

FLIGHT PLAN GENERATION FOR UNMANNED AERIAL  
VEHICLES

by

ANDREA L. NOONAN

B.S., Kansas State University, 2005

---

A THESIS

submitted in partial fulfillment of the  
requirements for the degree

MASTER OF SCIENCE

Department of Mechanical and Nuclear Engineering  
College of Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

2007

Approved by:

Major Professor  
Dr. Dale Schinstock

# Abstract

The goal of this research is to develop methods and tools for generating flight plans for an unmanned aerial vehicle (UAV). A method of generating flight plans is needed to describe data collection missions, such as taking aerial photographs. The flight plans are two-dimensional and exist in a plane a fixed distance above the Earth. Since the flight areas are typically small, the Earth's curvature is not accounted for in flight plan generation. Designed to completely cover a specified field area, the plans consist of a series of line and arc segments and are described in a format that is recognized by the Piccolo autopilot used by the Kansas State University Autonomous Vehicle Systems (AVS) Lab. Grids are designed to cover the field area, and turn maneuvers are designed to ensure efficient flight plans.

The flight plan generation process is broken into several parts. Once a field area is defined, path lines covering this area are calculated. Optimal turn maneuvers are calculated to smoothly connect the path lines in a continuous flight plan. Two methods of determining path line order are discussed. One method flies the lines in the order that they are arranged spatially; the other method decides line order by calculating the shortest turn maneuver to another path line. After the flight plan is generated, a text file is created in a format that is readable for the autopilot. In order to easily generate flight plans, a graphical user interface (GUI) has been created. This GUI allows a user to easily generate a flight plan without modifying any code. The flight plan generation software is used to build example flight plans for this thesis. These flight plans were flown with an UAV and test results are presented.

# Table of Contents

<b>Table of Contents</b>	<b>iii</b>
<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vii</b>
<b>Nomenclature</b>	<b>viii</b>
<b>Acknowledgements</b>	<b>x</b>
<b>Dedication</b>	<b>xi</b>
<b>1 Introduction and Background Information</b>	<b>1</b>
1.1 Project Description . . . . .	2
1.2 ECat UAV and Autopilot . . . . .	3
1.3 Coordinated Turn and Minimum Turn Radius . . . . .	4
1.4 Coordinate Systems and Conversions . . . . .	6
1.4.1 Conversion between ENU and NED . . . . .	6
1.4.2 Conversion between NED and ECEF . . . . .	7
1.4.3 Conversion between ECEF and LLA . . . . .	10
<b>2 Path Generation</b>	<b>13</b>
2.1 Description of Parameters . . . . .	13
2.1.1 Generating the Perimeter Lines . . . . .	14
2.1.2 Generating the Path Lines . . . . .	15

2.1.3	Path Line End Point Ordering . . . . .	18
2.2	Organizing the Flight Path Output File . . . . .	24
<b>3</b>	<b>Corner Turning Algorithm</b>	<b>27</b>
3.1	Motivation for the Corner Turning Algorithm . . . . .	28
3.2	Inputs Needed to Calculate an Optimal Turn Maneuver . . . . .	28
3.3	Three-Turn Solutions . . . . .	29
3.4	Two Turn Solutions with Straight Line Segment . . . . .	33
3.5	The Quickest Solution . . . . .	37
<b>4</b>	<b>Flight Plan Generation Program and Flight Test Results</b>	<b>39</b>
4.1	Three Passes GUI . . . . .	40
4.2	Generate Box GUI . . . . .	43
4.3	Generate Polygon GUI . . . . .	44
4.4	Flight Test Results . . . . .	46
<b>5</b>	<b>Conclusions and Recommendations</b>	<b>49</b>
	<b>Bibliography</b>	<b>52</b>
<b>A</b>	<b>Matlab Code</b>	<b>53</b>
A.1	Calculating Intersections between Path Lines and Perimeter Lines . . . . .	53
A.2	Efficiently Ordering Path Lines . . . . .	55
A.3	Turn Maneuver Generation Algorithm . . . . .	57

# List of Figures

1.1	ECat UAV . . . . .	3
1.2	Determining minimum turn radius . . . . .	5
1.3	Conversion from ECEF to NED . . . . .	9
1.4	Conversion from ECEF to LLA . . . . .	11
2.1	Demonstration of Variables Used in Path Generation Process . . . . .	14
2.2	Vertices and Perimeter of Field Area . . . . .	15
2.3	Box Surrounding Field Area and Unaltered Path Lines . . . . .	16
2.4	X and Y Spacing between Path Lines . . . . .	17
2.5	Path Lines Intersecting Perimeter Lines . . . . .	19
2.6	First Step in Ordering Path Line End Points . . . . .	20
2.7	Second Step in Ordering Path Line End Points . . . . .	21
2.8	Path Lines with Turn Maneuvers . . . . .	22
2.9	Possible Turn Maneuvers after Flying First Path Line . . . . .	23
2.10	Flight Path with Efficiently Ordered Path Lines . . . . .	24
3.1	Comparison of Autopilot-Generated and Optimal Turn Maneuvers . . . . .	28
3.2	An Arbitrary Three Turn Solution . . . . .	30
3.3	Illustration of (3.10) . . . . .	33
3.4	An Arbitrary Two Turn Solution . . . . .	34
3.5	Flowchart for Determining Quickest Turning Maneuver Solution . . . . .	38
4.1	Start Window of the Flight Plan Generation GUI . . . . .	40

4.2	Three Passes Flight Plan Generation GUI . . . . .	42
4.3	Generate Box Flight Plan Generation GUI . . . . .	44
4.4	Generate Polygon Flight Plan Generation GUI . . . . .	46
4.5	Test Flight with Path Lines Flown in Order . . . . .	47
4.6	Test Flight with Path Lines Flown Efficiently . . . . .	48

# List of Tables

2.1	Lengths of Turn Maneuvers in Figure (2.9) . . . . .	23
2.2	Total Flight Path Lengths in Figures (2.8) and (2.10) . . . . .	24

# Nomenclature

## Symbols

$\alpha$  - angle between due north and a path line, measured from due north

$\theta_A$  - angle subtended by the first arc in a three- or two-turn maneuver

$\theta_B$  - angle subtended by the second arc in a three-turn maneuver

$\theta_C$  - angle subtended by the third arc in a three- or two-turn maneuver

$\theta_d$  - angle of the vector from the initial point to the final point of a turn maneuver

$\theta_f$  - final heading of a turn maneuver

$\theta_i$  - initial heading of a turn maneuver

$\phi_{lat}$  - geodetic latitude

$\phi_{roll}$  - bank (or roll) angle of an aircraft

$a$  - length of the semimajor axis of the Earth,  $a = 6378137m$

$b$  - length of the semiminor axis of the Earth,  $b = 6356752m$

$C_{e/n}$  - rotation matrix to convert from NED to ECEF

$C_{n/e}$  - rotation matrix to convert from ECEF to NED

$D$  - magnitude of the vector from initial point to final point of a turn maneuver

$e$  - eccentricity of the Earth,  $e = 0.08181919$

$g$  - acceleration due to gravity,  $g = 9.81m/s^2$

$h$  - geodetic height (height above the spheroid)

$L$  - lift vector of the UAV

$l$  - longitude

$L_H$  - horizontal component of lift

$L_V$  - vertical component of lift

$N$  - prime vertical radius of curvature

$r$  - minimum turning radius of the UAV

## Acronymns

**AVS Lab** - Autonomous Vehicle Systems Laboratory

**CCW** - counter clockwise

**CW** - clockwise

**ECEF** - Earth Centered Earth Fixed

**ENU** - local coordinate system, East, North, Up correspond to XYZ

**GUI** - graphical user interface

**LLA** - Latitude, Longitude, Altitude

**NED** - North, East Down

**UAV** - unmanned aerial vehicle

**UGV** - unmanned ground vehicle

**WGS-84** - World Geodetic System 1984, model of the Earth used in this work.

## Definitions

**ECat** - electric UAV used by the AVS Lab

**Field Area** - target area for the UAV to survey

**Path Line** - a straight line portion of the flight plan that transverses the field area, all path lines of a flight plan are parallel

**Perimeter** - boundary of a field area

**Spacing** - the perpendicular distance seperating path lines

**Three-turn maneuver** - turning maneuver consisting of three arcs flown in alternating directions

**Two-turn maneuver** - turning maneuver consisting of an arc, staight line, arc sequence

**Waypoint** - the coodinates of a specific point in a flight plan

# Acknowledgments

My thanks are due to the professors of Kansas State University, especially my major professor: Dr. Dale Schinstock; the professors involved with the AVS Lab: Dr. Chris Lewis and Dr. Garth Thompson; and the professor responsible for many of my graduate courses: Dr. Warren White. Thank you for all your patience, help, and support.

Thanks also to Craig and Dustin for for keeping the AVS Lab atmosphere, well, interesting.

# Dedication

This thesis is dedicated to my family, including the family I have had for years and the one I have recently gained. And especially to my swell husband Francis. Thank you for your continued love, support and guidance.

# Chapter 1

## Introduction and Background

### Information

This thesis describes a method of generating flight plans for a small UAV used in remote sensing applications. The flight plans are designed to allow a sensor (such as a camera) to collect enough data to completely cover a desired field area. The flight plans take into account the turning capabilities of the UAV. They are constructed in a two-dimensional plane with a fixed altitude. Since the flight plans are assumed to be small (typically covering less than two square kilometers), the curvature of the earth is neglected over the area of the flight plan. This thesis describes the process of constructing parallel path lines, with appropriate spacing, over the field area, as well as the process of connecting the path line segments with turning maneuvers.

There is considerable work presented in the open literature addressing path planning for unmanned vehicles, including UAVs and unmanned ground vehicles (UGVs). For UGVs much of this work deals with path planning for tasks such as navigating in buildings. This work has little relation to the path planning performed here. There is some related work for UGVs dealing with the complete coverage of an area, such as work for de-mining and

cleaning robots. See [1]. These robots operate on the ground, typically near dangerous obstacles (such as staircases or mines). These obstacles can be known beforehand or the robot may discover them during its mission. This literature tends to focus on methods for detecting and avoiding obstacles. Also, since a UGV can stop and turn itself around, turn maneuvers are not usually addressed. In this thesis, UAV flight areas are assumed to be free of obstacles, and turning maneuvers must be addressed since a fixed-wing UAV cannot simply stop and turn. There is some UAV path planning literature that borrows ideas from ground robot path planning to generate flight paths that follow a course through an area with dangerous obstacles (i.e. anti-aircraft devices). However, no literature or method was found to generate a complete coverage flight path over obstacle-free skies. The Cloud Cap Technologies Piccolo II autopilot used in this research includes a waypoint navigator and a process of entering waypoints, but a means of automatically generating flight paths is not included. This thesis presents a method to generate a set of waypoints that describe a complete coverage flight plan that also takes aircraft turning capabilities into account.

The remainder of this chapter is dedicated to describing project background information pertinent to the flight plan generation process. Chapter 2 discusses the process of generating path line segments to cover a field area. Chapter 3 describes a method of calculating optimal turn maneuvers to connect the path lines. The flight path generation software and flight test results are described in Chapter 4. Chapter 5 outlines conclusions and recommendations for this project. The appendix contains Matlab code written in support of this project

## 1.1 Project Description

One of the current projects of the AVS Lab at Kansas State University is remote sensing of the Konza Prairie near Manhattan, Kansas. The AVS Lab utilizes a small unmanned aerial vehicle to fly over the prairie and collect photographs. These photographs are assembled into a large composite or mosaic and are used to monitor the status of the prairie.



**Figure 1.1:** *ECat UAV*

Enough properly placed photographs need to be taken to ensure that the entire target area is completely covered with sufficient overlap.

The ECat is equipped with an autopilot and navigator that is able to fly a flight plan consisting of lines and arcs. This thesis describes a method of generating the lines and arcs that comprise the flight plan. It discusses a method of constructing straight line segments that cover a target field area as well as methods of connecting the straight line segments with turning maneuvers. Assembling the lines and arcs into a flight plan that can be imported into the autopilot is also described.

## **1.2 ECat UAV and Autopilot**

ECat is the name of the electric UAV used by the AVS Lab. ECat has a wingspan of approximately 2 meters and weighs approximately 7 kilograms. The ECat airframe is modified from a Sig Kadet Senior hobby-grade remote control airplane kit. The modifications include a larger payload bay and landing skids. ECat is pictured in Figure 1.1. ECat is equipped with the commercially available Cloud Cap Technologies Piccolo II avionics package.

