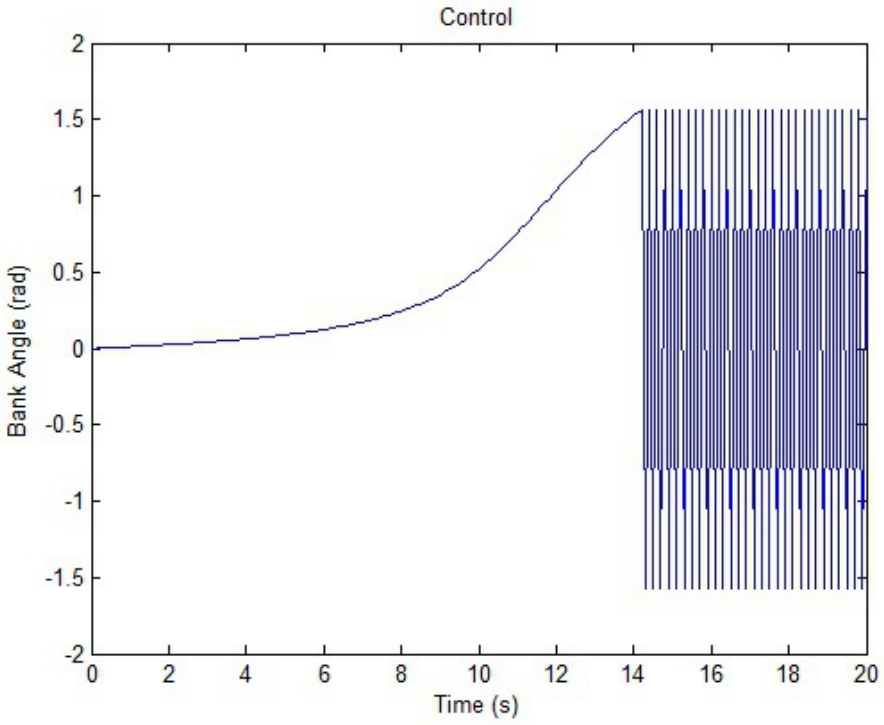

Figure 4.3: Control of the feedback linearisation law.

In order to achieve perfect tracking, figure 4.3 shows that the system must use infinite control. Figure 4.3 shows that to numerical precision, the bank angle of the aircraft switches with infinite frequency between a positive and negative 90 degree bank angle, which corresponds to an infinite turn-rate to the left and right. Figure 4.4 shows the path of a simulated UAV using this control strategy.

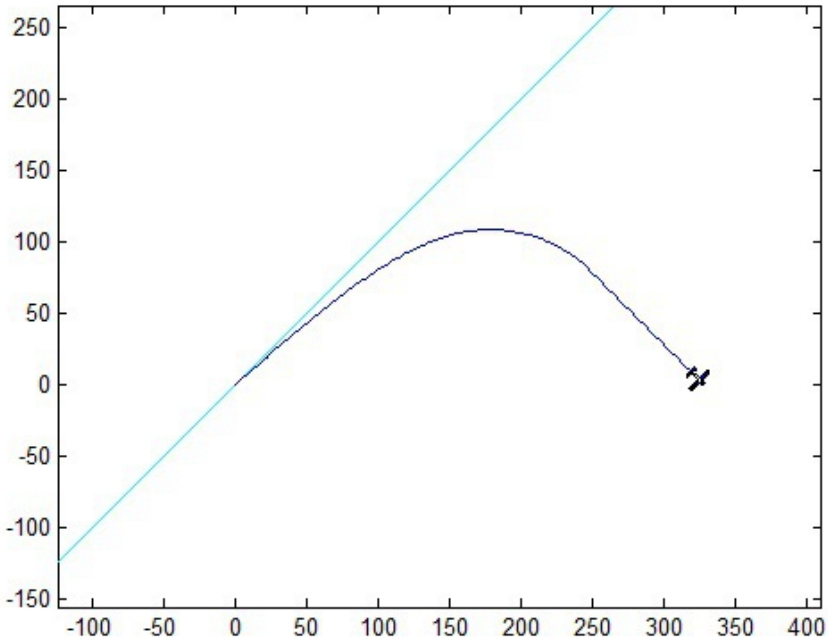

Figure 4.4: Simulation of a UAV using feedback linearisation.

As one would expect, even thought the sensor path is tracked perfectly, the UAV does not behave as intended. The results of this simulation confirm that this system is in fact a non-minimum phase, non-linear system as defined by [47]. As a result, except for a few degenerate examples, any controller that attempts to achieve perfect tracking will be unstable.

4.2 Myopic Solutions

Myopic solutions are the set of controllers that 'greedily' attempt to minimize the output error at a given instant without regard for any potential future error. By definition, myopic controllers that are applied to the sensor footprint control problem described in section 4.1.2 will cause the system to be unstable. On the other hand, many myopic solutions are easy to implement, and computational inexpensive. But, in order to apply any myopic controller to the sensor footprint tracking problem, the problem must be redefined to one which is asymptomatically minimum phase.

One way to do that, is to find the optimal aircraft path that when followed exactly, will minimized the error between the sensor footprint and the desired target. Unfortunately, as it was discussed in chapter 3, finding that optimal aircraft path is NP-hard. But given that

the tracking problem is non-minimum phase, and that therefore the goal of the controller is not perfect tracking, but only to minimize the error, a near optimal solution to the tracking problem will suffice.

There are a number of ways of translating a sensor path into an aircraft path, some of them are presented in chapter 3, but assuming the aircraft path is near optimal, tracking it perfectly should, by definition, also cause the UAV's sensor footprint to track the desired sensor path. This assumption, however, relies on the principle that the aircraft path can be tracked perfectly, which for any real system is impossible, if for no other reason then that the running time of the controller would introduce delay, which would result in tracking error.

The danger of assuming that perfect tracking of the aircraft path results in perfect tracking of the sensor path is in making the further assumption that minor tracking errors of the aircraft path result in minor tracking errors of the sensor path, which is not true. Take, for instance the projected sensor path of the sinusoidal aircraft path in figure 4.5. If an aircraft were to fly straight with its wings level down the center of the aircraft path, the mean squared cross track error of the aircraft path would only be 49 m^2, while the mean squared cross track error of the sensor path would be 1161 m^2.

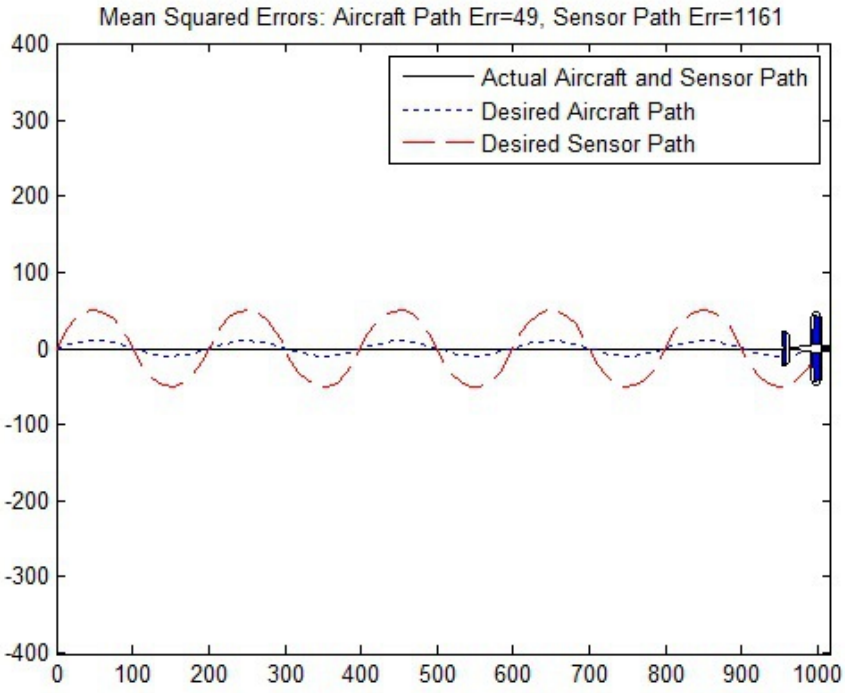

Figure 4.5: Example of nominally bad aircraft tracking, but very bad sensor tracking.

While this example may be slightly contrived, its purpose is to demonstrate the fact

that nominally following an aircraft path does not necessarily imply nominally following the sensor path. The reason, of course, is nominally following the aircraft path does not required the UAV to track a specific bank angle. For instance, a controller that oscillates back and forth across the aircraft path may produce error similar to figure 4.5, but in reverse. In order for a myopic controller to track a desired sensor path well, by tracking a desired aircraft path, it must also come close to tracking the radius of curvature of the desired aircraft path. By closely matching the radius of curvature of the desired, the aircraft will then come close to matching the desired bank angle at every point along that path.

The remainder of this section will discuss the performance of a few myopic controllers, and how well they are able to track a desired sensor path, given an optimal aircraft path. In each case, the the optimal aircraft path is a sine curve, and the desired sensor path is the projection of that sine curve onto the ground plane. The minimum radius of curvature of the the aircraft path is only about 80% of the aircraft's minimum turning radius, so the path is feasible for any controller to track.

4.2.1 Tracking the Radius of Curvature of the Optimal Path

Assuming that the optimal aircraft path for a desired sensor path has already been generated by a higher level path planner, it is not unreasonable to assume that if that path could be turned into a control law, then that control would produce reasonable tracking performance. By looking at the instantaneous radius of curvature at a given point along the desired aircraft path, the desired turn-rate at a point along the path can be calculated using equation 4.9, which describes the rate of change of a tangent line along a curve.

$$\kappa = \frac{d\psi}{ds} \tag{4.9}$$

κ is the curvature of, and inversely proportional to, the osculating circle at any point along the path s. Applying the chain rule to equation 4.9 gives the following result.

$$\kappa = \frac{\frac{d\psi}{dt}}{\frac{ds}{dt}} \tag{4.10}$$

or

$$\dot{\psi} = \kappa V \tag{4.11}$$

For a curve described by a function such that $y = f(x)$, the curvature, $\kappa(x)$ can be calculated by the following equation.

$$\kappa = \frac{\frac{d^2y}{dx^2}}{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{3/2}} \tag{4.12}$$

The open loop control law is then found by substituting equation 4.12 into equation 4.11.

$$u(x) = \dot{\psi} = V \frac{\frac{d^2y}{dx^2}}{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{3/2}} \tag{4.13}$$

Assuming a UAV had the proper initial conditions, and was perfectly modelled as a unicycle, applying the continuous control law from equation 4.13 should result in perfect tracking. Figure 4.6 shows the results of applying this control at 30hz.

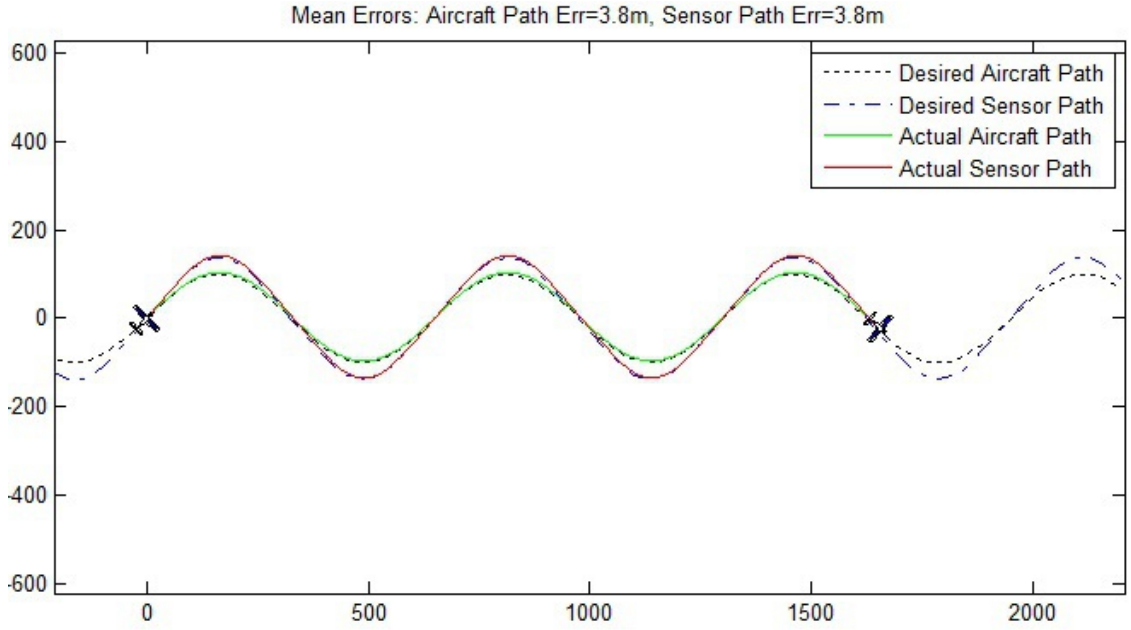

Figure 4.6: Steady state tracking performance of an open loop controller.

In simulation, the performance of the open loop controller is quite good, but despite the best conditions, the controller still suffers from lag and delay. Since the UAV is not a true unicycle, lag from the turn-rate response slowly adds error to the system. Likewise, since the control is not applied continuously, even at 30hz, small delays in the control adds up over time. Most importantly, this pure open loop control has no method of rejecting disturbances. As figure 4.7 shows, any wind at all will through the UAV irrecoverably off course.

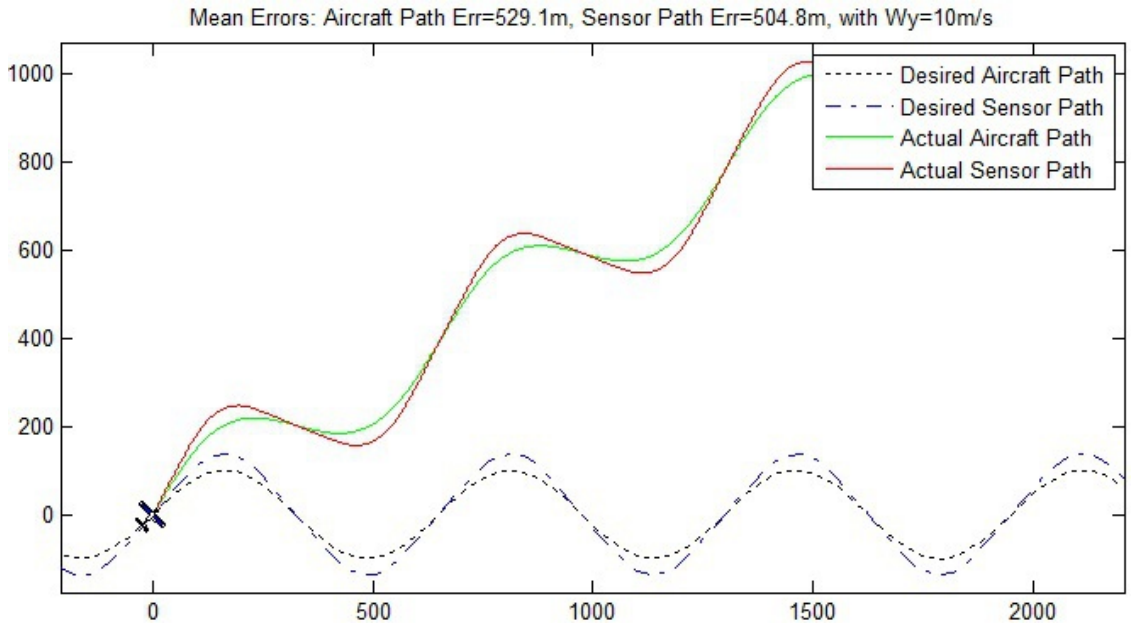

Figure 4.7: Tracking performance of a simple PID controller in the presence of wind.

Though an open loop controller like this cannot be used by itself for long periods of time, many feed-forward and model predictive controllers will use this type of open loop control over very short periods of time. By re-planning optimal paths quickly and adjusting the control accordingly, a degree of disturbance rejection can be achieved.

4.2.2 Simple PID Control

This PID tracking controller is presented here to provide context to sensor tracking problem. It is a simple waypoint tracking controller that demonstrates the potential down fall of tracking an aircraft path without regard for the tracking error of the sensor footprint. As such it also represents an upper bound on the tolerable error of the other tracking controllers presented here.

This controller uses closed PID loop control to steer the aircraft towards a waypoint ahead of the UAV, along the desired aircraft path. This type of control is analogous to steering the UAV with a carrot and a stick. Figure 4.8 gives the physical interpretation of the error shown in equation 4.14 that the PID controller attempts to drive to zero.

$$e = \psi_d - \psi \tag{4.14}$$

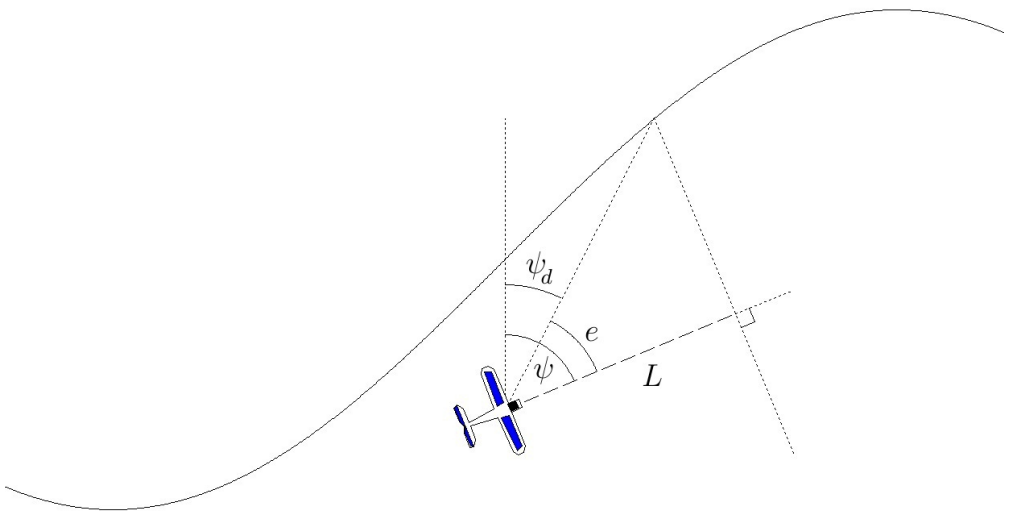

Figure 4.8: A graphical depiction of the heading error associated with the PID controller.

Given a predefined *lookahead distance*, L, the controller defines a waypoint along the aircraft path that intersects a line that is L meters ahead of the UAV and perpendicular to its heading. Variations of this type of control have better performance, but suffer from similar degradation interns of sensor tracking performance. One proposed variation uses an adaptive lookahead distance[25]. While others propose fitting a connecting curve from the aircraft's current state to a lookahead point on the desired aircraft path, and then tracking that connecting curve until the aircraft path is reached[25][17][37][2].

Though this is a relatively simple controller, waypoint navigation is the primary tracking controller for many small and inexpensive UAV autopilots[9][10][32]. This method of tracking control merely replicates an aircraft path defined by densely spaced waypoints, with a pre-turn radius equal to L.

Performance Analysis

Since the PID controller only responds when there is an error, as figure 4.9 shows, this type of control will always lag behind the desired path. The PID controller has no knowledge of the future path, but as the error grows from zero, the derivative element of the control attempts to minimize the 'predicted' future error by applying control based on the growth of the error.