### 1.3 Coordinated Turn and Minimum Turn Radius

For an aircraft in a coordinated turn, the turning radius, bank angle, and speed are all related as indicated in Figure 1.2. Typically, the speed of the UAV is known. Also, the bank angle limits of the autopilot are known. In order to find an optimal turning maneuver for the UAV, the minimum turning radius of the aircraft must be determined. Figure 1.2 provides a reference for this calculation. If the aircraft is banked by an angle  $\phi_{roll}$ , the vertical component of lift,  $L_V$ , is given by

$$L_V = L \cdot \cos(\phi_{roll}), \quad (1.1)$$

where  $L$  is the lift force. Assuming the altitude is constant during the turn, the vertical component of lift is equal to the gravitational force,  $mg$ . Therefore

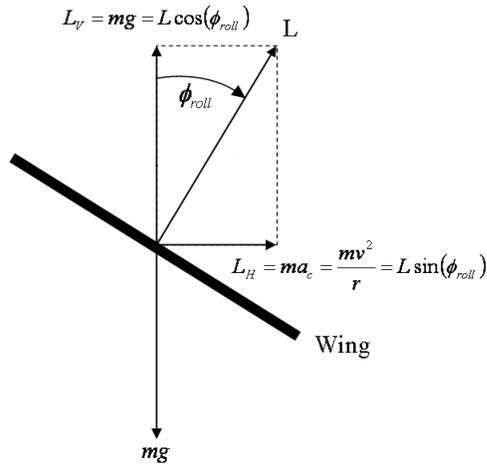
$$mg = L \cdot \cos(\phi_{roll}). \quad (1.2)$$

During a coordinated turn, the lift force is perpendicular to the wing. Therefore, with a constant velocity, all of the aircraft's acceleration is due to centripetal acceleration. Thus the horizontal component of lift  $L_H$  is

$$\frac{mv^2}{r} = L \cdot \sin(\phi_{roll}), \quad (1.3)$$

where  $m$  and  $v$  are the mass and speed of the UAV, and  $r$  is the turn radius. Dividing 1.3 by 1.2, and rearranging yields the minimum turning radius as:

$$r = \frac{v^2}{g \tan(\phi_{roll})}. \quad (1.4)$$



**Figure 1.2:** *Determining minimum turn radius*

The ECat typically flies at approximately 15 m/s. In order to achieve a minimum turn radius, the aircraft should bank as much as possible. The autopilot limits for the bank angle are set to  $30^\circ$ . Choosing a maximum bank angle of  $30^\circ$  will not give the autopilot authority to make corrections, so a bank angle less than  $30^\circ$  is needed for reasonable performance. A bank angle of  $15^\circ$  is a reasonable choice. It allows for autopilot corrections and provides for good performance with mild winds. For a  $15^\circ$  bank angle and 15 m/s airspeed, the turn radius is approximately 85 m. This turn radius has proved reasonable in flight conditions with mild winds. However, for stronger winds, increasing the turning radius results in a less aggressive turning command and allows the autopilot more control authority, therefore improving the aircraft's tracking ability.

## 1.4 Coordinate Systems and Conversions

There are four coordinate systems used for creation of the flight paths presented in this thesis. These include LLA, ECEF, NED, and ENU. The NED and ENU frames are local coordinate frames. The origin of these local frames is arbitrary. Usually the origin is chosen to be on the surface of the Earth, conveniently placed to calculate the flight plan. The X, Y, and Z axes of the ENU frame point east, north, and up, respectively. The flight plans are calculated in the ENU frame. This choice was made because this frame is more intuitive than the NED frame. The autopilot requires that the coordinates of waypoints be in LLA coordinates, so it is necessary to convert from a local ENU frame to a global LLA frame. Converting to NED and ECEF frames are intermediate steps in this process. These coordinate systems and the transformations between them are well-known and presented in many other works including [2]. The information is re-presented here for easy reference.

### 1.4.1 Conversion between ENU and NED

Converting from the ENU to the NED frame requires two rotations. A  $180^\circ$  rotation about the ENU frame's X-axis will align the ENU Z-axis with the NED Z-axis. Next, a  $-90^\circ$  rotation about the new Z-axis will align the X- and Y- axis with the NED frame. Thus the rotation matrix that describes the conversion from ENU to NED is written as

$$C_{NED/ECEF} = \begin{bmatrix} \cos(-90^\circ) & \sin(-90^\circ) & 0 \\ -\sin(-90^\circ) & \cos(-90^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(180^\circ) & \sin(180^\circ) \\ 0 & -\sin(180^\circ) & \cos(180^\circ) \end{bmatrix} \quad (1.5)$$

or

$$C_{NED/ECEF} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}. \quad (1.6)$$

Converting from NED to ENU will require taking the transpose of equation 1.6. The conversion from ENU to NED is simple and can also be performed by switching the X- and Y-coordinates of a waypoint and multiplying the Z-coordinate by  $-1$ . Thus a waypoint with ENU coordinates  $[A \ B \ C]$  becomes  $[B \ A \ -C]$  in NED coordinates.

### 1.4.2 Conversion between NED and ECEF

Figure 1.3 depicts the relationship between the ECEF and NED frames. The origin of the ECEF coordinate system is at the center of the earth. The X-axis of the ECEF frame points to the intersection of the equator and prime meridian, and the Z-axis points to the north pole. The Y-axis is perpendicular to the X-axis and Z-axis and its direction is determined by a right-handed coordinate system. Since the NED frame typically has a different orientation than the ECEF frame and its origin is typically not located at the earth's center, conversion from the ECEF frame to NED frame requires both a rotation and a translation.

The rotation that describes the ECEF frame with respect to the the NED frame, denoted by  $C_{n/e}$ , can be described by a series of three rotations. The angles  $l$  and  $\phi_{lat}$  correspond to the longitude and latitude coordinates of the origin of the NED frame. Typically, the location of the NED frame is chosen to coincide with the LLA coordinates of the autopilot's ground station. To change the orientation of a frame coincident with the ECEF frame to align with the NED frame requires three rotations. The first rotation is a negative rotation about the ECEF Z-axis by the angle  $l$ . Next, a  $-90^\circ$  rotation about the new Y-axis aligns the X-axis with north. Finally another rotation about the Y-axis by the angle  $\phi_{lat}$  aligns

the Z-axis with the down direction. The set of these three rotations can be written as:

$$C_{n/e} = \begin{bmatrix} \cos(\phi_{lat}) & 0 & \sin(\phi_{lat}) \\ 0 & 1 & 0 \\ -\sin(\phi_{lat}) & 0 & \cos(\phi_{lat}) \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \cos(l) & \sin(l) & 0 \\ -\sin(l) & \cos(l) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1.7)$$

or

$$C_{n/e} = \begin{bmatrix} -\sin(\phi_{lat}) \cos(l) & -\sin(\phi_{lat}) \sin(l) & \cos(\phi_{lat}) \\ -\sin(l) & \cos(l) & 0 \\ -\cos(\phi_{lat}) \cos(l) & -\cos(\phi_{lat}) \sin(l) & -\sin(\phi_{lat}) \end{bmatrix}. \quad (1.8)$$

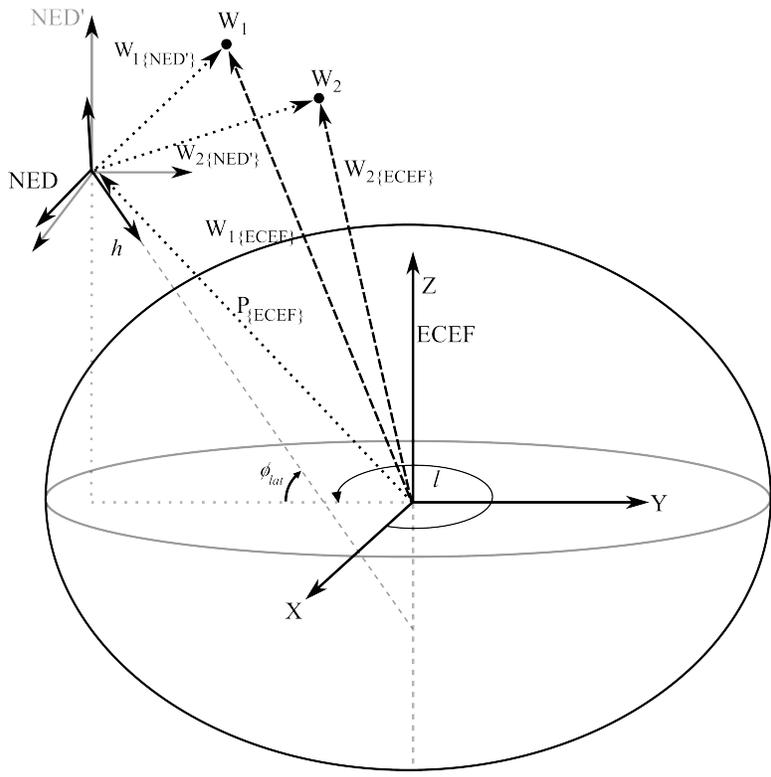
The transpose of equation 1.8,  $C_{n/e}^T = C_{e/n}$ , gives the rotation matrix describing the orientation of the ECEF frame with respect to the NED frame.

The rotation matrix  $C_{e/n}$  describes a set of rotations required rotate the NED frame to the orientation of the NED' frame as shown in Figure 1.3. The NED' frame is aligned with the ECEF frame. Waypoints  $W_1$  and  $W_2$ , originally described in the NED frame in Figure 1.3, are now described in the NED' frame as  $W_{1\{NED'\}}$  and  $W_{2\{NED'\}}$ . The location of the points  $W_1$  and  $W_2$  has not changed, but the method of describing their location has.

In order to describe these waypoints in the ECEF frame,  $P_{\{ECEF\}}$  must be added to both  $W_{1\{NED'\}}$  and  $W_{2\{NED'\}}$ . Equation 1.9 summarizes this operation.

$$\begin{aligned} W_{1\{ECEF\}} &= C_{e/n} \cdot W_{1\{NED\}} + P_{\{ECEF\}} = W_{1\{NED'\}} + P_{\{ECEF\}} \\ W_{2\{ECEF\}} &= C_{e/n} \cdot W_{2\{NED\}} + P_{\{ECEF\}} = W_{2\{NED'\}} + P_{\{ECEF\}} \end{aligned} \quad (1.9)$$

If the waypoint locations are known in the ECEF frame and the waypoints' locations with



**Figure 1.3:** Conversion from ECEF to NED

respect to the NED are needed, the following equation is used:

$$\begin{aligned} W_{1\{NED\}} &= C_{n/e} \cdot (W_{1\{ECEF\}} - P_{\{ECEF\}}) \\ W_{2\{NED\}} &= C_{n/e} \cdot (W_{2\{ECEF\}} - P_{\{ECEF\}}) \end{aligned} \quad (1.10)$$

### 1.4.3 Conversion between ECEF and LLA

Conversion between ECEF and LLA takes the ellipsoidal shape of the earth into account. WGS-84 model of the Earth describes the Earth as an ellipsoid with the length of the semimajor ellipsoid axis as  $a = 6378137$  m and the length of the semiminor axis as  $b = 6356752$  m. Using these values for  $a$  and  $b$ , the eccentricity of the Earth is derived as

$$e = \frac{(a^2 - b^2)^{1/2}}{a} \approx 0.08181919. \quad (1.11)$$

The prime vertical radius of curvature of the Earth is

$$N = \frac{a}{(1 - e^2 \sin^2(\phi_{lat}))^{1/2}}. \quad (1.12)$$

$N$  is the length of a line normal to the spheroid from the surface of the spheroid to the semiminor axis. Conversion from LLA to ECEF is calculated by

$$W_{ECEF} = \begin{bmatrix} (N + h) \cdot \cos(\phi_{lat}) \cdot \cos(l) \\ (N + h) \cdot \cos(\phi_{lat}) \cdot \sin(l) \\ (N(1 - e^2) + h) \cdot \sin(\phi_{lat}) \end{bmatrix} \quad (1.13)$$

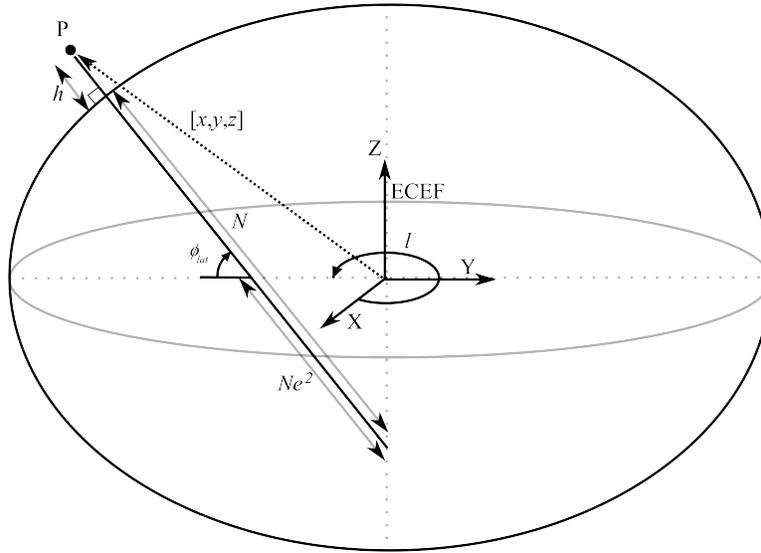
where latitude, longitude, and altitude are represented as  $\phi_{lat}$ ,  $l$  and  $h$ , respectively.

Converting from ECEF to LLA is accomplished by using an iterative algorithm. Refer-

ring to Figure 1.4,  $\sin(\phi_{lat})$  can be written

$$\sin(\phi_{lat}) = \frac{z}{N - Ne^2 + h} \quad (1.14)$$

where  $z$  is the Z-coordinate of point  $P$  in the ECEF frame. Likewise, from the triangle



**Figure 1.4:** Conversion from ECEF to LLA

formed with hypotenuse  $(h + N)$  and sides  $(z + Ne^2 \sin(\phi_{lat}))$  and  $(\sqrt{x^2 + y^2})$ ,

$$\tan(\phi_{lat}) = \frac{z + Ne^2 \sin(\phi_{lat})}{\sqrt{x^2 + y^2}} \quad (1.15)$$

can be written. Substituting equation 1.14 into equation 1.15 for  $\sin(\phi_{lat})$  yields

$$\tan(\phi_{lat}) = \frac{z}{(\sqrt{x^2 + y^2}) [1 - Ne^2 / (N + h)]}. \quad (1.16)$$

Since  $N$  is a function of  $\phi_{lat}$ , this iterative algorithm is used to solve for  $\phi_{lat}$ :

$$L = \tan^{-1}(y/x)$$

$$N = a$$

$$h = 0$$

$$\phi_{lat} = 1$$

**while**  $\Delta > tolerance$  **do**

$$\Delta = \phi_{lat} - \tan^{-1} \left[ \frac{z}{\sqrt{x^2+y^2}[1-Ne^2/(N+h)]} \right]$$

$$\phi_{lat} = \phi_{lat} - \Delta$$

$$N = \frac{a}{(1-e^2 \sin^2(\phi_{lat}))^{1/2}}$$

$$(N + h) = \frac{\sqrt{x^2+y^2}}{\cos(\phi_{lat})}$$

**end while**

$$\text{altitude} = h$$

$$\text{longitude} = L$$

$$\text{latitude} = \phi_{lat}$$

This algorithm is implemented in Matlab and a tolerance of  $1 \times 10^{-14}$  is used. Typically, the algorithm converges in 3 to 6 iterations.

# Chapter 2

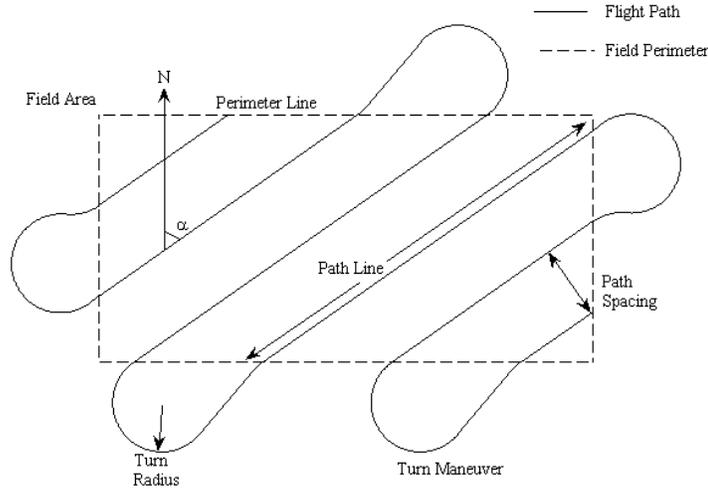
## Path Generation

This chapter discusses the flight plan generation process. The flight plans are created in a two-dimensional plane parallel to the surface of the Earth. Flight plans are assumed to be small so the curvature of the Earth is not taken into account. This chapter begins with a description of parameters used in the flight generation process and explains the steps for creating the flight paths over convex polygon field areas. In order to illustrate the path generation process, an example flight plan is generated with an example field area. Flight plans are created in the local ENU frame and are later converted to LLA coordinates.

### 2.1 Description of Parameters

Figure 2.1 illustrates variables used for calculating a flight path. The rectangle outlined with a dotted line represents the perimeter of an area to be surveyed. This area can be any convex polygon. Each of the sides of this area is known as a perimeter line. The flight path is shown as a solid line. The flight path is a combination of path lines and turn maneuvers.

A path line is a straight line that transverses the field area. All of the path lines are parallel. The endpoints of the path lines are on the perimeter of the field area. All path lines



**Figure 2.1:** *Demonstration of Variables Used in Path Generation Process*

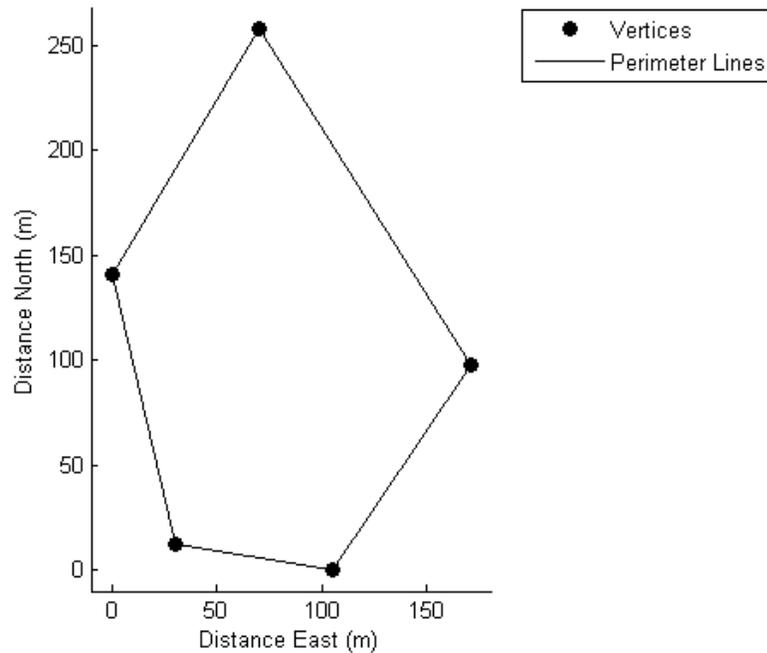
are oriented at an angle  $\alpha$  measured from North. The angle  $\alpha$  is restricted to be between 0 and  $\pi$  radians (0 and 180° degrees). It can be chosen so that the path lines are aligned with the direction of the wind or based on other restrictions. Path spacing is the perpendicular distance between two path lines. This distance is usually chosen to allow proper overlap between camera images.

Turn maneuvers connect path lines to form a continuous flight path. There are two types of turn maneuvers: three-turn and two-turn maneuvers. Calculation of the turn maneuvers is discussed in Chapter 3.

### 2.1.1 Generating the Perimeter Lines

The first step in generating a flight path is determining the field area. Field areas may be any convex polygon. A shape is convex if a line connecting any two of its vertices lies

completely within, or on the perimeter of, the field area. Figure 2.2 shows vertices and perimeter lines of an example field area. Without loss of generality, the origin of the ENU coordinate system is located so that all vertices are located in the first quadrant of the Cartesian plane. The vertices are listed in order going clockwise around the field area. This order is chosen for convenience, but without loss of generality. For this example, the vertices are  $(70, 258.3)$ ;  $(171, 98)$ ;  $(104.8, 0)$ ;  $(30, 12.8)$ ;  $(0, 140.7)$ . Local coordinates are used for convenience, and all flight plan waypoints will later be converted to LLA coordinates. If the vertices of the field area are known in LLA (or in some other coordinate system) they must be converted to ENU before the flight plan is calculated.



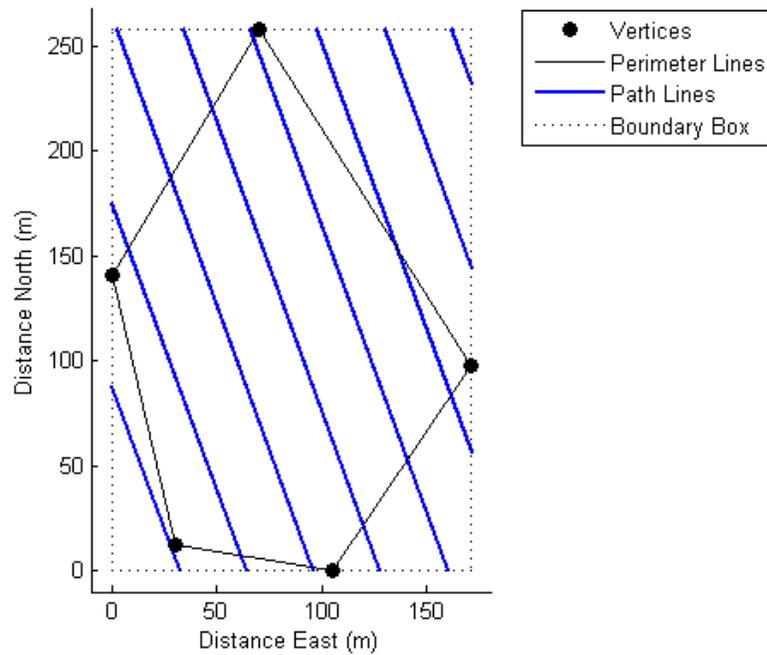
**Figure 2.2:** *Vertices and Perimeter of Field Area*

### 2.1.2 Generating the Path Lines

Once the field area and its perimeter is defined, the path lines that transverse the field area are generated. This task is completed in two steps. First, a rectangle with vertical and

horizontal sides that contain all the vertices is constructed and path lines are created to fill in this rectangle, as depicted in Figure 2.3. Then the intersections between the path lines and perimeter lines are calculated to form the altered path lines shown in Figure 2.5.

Constructing the boundary box is straightforward. The upper and lower edges of the boundary box correspond to maximum and minimum y-coordinates of the vertices. The right and left edges correspond to the maximum and minimum x-coordinates of the vertices.



**Figure 2.3:** *Box Surrounding Field Area and Unaltered Path Lines*

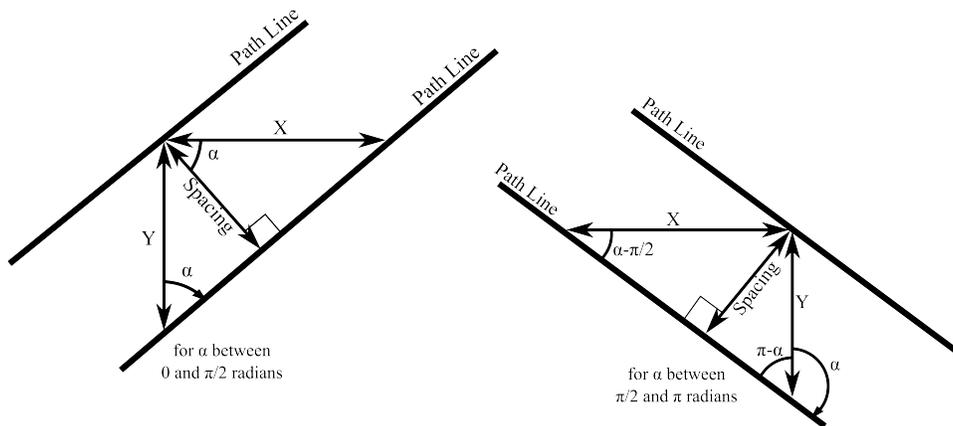
The path lines that fill the boundary box are defined by  $\alpha$ , by their end points on the boundary box as well as the spacing between the lines. If  $0 < \alpha < \pi/2$ , then the path lines are constructed beginning with the top left corner of the boundary box working toward the bottom right of the boundary box. If  $\pi/2 < \alpha < \pi$ , then the path lines are constructed from the bottom left corner working toward the top right corner of the boundary box. (If the path lines are vertical, i.e.  $\alpha = 0$ , path lines start at the left and work right. If path

lines are horizontal, i.e.  $\alpha = \pi/2$ , path lines start at the top and work down.) Referring to Figure 2.4 the horizontal and vertical spacing between the lines,  $X_{sp}$  and  $Y_{sp}$  respectively, can be determined with the following equations:

$$X_{sp} = \left| \frac{spacing}{\cos(\alpha)} \right|, \quad (2.1)$$

$$Y_{sp} = \left| \frac{spacing}{\sin(\alpha)} \right|. \quad (2.2)$$

For the case shown in Figure 2.3,  $\alpha$  is equal to 2.79 radians ( $160^\circ$ ). This is between  $\pi/2$  and  $\pi$  so creation of the path lines begins in the lower left corner. The first path line created intersects the y-axis at a distance  $Y_{sp}$  above the bottom left boundary box corner and the second path line has y-intercept at  $2 \cdot Y_{sp}$  above the bottom left corner and so on. The end points of the path lines are chosen to intersect the boundary box.



**Figure 2.4:** *X and Y Spacing between Path Lines*

At this point, the path lines shown in Figure 2.3 are constructed. Now the intersections between path lines and perimeter lines must be found and the path lines appropriately redefined. To accomplish this, each path line is looked at in turn. As long as a path line

and a perimeter line are not parallel, they will intersect. If  $m_1$  and  $b_1$  represent the slope and y-intercept of the perimeter line and  $m_2$  and  $b_2$  represent the slope and y-intercept of the path line, the x-coordinate of the intersection will be

$$x = \frac{b_2 - b_1}{m_1 - m_2}. \quad (2.3)$$

The y-coordinate of the intersection is found using the equation for a line and equation 2.3 to yield

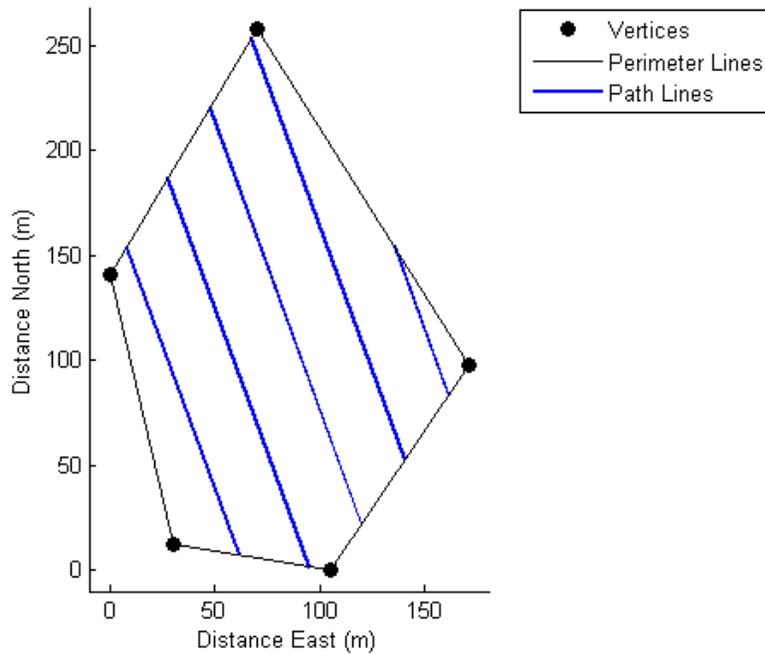
$$y = m_1 \frac{b_2 - b_1}{m_1 - m_2} + b_1. \quad (2.4)$$

If this intersection point,  $(x, y)$ , is on a perimeter line between two vertices, the intersection point and index of the path line are kept. If the intersection point coincides with an end point of a perimeter line, it is not automatically stored. The endpoints of the perimeter lines are listed in clockwise order around the perimeter of the field area. The intersection point is stored only if the intersection point corresponds to the first vertex of a perimeter line segment. This prevents a path line from intersecting more than two perimeter lines. Appendix A.1 includes Matlab code to calculate intersections between path lines and perimeter lines.

Once all of the intersections between perimeter lines and path lines are calculated, the path lines' end points are modified to represent these intersections. Figure 2.5 shows the result of this operation for the example field area.