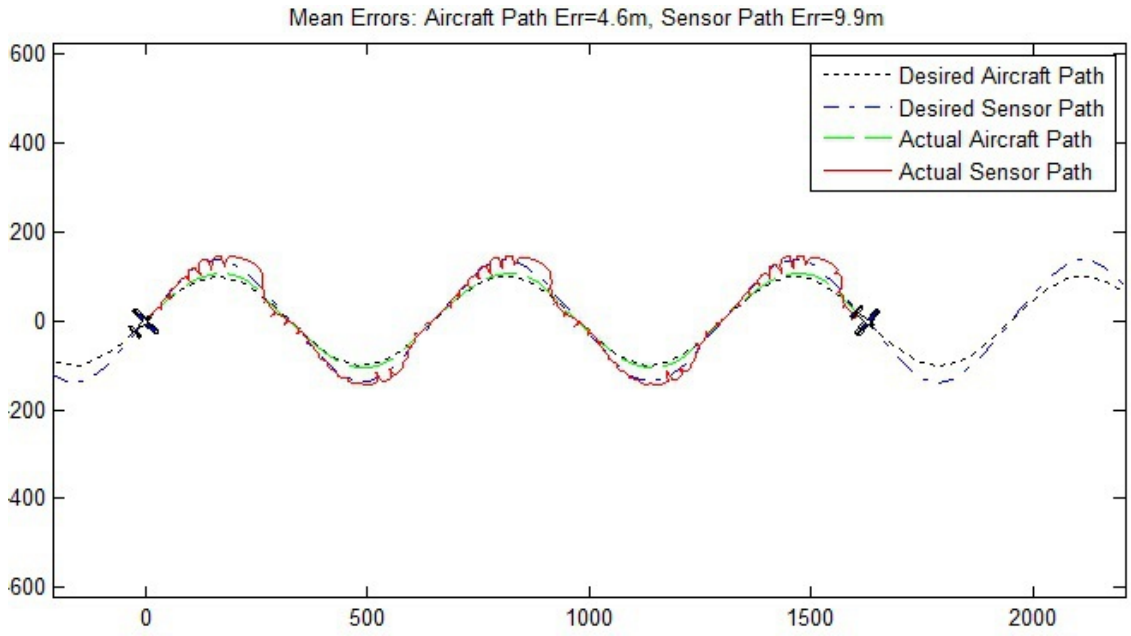

Figure 4.9: Steady state tracking performance of a simple PID controller.

The mean steady state cross-track error of the aircraft path is not bad, and even the cross-track error of the sensor path is not terrible, but the quality of the tracking performance is not good. Qualitatively, the target path is continuously oscillating with respect to the center of the sensor footprint. Quantitatively, though the average cross-track error of the sensor footprint is only $9.9m$, the standard deviation of that error is 6.5 m^2, meaning that it swings wildly about the target path. Those oscillation would not only make it more difficult to correlate images from one frame to another, it would also significantly degrade the image quality causing blur every time the camera capture an image while the aircraft is rolling.

Figure 4.10 gives an idea of the transient response of the PID controller. The transient respond is largely effected by lookahead distance of the controller. As the look ahead distance increases, the settling time of the cross-track error increases, but the oscillations about aircraft path decreases. Essentially a longer lookahead distance eases the UAV into the aircraft path. Unfortunately, a longer look ahead distance also increases the steady state cross-track error. The UAV is always trying to get to the waypoint at the end of its lookahead distance, if that way point is close to the UAV, then flying towards that waypoint also minimizes the cross-track error. If the point is far away, the UAV will potentially fly away from the aircraft path in order to fly towards the waypoint.

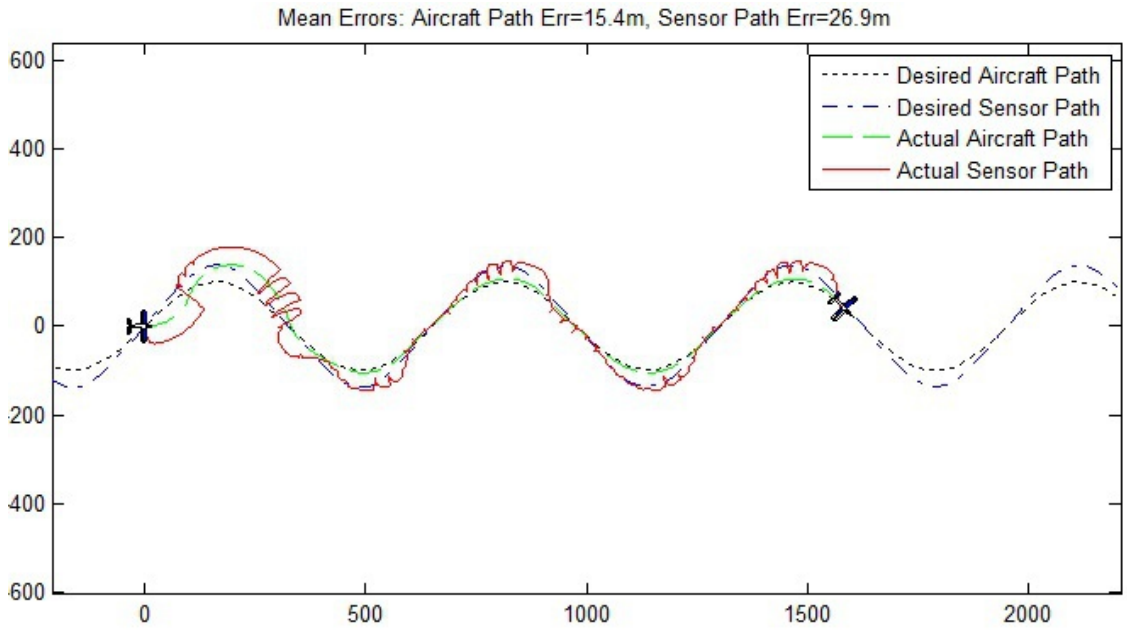

Figure 4.10: Transient response of a simple PID controller.

Another important thing to note about the transient response of the PID controller, is that during the controller's settling time, it does nothing to minimise the sensor path error. In other words, if the desired sensor path was updated with any frequency, this controller would have a very difficult time tracking it.

Finally, figure 4.11 shows the response of the PID tracking controller when it is given a step input disturbance. Initially the UAV is at steady state, but at the beginning of this simulation a $10\frac{m}{s}$ wind to the north is introduced.

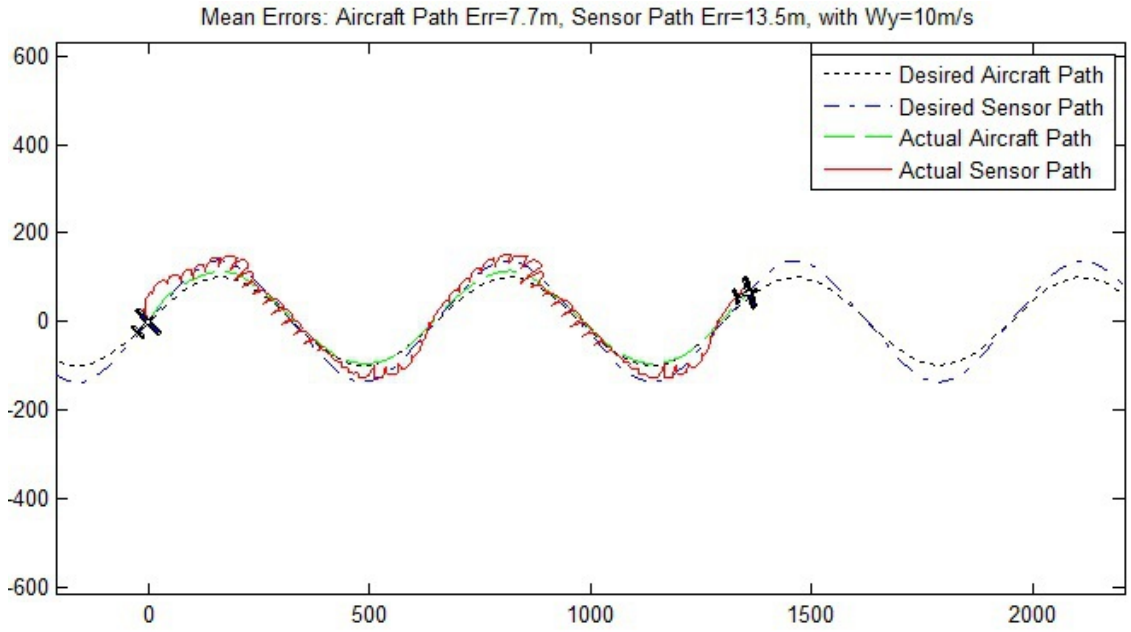

Figure 4.11: Disturbance response of a simple PID controller.

4.2.3 Spatial Sliding Mode

Sliding mode control offers a robust method of controlling nonlinear systems despite bounded model error, and uncertainties. Unlike the PID controller presented in section 4.2.2, sliding mode controllers use information about the whole path, in the form of derivatives, in order to theoretically achieve perfect tracking. More about sliding mode control can be found here [47]

Sliding mode controllers have been used in a variety of UAV tracking applications [41][46], and a detail analysis of sliding mode controllers applied to UAVs can be found here [33]. Additionally, in [33] McGee introduces a novel spatial sliding mode controller for UAVs. By reformulating the sliding mode controller with regard to a spatial, versus temporal, coordinate, the controller avoids potential singularities that make the UAV controller unstable.

As with traditional temporal sliding mode controllers, spatial sliding mode controllers can also theoretically achieve perfect tracking. But, perfect tracking requires very active control, which for UAVs means high frequency roll maneuvers. Not only would this highly active control be bad for sensor tracking, it would certainly fatigue the UAVs airframe, and possibly excite some unmodel dynamics of either airframe itself or the low level controllers. In order to avoid that type of high frequency control, a *boundary layer*, Φ, is added around the sliding surface. Inside the boundary layer the control is an affine function with respect to the sliding surface, s, instead of the step function, $\text{sgn}(s)$.

The spacial sliding mode (SSM) controller presented here performs very well. The steady state cross-track of both the aircraft path and the sensor path is less than a meter. Unfortunately, just as with the feedback linearisation controller, if the SSM controller is used to track the sensor path directly, the internal dynamics cause the system to become unstable. Nevertheless, as the PID controller was presented to give context to the sensor tracking problem by demonstrating the greatest tolerable error, the SSM controller is presented here not as a lower bound, but as an example of very good tracking control. Then section 4.3 will present a novel non-linear model predictive controller that is capable of directly tracking the sensor path with nearly the same performance.

Spatial Sliding Mode Formulation

The formulation of the SSM controller requires that the desired aircraft path must be a continuous function, $y^d = f(x)$, with continuous first and second derivatives, but, in practice, if the desired path does not fit those requirements, a local function that does have continuous first and second derivatives can be used to approximate the desired path [33].

Sliding mode control is acheived by defining a sliding surface s such that,

$$ss' \leq -\eta|s| \tag{4.15}$$

Where s' represents the spatial derivative of s with respect to x. In other words,

$$s' = \frac{ds}{dx} \tag{4.16}$$

Rearranging the sliding condition in equation 4.15 give the following result.

$$\begin{aligned} s' &\leq -\eta\frac{|s|}{s} \\ s' &\leq -\eta\,\mathrm{sgn}\,(s) \end{aligned} \tag{4.17}$$

If s is chosen such that equation 4.17 is true at all times, then the function s will be driven to the invariant set $s = 0$ in finite time. By making s a function of the error dynamics of the system, then once the s has been driven to 0, the error will also go to zero according to the chosen error dynamics. By defining s such that,

$$s(\mathbf{x}, u) = (\frac{d}{dx} + \lambda)^{r-1} e \tag{4.18}$$

where r is the relative degree of the system, then choosing u such that equation 4.17 is true implies that $s \to 0$ and $e \to 0$.

Given an error function $e = y - y^d = y - f(x)$, in order to implement the sliding controller, the sliding surface must be formulated such that the control input, u appears in the first spacial derivative of s, s'. Note that $'$ denotes the derivative of s with respect to x.

If the UAV is modelled as a unicycle within a wind frame, then the dynamics of the UAV with respect to time are, given by the following equations.

$$\begin{aligned} \dot{x} &= V\cos\psi + W_x \\ \dot{y} &= V\sin\psi + W_y \\ \dot{\psi} &= u \end{aligned} \tag{4.19}$$

Then by the chain rule,

$$\frac{dy}{dx} = \frac{dy}{dt}\frac{dt}{dx} = \frac{\frac{dy}{dt}}{\frac{dx}{dt}} = \frac{\dot{y}}{\dot{x}} \tag{4.20}$$

Then it is easy to see that the dynamics of the UAV with respect to the spacial coordinate x are,

$$\begin{aligned} x' &= 1 \\ y' &= \frac{V\sin\psi + W_y}{V\cos\psi + W_x} \\ \psi' &= \frac{u}{V\cos\psi + W_x} \end{aligned} \tag{4.21}$$

From equations 4.21 is apparent that the system has a relative degree of 2, which means that the sliding surface should be a first order model.

$$s = e' + \lambda e \tag{4.22}$$

So that the derivative of s is,

$$s' = e'' + \lambda e' \tag{4.23}$$

Substituting $e = y - f(t)$ into equations 4.22 and 4.23 yeilds,

$$\begin{aligned} s &= (y' - f'(x)) + \lambda\,(y - f(x)) \\ s' &= (y'' - f''(x)) + \lambda\,(y' - f'(x)) \end{aligned} \tag{4.24}$$

The control term is in the expansion of y''.

$$
\begin{aligned}
y'' &= \frac{(V\sin\psi + W_y)'(V\cos\psi + W_x) - (V\cos\psi + W_x)'(V\sin\psi + W_y)}{(V\cos\psi + W_x)^2} \\
y'' &= \frac{(V\psi'\cos\psi + W_y')(V\cos\psi + W_x) - (-V\psi'\sin\psi + W_x')(V\sin\psi + W_y)}{(V\cos\psi + W_x)^2} \\
y'' &= \frac{(V\psi'\cos\psi + W_y')\dot{x} - (-V\psi'\sin\psi + W_x')\dot{y}}{\dot{x}^2} \\
y'' &= \frac{\frac{1}{\dot{x}}(V\dot{\psi}\cos\psi + \dot{W}_y)\dot{x} - \frac{1}{\dot{x}}(-V\dot{\psi}\sin\psi + \dot{W}_x)\dot{y}}{\dot{x}^2} \\
y'' &= \frac{(Vu\cos\psi + \dot{W}_y)\dot{x} - (-Vu\sin\psi + \dot{W}_x)\dot{y}}{\dot{x}^3} \\
y'' &= \frac{(V\dot{x}\cos\psi + V\dot{y}\sin\psi)u + (\dot{x}\dot{W}_y - \dot{y}\dot{W}_x)}{\dot{x}^3} \\
y'' &= au + b
\end{aligned} \tag{4.25}
$$

The term b accounts for the rate of change of the wind frame.

$$b = \frac{\dot{x}\dot{W}_y - \dot{y}\dot{W}_x}{\dot{x}^3} \tag{4.26}$$

Simplifying a yields,

$$
\begin{aligned}
a &= \frac{(V\dot{x}\cos\psi + V\dot{y}\sin\psi)}{\dot{x}^3} \\
a &= \frac{Vcos\psi(V\cos\psi + W_x) + V\sin\psi(V\sin\psi + W_y)}{\dot{x}^3} \\
a &= \frac{V^2\cos^2\psi + Vcos\psi W_x + V^2\sin^2\psi + V\sin\psi W_y}{\dot{x}^3} \\
a &= \frac{V^2(\cos^2\psi + \sin^2\psi) + \dot{x}W_x - W_x^2 + \dot{y}W_y - W_y^2}{\dot{x}^3} \\
a &= \frac{V^2 + \dot{x}W_x + \dot{y}W_y - (W_x^2 + W_y^2)}{\dot{x}^3} \\
a &= \frac{V^2 + \dot{x}W_x + \dot{y}W_y - W^2}{\dot{x}^3}
\end{aligned}
$$

Where $W^2 = W_x^2 + W_y^2$. Then noting that the velocity of the UAV with respect to the ground is given by,

$$\begin{aligned}
V_g^2 &= \dot{x}^2 + \dot{y}^2 \\
V_g^2 &= (V\cos\psi + W_x)(V\cos\psi + W_x) + (V\sin\psi + W_y)(V\sin\psi + W_y) \\
V_g^2 &= V^2(\cos^2\psi + \sin^2\psi) + 2V\cos\psi W_x + 2V\sin\psi W_y + W_x^2 + W_y^2 \\
V_g^2 &= V^2 + 2V\cos\psi W_x + 2V\sin\psi W_y + W^2 \\
V_g^2 &= V^2 + 2(\dot{x}W_x - W_x^2) + 2(\dot{y}W_y - W_y^2) + W^2 \\
V_g^2 &= V^2 + 2\dot{x}W_x + 2\dot{y}W_y - W^2
\end{aligned}$$

Then it is clear that,

$$a = \frac{V_g^2 + V^2 - W^2}{2\dot{x}^3} \tag{4.27}$$

Therefore the derivative of the sliding surface is,

$$s' = au + b + c$$

Where,

$$c = -f''(x) + \lambda(y' - f'(x)) \tag{4.28}$$

In practice, the actual wind values, $(W_x, W_y, \dot{W}_x, \dot{W}_y)$, are unknown, but the controller does have access to an estimate of the their mean values, $(\bar{W}_x, \bar{W}_y, \bar{\dot{W}}_x, \bar{\dot{W}}_y)$. Assuming that the error of the estimate is bounded such that $|W_x - \bar{W}_x| \leq \delta W_x$, then it is still possible to design a controller that is capable of perfect tracking despite wind and wind gusts, as long as the disturbances are bounded. The values of a, b and c based on the wind estimates and differentiated GPS values, $(\dot{x}, \dot{y})$ are,

$$\begin{aligned}
\bar{a} &= \frac{V_g^2 + V^2 - \bar{W}^2}{2\dot{x}^3} \\
\bar{b} &= \frac{\dot{x}\bar{\dot{W}}_y - \dot{y}\bar{\dot{W}}_x}{\dot{x}^3} \\
c &= -f''(x) + \lambda(\frac{\dot{y}}{\dot{x}} - f'(x))
\end{aligned} \tag{4.29}$$

Substituting s' into the sliding condition given in 4.17 yields,

$$au + b + c \leq -\eta\,\mathrm{sgn}\,(s) \tag{4.30}$$

Given the form of s', a natural choice for u is,

$$u = \frac{-\bar{b} - c - K\,\mathrm{sgn}\,(s)}{\bar{a}} \tag{4.31}$$

Substituting 4.31 into 4.30 yeilds,

$$\begin{aligned}
a\left(\frac{-\bar{b} - c - K\,\mathrm{sgn}\,(s)}{\bar{a}}\right) + b + c &\leq -\eta\,\mathrm{sgn}\,(s) \\
-\frac{a}{\bar{a}}\bar{b} - \frac{a}{\bar{a}}c - \frac{a}{\bar{a}}K\,\mathrm{sgn}\,(s) + b + c &\leq -\eta\,\mathrm{sgn}\,(s) \\
\left(1 - \frac{a}{\bar{a}}\right)(\bar{b} - c) + (b - \bar{b}) - \frac{a}{\bar{a}}K\,\mathrm{sgn}\,(s) &\leq -\eta\,\mathrm{sgn}\,(s) \\
\left(1 - \frac{a}{\bar{a}}\right)(\bar{b} - c) + \delta b - \frac{a}{\bar{a}}K\,\mathrm{sgn}\,(s) &\leq -\eta\,\mathrm{sgn}\,(s)
\end{aligned}$$