### 2.1.3 Path Line End Point Ordering

The end points of the path lines need to be in a predictable format so that the travel direction along a path line can be determined. For path lines that are not vertical, the eastern-most

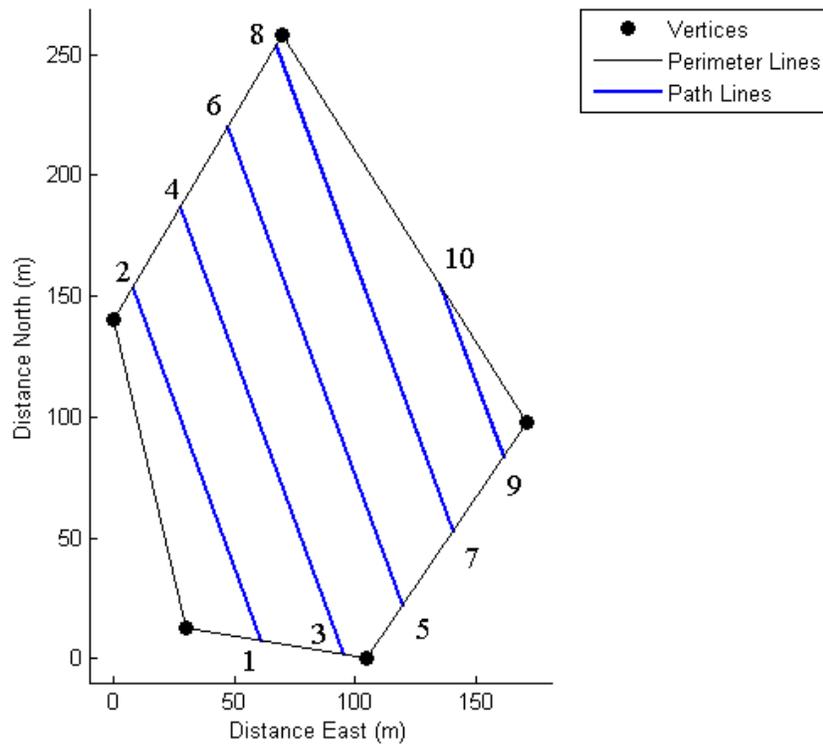


**Figure 2.5:** *Path Lines Intersecting Perimeter Lines*

end point of a path line is listed first. If the lines are vertical, then the southern-most point is listed first. This insures that the end points are listed in the order shown in Figure 2.6.

It is inefficient to fly to each of the end points in the order shown in Figure 2.6. There are two methods for choosing how to order these end points. The first method is straightforward and requires that the endpoints of every other line are re-ordered so that they are listed as shown in Figure 2.7. This results in the path lines being flown in order from one side of the field area to the other. After the end points are listed in order, turn maneuvers are calculated to connect the end points. The calculations necessary to generate the turn maneuvers are discussed Chapter 3. Calculating the turn maneuvers for this example results in the flight path shown in Figure 2.8.

The second method of ordering usually results in a more efficient (i.e. shorter) flight path. This second method takes the length of a turn maneuver into account. The endpoints of the first path line begin a list of waypoints that have been visited. A second list contains



**Figure 2.6:** *First Step in Ordering Path Line End Points*

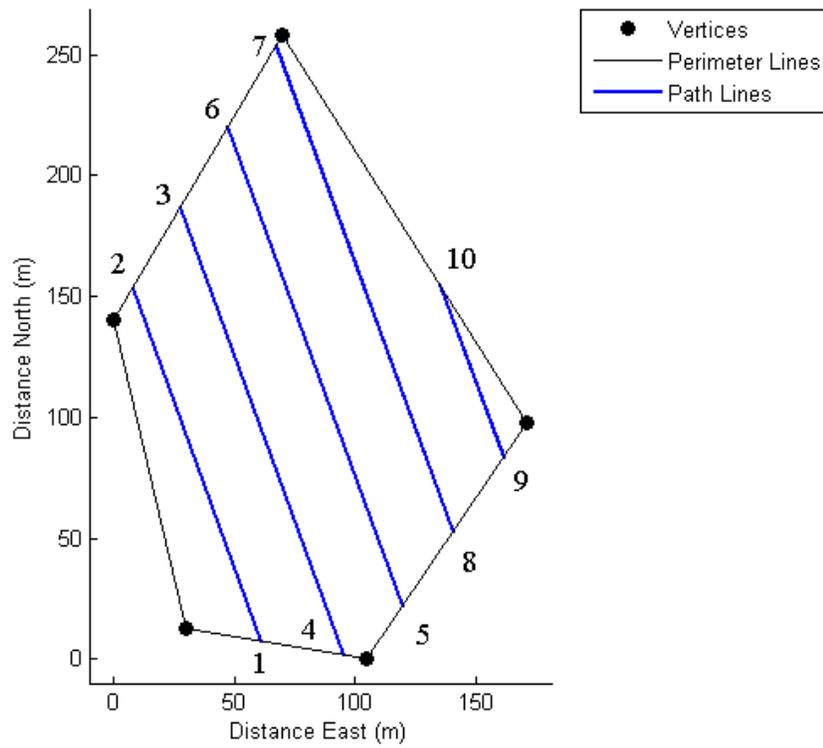
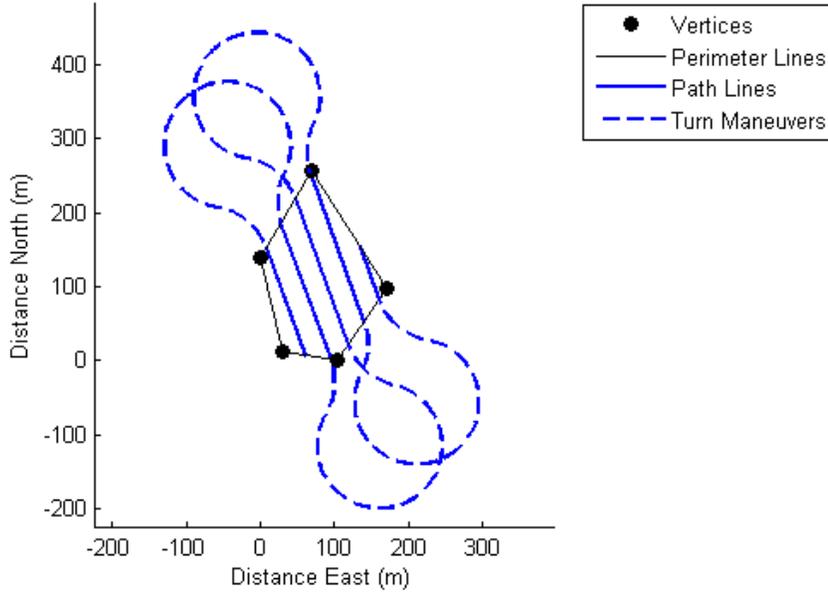


Figure 2.7: Second Step in Ordering Path Line End Points

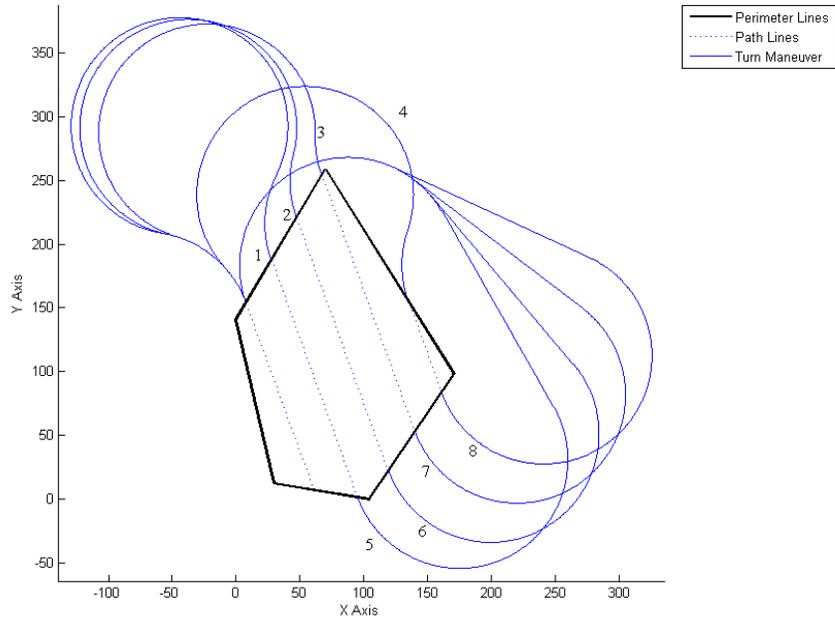


**Figure 2.8:** *Path Lines with Turn Maneuvers*

all the other endpoints not yet visited. After a path line is flown, its endpoints are added to the visited list and turn maneuvers from the last endpoint to each non-visited endpoints are calculated. The shortest of these possible turn maneuvers is included as the next turn maneuver and the endpoints of the next path line are included in the list of visited endpoints. This process repeats until all endpoints have been visited. The code describing this process is included in Appendix [A.2](#).

Figure [2.9](#) shows eight turn maneuver possibilities after the first path line has been flown for the example flight path. Table [2.1](#) lists the lengths of each turn maneuver shown in Figure [2.9](#). Comparing all eight possible turn maneuvers shows that turn number four to from the first path line to the fifth path line is the shortest maneuver. This path is chosen as the next turn maneuver. At the end of the fifth path line, the turn possibilities to the second, third, and fourth path lines are considered.

A flight path generated using this second path line ordering method results in the path

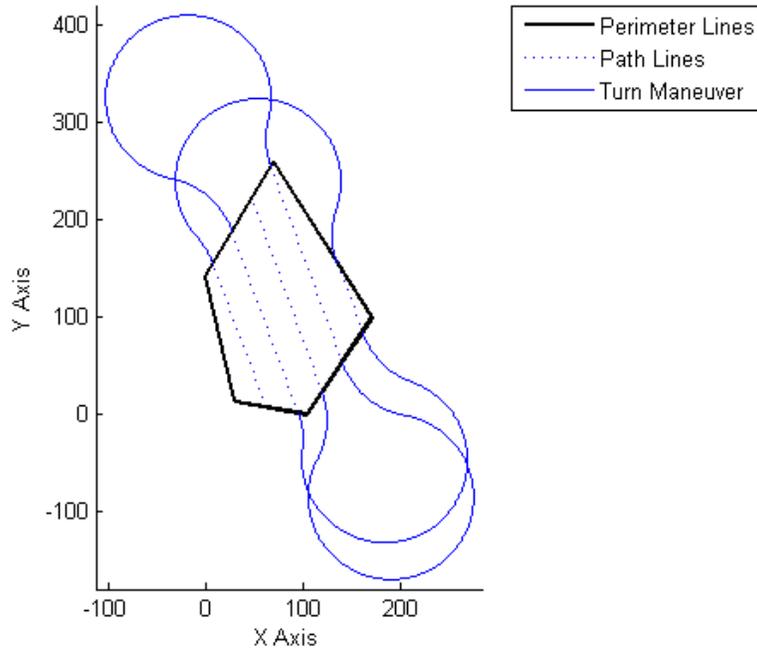


**Figure 2.9:** *Possible Turn Maneuvers after Flying First Path Line*

**Table 2.1:** *Lengths of Turn Maneuvers in Figure (2.9)*

Turn Number	Length of Turn
1	585.4 m
2	541.4 m
3	488.8 m
4	447.7 m
5	709.6 m
6	706.9 m
7	700.8 m
8	705.8 m

shown in Figure 2.10 The total flight path lengths for the flight paths in Figures 2.8 and 2.10 are listed in Table 2.2. Using the second method of choosing the order the path lines results in a flight path that is 272.4 meters shorter for this example.



**Figure 2.10:** *Flight Path with Efficiently Ordered Path Lines*

**Table 2.2:** *Total Flight Path Lengths in Figures (2.8) and (2.10)*

Method	Flight Path Length
Fly Path Lines in Order	3197.8 meters
Using Turn Maneuver Length	2925.4 meters

## 2.2 Organizing the Flight Path Output File

This section discusses how the flight plan generated in section 2.1 is assembled as an output file. This output file is then imported into the Cloud Cap Operator Interface.

The flight plans created in the previous section are a list of lines and arcs. Each line endpoint and arc center are waypoints in the flight plan. These waypoints are listed in the order that they will be flown and are described in LLA coordinates. In addition to location, information about turn direction, arc radius, and whether or not to take pictures is also included for each waypoint.

The Piccolo Operator Interface can import a text file containing all the waypoint data. This data is organized into an  $m \times 20$  matrix where  $m$  is the total number of waypoints in the flight plan. There is enough memory storage on the Piccolo for 55 waypoints. Column 1 represents the waypoint's index and is represented as

$$C_1 = [0 \quad 1 \quad 2 \quad \cdots \quad m - 1]^T. \quad (2.5)$$

The second column is the index of the next waypoint:

$$C_2 = [1 \quad 2 \quad \cdots \quad m - 1 \quad 0]^T \quad (2.6)$$

Columns 3, 4, and 5 contain the latitude, longitude and altitude, respectively, of each waypoint. The sixth and seventh columns are currently unused and all entries are set to 0. Column 6 entries are all set to  $-1$  and column 7 entries are all 0. Column 8 controls whether pictures should or should not be taken. Pictures are taken on straight line segments that are not part of a turn maneuver. Two points describe a line segment, and in order to take pictures along the line, the entry in the seventh column in the row corresponding to the second waypoint of the line segment is set to 1. Other row's entries are set to zero in this column. The next five columns are unused, and their entries are set to zero. Column 14 describes the direction of an arc in a turn maneuver. Clockwise (or right) turns and straight line segments are represented with a 0. Counterclockwise turns are denoted by 1. If a waypoint is an arc waypoint, column 15 contains the radius of that arc and column 16

contains the angle of that arc. The remaining four columns are also unused and their entries are all zero.

# Chapter 3

## Corner Turning Algorithm

This chapter discusses the process of calculating optimal turn maneuvers to connect the path lines described in Chapter 2. These turn maneuvers are two-dimensional (they are calculated in a horizontal plane above the ground), and they describe the shortest path from an initial point and heading to a final point and heading, given a limited turning radius. Since the aircraft's speed is held constant, these shortest-length paths are time-optimal. There are two categories of turn maneuvers that are generated: three-turn and two-turn maneuvers. The three-turn maneuvers consist of three successive arcs, and two-turn maneuvers involve flying a combination of arc-line-arc submaneuvers. Calculation of the three-turn maneuvers is described in Section 3.3 and the two-turn maneuvers are found in Section 3.4.