Then the sliding condition will be satisfied if K is chosen such that,

$$\begin{aligned}
K &= \max\left[\frac{\bar{a}}{a}\left(\left(1 - \frac{a}{\bar{a}}\right)(\bar{b} - c) + \delta b + \eta\right)\right] \\
K &= \max\left[\left(\frac{\bar{a}}{a} - 1\right)(\bar{b} - c) + \frac{\bar{a}}{a}(\delta b + \eta)\right]
\end{aligned}$$

And, since,

$$\max\left[\left(\frac{\bar{a}}{a} - 1\right)(\bar{b} - c) + \frac{\bar{a}}{a}(\delta b + \eta)\right] \leq \max\left[\left(\frac{\bar{a}}{a} - 1\right)(\bar{b} - c)\right] + \max\left[\frac{\bar{a}}{a}(\delta b + \eta)\right]$$

Then,

$$K = \max\left[\left(\frac{\bar{a}}{a} - 1\right)(\bar{b} - c)\right] + \max\left[\frac{\bar{a}}{a}(\delta b + \eta)\right]$$

will also satisfy the sliding condition in equation 4.30.
If wind is quasi-static, meaning $\dot{\bar{W}}^2 = 0$, and $\bar{b} = 0$, then the control law becomes,

$$K = \max\left[c\left(\left(\frac{\bar{a}}{a}\right)_{max} - 1\right), c\left(1 - \left(\frac{\bar{a}}{a}\right)_{min}\right)\right] + \left(\frac{\bar{a}}{a}\right)_{max}(\delta b_{max} + \eta) \tag{4.32}$$

Were,

$$\frac{\bar{a}}{a} = \frac{V_g^2 + V^2 - \bar{W}^2}{V_g^2 + V^2 - W^2}$$

and,

$$\begin{aligned}
\left(\frac{\bar{a}}{a}\right)_{max} &= \frac{V_g^2 + V^2 - (W^2 + 2W\delta W + \delta W^2)}{V_g^2 + V^2 - W^2} \\
\left(\frac{\bar{a}}{a}\right)_{max} &= 1 + \frac{V_g^2 + V^2 + 2W\delta W - \delta W^2}{V_g^2 + V^2 - (\bar{W} + \delta W \operatorname{sgn}\left(\bar{W}\right))^2} \\
\left(\frac{\bar{a}}{a}\right)_{max} &= 1 + \frac{V_g^2 + V^2 + 2W\delta W}{V_g^2 + V^2 - (\bar{W} + \delta W \operatorname{sgn}\left(\bar{W}\right))^2} > 1
\end{aligned} \tag{4.33}$$

and,

$$\left(\frac{\bar{a}}{a}\right)_{min} = \frac{V_g^2 + V^2 - \bar{W}^2}{V_g^2 + V^2} < 1$$

Given these values for K the sliding condition will always be satisfied, which means $s \to 0$, $e \to 0$ and the controller will theoretically achieve perfect tracking.

Performance Analysis

Figure 4.12 shows the effect of the discontinuity of the function $\operatorname{sgn}(() s)$. As expected, it causes aggressive, high frequency control action once the UAV has reached the invariant set $s = 0$.

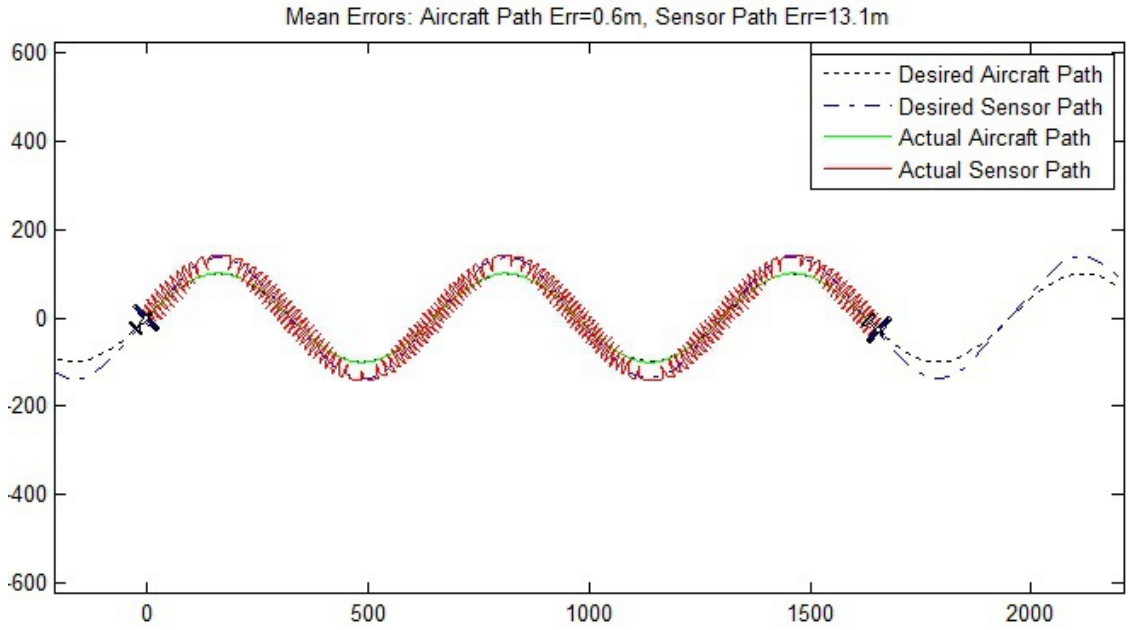

Figure 4.12: Steady state tracking performance of a spatial sliding mode controller.

Normally high frequency control action, called *chatter*, like that shown in figure 4.12 would be bad simply because it would fatigue the aircraft, and possibly excite unmodelled dynamics, but in this case the chatter exemplifies how tracking the aircraft path does not necessarily imply that the sensor path is also tracked. Essentially, figure 4.12 is the opposite of the example shown in figure 4.5.

Fortunately, chatter is a well know limitation of sliding mode controllers, with well known solutions. To avoid chatter about the sliding surface, the function $\text{sgn}(s)$ in the controller is replaced with the function $\text{sat}\left(\frac{s}{\Phi}\right)$. Where,

$$\text{sat}\left(\frac{s}{\Phi}\right) = \begin{cases} \frac{s}{\Phi} & \text{if } \left|\frac{s}{\Phi}\right| \leq 1 \\ \text{sgn}\left(\frac{s}{\Phi}\right) & \text{otherwise} \end{cases} \tag{4.34}$$

Using the function $\text{sat}\left(\frac{s}{\Phi}\right)$ instead of $\text{sgn}(s)$ will significantly reduce chatter as Φ increases, but the controller is no longer guaranteed to drive $s \to 0$, which means that there error no longer guaranteed to go to zero either. Instead, the controller guarantees to drive s to an invariant set described as a boundary layer around $s = 0$. Correspondingly, the error is driven to within a maximum tolerance, ϵ, away from $e = 0$.

Figure 4.13 shows the steady state tracking performance of a SSM controller with a well tuned boundary layer.

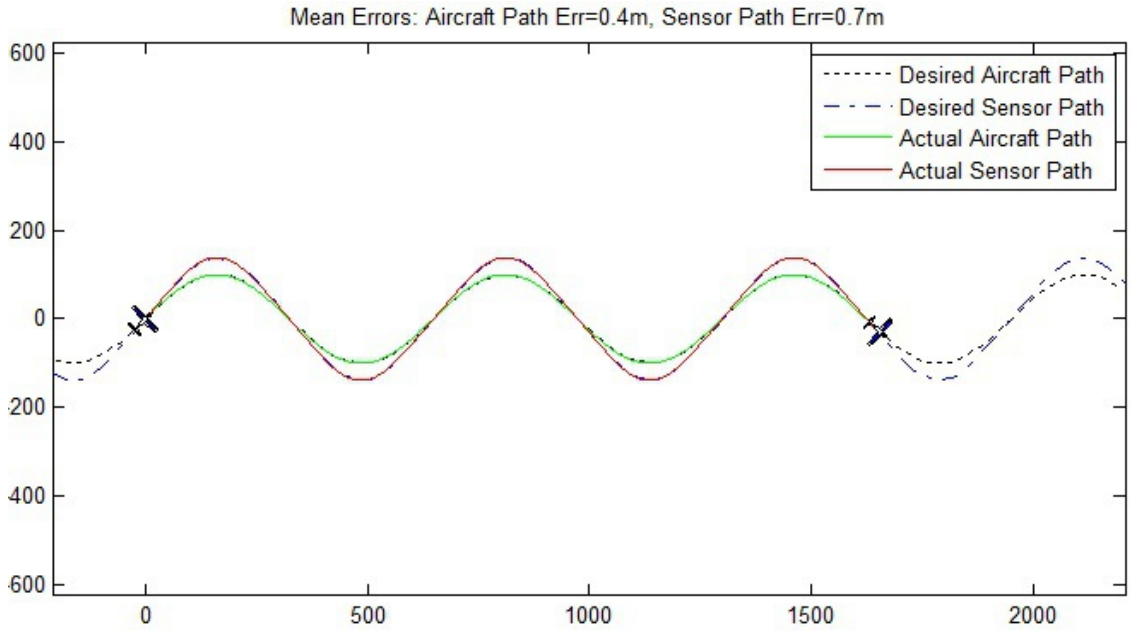

Figure 4.13: Steady state tracking performance of a spatial sliding mode controller with a boundary layer.

As expected, the tracking performance of this controller is superb. In perfect conditions, the sensor tracking performance is only slightly worse than the aircraft tracking performance. Once again, this is the type of performance the controller presented in section 4.3 will attempt to achieve, but without the need of having a well defined aircraft path.

Figure 4.14 shows the transient response of the SSM tracking controller. While its performance is still good, it is worth pointing out that relative to the steady state tracking performance the sensor tracking performance is 13% worse when compared to the aircraft tracking performance. This is as expected, since good sensor tracking performance is not the controller's goal, but an ancillary effect of good aircraft tracking performance.

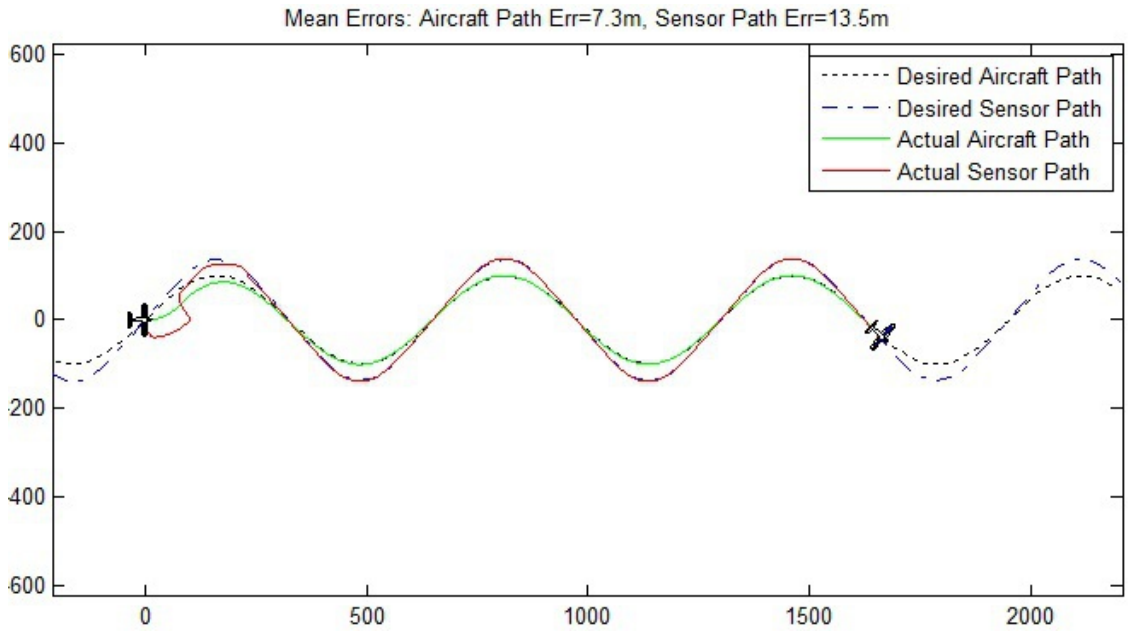

Figure 4.14: Transient response of a spatial sliding mode controller.

Finally, figure 4.15 shows the response of the SSM controller to a step disturbance, which is once again a $10\frac{m}{s}$ wind toward the north.

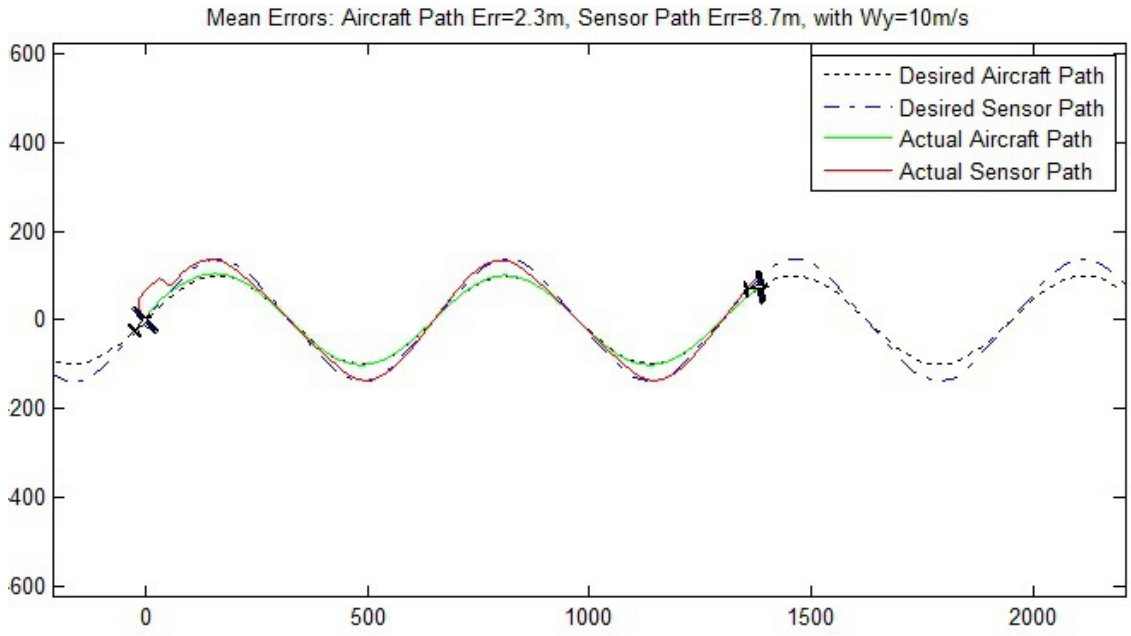

Figure 4.15: Disturbance response of a spatial sliding mode controller.

In this case, it is quite obvious that the tracking controller's goal is to minimize the cross-track error of the aircraft path, without regard for the sensor path. Where in the previous examples the sensor cross-track error was comparable to the cross-track error of the aircraft path, in this case it is several times worse.

Improving the tracking performance of the sensor is one of the main goals of the controller presented in the section 4.3. By attempting to directly control the sensor footprint to track a desired sensor path, the controller aims to reverse the performance degradation seen in figure 4.15. Meaning that as the aircraft is disturbed, the controller will attempt to minimize the cross-track error of the sensor path, without regard for the cross-track error of the aircraft path.

4.3 Direct Control of a UAV's Sensor Footprint

The controller presented here takes a novel approach to solving the sensor tracking problem described in section 4.1.2. Rather than redefining the target sensor path as an aircraft path and then using a traditional tracking controller to track that aircraft path, this controller directly tracks the sensor path removing the need for a translation step and improving the sensor tracking performance in the presence of disturbances.

In a very general sense, this controller could be classified as a receding horizon, nonlinear model predicitive controller(NMPC)[40]. It uses a model of a UAV to integrate forward in

time to a limited horizon, applies a cost function to find a near optimal path, and then uses the initial control law from that path as the control input, until is able to compute a new optimal path. The main challenge of controlling the sensor footprint is that the difficulty of finding an optimal path is superseded by the difficulty of finding even a feasible path, where a feasible path is defined as one that keeps the desired sensor path within the UAVs sensor footprint. The difficult of finding a feasible path was discussed previously in section 3.4.

To solve the tracking problem this NMPC controller first attempts to find a set of feasible paths and then from that set of feasible paths, chooses the path that is closest to the optimal path. The controller finds feasible paths by applying the RRT with iterative Gaussian sampling path planner that was discussed in section 3.5.1.

4.3.1 Formulating an Optimal Control Problem

The optimal control, $\mathbf{u}^o$, is defined as the set of controls that uses the least control effort necessary to keep the desired sensor target as near to the center of a sensors field of view as possible. The optimal controller will find a sequence of controls that keeps the desired sensor target within the UAV's field of view at all times, but if that is not possible, it will always find a path that steers the UAV towards a configuration that will eventually put the target with in the UAV's field of view, if such a solution exists.

The trajectory of the aircraft, and its sensor footprint, is parametrized by the control sequence,

$$\mathbf{u} = [u_1 u_2 \dots u_\infty] \tag{4.35}$$

where each element of $\mathbf{u}$ is from the set of desired turn rates,

$$u_i \in [\dot{\psi}^d_{min}, \dot{\psi}^d_{max}]$$

Then, given a model of the aircraft of the form,

$$\dot{\mathbf{x}} = f(\mathbf{x}, u(t))$$

.

the trajectory of the aircraft $\mathbf{X}$ is defined as

$$\mathbf{X} = [\mathbf{x}_1 \dots \mathbf{x}_n \dots \mathbf{x}_{inf}]$$

where,

$$\mathbf{x}_i = \sum_{k=1}^{i} \int_{(k-1)\Delta t}^{k\Delta t} f(\mathbf{x}_0, u_k) \mathrm{d}t \tag{4.36}$$

The cost associated with any trajectory $\mathbf{X}$ is defined as,

$$J(\mathbf{X}) = \sum_{i=1}^{\infty} g(\mathbf{x}_i, u_i, u_{i-1}) \tag{4.37}$$

where $g(\mathbf{x}_i, u_i, u_{i-1})$ is the stage cost at each step.

The stage cost is a function of the distance of the center of the sensor footprint from the desired sensor path, and the control effort required to get to that point. The sensor error at each stage, e_t, is the square of the perpendicular distance from the desired ground path to the center of the sensor footprint.

$$e_t^2 = \min_{\mathbf{x}_d^s} \left\| \begin{bmatrix} x_t^s \\ y_t^s \end{bmatrix} - \begin{bmatrix} x_d^s \\ y_d^s \end{bmatrix} \right\|_2^2 \tag{4.38}$$

where x_t^s and y_t^s are defined the same as they were in equation 4.3, which in vector form is,

$$\begin{bmatrix} x_t^s \\ y_t^s \end{bmatrix} = \begin{bmatrix} x_t + C\dot{\psi}_t \sin\psi_t \\ y_t - C\dot{\psi}_t \cos\psi_t \end{bmatrix} \tag{4.39}$$

Figure 4.16 is a graphical representation of the stage cost due to sensor error at every stage.

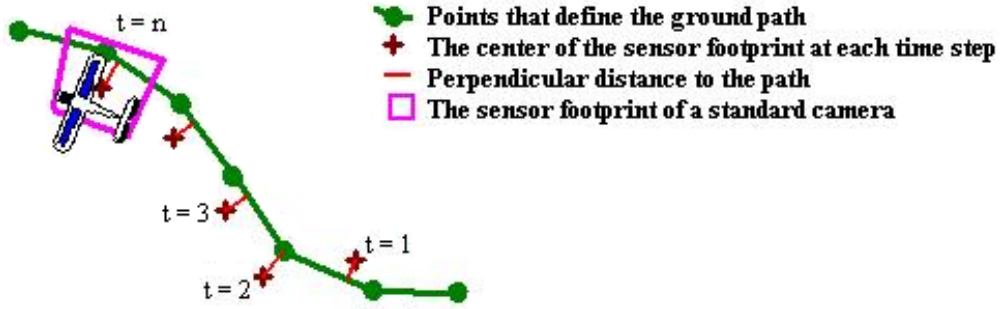

Figure 4.16: A depiction of the cost associated with the sensor footprint. The total cost associated with the sensor footprint is the sum of the distance, e_t, squared at every time step.

The second term in the stage cost, $g(\mathbf{x}_i, u_i, u_{i-1})$, acts to dampen the control effort, effectively smoothing the optimal path. The cost at each stage due to the control effort is

$$e_t^2 (u_t - u_{t-1})^2 \tag{4.40}$$

Without it, the aircraft will constantly change its turn-rate for a negligible increase in

its tracking performance. Constantly changing its turn-rate causes the aircraft to roll back and forth, and provide poor surveillance data. When the aircraft is far from the desired ground path, small differences in the UAV's roll angle do very little to improve the tracking performance. But, when the aircraft is directly over the desired ground path, the UAV tracks the desired path primarily by changing its roll angle. Therefore the difference in control action between every time step is weighted by the aircraft's distance from the desired path. Putting together both terms of the stage cost gives

$$g(\mathbf{x}_i, u_i, u_{i-1}) = e_t^2 \left(1 + (u_t - u_{t-1})^2\right) \tag{4.41}$$

Since, the goal of the controller is to place the center of the UAV's sensor footprint on the desired sensor target, if it accomplishes that, then the stage cost goes to zero. So, even though the summation within the trajectory cost, $J(\mathbf{X})$ goes to infinity, assuming that a feasible path exist, the cost of the optimal path is finite, because the stage cost becomes zero.

4.3.2 Approximating the Optimal Solution in Real-time

Since the aircraft may be very far from placing its sensor footprint on the desired sensor path, and because the wind disturbances are not well modelled, it would be inefficient to spend resources calculating the exact cost of each feasible solution. Instead, the controller uses a receding horizon approach that calculates the exact cost of the first n steps of each feasible solution, and then calculates an estimated cost for the remainder of each solution.

Each potential aircraft path is therefore parametrised by the finite vector,

$$\mathbf{u} = [u_1 u_2 \dots u_n] \tag{4.42}$$

Where n is the number of steps in the receding horizon. Since, each element of the control sequence is integrated over a fixed time step, Δt, the receding horizon plans $n\Delta t$ seconds into the future. The cost of the remainder of each path, the *cost-to-go*, is estimated by the infinite horizon heuristic $h(\mathbf{x}_n)$, which is a function of the final state of the UAV at the end of the receding horizon. Therefore, the estimated cost of the UAV's trajectory is calculated by the heuristic function,

$$H(\mathbf{X}) = \sum_{i=1}^{n} g(\mathbf{x}_i, \mathbf{u}) + h(\mathbf{x}_n) \tag{4.43}$$

Unfortunately, finding the optimal solution to the sensor tracking problem is even more difficult than finding a feasible solution to the path planning problem discussed in section 3.5, which was already NP-hard. Rather than directly trying to optimize the UAV's trajectory, the NMPC uses the RRT iterative Gaussian sampling path planner to find a set of feasible paths, and then chooses the path with the lowest cost. The exact NMPC algorithm is as follows.

Algorithm 3 NMPC using an RRT with Itereative Gaussian Sampling

procedure IGSRRT_NMPC($x_{\text{init}}, \mathbf{u}_{last}, K, \Delta t$)
  $\mathbf{u}_{best} \leftarrow \mathbf{u}_{inti}$
  $H_{best} \leftarrow$ **PATH_COST**($\mathbf{x}_{inti}, \mathbf{u}_{best}$)
  for k=1 to K do
    $\mathbf{u}_{logic} \leftarrow$ **LOGICAL_CONTROL**($\mathbf{u}_{best}$);
    $H_{rand} \leftarrow$ **PATH_COST**($\mathbf{x}_{init}, \mathbf{u}_{logic}$)
    if $H_{logic} < H_{best}$ **then**
      $\mathbf{u}_{best} \leftarrow \mathbf{u}_{logic}$
    end if
  end for
  for k=1 to K do
    $\mathbf{u}_{rand} \leftarrow$ **RANDOM_CONTROL**($\mathbf{u}_{best}$);
    $H_{rand} \leftarrow$ **PATH_COST**($\mathbf{x}_{init}, \mathbf{u}_{rand}$)
    if $H_{rand} < H_{best}$ **then**
      $\mathbf{u}_{best} \leftarrow \mathbf{u}_{rand}$
    end if
  end for
  return $u_1 \in \mathbf{u}_{best}$
end procedure

In the IGSRRT_NMPC algorithm, the PATH_COST() function integrates the control sequence $\mathbf{u}$ in order to generate $\mathbf{X}$, which it uses to calculate the heuristic cost $H(\mathbb{X})$. The first for-loop in the algorithm cycles though a sequence of logical control laws that often times generate trajectory with the lowest cost, but are unlikely to be generated by random sampling. They also do a very rough job of sampling uniformly across the configuration space, which helps the controller avoid local minimums. Figure 4.17 shows a sample of the logical paths that IGSRRT_NMPC controller generates.

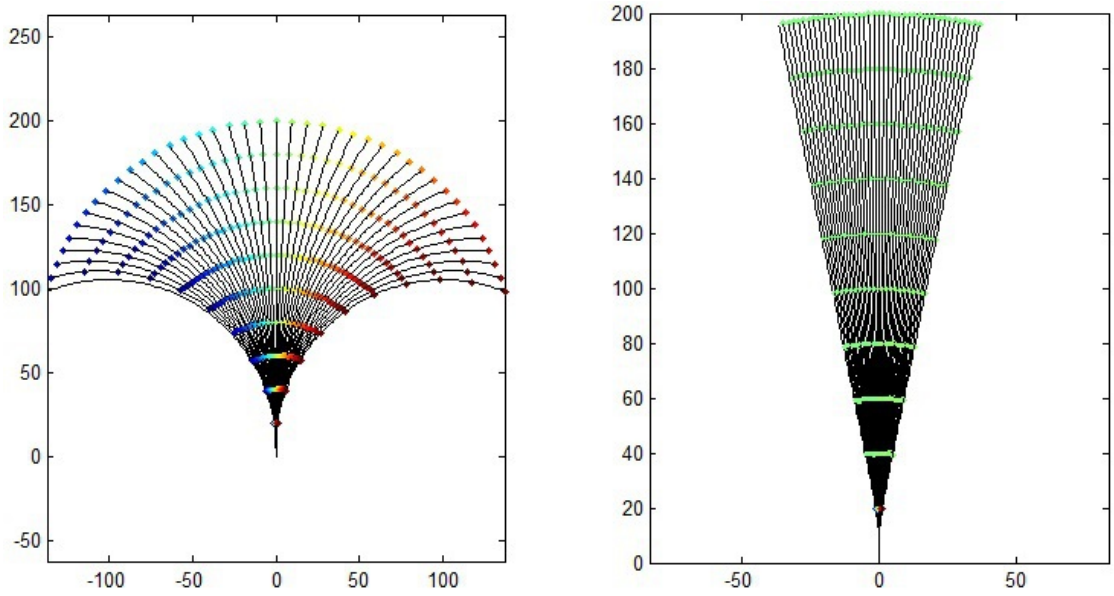

Figure 4.17: A sample of the set of logical paths generated by the IGSRRT_NMPC controller.

By including a set of logical control vectors, the controller produces smoother trajectories when tracking straight paths and orbits, for which the optimal aircraft trajectories are lines and circles.

The IGSRRT_NMPC is initiated with the best control sequence from the previous iteration. So, not only does the IGSRRT iterate through out its runtime, causing it to converge to paths with lower and lower cost, the controller also continues to iterate to a lower cost solution every time it is called. Essentially, as the frequency of the controller increases, the number of iterations of the IGSRRT path planner also increases. Similar to the RRT-Anytime algorith [44], this controller will essentially continue to build a single IGSRRT that continuously increases its density while updating the control law any time it finds a more optimal path.

The choice of the heuristic cost function, $h(\mathbf{x})$, plays a large role in the run time efficiency and stability of the IGSRRT_NMPC controller. Because the controller only cycles through a fixed number of iterations each time it is called, and the number of computations that are made in each iteration is always the same, the controller will always generate a control output in a fixed finite time. Assuming that the algorithm runs sufficiently quickly, it meets the requirements of being used as a real-time controller[18]. Of course, running quickly means having a fast and simple heuristic for calculating the cost-to go, but for stability, the heuristic function must also have atleast limited precision.

4.3.3 Stability Analysis

The stability of the NMPC tracking controllers is a complicated subject and is the subject of on going research [43][40][8]. The first question is what does stability mean in this context? Depending on the desired sensor path, it may not be dynamically feasible to track the sensor path with zero error. In fact, as pointed out here [47] the goal of a non-minimum phase tracking problem is not to reduce the error to zero, because in most cases that is not possible. Furthermore what does it mean for the controller to be unstable. The output of the controller in a desired turn-rate that is sent to a lower level turn-rate controller. It is assumed that the turn-rate controller is already bounded input, bounded output stable, therefore the IGSRRT_NMPC cannot cause the UAV to become dynamically unstable. The worst behaviour that the tracking controller can exhibit is to cause the cross-track error to go to infinity. So it makes sense that the stability of the IGSRRT_NMPC tracking controller should be measured by its ability to minimize the cross-track error.

Given that there is an optimal solution that does minimize the cross-track error, as described in section 4.3.1, a measure of the stability of the tracking controller should then be its ability to converge to that optimal solution. Given the probabilistic completeness of the IGSRRT algorithm if the controller used an infinite horizon path planner, it is clear that as the iterations of the IGSRRT path planner increases, the control output will eventually converge to the the optimal control law.

Theorem 4.3.1 *Given a smooth function, $J(\mathbf{X})$, that describes the optimal path, $\mathbf{X}^o$, such that,*

$$J(\mathbf{X}^o) < J(\mathbf{X}_i) \quad \forall \ \{\mathbf{X}_i : \mathbf{X}_i \neq \mathbf{X}^o\}$$

Then the lowest cost path generated by probabilistically complete path planner will asymptotically converge to the optimal solution as the number of samples taken by the path planner goes to infinity.

Proof 4.3.2 *Let $J(\mathbf{X})$ be a smooth function that describes the optimal path, $\mathbf{X}^o$ and the optimal cost $J^o(\mathbf{X}^o)$. Then there exists a bounded set of paths, $\mathbb{X}^\epsilon$ such that,*

$$J(\mathbf{X}_i) < J^o + \epsilon \quad \forall \ \mathbf{X}_i : \mathbf{X}_i \in \mathbb{X}^\epsilon$$

Then the set of paths, $\mathbb{X}^\epsilon$ defines a bounded volume in the configuration space of the vehicle. Because the path planner is probabilistically complete, the probability of randomly sampling a path in $\mathbb{X}^\epsilon$ goes to one as the number of samples goes to infinity.

Furthermore, there exists a set $\mathbb{X}^\delta \subset \mathbb{X}^\epsilon$ and a $\delta < \epsilon$ such that,

$$J(\mathbf{X}_i) < J^o + \delta < J^o + \epsilon \quad \forall \ \mathbf{X}_i : \mathbf{X}_i \in \mathbb{X}^\delta$$

Where the probability of a randomly sampled path being in $\mathbb{X}^\delta$ is greater than zero, because

$\mathbf{X}^o \in \mathbb{X}^\delta$, *but less than the probability of the path being in* $\mathbb{X}^\epsilon$.

$$\because P(\mathbf{X}_i \in \mathbb{X}^\delta) = P(\mathbf{X}_i \in \mathbb{X}^\delta | \mathbf{X}_i \in \mathbb{X}^\epsilon) P(\mathbf{X}_i \in \mathbb{X}^\epsilon) < P(\mathbf{X}_i \in \mathbb{X}^\epsilon)$$

Again, because the path planner is probabilistically complete the probability of sampling a path in $\mathbb{X}^\delta$ *goes to one as the number of samples goes to infinity. Therefore, for any path sampled in the configuration space that has a cost greater than the optimal cost by* ϵ, *as the number of samples goes to infinity, the path planner is probabilistically guaranteed to sample a lower cost path from the set of paths that have atleast* $\epsilon - \delta$ *lower cost.*

Stability of the Receding Horizon Controller

The stability of the NMPC controller must be discussed in terms of its convergence to the optimal path. The previous section showed that the infinite horizon IGSRRT path planner probabilistically and asymptomatic converges to the optimal path, but for the NMPC controller to converge to the optimal path, then it must do so using a finite horizon path planner. This section will show that if the infinite horizon estimator, $h(\mathbf{x})$, satisfies certain conditions, then the NMPC controller will converge to the optimal path to within a margin of error that is proportional to the error of the infinite horizon estimator.

Lemma 4.3.3 *If* $h(\mathbf{x})$ *has a single infimum, then the domain of,* $\{h(\mathbf{x}) : h(\mathbf{x}) \leq z\}$ *is not disjoint for any* $z > h_{min}$.

Proof 4.3.4 *Given a function,* $h(\mathbf{x})$, *with a single infimum, assume that the domain of* $\{h(\mathbf{x}) : h(\mathbf{x}) \leq z\}$ *is disjoint. Then there would be a infimum in each set in the domain of* $\{h(\mathbf{x}) : h(\mathbf{x}) \leq z\}$. *Since it is given that there is only one infimum, the assumption must be wrong. Therefore the domain must not be disjoint.*

Lemma 4.3.5 *If* $h(\mathbf{x})$ *has a single infimum,* $h_{inf} = h(\mathbf{x}_{inf})$, *and* $h(\mathbf{x})$ *is defined over the range* $\{h(\mathbf{x}) : h(\mathbf{x}) \leq z\}$, *then as* z *decreases,* ε *also decreases. where* ε *is defined as,*

$$\varepsilon = \max_{\mathbf{x}} \|\mathbf{x} - \mathbf{x}_{inf}\| \tag{4.44}$$

Proof 4.3.6 *Let* ε_z *be the the* ε *for some* z, *and let* $z_1 < z_2$. *Then, assume that* $\varepsilon_{z_1} > \varepsilon_{z_2}$. *Then there must exist some*

$$\mathbf{x}_1 = \underset{\mathbf{x},\, z_1}{\operatorname{argmax}} \|\mathbf{x} - \mathbf{x}_{inf}\| \tag{4.45}$$

such that, $\mathbf{x}_1 \in \{h(\mathbf{x}) : h(\mathbf{x}) \leq z_1\}$ *and* $\mathbf{x}_1 \notin \{h(\mathbf{x}) : h(\mathbf{x}) \leq z_2\}$. *But that can not be true since*

$$Domain\{h(\mathbf{x}) : h(\mathbf{x}) \leq z_1\} \subset Domain\{h(\mathbf{x}) : h(\mathbf{x}) \leq z_2\} \tag{4.46}$$

Therefore the assumption must be wrong, and $\varepsilon_{z_1} \leq \varepsilon_{z_2}$.

Theorem 4.3.7 *Given a smooth function, $J(\mathbf{X})$, that describes the optimal path, $\mathbf{X}^o$, such that,*

$$J(\mathbf{X}) = \sum_{k=1}^{\infty} g(\mathbf{x}_k)$$

And, a heuristic, $H(\mathbf{X})$,

$$H(\mathbf{X}) = \sum_{k=1}^{n} g(\mathbf{x}_k) + h(\mathbf{x}_n)$$

that estimates $J(\mathbf{X})$, such that,

$$h(\mathbf{x}_n^i) \geq \sum_{k=n+1}^{\infty} g(\mathbf{x}_k^i) \quad \forall\, \mathbf{X_i}$$

and that $h(x)$ has one single infimum such that,

$$h(\mathbf{x}_n^o) \leq h(\mathbf{x}_n^i) \Rightarrow \sum_{k=n+1}^{\infty} g(\mathbf{x}_k^o) \leq \sum_{k=n+1}^{\infty} g(\mathbf{x}_k^i)$$

Then the lowest cost path according to $H(\mathbf{X})$ generated by probabilistically complete path planner will asymptotically converge to within,

$$h_{err}(\mathbf{X}_i) = h(\mathbf{x}_n^i) - \sum_{k=n+1}^{\infty} g(\mathbf{x}_k^i)$$

of the optimal solution as the number of samples taken by the path planner goes to infinity.

Proof 4.3.8 *Let $h(x)$ be a heuristic with a single infimum such that,*

$$h(\mathbf{x}_n^o) \leq h(\mathbf{x}_n^i) \Rightarrow \sum_{k=n+1}^{\infty} g(\mathbf{x}_k^o) \leq \sum_{k=n+1}^{\infty} g(\mathbf{x}_k^i)$$

Then, there exists a bounded set of paths, $\mathbb{X}^\epsilon$ such that,

$$H(\mathbf{X}_i) < H(\mathbf{X}^o) + \epsilon \quad \forall\, \mathbf{X}_i : \mathbf{X}_i \in \mathbb{X}^\epsilon$$

where $\mathbf{X}^o$ is defined as,

$$J(\mathbf{X}^o) < J(\mathbf{X}_i) \quad \forall\, \{\mathbf{X}_i : \mathbf{X}_i \neq \mathbf{X}^o\}$$

Furthermore, since,

$$H(\mathbf{X}_i) = J(\mathbf{X}_i) + h(\mathbf{x}_n^i) - \sum_{k=n+1}^{\infty} g(\mathbf{x}_k^i) = J(\mathbf{X}_i) + h_{err}(\mathbf{X}_i)$$

then,

$$H(\mathbf{X}_i) < J^o + h_{err}(\mathbf{X}_i) + \epsilon \quad \forall\, \mathbf{X}_i : \mathbf{X}_i \in \mathbb{X}^\epsilon$$

Then the set of paths, $\mathbb{X}^\epsilon$ defines a bounded volume in the configuration space of the vehicle. Because the path planner is probabilistically complete, the probability of randomly sampling a path in $\mathbb{X}^\epsilon$ goes to one as the number of samples goes to infinity.