Optimal turn maneuver calculation techniques are addressed in other works. The method presented here is similar to [3] but does not use an iterative solution method. Other path planning methods, such as [4] determine waypoints on a course through a dangerous area based on risk assessment algorithms, but do not necessarily take the vehicle's turning capabilities into account. A method for smoothing flight path trajectories is presented in [5], but it relaxes the constraint that the vehicle passes through specified waypoints. Another method [6] addresses optimal turning but does not discuss three-turn solutions. Other meth-

ods, such as [5] and [7], find a minimum length by minimizing a cost function. Another work proves that a solution involving either three arcs of a minimum radius or two arcs and a line is optimal [8]. See also [9].

### 3.1 Motivation for the Corner Turning Algorithm

Figure 3.1 shows a field area for the ECat UAV to survey. If the only waypoints provided to the autopilot are the path line endpoints, the UAV will fly a path similar to the path near the bottom of Figure 3.1. The UAV can turn around but it takes a while for it to track the path line very well. However, using a series of arcs to describe an optimal turn maneuver provides an efficient way for the UAV to turn itself around, as shown by the optimal turn maneuvers near the top of Figure 3.1. The optimal turn maneuvers also properly align the UAV for the next path line.

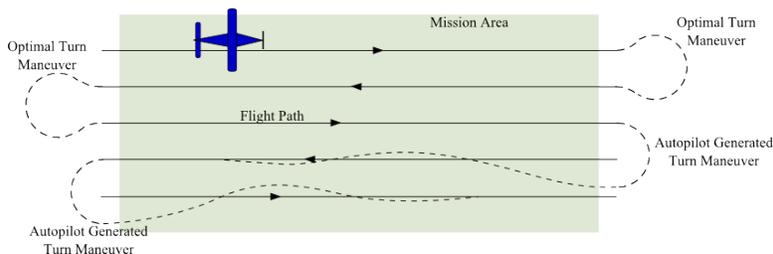


Figure 3.1: Comparison of Autopilot-Generated and Optimal Turn Maneuvers

### 3.2 Inputs Needed to Calculate an Optimal Turn Maneuver

In order to solve for the optimal turn solution presented here, the following quantities are needed: initial position; final position; initial heading,  $\theta_i$ ; final heading  $\theta_f$ ; and minimum turn radius. The minimum turn radius can be calculated by using the maximum allowable

bank angle and aircraft speed as described in section 1.3. Up to four three-turn solutions and up to four two-turn solutions are calculated. The shortest overall solution is chosen from these possible solutions.

### 3.3 Three-Turn Solutions

This section describes a method to find an optimal turn maneuver that consists of three arcs. Figure 3.2 shows an arbitrary three-turn solution to illustrate the parameters used in solving for the optimal turn solution. The initial and final headings,  $\theta_i$  and  $\theta_f$ , are known. The points  $\vec{A}$ ,  $\vec{B}$ , and  $\vec{C}$  are the centers of the three arcs that form this turn solution. The vector from the initial point to the final point is denoted by  $\vec{D}$ . The unknown subtended angles of each arc are  $\theta_A$ ,  $\theta_B$ , and  $\theta_C$ . All distance measurements are scaled to the turn radius, i.e., one unit of distance equals one turn radius.

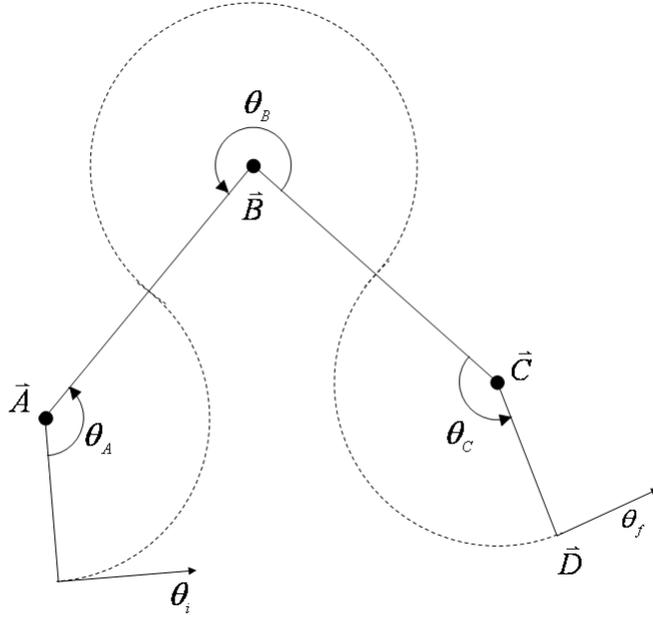
Taking the start of the first turn as the origin of the coordinate system, without loss of generality, the three arc path can be represented as the sum of five complex vectors as follows:

$$e^{j \cdot (\theta_i + s \cdot \frac{\pi}{2})} + 2 \cdot e^{j \cdot (\theta_i + s \cdot (\theta_A - \frac{\pi}{2}))} + 2 \cdot e^{j \cdot (\theta_i + s \cdot (\frac{\pi}{2} + \theta_A - \theta_B))} = \left| \vec{D} \right| e^{j \cdot \theta_D} + e^{j \cdot (\theta_f + s \cdot \frac{\pi}{2})} \quad (3.1)$$

where  $s$  denotes the direction of the starting turn and is equal to +1 for a CCW turn or -1 for a CW turn. Simplifying 3.1 by rotating by  $-\theta_i$  and by  $s \frac{\pi}{2}$ , rearranging the equation, and noting that  $e^{j s \pi} = -1$  regardless of  $s$  yields

$$e^{j \cdot s \cdot \theta_A} - e^{j \cdot s \cdot (\theta_A - \theta_B)} = \frac{1}{2} \left[ \left| \vec{D} \right| e^{j \cdot (\theta_D - \theta_i + s \cdot \frac{\pi}{2})} - e^{j \cdot (\theta_f - \theta_i)} + 1 \right]. \quad (3.2)$$

Since the right-hand side of 3.2 is known, the equation can be rewritten in the following



**Figure 3.2:** *An Arbitrary Three Turn Solution*

form:

$$e^{j \cdot s \cdot \theta_A} - e^{j \cdot s \cdot (\theta_A - \theta_B)} = G e^{j \cdot \theta_G} \quad (3.3)$$

where

$$G e^{j \cdot \theta_G} = \frac{1}{2} \left[ \left| \vec{D} \right| e^{j(\theta_D - \theta_i + s \cdot \frac{\pi}{2})} - e^{j(\theta_f - \theta_i)} + 1 \right]. \quad (3.4)$$

Rotating by  $-\theta_G$ , equation 3.3 can be rewritten as

$$e^{j \cdot a} - e^{j \cdot (a+b)} = G \cdot e^{j \cdot 0} \quad (3.5)$$

where

$$\begin{aligned} a &= s\theta_A - \theta_G \\ b &= -s\theta_B. \end{aligned} \tag{3.6}$$

From 3.4 there are two values for  $G$  and  $\theta_G$  corresponding to  $(G_1, \theta_{G_1})$  for  $s = 1$  and  $(G_2, \theta_{G_2})$  for  $s = -1$ . If  $G > 2$  then 3.5 has no solution and an arc-line-arc line solution is needed. This situation will be addressed in Section 3.4. Otherwise, the unknowns  $a$  and  $b$  may be solved for using Euler's identity ( $e^{j \cdot x} = \cos(x) + j \sin(x)$ ) and equating real and imaginary parts of 3.5 as follows:

$$\cos(a) - \cos(a + b) = G \tag{3.7}$$

and

$$\sin(a) = \sin(a + b). \tag{3.8}$$

Equation 3.8 reduces to

$$b = 2n\pi \tag{3.9}$$

where  $n = 0, 1, 2, \dots$  or

$$b = (2n + 1)\pi - 2a. \tag{3.10}$$

Equation 3.9 is an obvious solution of 3.8, and represents the case where a single arc is sufficient to optimally complete the maneuver. This case will be handled in the Two-Turn

solution where the line length equals zero and the subtended angle for this situation will be equal to  $\theta_f - \theta_i$ . Otherwise, equation 3.10 applies. Equation 3.10 is illustrated in Figure 3.3. A line intersecting the sine function as shown in Figure 3.3 indicates that  $\sin(a)$  equals  $\sin(a + b)$ , as stated in equation 3.8. From this, equation 3.10 is written. Substituting 3.10 into 3.7, and simplifying yields four solutions for  $a$ :

$$\begin{aligned}
a_1 &= +\cos^{-1}\left(\frac{G_1}{2}\right) \\
a_2 &= +\cos^{-1}\left(\frac{G_2}{2}\right) \\
a_3 &= -\cos^{-1}\left(\frac{G_1}{2}\right) \\
a_4 &= -\cos^{-1}\left(\frac{G_2}{2}\right).
\end{aligned} \tag{3.11}$$

Substituting 3.11 into 3.6 yields the angles  $\theta_A$ ,  $\theta_B$ . Noting that  $\theta_C$  is dependent on the difference between  $\theta_f$  and  $\theta_i$  as well as the angles  $\theta_A$  and  $\theta_B$ ,  $\theta_f - \theta_i = s \cdot (\theta_A - \theta_B + \theta_C)$  can be written. Solving for  $\theta_A$ ,  $\theta_B$ , and  $\theta_C$  yields

$$\begin{aligned}
\theta_{A1} &= +(a_1 + \theta_{G1}), & \theta_{B1} &= +(2a_1 - \pi), & \theta_{C1} &= +(\theta_f - \theta_i + a_1 - \pi - \theta_{G1}) \\
\theta_{A2} &= -(a_2 + \theta_{G2}), & \theta_{B2} &= -(2a_2 - \pi), & \theta_{C2} &= -(\theta_f - \theta_i + a_2 - \pi - \theta_{G2}) \\
\theta_{A3} &= +(a_3 + \theta_{G1}), & \theta_{B3} &= +(2a_3 - \pi), & \theta_{C3} &= +(\theta_f - \theta_i + a_3 - \pi - \theta_{G1}) \\
\theta_{A4} &= -(a_4 + \theta_{G2}), & \theta_{B4} &= -(2a_4 - \pi), & \theta_{C4} &= -(\theta_f - \theta_i + a_4 - \pi - \theta_{G2}).
\end{aligned} \tag{3.12}$$

Since values for  $\theta_A$ ,  $\theta_B$ , and  $\theta_C$  greater than  $2\pi$  do not provide an optimal solution, the magnitude of all angles is restricted to be between 0 and  $2\pi$ .

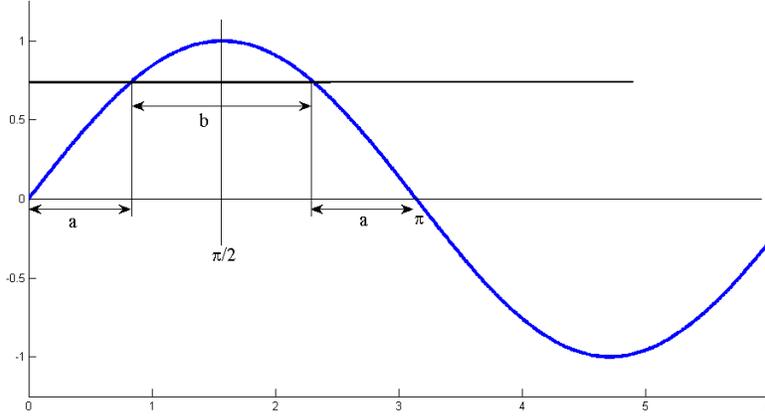


Figure 3.3: Illustration of (3.10)

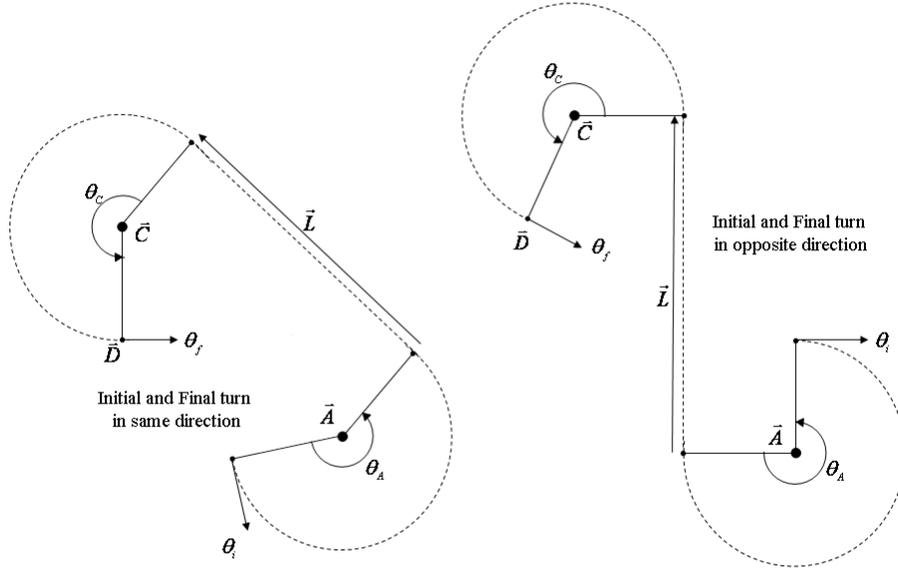
### 3.4 Two Turn Solutions with Straight Line Segment

As mentioned in section 3.3, if  $G > 2$ , then a straight line solution is needed. Also, there are some solutions where  $G < 2$  and a straight line is a part of the shortest path. In order for a straight line solution to occur with  $G < 2$ , the heading at the end of the initial turn and at the beginning of the final turn must be the same. When  $G = 2$ , a single arc is sufficient to complete the turn. The initial turn may be CW or CCW. Regardless of whether the initial and final turns are in the same direction or in opposite directions, the two-turn solutions are calculated the same way.