Furthermore, there exists a set $\mathbb{X}^\delta \subset \mathbb{X}^\epsilon$ and a $\delta < \epsilon$ such that,

$$H(\mathbf{X}_i) < J^o + h_{err}(\mathbf{X}_i) + \delta < J^o + h_{err}(\mathbf{X}_i) + \epsilon \quad \forall\, \mathbf{X}_i : \mathbf{X}_i \in \mathbb{X}^\delta$$

Where the probability of a randomly sampled path being in $\mathbb{X}^\delta$ is greater than zero as long as δ is greater than zero, but less than the probability of the path being in $\mathbb{X}^\epsilon$.

$$\because P(\mathbf{X}_i \in \mathbb{X}^\delta) = P(\mathbf{X}_i \in \mathbb{X}^\delta | \mathbf{X}_i \in \mathbb{X}^\epsilon) P(\mathbf{X}_i \in \mathbb{X}^\epsilon) < P(\mathbf{X}_i \in \mathbb{X}^\epsilon)$$

Again, because the path planner is probabilistically complete the probability of sampling a path in $\mathbb{X}^\delta$ goes to one as the number of samples goes to infinity. Therefore, for any path sampled in the configuration space that has a cost greater than the optimal cost by ϵ, as the number of samples goes to infinity, the path planner is probabilistically guaranteed to sample a lower cost path from the set of paths that have atleast $\epsilon - \delta$ lower cost. Therefore as δ goes to zero, $\mathbb{X}^\delta$ becomes the set of paths bounded by the error $J^o + h_{err}(\mathbf{X}_i)$, where $h_{err}(\mathbf{X}_i)$ decreases as $\mathbf{X}_i \to \mathbf{X}^o$.

4.3.4 Performance Analysis

The performance of the IGSRRT_NMPC controller is comparable to SSM controller, despite the fact that is it solving the non-minimum phase sensor tracking problem, rather than the minimum phase aircraft tracking problem. The IGSRRT_NMPC controller in these simulations uses a time step of $\Delta t = 1$s and a receding horizon of 15 steps. In other words, it calculates the exact modelled cost of the first 15 seconds, and then estimates the remainder of the infinite horizon cost.

Compared to the PID and SSM controllers, which ran at 30Hz, the IGSRRT_NMPC controller only runs at 10Hz. This is for two reasons. Firstly, the IGSRRT_NMPC is more computationally expensive to run. Though it does not require a second path planner to translate the sensor path into and optimal aircraft path, as the PID and SSM controllers require, running it at 30Hz would require a significant percentage of the processing power available to a small UAV. Secondly, unlike the PID and SSM controllers which are theoretically continuous control laws, and who's proof of stability assumes that they will be implemented as

close as possible to a theoretical continuous control law, the IGSRRT_NMPC only expects to be updated once at the end of every time step. In between each time step, the controller acts like an open loop controller similar to the one presented in section 4.2.1.

Figure 4.18 shows the steady state tracking performance of the IGSRRT_NMPC controller. The cross-track error is slightly greater than the SSM controller, which is to be expected, since under ideal conditions the SSM controller is capable of perfect tracking.

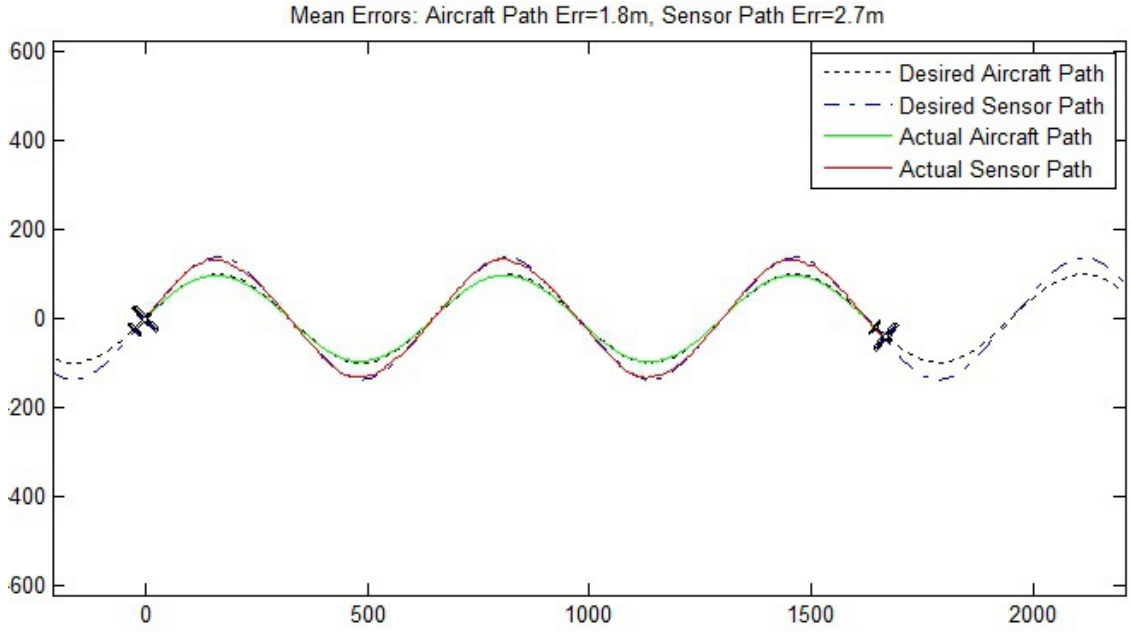

Figure 4.18: Steady state tracking performance of the RRT_NMPC controller.

The motivation for using the IGSRRT_NMPC controller is to track desired sensor paths that cannot be tracked perfectly, but since the SSM cannot track those paths at all, the purpose of these simulations is to show that under nominal conditions, the IGSRRT_NMPC tracks as well as the SSM controller.

Figure 4.19 shows the transient response of the IGSRRT_NMPC controller, which is once again comparable to the SSM controller.

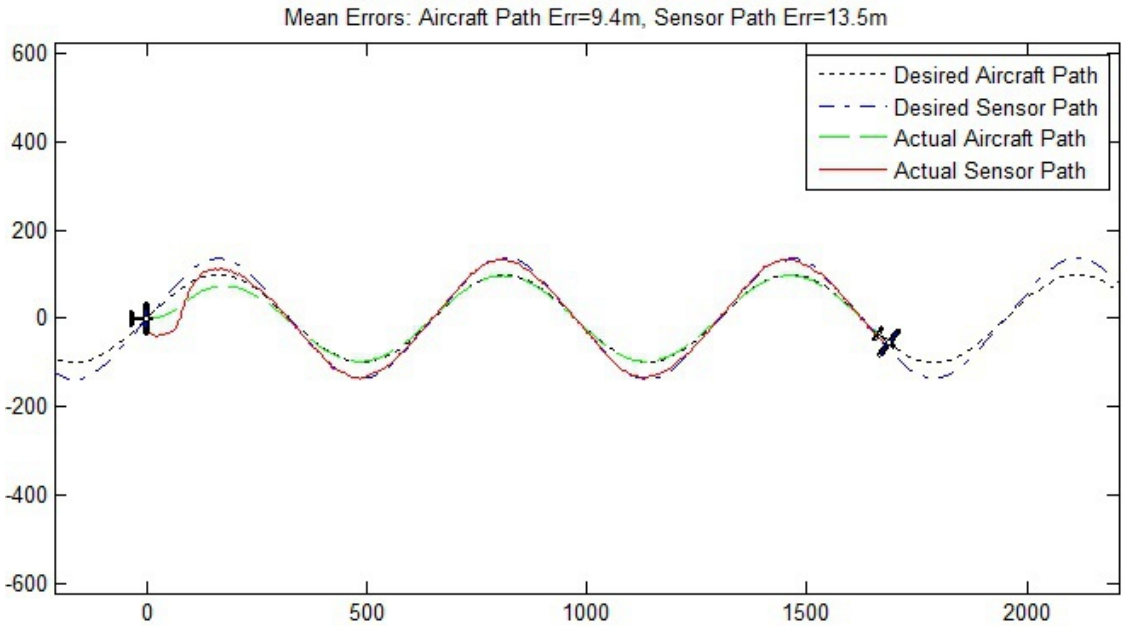

Figure 4.19: Transient response of the RRT_NMPC controller.

Finally, figure 4.20 shows the response of the IGSRRT_NMPC to a step disturbance. In this case, the IGSRRT_NMPC controller actually performs better than the SSM controller.

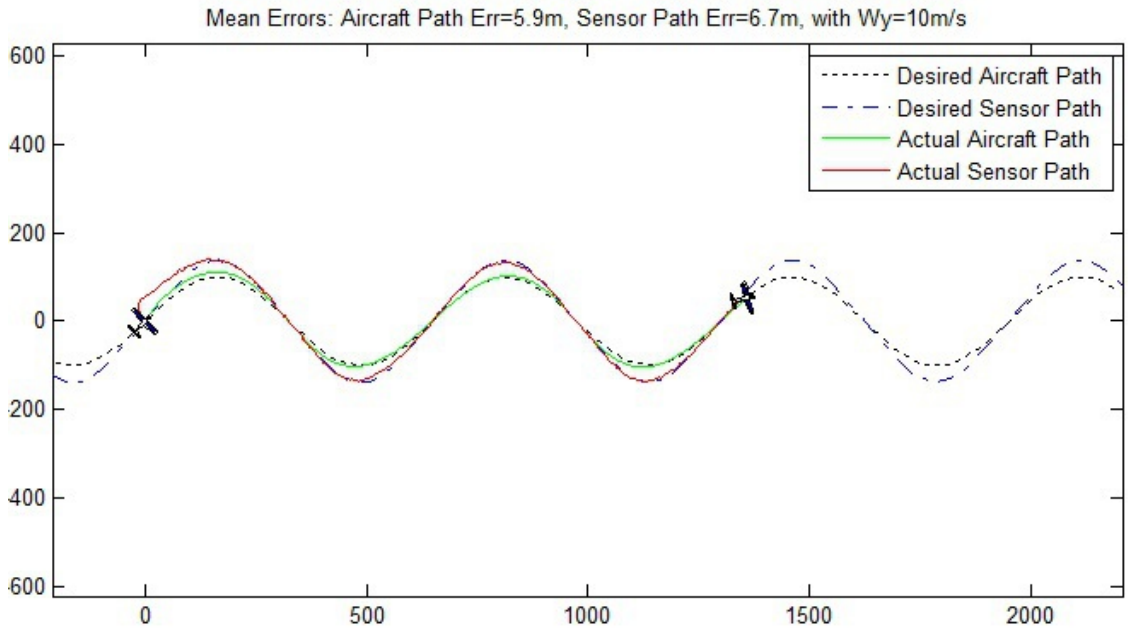

Figure 4.20: Disturbance response of the RRT_NMPC controller.

In the presence of a disturbance, the SSM attmepts to minimize the cross-track error of the aircraft path, while the IGSRRT_NMPC controller attempts to minimize the cross-track error of the sensor path. As a result, the SSM produces the lowest cross-track error of the aircraft path, while the IGSRRT_NMPC produces the lowest cross-track error of the sensor path. These results, along with the rest of the simulation results, are listed in table 4.3.4.

	Aircraft Error(m)			Sensor Error(m)		
Control	Tracking	Transient	Disturbance	Tracking	Transient	Disturbance
PID	4.6	15.4	7.7	9.9	26.9	13.5
SSM	0.4	7.3	2.3	0.7	13.5	8.7
NMPC	1.8	9.4	5.9	2.7	13.5	6.7

Hardware-in-the-loop Simulations

In addition to Matlab simulations, both the SSM and IGSRRT_NMPC controllers flew identical sinusoidal paths in C3UV's hardware-in-the-loop (HIL) simulator. The HIL simulator allows control algorithms to run on the same processors, and use the same Piccolo autopilot that it would use in an actual flight. The aircraft dynamics are simulated on a third computer, which runs a 12 degree-of-freedom UAV model provided by cloud cap.

Comparing the performance of these two controllers within a HIL simulation is the only way to get high fidelity results while still being able to control environmental parameters, which is necessary in order to ensure that neither controller is effected by extraneous disturbances.

In both HIL simulations, the controllers were given the same sinusoidal aircraft path, and projected sensor footprint that were used in the MatLab simulations. Figure 4.21 shows the steady state results of the SSM controller.

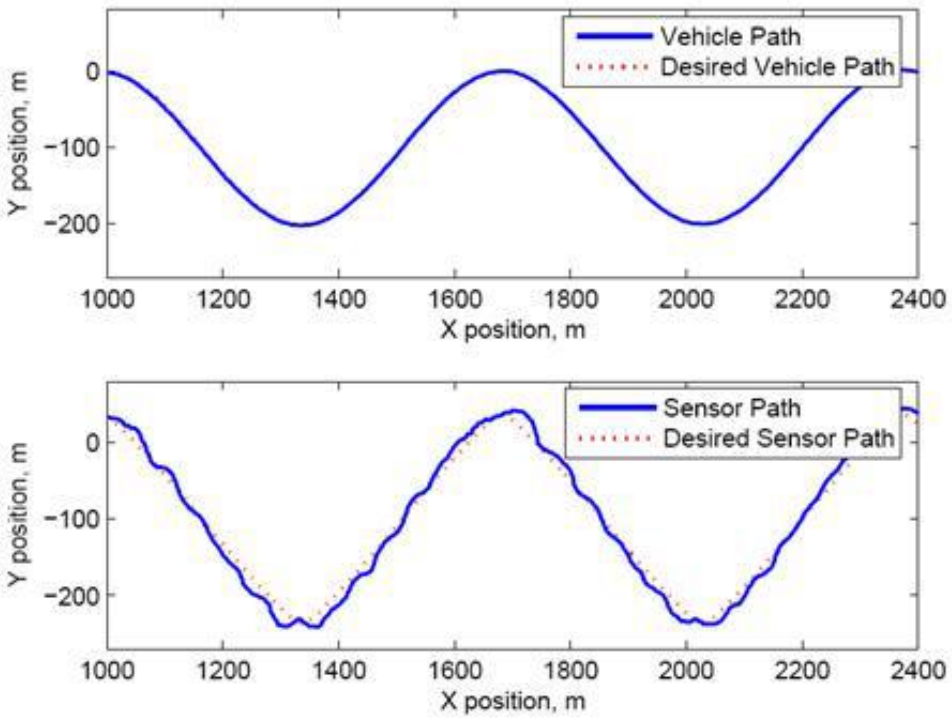

Figure 4.21: Sinusoidal Tracking results of the spatial sliding mode controller.

As one would expect, the further the simulation strays from the idealized model of a small UAV the poor the performance becomes. Qualitatively, it is clear that the SSM controller demonstrates more chatter in the HIL simulation than the Matlab simulation.

Quantitatively the average cross track error of the aircraft path is 1.3m, while the average cross-track error of the sensor path is 4.0m. Both of which are considerably worse than the near perfect tracking in the MatLab simulation, but are still reasonably good.

Figure 4.22 shows the steady state tracking performance of the IGSRRT_NMPC controller.

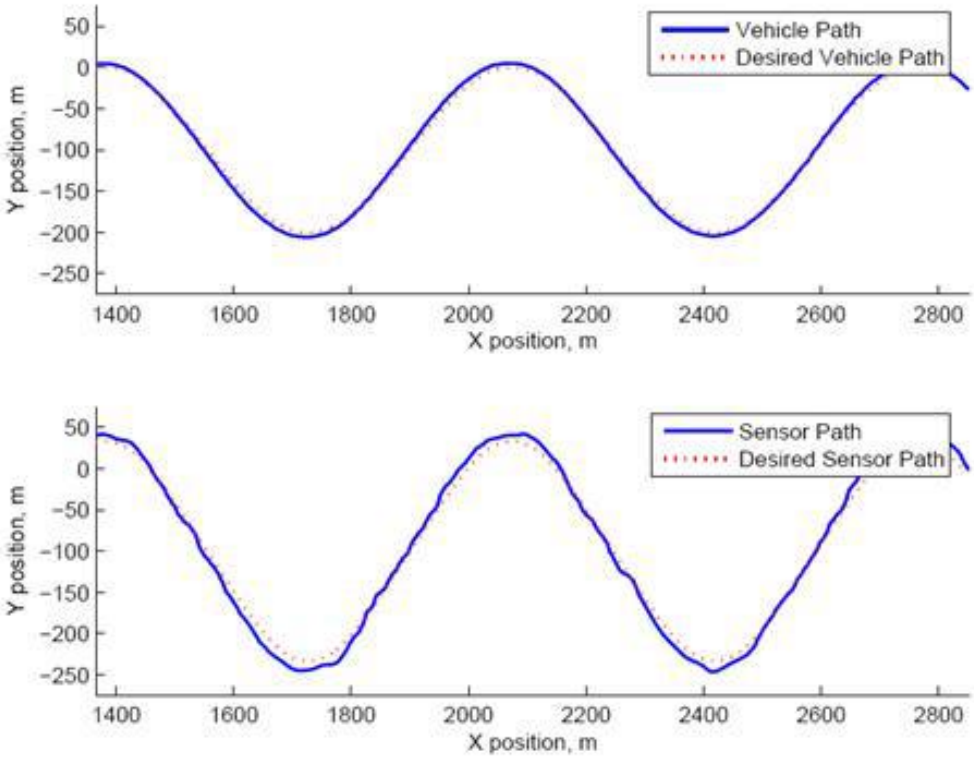

Figure 4.22: Sinusoidal tracking results for the kinodynamic controller.

For the kinodynamic controller, the average aircraft path error was 3.7 meters, while the average sensor path error was 6.1 meters. Both controllers demonstrate the ability to track the specified path. It is difficult to truly assess the relative performance of the two controllers in this simulation environment, but it is worth noticing that despite the kinodynamic controller's slightly poorer tracking performance, it tracks the line much more smoothly. Since the ultimate goal is to collect video data of the target path, the quality of the data would benefit from the smoother tracking. Both ground paths are well within the sensor footprint of the UAV. The following table summarises the HIL results.

	Aircraft Error	Sensor Error
SSM	1.3	4.0
NMPC	3.7	6.1

4.4 Conclusion

This chapter presented a novel NMPC turn-rate controller that steers a UAV in order to have the UAV's sensor footprint track a desired target. It was shown that a greedy solution

to the sensor tracking problem results in a non-linear non-minimum phase system, where in most cases, perfect tracking is not possible. Both a qualitative and quantitative analysis of good and bad tracking was presented, and it was shown that the tracking performance of the NMPC controller is comparable to a spatial sliding mode controller, and that the NMPC controller is capable of solving the sensor tracking problem by finding a near optimal solution with real-time performance characteristics.

Chapter 5

Vision in the Loop Tracking with Small UAVs

The following chapter presents the experimental results of flights conducted at the CIR-PAS airfield in Camp Roberts, CA, using the Berkeley Sig Rascal UAV. The purpose of these experiments was to quantify the real world performance of the NMPC controller presented in Chapter 4. In the following sections, the NMPC controller is applied to a variety of tracking tasks including:

- tracking various paths
- flying directly over a point to acquire images
- orbiting a point for persistent surveillance

Several significant modelling assumptions were made in order to develop the NMPC controller described in Chapter 4, namely that:

- the dynamics of the UAV are well modelled by the kinodynamic unicycle model
- the wind disturbance is purely additive
- the UAV only make coordinated turns
- the UAV's pitch remains nominally zero

Chapter 2 has already addresses each of these assumptions individually. The goal of the following experiments is to assess the tracking performance of the NMPC controller despite the limitations of those assumptions.

5.1 Tracking Paths

This section discusses the performance of the NMPC controller tracking a given sensor path from a UAV with a fixed, downward looking camera. These examples are a direct application of the work that has been discussed in the previous chapters.

The first experiment, tracking a river, summarizes the culmination of the work that preceded the NMPC controller, and strongly motivated the need for a sensor tracking controller like the one presented here. The second experiment simply verifies the straight line tracking performance and stability of the NMPC controller. The third experiment presents the real world sinusoidal tracking performance of the NMPC controller, verifying the steady state sinusoidal tracking simulation presented in chapter 4. The final experiment presents the tracking performance of the NMPC controller when it is given nearly random, discontinuous paths. The paths are generated by a higher level planner running a search algorithm over an area. This final experiment also compares the tracking performance of the NMPC controller with the SSM controller.