Similar to the solution method in Section 3.3, referring to Figure 3.4 the following vector loop equation is written:

$$e^{j \cdot (\theta_i + s_1 \frac{\pi}{2})} + e^{j \cdot (\theta_i - s_1 \frac{\pi}{2} + \theta_A)} + L \cdot e^{j \cdot (\theta_i + \theta_A)} + e^{j \cdot (\theta_i + \theta_A + s_2 \cdot \frac{\pi}{2})} = D \cdot e^{j \cdot \theta_D} + e^{j \cdot (\theta_f + s_2 \frac{\pi}{2})}. \quad (3.13)$$

Here  $s_1$  denotes the direction of the initial turn and  $s_2$  is the direction of the final turn.



**Figure 3.4:** *An Arbitrary Two Turn Solution*

Rearranging 3.13 results in

$$D \cdot e^{j \cdot \theta_D} + e^{j \cdot (\theta_f + s_2 \frac{\pi}{2})} - e^{j \cdot (\theta_i + s_1 \frac{\pi}{2})} = e^{j \cdot (\theta_i - s_1 \frac{\pi}{2} + \theta_A)} + L \cdot e^{j \cdot (\theta_i + \theta_A)} + e^{j \cdot (\theta_i + \theta_A + s_2 \frac{\pi}{2})} \quad (3.14)$$

or

$$G_{two-turn} \cdot e^{j \cdot \theta_{Gtwo-turn}} = e^{j \cdot (\theta_i - s_1 \frac{\pi}{2} + \theta_A)} + L \cdot e^{j \cdot (\theta_i + \theta_A)} + e^{j \cdot (\theta_i + \theta_A + s_2 \frac{\pi}{2})} \quad (3.15)$$

where

$$G_{two-turn} \cdot e^{j \cdot \theta_{Gtwo-turn}} = D \cdot e^{j \cdot \theta_D} + e^{j \cdot (\theta_f + s_2 \frac{\pi}{2})} - e^{j \cdot (\theta_i + s_1 \frac{\pi}{2})}. \quad (3.16)$$

Based on the values of  $s_1$  and  $s_2$ , there are up to four possibilities for  $G_{two-turn} e^{j \cdot \theta_{Gtwo-turn}}$ .

These possibilities correspond to the following values of  $s_1$  and  $s_2$ :

$$\begin{aligned}
s_1 &= +1, & s_2 &= +1 \\
s_1 &= -1, & s_2 &= -1 \\
s_1 &= +1, & s_2 &= -1 \\
s_1 &= -1, & s_2 &= +1.
\end{aligned} \tag{3.17}$$

The cases where  $s_1 = s_2$  and  $s_1 = -s_2$  will be looked at separately.

First, let  $s_1 = s_2$ . The first and third terms on the right-hand side of 3.15 represent vectors of equal magnitudes pointing in opposite directions. Therefore these terms cancel yielding

$$G_{two-turn} \cdot e^{j\theta_{Gtwo-turn}} = L \cdot e^{j(\theta_i + \theta_A)}. \tag{3.18}$$

For two vectors to be equal, both the magnitude and direction must be equal. Therefore

$$G_{two-turn} = L \tag{3.19}$$

and

$$\theta_A = \theta_{Gtwo-turn} - \theta_i \tag{3.20}$$

for  $s_1 = s_2$ . Since  $G_{two-turn}$  has two values ( one for  $s_1 = s_2 = +1$  and  $s_1 = s_2 = -1$ ),  $\theta_A$  will have two values.

Now let  $s_1 = -s_2$ . The first and third terms on the right-hand side of 3.15 are vectors

of equal length pointing in the same direction. Thus 3.15 becomes

$$G_{two-turn} \cdot e^{j \cdot \theta_{Gtwo-turn}} = 2e^{j \cdot (\theta_i + s_2 \frac{\pi}{2} + \theta_A)} + L \cdot e^{j \cdot (\theta_i + \theta_A)}. \quad (3.21)$$

Rotating 3.21 by  $-\theta_i - \theta_A$  yields

$$G_{two-turn} \cdot e^{j \cdot \theta_{Gtwo-turn} - \theta_i - \theta_A} = 2e^{j \cdot (s_2 \frac{\pi}{2})} + L \cdot e^{j \cdot 0}. \quad (3.22)$$

Using Euler's identity and simplifying yields

$$G_{two-turn} \cos(\theta_{Gtwo-turn} - \theta_i - \theta_A) = L \quad (3.23)$$

and

$$G_{two-turn} \sin(\theta_{Gtwo-turn} - \theta_i - \theta_A) = 2 \sin\left(s_2 \frac{\pi}{2}\right). \quad (3.24)$$

Since  $2 \cdot \sin\left(s_2 \frac{\pi}{2}\right)$  will equal  $-1$  for  $s_2 = -1$  and will equal  $1$  for  $s_2 = +1$ , 3.24 becomes

$$G_{two-turn} \sin(\theta_{Gtwo-turn} - \theta_i - \theta_A) = 2s_2 \quad (3.25)$$

Solving for  $\theta_A$  is accomplished by dividing equation 3.25 by 3.23 and by simplifying to get

$$\theta_A = \theta_{Gtwo-turn} - \theta_i - \tan^{-1}\left(\frac{2s_2}{L}\right). \quad (3.26)$$

Squaring 3.23 and 3.25 adding and rearranging the result yields

$$L = \pm \sqrt{G_{two-turn}^2 - 2^2}. \quad (3.27)$$

Since  $L$  represents a magnitude, the negative solution in 3.27 is invalid. If  $G_{two-turn} \geq 2$ , the  $L$  can be determined. If  $G_{two-turn} = 2$ , then  $L = 0$  and a single arc completes the turn maneuver. These equations are valid if  $s_1 = -s_2$ .

From equations 3.20 and 3.26, values for  $\theta_A$  can be found. Referring to Figure 3.4, angles can be added to get  $\theta_i + \theta_A + \theta_C = \theta_f$  or this can be rewritten as

$$\theta_C = \theta_f - \theta_i - \theta_A. \quad (3.28)$$

This yields four solution sets for  $(\theta_A, \theta_C)$ :

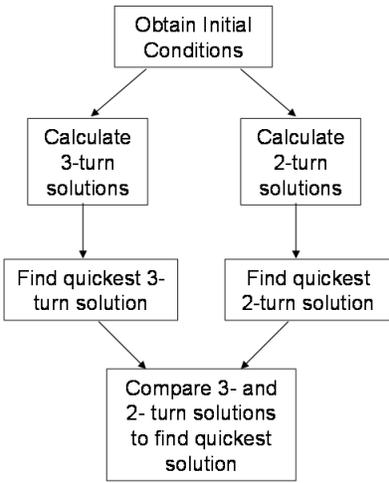
$$\begin{aligned} \theta_{A1} &= \theta_{G(s_1=s_2=+1)} - \theta_i, & \theta_{C1} &= \theta_f - \theta_i - \theta_{A1} \\ \theta_{A2} &= \theta_{G(s_1=s_2=-1)} - \theta_i, & \theta_{C2} &= \theta_f - \theta_i - \theta_{A2} \\ \theta_{A3} &= \theta_{G(s_2=-s_1=+1)} - \theta_i - \tan^{-1}\left(\frac{2}{L}\right), & \theta_{C3} &= \theta_f - \theta_i - \theta_{A3} \\ \theta_{A4} &= \theta_{G(s_2=-s_1=-1)} - \theta_i - \tan^{-1}\left(-\frac{2}{L}\right), & \theta_{C4} &= \theta_f - \theta_i - \theta_{A4} \end{aligned} \quad (3.29)$$

where  $\theta_{G_{two-turn}}$  is calculated from 3.16 using the indicated values for  $s_1$  and  $s_2$ .

## 3.5 The Quickest Solution

There are up to eight possible solutions for any turn maneuver – up to 4 three-turn solutions and up to 4 two-turn solutions. The quickest (i.e. shortest) path of these 8 possible choices must be chosen. For the three-turn solutions, since all the turn radii are equal,  $\theta_A$ ,  $\theta_B$ , and  $\theta_C$  can be added together for each solution and the minimum sum can be chosen. The  $\theta$ 's that give this quickest solution are stored.

All units in the preceding equations are normalized to the turn radius, thus one unit of linear travel is equal to one unit (i.e. radian) of angular travel. Thus, for each two-turn solution,  $\theta_A + \theta_C + |L|$  can be calculated, the smallest sum chosen, and the  $\theta_A$ ,  $\theta_C$ , and  $|L|$



**Figure 3.5:** *Flowchart for Determining Quickest Turning Maneuver Solution*

that give the shortest solution determined. Comparing the shortest  $\theta_A + \theta_B + \theta_C$  (three-turn solution) and the shortest  $\theta_A + \theta_C + |L|$  (two-turn solution), the smallest sum can be chosen to yield the overall solution. Figure 3.5 provides a visual summary for this process. Matlab code for the corner turning algorithm is provided in Appendix A.3.

## Chapter 4

# Flight Plan Generation Program and Flight Test Results

This chapter describes the flight path generation software that accompanies this thesis. All flight plan generation programs generate path lines, turning maneuvers, and output files described in Chapters 2 and 3. Each flight plan also includes a final turn maneuver that connects the end of the last path line to the initial point of the flight plan. The flight path generation algorithms are assembled into a single Matlab Graphical User Interface (GUI). The GUI allows a user to create a flight plan easily without changing any Matlab code. Source code needed to run the GUI is located in a single folder, “Flight Plan Generator GUI”.

After the folder “Flight Plan Generator GUI” is set as the Matlab working directory, it is run by typing “FlightPlanGeneratorGUI” (without quotes) at the command prompt. This opens a window with three buttons as shown in Figure 4.1. The GUI windows are best viewed at monitor resolutions greater than 800x600. Maximizing the GUI window helps ensure that the entire window is visible. The window shown in Figure 4.1 provides a gateway to three different flight plan generation GUIs. “Three Passes GUI” generates



**Figure 4.1:** *Start Window of the Flight Plan Generation GUI*

flight plans over small, specific plots on the Konza Prairie. “Generate Box GUI” allows the user to specify a rectangular field area and “Generate Polygon GUI” utilizes a user-defined polygon field area. Both generate flight plans over the specified field areas. Each GUI has a series of flight plan parameters for the user to define. The GUIs also plot the flight plan over a geo-referenced image of the flight area. The axis labels of the geo-referenced image may be in either latitude-longitude coordinates or in meters. A toolbar at the top of the GUI window provides a means for the user to zoom, pan, and add data tips to the flight plan image. After any flight plan parameter is changed, the “Calculate Flight Plan” button is pushed and the modified flight plan is displayed. Further details of these GUIs are discussed in in the following three sections of this chapter.

## 4.1 Three Passes GUI

At the Konza Prairie, there are several pairs small subplots or features that require periodic monitoring. “Three Passes GUI” generates flight plans to fly over these features and subplots. Figure 4.2 shows a screen shot of this flight plan generator. Each flight plan is similiar in that it has three straight line segments or path lines. One path line passes through centers of the subplots, the other two path lines are on either side of the first path line.

While much of the information needed to generate the flight plans is already defined in the source code, the user is able to define the parameters shown on the left side of Figure 4.2. The “Output File Name” text box allows the user to input a file name to save the flight plan data. The default file name is “outputfile”. The flight plan parameters will also be saved to a file named with the same output file name with the word “parameters” appended to the name. The “Choose Map” pulldown menu has three map options: “20B”, “2C”, and “1D”. These names correspond to watersheds on the Konza Prairie that have small, monitored subplots. The “Flight Path Parameters” panel allows the shape of the flight path to be altered. “Spacing” describes the distance between the path lines in meters. “Turn Radius” lists the desired turning radius in meters. “Setup Distance” describes the distance between the center of a subplot and the beginning or end of the path line. This distance is also in meters. “Flight Altitude” is the LLA altitude, or altitude above the spheroid. The “Starting Subplot” pulldown menu allows the user to choose to have the first waypoint be the eastern- or western-most subplot. The user may also choose to display the figure’s axes in latitude-longitude coordinates or in local coordinates by selecting or deselecting the “Axis Labels in Meters” radio button.

To accept the input values, the user presses the “Calculate Flight Path” button. This action calculates the flight path, outputs it to the specified output file, and displays it over the image of the appropriate area. The yellow line in the flight plan plot denotes the flight path. The cyan dots represent the corners of a subplot (if applicable) and the magenta asterisk is the center of the subplot (or the location of the feature). The distance between the yellow and magenta asterisks is the setup distance. The starting point of the flight plan is indicated with a yellow circle.