5.1.1 Tracking a River

Tracking linear structures, both natural and man-made, is an obvious application for the NMPC controller. In the summer of 2006, an experiment was performed to track a river using vision in the loop feedback from a near infra-red camera. As part of the tracking challenge, the controller had no prior knowledge of the river's location. The experiment was initiated by sending the aircraft along a heading that would intercept the river. Then, the UAV had to use vision in the loop data to acquire the river, and then track it. The goal was to image the length of the river, in order to later map it offline. The next two sections present the results of the controller used in 2006, along with simulated results using the NMPC controller.

Previous Work

The objective of the controller used in 2006 was to minimize the cross-track error of UAV's position over the river[36]. Without going into too much detail, the assumption was that if the UAV remained over the river, and used minimum control effort, the sensor footprint would remain on the river. The controller used a connecting spline and back stepping to derive its control law. Unfortunately, the controller's transient performance made it nearly impossible for the UAV to track the river, unless it was initially nominally aligned with the river. As a result, the controller required the elaborate alignment procedure shown in figure 5.1.

Figure 5.1: Alignment procedure for the original river tracking algorithm.

The red, looping line shows the path the UAV took to align itself with the river, prior to using the spline controller, or vision in the loop feedback. Figure 5.2 shows the coverage of the river this controller was able to achieve. The whited-out portions of the river, are points along the river that were imaged.

Figure 5.2: Coverage of the river using the spline controller in 2006.

In particular, the portions of the river where the river take a sharp bend, were not imaged. In these experiments, the UAV was flying from the North end of the river to the South. As ths UAV came to each bend in the river, it would turn sharply to stay on track, as it did so, it banked so hard that the river went out of view. The assumption, that minimizing the UAV's cross-track error with the river, would keep the river in view, turned out to be false.

Simulated Results of the NMPC Controller

As a first attempt at applying the NMPC controller, and in order to assess its potential advantage over the spline controller, the river tracking experiment was recreated in simulation, in order to explore the NMPC's performance.

In order to simulate river tracking with vision in the loop, a vision simulator was created. An image of the river was taken from Google maps and the center line of the river was then manually identified and stored in a lookup table. The center line of the river is shown in 5.3 by a blue line. The vision simulator then calculated which points from the lookup table were within the sensor footprint of the UAV, and then sent those points to the controller.

The infinite horizon estimator is simply the cost the UAV would accumulate while travelling to the path from it's current position, and worst case heading angle.

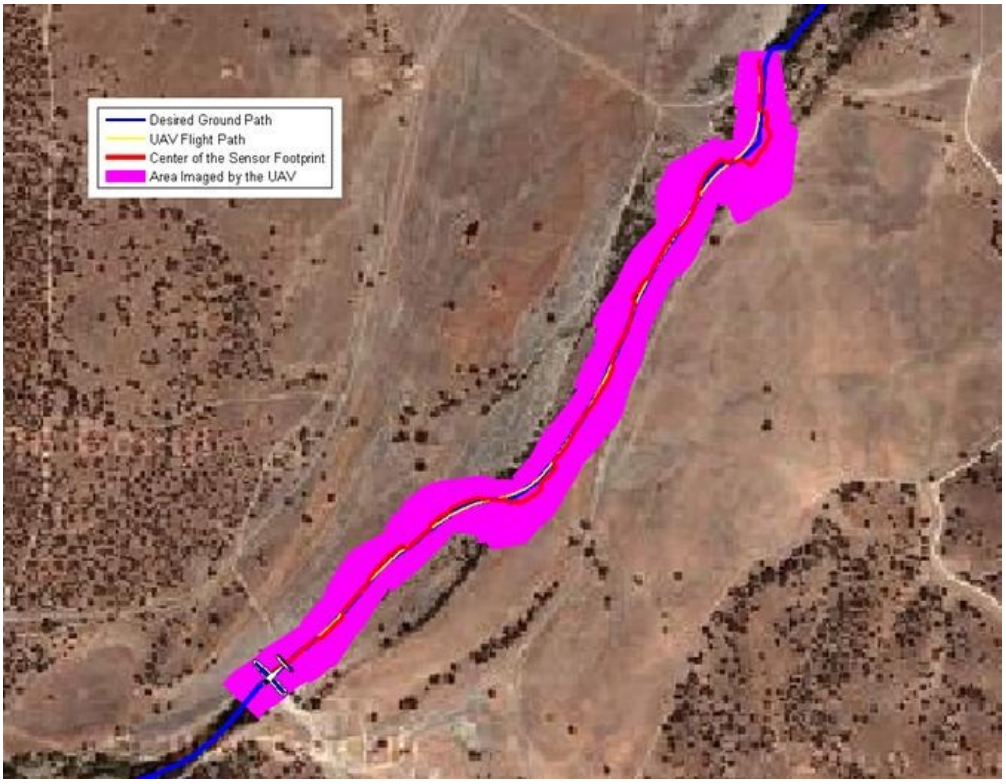

Figure 5.3: Coverage of the river using the NMPC controller.

The magenta area shows that the entire length of the river was well imaged, including the bends in the river that were previously missed by the old controller. The tracking performance had a mean cross-track error of 5.8 m and a standard deviation of 6.0. Given the limited information horizon, about 6 seconds, provided to the controller, it showed very good performance.

Figure 5.4 shows the UAV aligning with the river using the new tracking controller.

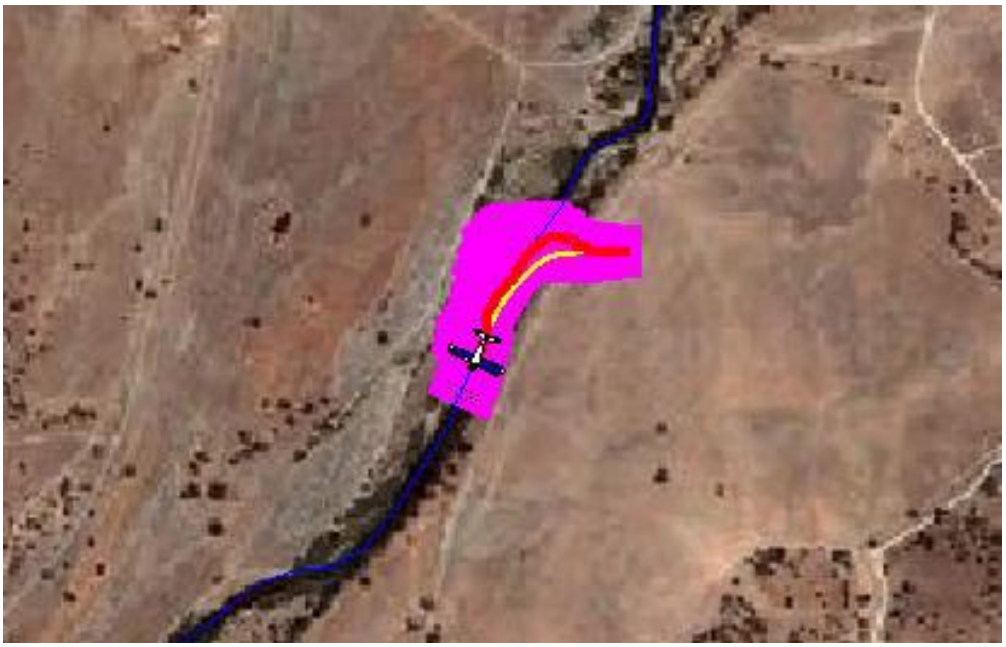

Figure 5.4: The UAV aligning with the river using the NMPC controller.

The aircraft starts by flying directly west and has no information about the river's location. As soon as the river comes within the field of view of the cameras, the UAV makes a minimum radius turn to align itself. It then aligns itself with the river with minimal over shoot. As was previously shown in Chapter 4, the transient behaviour of the NMPC controller is quite good, which in this case allows the UAV to skip the elaborate alignment procedure that was used by the spline controller.

5.1.2 Tracking a Runway

The goal of the first experimental flight was to track a straight path on the ground. In order to give that path some relative meaning in the physical world, it was superimposed on to the runway at the CIRPAS facility. In order to decouple the errors associated with vision detection and tracking, the GPS coordinates of the air field were manually entered into the tracking controller. The aircraft flew along the runway for 16 min. When the aircraft reached the end of the runway, the tracking controller, in order to reduce the sensor footprint's distance from the runway, would turn the aircraft around for another pass. Quantitatively, the Controller shows very good tracking performance. Once the sensor footprint settled on the runway, it had an average cross track error of only 1.0m and a standard deviation of 1.7m^2.

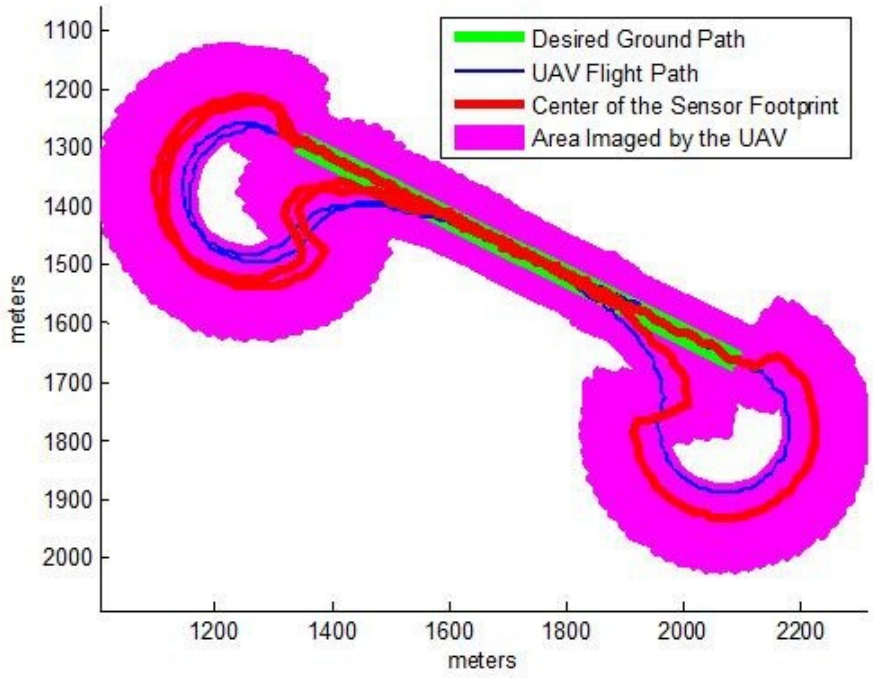

Figure 5.5: Plot of line tracking flight results.

Figure 5.5 shows the path of the UAV throughout the experimental flight. The tight turns at the both ends of the runway are the minimum turning radius of the vehicle. Even though the goal of the controller is to place the sensor footprint on the desired path, the settling time of the UAV also turned out to be quite good, about three times better than the line tracking controller provided by the Piccolo autopilot.

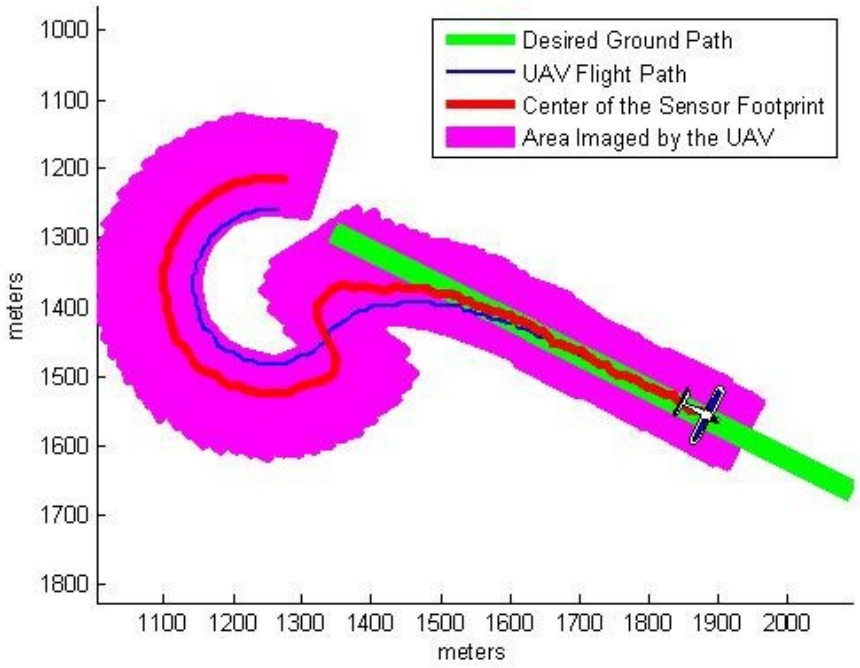

Figure 5.6: This controller provides 100% surveillance of the path in a single pass.

Figure 5.6 shows how the UAV banks in order to place its sensor footprint as close to the runway as possible. As the UAV approaches from the left it banks sharply to the right, shifting its footprint from its right side to its left. By taking minimum radius turn to the left and then, at the optimal moment, shifting to a minimum radius turn to the right, the UAV is able to survey the entire length of the runway on every pass.

Figure 5.7: Animations of the UAV acquiring the runway.

Figure 5.7 shows the UAV acquiring the runway. The white trapezoids are projections

of the camera's field of view. As the sensor foot print of the UAV settles on to the runway, the controller produces no overshoot in either the camera position or the aircraft's position.

5.1.3 Tracking a Sinusoidal Path

The third path track experiment recreates the steady state sinusoidal track experiment that was discussed in Chapter 4. Figure 5.8 shows the results.

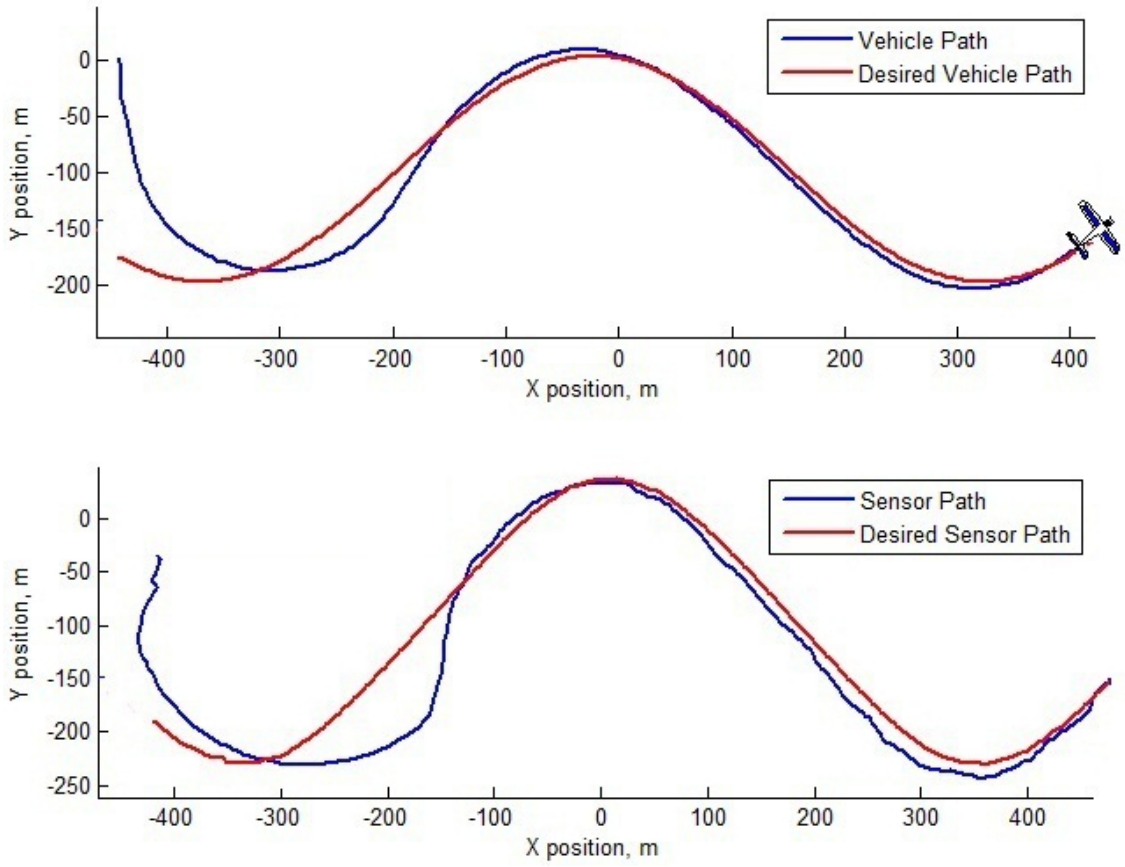

Figure 5.8: Real world sine tracking performance.

Once the UAV had settle, it tracked the projected sine path with an average cross-track error of 7.2m. During this experiment, the mean wind speed was less than 1m/s. The following table summarizes the three different sine tracking experiments.

NMPC	Aircraft Error(m)	Sensor Error(m)
MatLab	1.8	2.7
HIL	3.7	6.1
Flight	5.1	7.2

5.1.4 Tracking a Nearly Random, Discontinuous Path

The final path tracking experiment required the NMPC controller to find an optimal trajectory along a disjoint path. The disjoint paths were created by a higher level controller that was attempting to search an area by minimizing a probability function that defined the likelihood of finding a target. Further discussion on the searching algorithm can be found here[50]. Essentially, the goal of the search algorithm was to have the tracking controller look from left to right in order to 'sweep out' as much probability mass as possible. Figure 5.9 shows an example of the desired paths that were sent to the NMPC controller, along with the controller's tracking performance.

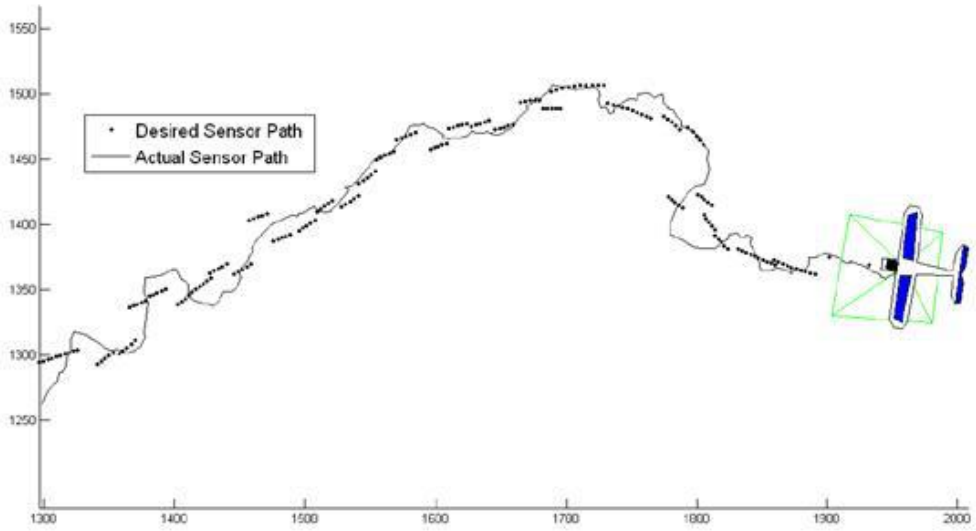

Figure 5.9: Example of the typical disjoint path.

The high level search algorithm generated both a desired sensor path and the corresponding optimal aircraft path. As such, it was possible to compare the tracking performance of the NMPC controller with the SSM controller. Unlike the HIL simulation results, which were based on exactly the same initial conditions for both controllers, difference in the initial conditions and environmental conditions for the two controllers had to be accounted for by averaging over several flights. The following table presents the tracking average tracking results of each controller.

	Sensor Error(m)	
	Average Error	Standard Deviation
SSM	9.2	7.2
NMPC	7.5	7.2

Once again, both controllers, perform about the same, though in this case the NMPC controller ends up tracking the sensor path slightly better. This data provides compelling evidence, that even for the most disjoint desired paths, the NMPC controller is still capable of finding near optimal paths that place the UAV's sensor footprint on the desired sensor path. It is even more compelling given the fact that the SSM is given the optimal path and it is still out performed by the NMPC controller.

5.2 Flying Over a Point

This next section describes the tracking results of a slightly different tracking task. The goal of these tracking tasks is to flight the aircraft, wings level over a target so that the UAV can take picture of the target from directly over head. Pictures from directly over a target are particularly useful for localization and photo mosaicking. Photos taken orthogonal to the ground are less distorted than those taken at a skew angle.

In order to take a wings level over the target both of these tracking tasks superimpose a line over the desired point. By making the line long enough that the controller has enough time to settle, the UAV should fly wings level over the target, just as the UAV flew wings level over the runway.

5.2.1 Tracking Orthogonal Paths Over a Target

In this experiment, the goal was to fly a figure eight pattern over the target, so that every pass would provide an image that was orthogonal to the last. By using a camera that was slightly pitch forward, the camera would get several pictures of the target as the UAV approached it. In principle getting perpendicular views of the target would be the best way to localize it.

The goal was to track a figure eight centred over a desired point. The figure eight was divided into 4 sub paths, shown in figure 5.10 as fine yellow lines. The white dot in figure 5.10 represents the center of the figure eight and the point of interest the controller is trying to image. The bold grey line is the path taken by the sensor footprint. This experiment was done in the presence of a 5m/s wind to the North West, which accounts for the error at the top of the figure eight; the aircraft could not turn any sharper in order to avoid being blown off course.

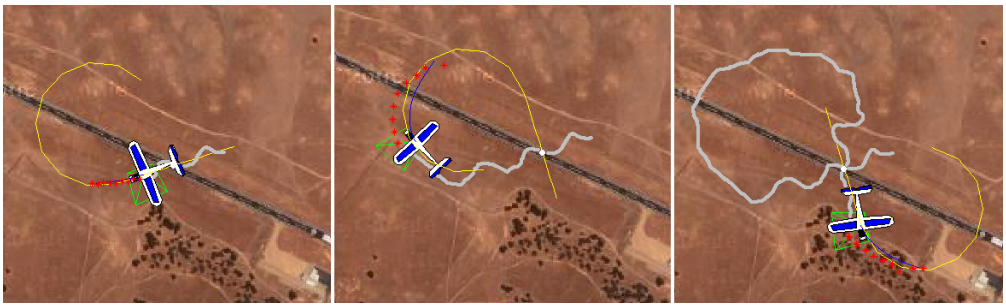

Figure 5.10: The UAV flying a figure eight pattern over a point.

The wind also caused and unexpected saw-tooth pattern when the plane was flying directly into it. This is likely caused by a combination of the two following effects. Previous simulations produced a similar pattern that was later traced back to poor GPS estimates. The data from the flight showed that GPS filter did not perform well in the rather strong winds, and it is very likely that the controller was getting poor GPS estimates. The unmodeled aircraft dynamics could have also caused this strange behaviour, as it was shown in Chapter 2 each of the assumptions that lead to the kinodynamic unicycle model of a UAV begin to break down in winds over 5m/s.

In this experiment, the average cross track error was 15.3m with a standard deviation of 22.5m2. Conditions with less wind would have likely produced much better data. Despite the saw-tooth pattern, the biggest source of error occurred at the top of the figure eight. The aircraft was already turning at its maximum rate at the top of the figure eight; it was not physically possible for the UAV to stay on track. Despite being blown off course, the controller was able to recover from the disturbance and get back on track in time to take and image of the desired point.

5.2.2 Maximizing Time Over the Target using Wind

Rather than enforcing that the UAV fly perpendicular paths over the target, an alternative meathode of flying wings level over the target is to rotate the straight path over the desire point to what ever direction minimizes the cost of the tracking controller. This is a particular useful approach when there is wind, as the path will automatically rotate such that the UAV flies directly into the wind as it flies over the target. As a result, the UAV will spend a greater percentage of the flight, over the desired target. Figure 5.11 shows an example of this tracking stratagy when there is no wind. Each picture from left to right is snapshot of an animated simulation.

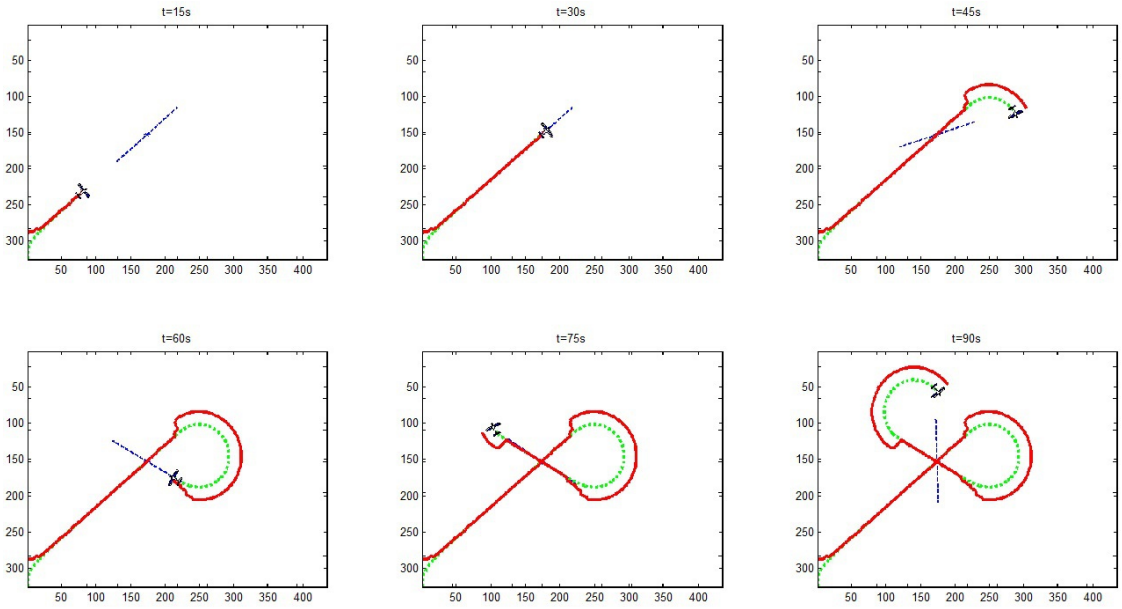

Figure 5.11: Frames taken from a simulation showing a wings level fly over controller.

Figure 5.12 shows the path of the UAV using the same controller, but when there is a 5m/s wind head to the Southwest, where North is up.

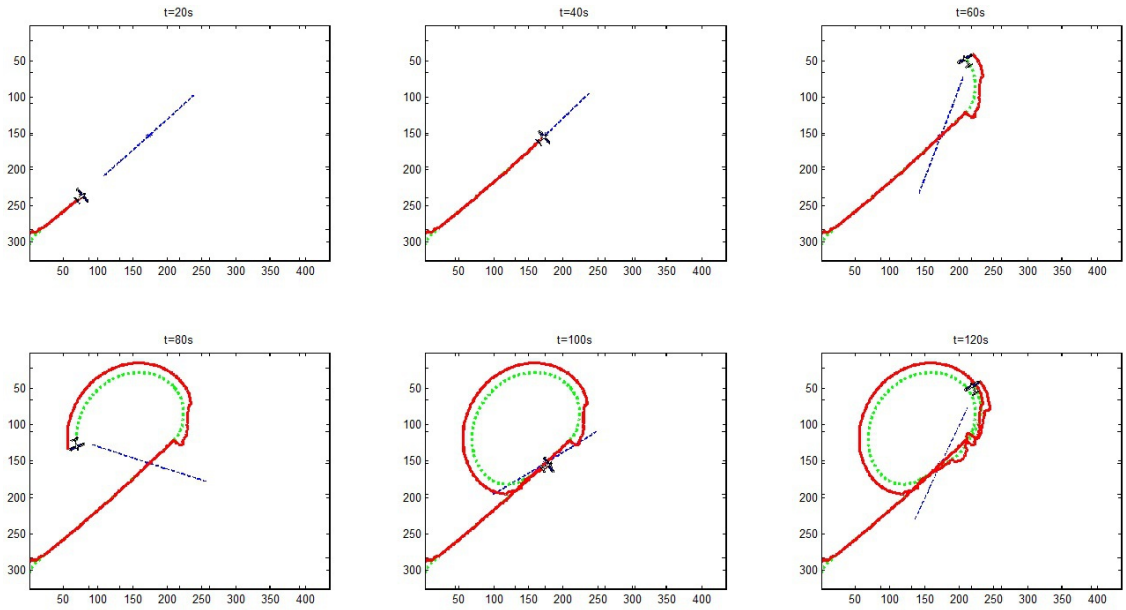

Figure 5.12: Frames taken from a simulation with wind showing a wings level fly over controller.

In the simulation with no wind, the UAV takes the fastest route to fly over the point. But, in the simulation with wind, the UAV lines up to fly over the point while heading into the wind.

5.3 Orbiting a Point

In many situations, it is advantageous to have persistent surveillance of a target, rather than periodic images from flying over the target. In order to achieve persistent surveillance the UAV must orbit around the target, which minimizes the average distance of the UAV to the target overtime. Unlike the previous tracking tasks which all used a downward looking cameras, orbiting a target requires a sideways looking camera.

Choosing the angle of the sideways looking camera is a critical decision. The choice of angle determines the radius of the orbit required to track a target, given that the aircraft is at a desired altitude. In most situations, it makes sense to let the desired orbit radius and altitude determine the angle of the camera. If the angle of the camera is defined as the positive rotation of a downward looking camera about the body-fixed x-axis, then the geometry of the camera angle is given by figure 5.13.

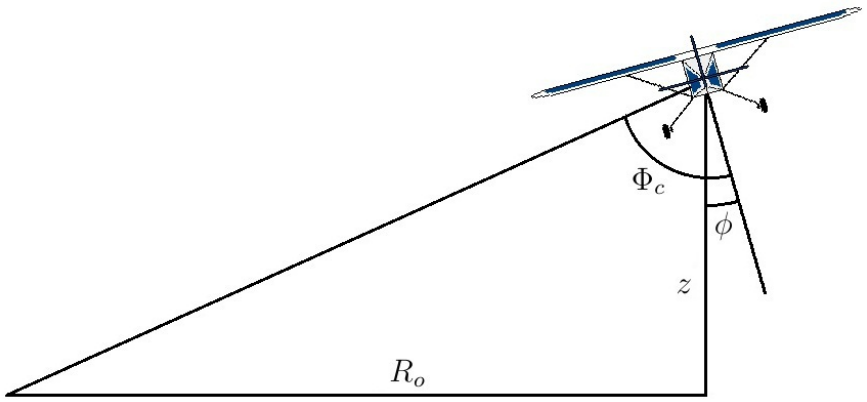

Figure 5.13: The geometry defining the camera angle for a given orbit radius.

The bank angle required to maintain a given orbit can be found by substituting,

$$\dot{\psi} = \frac{V}{R_o}$$

into the coordinated turn equation, which results in the following equation.

$$\phi = \tan^{-1}\left(\frac{V^2}{gR_o}\right)$$

Adding this bank angle to geometry from figure 5.13 results in the following equation for the camera angle.

$$\Phi = tan^{-1}\left(\frac{R_o}{z}\right) + \tan^{-1}\left(\frac{V^2}{gR_o}\right)$$

Other than changing the kinodynamic unicycle model to account for this new camera angle when calculating the projection of the sensor footprint, the NMPC controller remains the same for orbit tracking as it was for path tracking.

5.3.1 Finding an Infinite Horizon Estimator

The infinite cost horizon estimate is a bit different for the orbit tracking task versus the path tracking task. The biggest different is that the state of having zero cost is not when the UAV is on top of the target. The zero cost state is when the UAV is in a minimum radius, clockwise orbit about the target. Other than that the heuristic that calculates the infinite cost does so by calculate a Dubins like path from the UAV's current position to a position in orbit about the target. Figure 5.14 shows a few examples of those Dubins like paths. The main difference, between the actual Dubins paths and these paths is that the heuristic has

a bias towards turning right. From experiential observations, having the UAV take mostly right turns results in faster settling times, by forcing the UAV to approach the orbit without going through it.

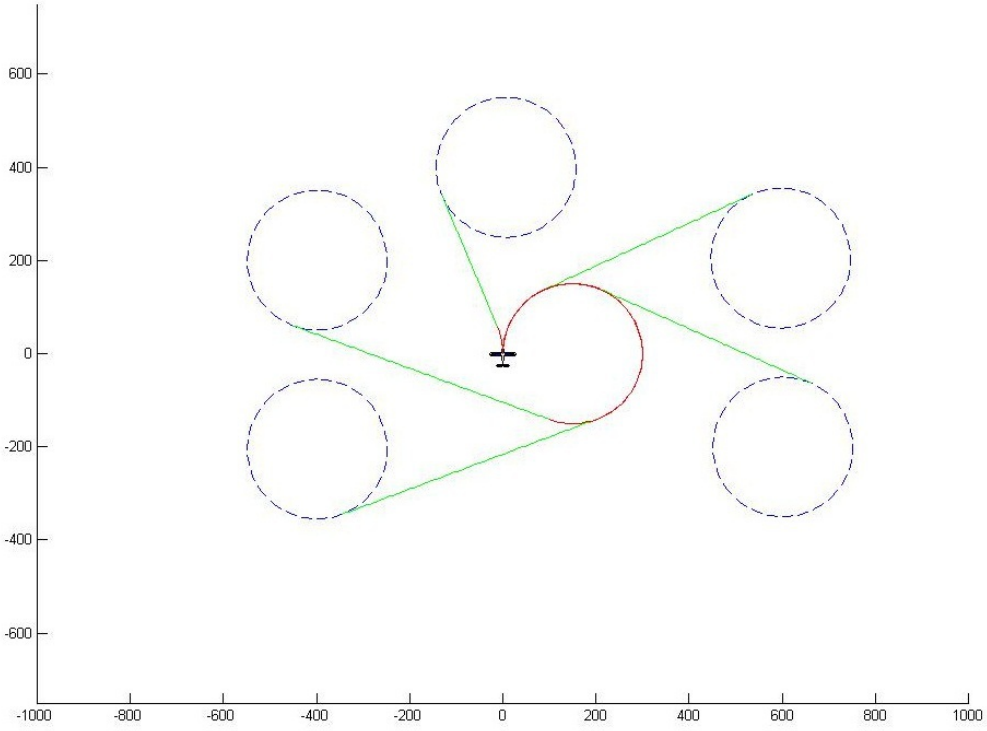

Figure 5.14: Sample paths showing the distance to the orbit.

Once the Dubins like distance, d has been calculated, the heuristic calculates the cost that would accrue at each time step on the way to the orbit. Given the distance to the target, the cost the would accrue is given by the following equation.

$$h(d) = \frac{1}{2}d + \left(\frac{d}{V\Delta t} + 1\right) \tag{5.1}$$

The first term in the equation for $h(d)$ is simply the average distance remaining for the set of remaining steps until the UAV reaches the orbit. The second term calculates the number of remaining steps until the UAV reaches the orbit.

Figure 5.15 shows the distance function with respect the UAV.

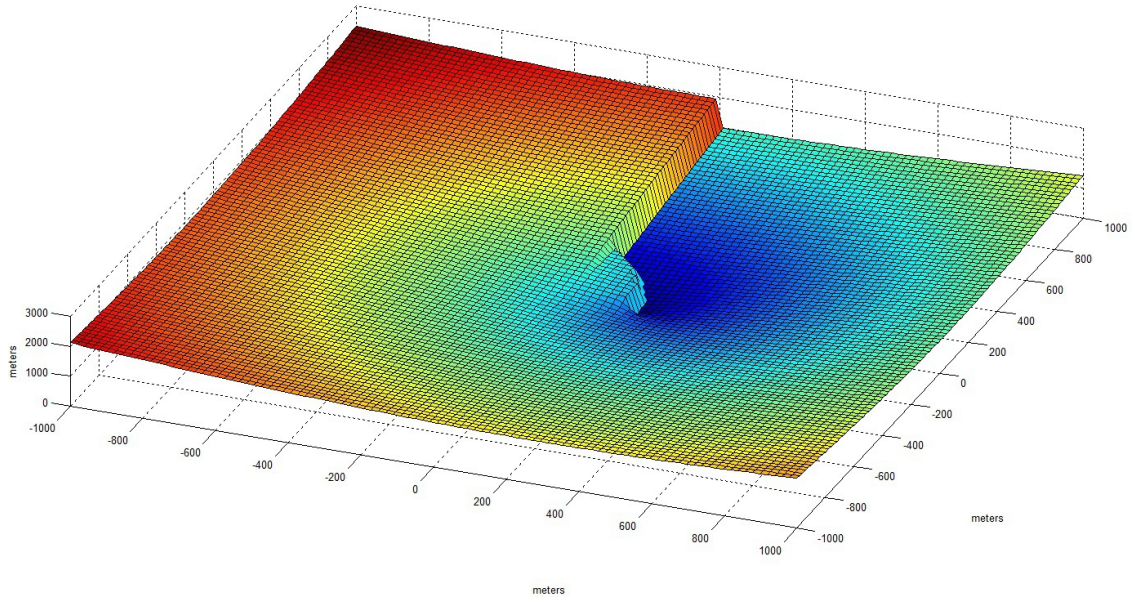

Figure 5.15: The distance function for the infinite cost heuristic for the orbit tracking task.

Because the minimum of this function is unique, the IGS_RRT algorithm rapidly converges to position about the target that minimizes this heuristic cost.

5.3.2 Pedestrian Tracking

The last experiments that were performed with this controller involved tracking a pedestrian using vision in the loop feedback. Much like the river tracking experiment, the UAV was commanded to look at a point, until it identified a pedestrian in the image, and then to track the pedestrian. Figure 5.16 shows the slides from an animation of the flight data from one of the orbit experiments. In the animation the yellow dot is the pedestrian, and the green trapezoid is the sensor footprint.