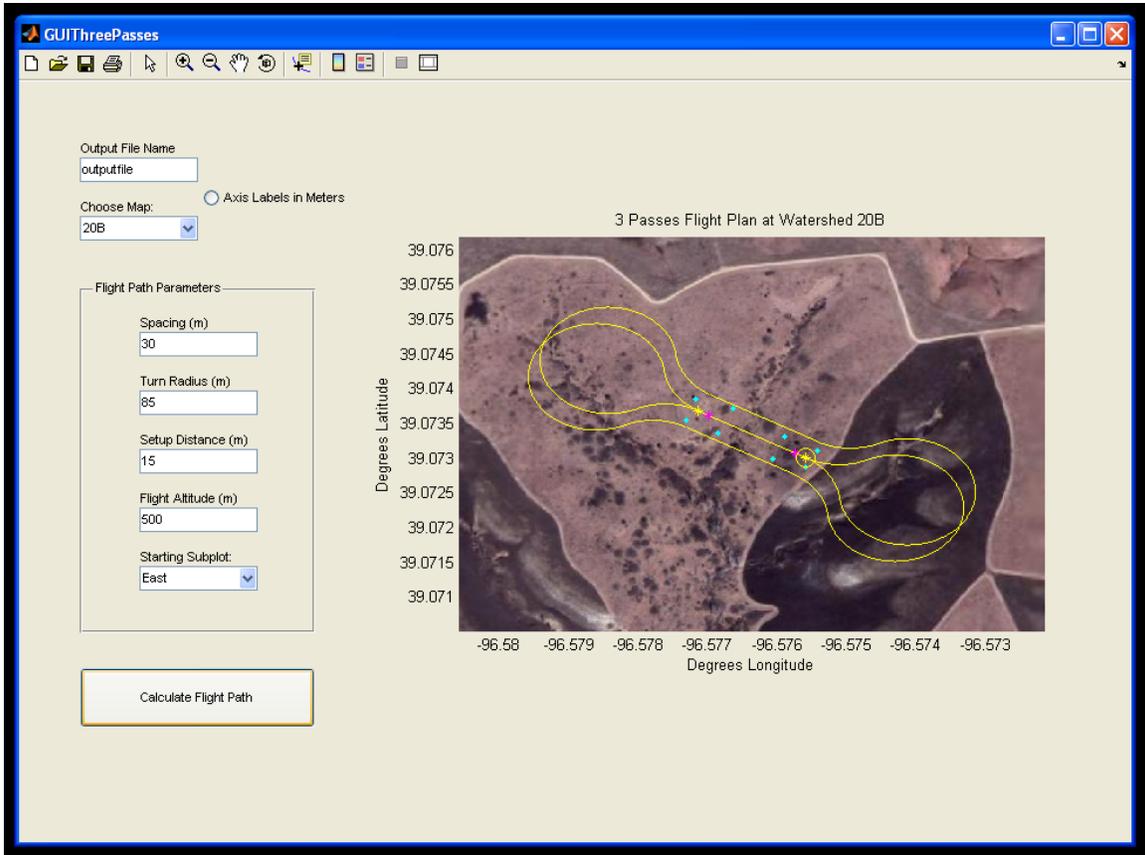


Figure 4.2: Three Passes Flight Plan Generation GUI

## 4.2 Generate Box GUI

“Generate Box GUI” creates a rectangular field area and creates a flight path over the area. A screen shot of this GUI is shown in Figure 4.3. The output file name and map are chosen from the “Output File Name” and “Choose Map” text box and pulldown menu in the top left corner of the GUI window. The rectangular field area is defined by its center in latitude-longitude coordinates as well as the north-south length and east-west width of the field area. The length and width are defined in meters. In addition to the spacing between path lines, turn radius, and altitude above the spheroid (all defined in meters), the angle of the path lines is also defined in degrees. This angle must be between 0 and 180 degrees, and an angle of  $0^\circ$  yields north-south path lines and  $90^\circ$  yields east-west path lines. The “Method to Fly Path Lines” pulldown menu lets the user choose between flying the path lines in order and flying the lines efficiently (as described in section 2.1.3). The user may also choose between labeling the resulting plot with either local or latitude-longitude coordinates by either selecting or deselecting the “Axis Labels in Meters” radio button near the top of the GUI window.

Clicking the “Calculate Flight Plan” button in the bottom left corner of the GUI window generates the flight plan, outputs it to the specified output file, and plots the flight plan on top of an image of the flight area. The flight plan parameters are also stored to a file named by the output file name with “parameters” appended to it. To the right of the image of the flight plan, the sum of the lengths of the path lines, the total length of all the turn maneuvers and the total flight plan length are all listed in meters. The total number of waypoints is also listed. If the total number of waypoints in the flight plan exceeds 55, a warning is displayed under the flight path image. The Piccolo autopilot does not have sufficient memory to handle more than 55 waypoints.

The flight plan is represented in yellow in the image window. The yellow asterisk indicates the first waypoint in the flight plan and the initial flight direction will be along the path line. The green lines are the boundaries of the field area and the center of the field

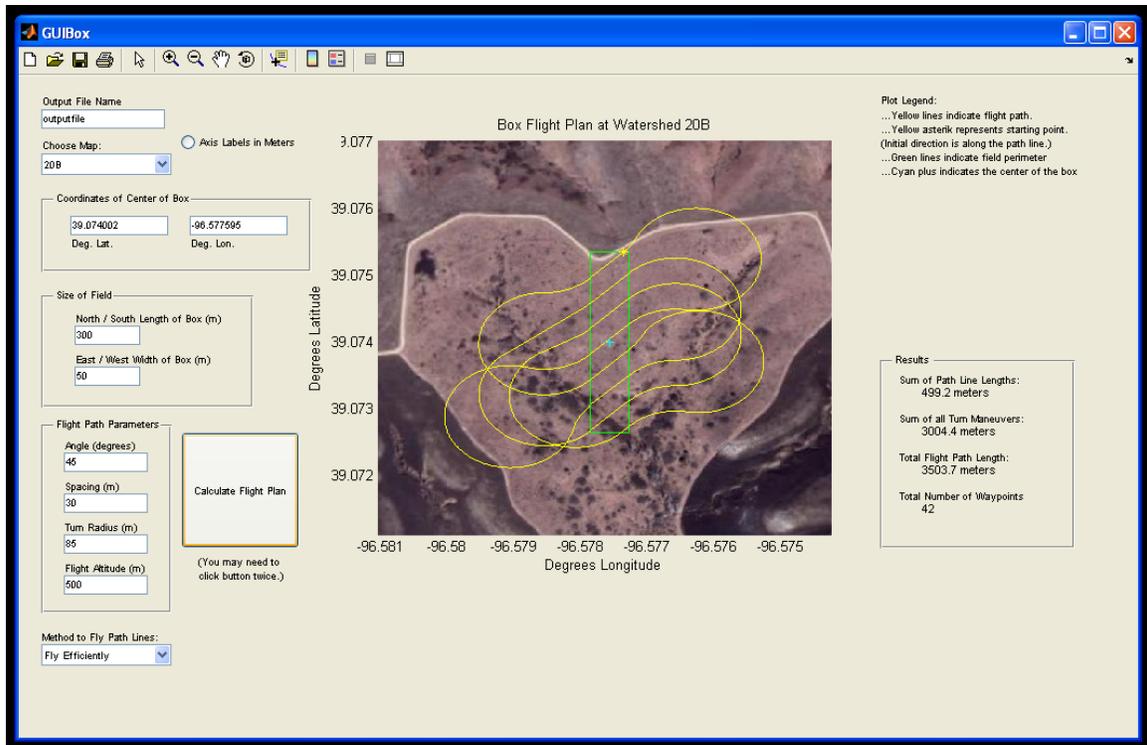


Figure 4.3: Generate Box Flight Plan Generation GUI

area is marked with a cyan plus sign.

### 4.3 Generate Polygon GUI

The third flight plan generator is “Generate Polygon GUI”, shown in Figure 4.4. Several options in this flight plan generator are the same as those in the “Generate Box GUI”. This “Generate Polygon GUI” also allows the user to input the output file name; choose an image map; specify flight the path parameters angle, spacing, turn radius, and flight altitude; as well as choose the method to fly the path lines. It also allows the user to specify the axis labels as either degrees latitude-longitude or meters.

Instead of defining a rectangular box, “Generate Polygon GUI” takes an arbitrary number of points, defined by the user, and treats them as vertices of the field area. The first step in generating a flight plan with this GUI is selecting the field area vertices. The user

inputs the number of sides, or vertices, of the field area in the appropriate box on the left side of the GUI window, near the top. Then the user clicks the button “Enter Vertices By Clicking”. This action displays the selected satellite image map and attaches a crosshair to the mouse pointer. Clicking on the satellite image places vertices. The user should take care to enter a convex field area shape and place the vertices in a clockwise manner around the desired field area polygon as the software is unable to reliably deal with non-convex polygons or field perimeter lines that cross. As the user clicks to place the vertices, a green dot will appear on the map, and a green line will connect the dots to form a closed polygon.

Alternatively, the user may enter vertices by using the text box in the panel labeled “Enter Vertices Manually”. The vertices must be entered in latitude-longitude coordinates and be listed in clockwise order going around the polygon. If a coordinate is a positive number, it should be preceded with a plus sign. Each line lists a latitude coordinate, a space, then the longitude coordinate. One vertex point is listed per line. There should be no blank lines between lines or before the first line. Clicking the “Get Current Vertices” button imports the current field area vertices, listed in the upper right corner of the GUI window, into the “Enter Vertices Manually” text box. This feature allows the user to make modifications to the current vertices.

Clicking the “Calculate Flight Plan” displays the flight plan in the GUI window, creates an output file with the name indicated in the “Output File Name” text box, and generates an appropriately named file of the flight path parameters. The field perimeter is indicated in green. The flight path is displayed in yellow with a yellow asterisk indicating the first waypoint. The initial direction is along the first path line. Information about the flight path length is also displayed on the right side of the GUI window. The total lengths of all path lines, the total length of all turn maneuvers and the total flight path length are all listed. The total number of waypoints is displayed, and if this number exceeds 55, a warning will appear near the bottom of the GUI window.

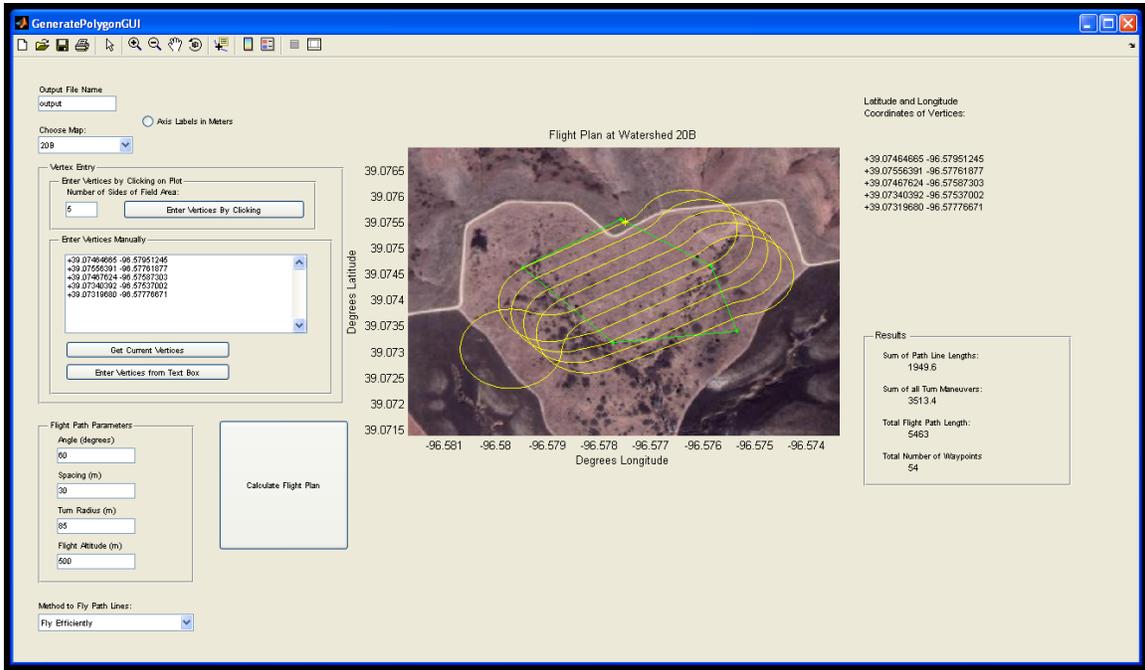
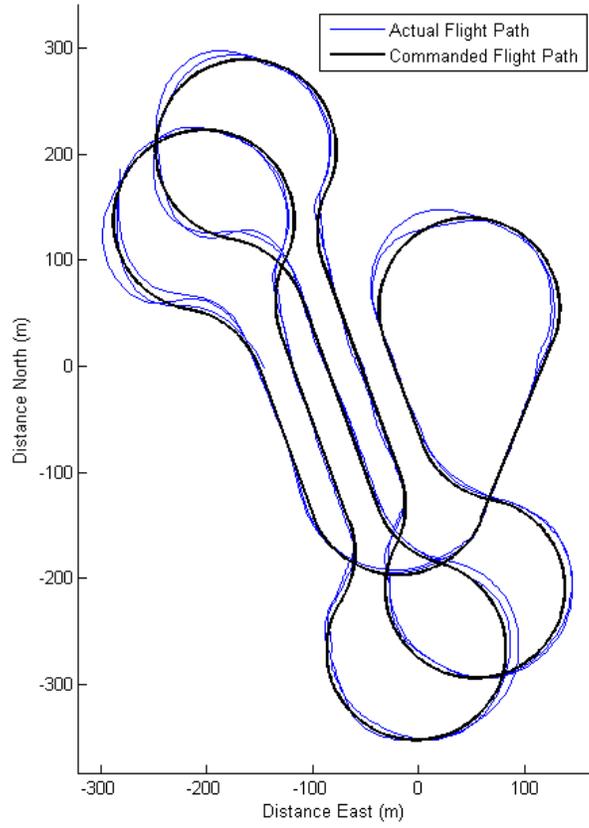


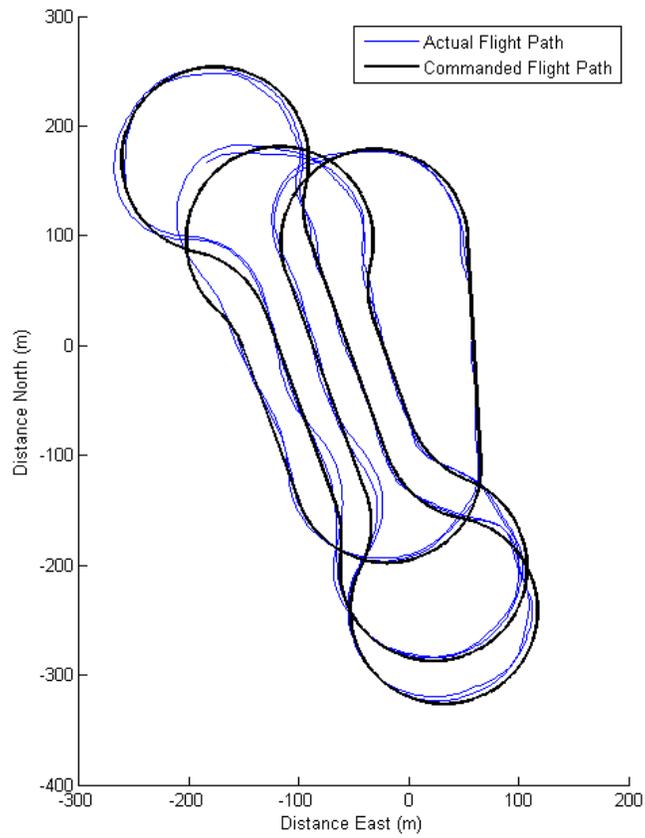
Figure 4.4: Generate Polygon Flight Plan Generation GUI

## 4.4 Flight Test Results

In order to verify the flight plan generator, the flight plans generated in Chapter 2 were flown in a test flight. Both flight plans were created with GeneratePolygonGUI and both share the same path lines. The flight plan in Figure 4.5 was created by flying the path lines in order from the west to the east. Figure 4.6 utilizes the efficient path line ordering method. Each flight path includes an additional turn maneuver to connect the last path line flown with the first path line. Approximately two circuits of the flight plan were completed for each test flight.