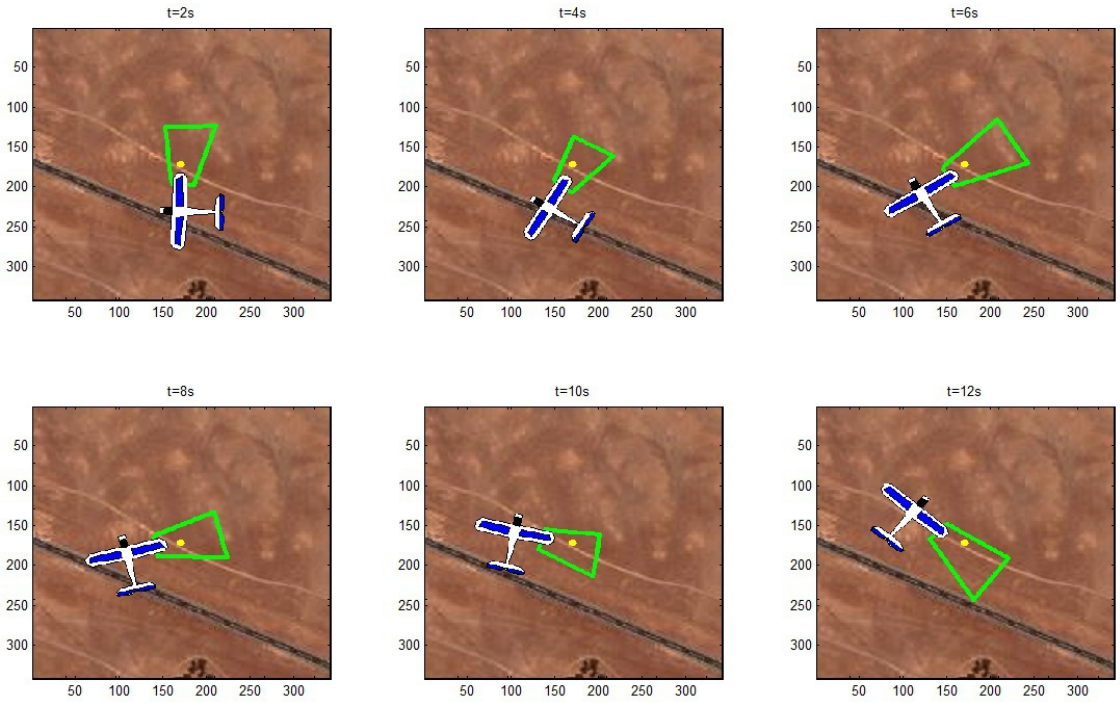

Figure 5.16: Slides from an animation of the flight data.

Again, like the river tracking experiment, the goal of this experiment was to provide coverage, or persistent surveillance of the pedestrian. Unfortunately, the wind during the experiments was consistently in excess of 5m/s, which lead to rather mediocre tracking. The goal of good tracking it being able to you a camera with a smaller field of view, which translates to an image with greater resolution. Figure 5.17 gives an idea of the tracking performance by showing the percentage of coverage as field of view goes from 0 to 30 degrees.

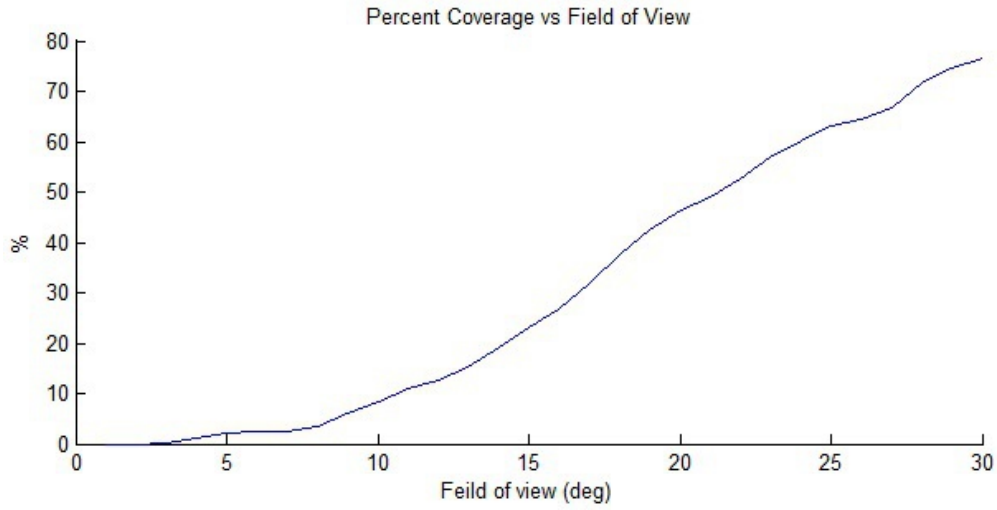

Figure 5.17: The percentage of frames with the target within the feild of view as a function of the size of the field of view.

As the field of view drops down to less than 10 degrees, a mere 5 degrees of error will put the target out of the field of view. In Chapter 2 it was shown that if the wind speed is over 5m/s there are about 5 degrees error in the coordinated turn approximation of the bank angle, let alone all of the other errors that accumulate do to the wind. As such it is not surprising that a field of view less than 10 degrees would almost never have the target in view.

5.4 Conclusions

This chapter presented the tracking results of the IGS-RRT NMPC tracking controller that was developed in the previous chapters. The first experiment showed that the controller was capable of positing the UAV's sensor footprint on a straight line, with an accuracy of 1.0m and a standard deviation of 1.7m^2. The second experiment showed that the controller was able to maintain a tracking performance of 7.5m on average even when the desired sensor path was discontinuous and nearly random. Furthermore, the experiment showed that given real world conditions, the NMPS controller was able to track slightly better than the spatial sliding mode controller presented in chapter 4. The third experiment used the NMPC controller to acquire aerial imagery of a point from directly over head, which it was able to do despite significant wind disturbances. And, the final experiment, showed the tracking performance of the NMPC control when used to persistently track a pedestrian, by orbiting a single point.

From the experimental data, it is clear the NMPC controller performs very well when

tracking paths, but not quite as well when orbiting points, which is somewhat expected based on the modelling conclusions from chapter 2. In the presence of win disturbances, many of the modelling assumptions become less valid, and significantly reduce the accuracy of the model especially when the aircraft is banking. As a result, when the UAV performs a nearly constant maximum bank angle turn, in order to orbit a point, the tracking performance of the NMPC controller is significantly degraded. It follows then that improving the model with respect to wind disturbances may significantly improve the NMPC's orbit tracking performance.

Chapter 6

Summary of Conclusions

6.1 Conclusions

The previous chapters presented a novel approach to controlling the sensor footprint of a UAV such that it optimizes the quality of the images the UAV collects. The approach was composed of three innovations. Firstly, it was shown that an innovative kinodynamic unicycle model that augments the traditional, kinematic unicycle model with a first order approximation of a UAV's roll dynamics, improves the model's accuracy by about 25%. Secondly, an iterative Gaussian sampling strategy for the rapidly exploring random tree algorithm was introduced that generates paths that are ten times more optimal, and actually runs faster than the original uniform sampling strategy. And lastly, a receding horizon, nonlinear model predictive controller was presented that leverages the other two innovations to solve the non-minimum phase problem of controlling a UAV's sensor footprint.

Chapter 2 presented a novel kinodynamic unicycle model, and used actual flight data to evaluated its accuracy. It also evaluated several common assumption that are often made in order to reduce the complexity of the dynamics of fixed wing UAVs. Both the coordinated turn assumption, and zero pitch assumption, turned out to be accurate when the mean wind speed was small, and both assumptions became less accurate as the wind increased. In fact, by every metric, the performance of both unicycle models degraded rapidly as the mean wind speed increased, which motivates the need for a better model of the effect wind disturbances.

Several common motion planners were presented in Chapter 3, along with a novel iterative Gaussian sampling strategy for the rapidly exploring random tree algorithm. The relative advantages and disadvantages of several motion planners with respect to motion planning for small UAVs was also discussed. In particular, the chapter discussed the difference between fast motion planning, and kinodynamic motion planning.

Chapter 3 also presented data from Monte Carol simulations that showed that, in addition to being computationally less complex than a uniform sampling strategy, the IGS-RRT algorythm also generates paths who's costs are ten times lower. Lastly, the IGS-RRT showed

that by initializing the tree with the 'best path' from the last query, it essentially continues to iterate on the same tree, and returning solutions like an anytime algorithm.

Chapter 4 presented a novel NMPC turn-rate controller that steers a UAV in order to have the UAV's sensor footprint track a desired target. It was shown that a greedy solution to the sensor tracking problem results in a non-linear non-minimum phase system, where in most cases, perfect tracking is not possible. Both a qualitative and quantitative analysis of good and bad tracking was presented, and it was shown that the tracking performance of the NMPC controller is comparable to a spatial sliding mode controller, and that the NMPC controller is capable of solving the sensor tracking problem by finding a near optimal solution with real-time performance characteristics.

Finally, chapter 5 presented the real world tracking results of several experiments that used the IGS-RRT NMPC controller to control the Berkeley Sig Rascal UAV. The first experiment showed that the controller was capable of positing the UAV's sensor footprint on a straight line, with an accuracy of 1.0m and a standard deviation of $1.7m^2$. The second experiment showed that the controller was able to maintain a tracking performance of 7.5m on average even when the desired sensor path was discontinuous and nearly random. Furthermore, the experiment showed that given real world conditions, the NMPS controller was able to track slightly better than the spatial sliding mode controller presented in chapter 4. The third experiment used the NMPC controller to acquire aerial imagery of a point from directly over head, which it was able to do despite significant wind disturbances. And, the final experiment, showed the tracking performance of the NMPC control when used to persistently track a pedestrian, by orbiting a single point. From the experimental data, it is clear the NMPC controller performs very well when tracking paths, but not quite as well when orbiting points, though it's orbit tracking performance may be greatly improved with a better disturbance model

6.2 Future Work

Though this paper presents the results of individual UAVs flying in collision free environments, by using the RRT algorithm, it lays the ground work for a much more complicated problem, possibly involving team work among UAVs, sensor tracking around obstacles, and optimizing the quality of an image based on occlusions, localization, and environmental conditions.

In particular, the disturbance model for the way a UAV is effected by the wind may be significantly improved if it accounted for the attitude of the aircraft relative to the wind. For instance, a neural network could be used to generate wind estimates, based on a large volume of previous light data.

The efficiency of the RRT path planner, could be improved by sampling the configuration space of the UAV using a discrete number of maneuvers, instead of, or in combination with, model integration. Thought using maneuvers would decrease the sampling resolution, after

the first few seconds, based on the accuracy of the model, it probably wouldn't make a difference.

Finally, it may be possible to get the spatial sliding mode controller to track the sensor path directly by using output redefinition techniques [47]. Essentially the output redefinition would require a path planner to find an optimal sensor path, and then reflect that path around the longitudinal axis of the UAV, which would put the reference path above the UAV. Then, by minimizing the error between the reference path and a vector pointed up from the UAV, the UAV's sensor footprint, should track the target.

Bibliography

[1] http://www.army-technology.com/projects/rq11-raven/.

[2] http://www.cloudcaptech.com/.

[3] http://www.diydrones.com/.

[4] Brandon Basso. Locally weighted models for fixed wing autonomous aircraft control. 2008.

[5] J. Bellingham, A. Richards, and J.P. How. Receding horizon control of autonomous aerial vehicles. In *American Control Conference, 2002. Proceedings of the 2002*, volume 5, pages 3741 – 3746 vol.5, 2002.

[6] V. Boor, M.H. Overmars, and A.F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1018 –1023 vol.2, 1999.

[7] Arthur E. Bryson. *Control of Spacecraft and Aircraft*. Princeton University Press, Princeton, New Jersey, 1994.

[8] Eduardo Camacho and Carlos Bordons. Nonlinear model predictive control: An introductory review. In Rolf Findeisen, Frank Allgwer, and Lorenz Biegler, editors, *Assessment and Future Directions of Nonlinear Model Predictive Control*, volume 358 of *Lecture Notes in Control and Information Sciences*, pages 1–16. Springer Berlin / Heidelberg, 2007.

[9] Haiyang Chao, Yongcan Cao, and YangQuan Chen. Autopilots for small fixed-wing unmanned air vehicles: A survey. In *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*, pages 3144 –3149, aug. 2007.

[10] HaiYang Chao, YongCan Cao, and YangQuan Chen. Autopilots for small unmanned aerial vehicles: A survey. *International Journal of Control, Automation and Systems*, 8:36–44, 2010. 10.1007/s12555-010-0105-z.

[11] Michael Clark Tugrul Galatali Juan P. Gonzalez Jay Gowdy Alexander Gutierrez Sam Harbaugh Matthew Johnson-Roberson Yu Hiroki Kato Phillip Koon Kevin Peterson Bryon Smith Spencer Spiker Erick Tryzelaar William Red Whittaker Chris Urmson, Joshua Anhalt. High speed navigation of unrehearsed terrain: Red team technology for grand challenge 2004. Technical report, The Robotics Institute at Carnegie Mellon University, 5000 Forbes AvenuePittsburgh, PA 15213, June 2004.

[12] John Doe. The unmanned aerial vehicles (uav) market 2010-2020: Technologies for isr and counter-insurgency. Technical report, Companies and Markets, 2010.

[13] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):pp. 497–516, 1957.

[14] Tim McGee Zu Whan Kim Jack Tisdale Raja Sengupta Karl Hedrick Eric Frew, Stephen Spry. Flight demonstrations of seld-directed collaborative navigation of small unmanned aircraft. In *AIAA Unmanned Unlimited Technical Conference*, 2004.

[15] E. Frazzoli, M.A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. In *American Control Conference, 2001. Proceedings of the 2001*, volume 1, pages 43 –49 vol.1, 2001.

[16] E. Frew, T. McGee, K. ZuWhan, X. Xiao, S. Jackson, M. Morimoto, S. Rathinam, J. Padial, and R. Sengupta. Vision based road-following using a small autonomous aircraft. In *Proceedings of the IEEE Aerospace Conference*, March 2004.

[17] Ruggero Frezza, Giorgio Picci, and Stefano Soatto. A lagrangian formulation of nonholonomic path following. In David Kriegman, Gregory Hager, and A. Morse, editors, *The confluence of vision and control*, volume 237 of *Lecture Notes in Control and Information Sciences*, pages 118–133. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0109667.

[18] A. Gambier. Real-time control systems: A tutorial. In *Proc. of the 5th Asian Control Conference*, 2004.

[19] Michael Hardy. Does the military have too many drones?, March 2010. http://www.defensesystems.com/Articles/2010/03/31/military-has-too-many-unmanned-planes-maybe.aspx.

[20] D. Hsu, R. Kindel, J. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *Int. Journal of Robotics Research*, 18:233–255, 2002.

[21] Yong K. Hwang and Narendra Ahuja. Gross motion planning a survey. *ACM Comput. Surv.*, 24:219–291, September 1992.

[22] Jonathan P. How Ian Garcia. Improving the efficiency of rapidly-exploring random trees using a potential function planner. IEEE Conference on Decision and Control,, 2005.

[23] S. Jackson, J. Tisdale, M. Kamgarpour, B. Basso, and J. K. Hedrick. Tracking controllers for small uavs with wind disturbances: Theory and flight results. In *Proceedings of the IEEE Conference on Decision and Control*, Cancun, 2008.

[24] Brian V. Bonnlander Matther J. Johnson Jerry E. Pratt John R Rebula, Peter D. Neuhause. A controller for the littledog quadruped walking on rough terrain. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1467–1473, April 2007.

[25] Dongwon Jung and Panagiotis Tsiotras. Bank-to-turn control for a small uav using backstepping and parameter adaptation. In *17th IFAC World Congress*.

[26] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566 –580, aug 1996.

[27] Jr. Kuffner, J.J. and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 995 –1001 vol.2, 2000.

[28] Yoshiaki Kuwata and Jonathan How. Three dimensional receding horizon control for uavs. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, pages 2004–5144, 2004.

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

[30] S.M. LaValle and Jr. Kuffner, J.J. Randomized kinodynamic planning. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 473 –479 vol.1, 1999.

[31] Steven M. Lavalle, James J. Kuffner, and Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[32] S. Leven, J.-C. Zufferey, and D. Floreano. A minimalist control strategy for small uavs. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 2873 –2878, oct. 2009.

[33] Timothy G. McGee. *Autonomous search and surveillance with small fixed wing aircraft*. PhD thesis, University of California, Berkeley, 2006.

[34] Chikatoyo Nagata, Eri Sakamoto, Masato Suzuki, and Seiji Aoyagi. Path generation and collision avoidance of robot manipulator for unknown moving obstacle using real-time rapidly-exploring random trees (rrt) method. In Keiichi Shirase and Seiji Aoyagi, editors, *Service Robotics and Mechatronics*. Springer London, 2010.

[35] D. Rathbun, S. Kragelund, A. Pongpunwattana, and B. Capozzi. An evolution based path planning algorithm for autonomous motion of a uav through uncertain environments. In *Digital Avionics Systems Conference, 2002. Proceedings. The 21st*, volume 2, pages 8D2–1 – 8D2–12 vol.2, 2002.

[36] S. Rathinam, P. Pinto de Almeida, Z. Kim, S. Jackson, A. Tinka, W. Grossman, and R. Sengupta. Autonomous searching and tracking of a river using an uav. In *IEEE American Controls Conference*.

[37] S. Rathinam, Zu Kim, Aram Soghikian, and Raja Sengupta. Vision based following of locally linear structures using an unmanned aerial vehicle. In *Proceedings of the IEEE Conference on Decision and Control*, December 2005.

[38] Sivakumar Rathinam, Zuwhan Kim, Raja Sengupta, Sivakumar Rathinam, Zuwhan Kim, and Raja Sengupta. Vision based following of structures using an uav, 2006.

[39] Barak Raveh, Angela Enosh, and Dan Halperin. A little more, a lot better: Improving path quality by a simple path merging algorithm. *CoRR*, abs/1001.2391, 2010.

[40] P.D. Roberts. A brief overview of model predictive control. In *Practical Experiences with Predictive Control (Ref. No. 2000/023), IEE Seminar on*, pages 1/1 –1/3, 2000.

[41] J. Sanchez and R. Fierro. Sliding mode control for robot formations. In *Intelligent Control. 2003 IEEE International Symposium on*, pages 438 –443, oct. 2003.

[42] Tom Schouwenaars and Eric Feron. Decentralized cooperative trajectory planning of multiple aircraft with hard safety guarantees. *Control*, (August):AIAA 2004–5141, 2004.

[43] P.O.M. Scokaert, D.Q. Mayne, and J.B. Rawlings. Suboptimal model predictive control (feasibility implies stability). *Automatic Control, IEEE Transactions on*, 44(3):648 – 654, mar 1999.

[44] Alejandro Perez Emilio Frazzoli Seth Teller Sertac Karaman, Matthew R. Walter. Anytime motion planning using the rrt*. IEEE Intl Conference on Robotics and Automation, 2011.

[45] D.H. Shim, H.J. Kim, and S. Sastry. Decentralized nonlinear model predictive control of multiple flying robots. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 4, pages 3621 – 3626 vol.4, dec. 2003.

[46] Sahjendra N. Singh, Meir Pachter, Phil Chandler, Siva Banda, Steve Rasmussen, and Corey Schumacher. Inputoutput invertibility and sliding mode control for close formation flying of multiple uavs. *International Journal of Robust and Nonlinear Control*, 10(10):779–797, 2000.

[47] J.J.E. Slotine and W. Li. *Applied nonlinear control*. Prentice Hall, 1991.

[48] Guang Song and N.M. Amato. Randomized motion planning for car-like robots with c-prm. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 37 –42 vol.1, 2001.

[49] Petr Svestka, Mark H. Overmars, Mark H. Overmars, and Mark H. Overmars. Motion planning for car-like robots using a probabilistic learning approach. *Int. J. of Robotics Research*, 16, 1995.

[50] Jack Tisdale. *Cooperative Sensing and Control with Unmanned Aerial Vehicles*. PhD thesis, University of California, Berkeley, 2008.

[51] J.N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *Automatic Control, IEEE Transactions on*, 40(9):1528 –1538, sep 1995.