**Figure 4.5:** *Test Flight with Path Lines Flown in Order*



**Figure 4.6:** *Test Flight with Path Lines Flown Efficiently*

# Chapter 5

## Conclusions and Recommendations

- A method for generating path line segments to cover a specified area was developed and implemented in Matlab. Two methods for choosing flight order of the path lines were used. One method flies the path lines in order from west to east, and the other method determines the next path line flown by determining the shortest distance from the current path line to another path line.
- An optimal turn maneuver algorithm was developed and implemented. This algorithm determines the most efficient way for the UAV to travel between two path lines given finite turning capabilities of the UAV.
- Software was developed in Matlab to automatically generate flight path algorithms. A graphical user interface was created to allow flight plans to be generated easily with little knowledge of the flight plan generation process.
- The method used for generating path lines can be improved. The placement of the path lines can be shifted up or down (i.e. change the y-intercepts of the path lines) to optimize the total length of all path lines.
- Locations near the field area perimeter may not be captured in photographs. A sug-

gested solution for this problem is to enlarge the field area, maintaining its shape, and generate the flight plan over a larger field area.

- In order for a new image map to be used in a GUI, the user must obtain the image as well as modify code. This process could be streamlined with a different method of importing geo-referenced images into the Matlab GUI. A new importing method would allow users unfamiliar with Matlab to easily add images on which flight plans may be plotted.
- The first waypoint of the flight plan is on the first path line's eastern-most endpoint. Since this waypoint location may not be a convenient place to begin the flight plan, Matlab code and the GUI could be modified to allow the user to choose the initial waypoint of the flight plan.

# Bibliography

- [1] H. Choset, Coverage for robotics – a survey of recent results, in *Annals of Mathematics and Artificial Intelligence*, 2001.
- [2] B. L. Stevens and F. L. Lewis, *Aircraft Control and Simulation, 2nd Edition*, John Wiley, 2003.
- [3] E. P. Anderson, R. W. Beard, and T. W. McLain, Realtime dynamic trajectory smoothing for unmanned air vehicles, in *IEEE Transactions on Control Systems Technology*, 2005.
- [4] D. Gu, W. Kaml, and I. Postlethwaite, A uav waypoint generator, in *AIAA First Intelligent Systems Technical Conference*, 2004.
- [5] I. H. Whang and T. W. Hwang, Horizontal waypoint guidance design using optimal control, in *IEEE Transactions on Aerospace and Electronic Systems*, 2002.
- [6] G. Yang and V. Kaplia, Optimal path planning for unmanned air vehicles with kinematic and tactical constraints, in *Proceedings of the 41st IEEE Conference on Decision and Control*, 2002.
- [7] G. Moon and Y. Kim, Flight path optimization passing through waypoints for autonomous flight control systems, in *Engineering Optimization*, 2005.
- [8] L. E. Dubins, On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents, in *American Journal of Mathematics*, 1957.

- [9] A. Noonan, D. Schinstock, C. Lewis, and B. Spletzer, Optimal turning path generation for unmanned aerial vehicles, in *Proceedings of the Ninth IASTED Control and Applications Conference*, 2007.

# Appendix A

## Matlab Code

### A.1 Calculating Intersections between Path Lines and Perimeter Lines

This Matlab function calculates the intersections between path lines and perimeter lines as described in Section 2.1.2. It takes path lines, such as those shown in Figure 2.3, and outputs modified path lines like those in Figure 2.5. The field area vertices and perimeter lines are defined as global variables, not arguments passed to the function.

The path lines, as well as the perimeter lines, are listed as an  $n \times 6$  matrix where  $n$  represents the number of line segments. The first and second columns of the matrix contain the slope and y-intercept of the line segment. Columns three and four contain the x- and y- coordinates of one endpoint and the fifth and sixth columns list the second endpoint's coordinates.

```
function [newPathLines] = intersections(oldPathLines)
global vertex
global Perimeter
```

```

P = Perimeter;

[num_paths,junk]=size(oldPathLines); [num_vertices,junk]=size(vertex);
k = 1; tol = 0.0000005;
for i = 1:num_paths
    for j = 1:num_vertices
        m2 = oldPathLines(i,1);    b2 = oldPathLines(i,2);
        m1 = P(j,1);                b1 = P(j,2);
        if m2 == Inf % PathLines are vertical
            if m1 == Inf
                x = Inf; y = Inf;
            else
                x = oldPathLines(i,3); y = m1*x + b1;
            end
        elseif m2 == 0 % PathLines are horizontal
            y = oldPathLines(i,4);
            if m1 == 0 % if perimeter line is horizontal
                x = Inf;
            elseif m1 == Inf
                x = P(j,3);
            else
                x = (y-b1)/m1;
            end
        elseif m1 ~= m2
            if m1 == Inf %if the Perimeter line is infinite slope
                x = P(j,3); y = m2*x + b2;
            else
                x = (b2-b1)/(m1-m2); % x and y are the point of intersection
                y = m1*x + b1; % of the path line and the polygon line
            end
        else %the PathLine and current Perimeter line are parallel -- no intersection
            x = Inf; y = Inf;
        end
        xmax = max(P(j,3),P(j,5));    ymax = max(P(j,4),P(j,6));
        xmin = min(P(j,3),P(j,5));    ymin = min(P(j,4),P(j,6));
        if (x < xmax) && (x > xmin) && (y < ymax) && (y > ymin) % if the intersection point lies on the line segment
            intPath(k,1:3) = [i x y]; k = k+1;
        elseif (x < (P(j,3)+tol) && (x > (P(j,3)-tol)) && (y < (P(j,4)+tol) && (y > (P(j,4)-tol)))
            % if [x y] is equal to the first endpoint listed in the Perimeter line segment
            intPath(k,1:3) = [i x y]; k = k+1;
        elseif (m1==Inf) && (y < ymax-tol) && (y > ymin+tol)
            % if [x y] is on a vertical perimeter line and is between the y limits
            intPath(k,1:3) = [i x y]; k = k+1;
        elseif (m1==0) && (x < xmax-tol) && (x > xmin+tol) %&& ((y<=b1+tol)&&(y>=b1-tol))

```

```

        % if [x y] is on a horizontal perimeter line and is between the x limits
        intPath(k,1:3) = [i x y]; k = k+1;
    end
end
end

[rows,junk] = size(intPath);
if rows <= 2
    msgbox('Path Spacing is too big -- only one path line is needed to cover area. Try decreasing Spacing.')
end
if intPath(1,1) ~= intPath(2,1)
    intPath = intPath(2:rows,:); rows = rows-1;
end
if intPath(rows-1,1) ~= intPath(rows,1)
    intPath = intPath(1:rows-1,:); rows = rows-1;
end
if mod(rows,2) == 1
    disp('error -- odd number of endpoints ')
end
[num_pts,junk] = size(intPath); k = 1;
for i = 1:2:num_pts
    index = intPath(i,1);
    EndPts(k,1:6) = [oldPathLines(index,1:2) intPath(i,2:3) intPath(i+1,2:3)];
    k = k+1;
end
newPathLines = EndPts;

```

## A.2 Efficiently Ordering Path Lines

This Matlab code illustrates the method of ordering the path lines efficiently. All path line information is stored in an  $n \times 6$  matrix “PathLines”. Columns one and two contain the slope and y-intercept of the line, columns three and four contain the east-most pathline endpoint, and the fifth and sixth columns contain the west-most endpoint. (If the path lines are vertical, the southern endpoint is listed first, and the northern waypoint is listed second.) The % sign denotes a comment.

```

[num,col] = size(PathLines);
k = 1;

```

```

% Create a list of the non-visited endpoints
for i = 2:num
    EndPts2Visit(k:k+1,1:3) = [i PathLines(i,3:4);...
                             i PathLines(i,5:6)];
    k = k+2;
end

% List of visited endpoints
VisitedList = [PathLines(1,3:4); PathLines(1,5:6)];
z=1; sum.turns = 0; stop = 0;
while stop ~ 1
    [num,junk] = size(VisitedList);
    if isempty(EndPts2Visit)
        break
    else
        [j,k] = size(EndPts2Visit);
    end
    for i = 1:j
        % calculate vector between endpoints (magnitude and direction):
        D(i) = sqrt((VisitedList(num,1)-EndPts2Visit(i,2))^2+(VisitedList(num,2)-EndPts2Visit(i,3))^2);
        theta_D(i) = atan2(-VisitedList(num,2)+EndPts2Visit(i,3),-VisitedList(num,1)+EndPts2Visit(i,2));
        % calculate initial heading for turn maneuver
        theta_i = atan2(VisitedList(num,2)-VisitedList(num-1,2),VisitedList(num,1)-VisitedList(num-1,1));
        % calculate final heading for turn maneuver
        if mod(i,2) == 1 % if going from east endpoint to west endpoint (or bottom to top)
            theta_f(i) = atan2(EndPts2Visit(i+1,3)-EndPts2Visit(i,3),EndPts2Visit(i+1,2)-EndPts2Visit(i,2));
        else % if going from west endpoint to east endpoint (or top to bottom)
            theta_f(i) = atan2(-EndPts2Visit(i,3)+EndPts2Visit(i-1,3),-EndPts2Visit(i,2)+EndPts2Visit(i-1,2));
        end
        CL(i) = CurveLenFcn(RADIUS,theta_i,theta_f(i),theta_D(i),D(i)); % calculate the curve length
    end
    [smallestCL,index_smallest] = min(CL);

    % Calculate the smallest turn maneuver:
    theta_f = theta_f(index_smallest);
    D = D(index_smallest);
    theta_D = theta_D(index_smallest);
    [Thetas(z,1:3),Centers(z,1:3),Pja,Pjb,Pjc,dir(z),type(z,1:5)] = CurveFcn_withTurnType(RADIUS,theta_i,theta_f,theta_D,D);
    % Data must be shifted to appropriate position:
    Centers(z,1:3) = Centers(z,1:3) + complex(VisitedList(num,1),VisitedList(num,2))*ones(1,3);
    [junk,cola]=size(Pja);
    Pa = Pja + complex(VisitedList(num,1),VisitedList(num,2))*ones(1,cola); turnscell.pa{z,:}=Pa;
    [junk,colb]=size(Pjb);
    Pb = Pjb + complex(VisitedList(num,1),VisitedList(num,2))*ones(1,colb); turnscell.pb{z,:}=Pb;
end

```

```

[junk,colc]=size(Pjc);
Pc = Pjc + complex(VisitedList(num,1),VisitedList(num,2))*ones(1,colc); turnscell.pc{z,:}=Pc;
TurnPart2(z,1:2) = [Pa(cola) Pc(1)]; % keep the end points of the 2nd part of turn maneuver

% add to the VisitedList of waypoints
if mod(index_smallest,2) == 1
    VisitedList = [VisitedList; EndPts2Visit(index_smallest,2:3); EndPts2Visit(index_smallest+1,2:3)];
    if index_smallest ~= 1
        EndPts2Visit = [EndPts2Visit(1:index_smallest-1,:); EndPts2Visit(index_smallest+2:end,:)];
    else
        EndPts2Visit = EndPts2Visit(3:end,:);
    end
end
else
    VisitedList = [VisitedList; EndPts2Visit(index_smallest,2:3); EndPts2Visit(index_smallest-1,2:3)];
    if index_smallest ~= 2
        EndPts2Visit = [EndPts2Visit(1:index_smallest-2,:); EndPts2Visit(index_smallest+1:end,:)];
    else
        EndPts2Visit = EndPts2Visit(3:end,:);
    end
end
end
z = z+1; clear D theta_i theta_f theta_D CL;
end

```

## A.3 Turn Maneuver Generation Algorithm

This Matlab code is a function that calculates the optimal turn maneuver for a given situation. The inputs to the function are the turn radius,  $R$ ; initial heading, “theta\_i”; final heading, “theta\_f”; heading from initial point to final point, ”theta\_D”; and distance from initial point to final point, “D”. The function takes these input arguments, calculates the optimal turn maneuver, and outputs variables needed to describe the turn maneuver. The output “Thetas” is a  $1 \times 3$  vector where each element is the angle of the arc swept out. If the second element of “Thetas” is a complex number, then the turn maneuver is an two-turn solution. (See section 3.4.) “Centers” is a  $1 \times 3$  vector containing the coordinates of the arc’s center in complex form. “Pja”, “Pjb”, are “Pjc” variables used to plot the curve results. The ouput “dir” indicates the direction of travel around an arc and “type” indicates if the

maneuver is a two- or three-turn maneuver. This algorithm places the start of the turn maneuver at (0,0), so shifting the position of “Centers”, “Pja”, “Pjb”, and “Pjc” may be necessary. This code can easily be modified to output the total length of a turn maneuver

```
function [Thetas,Centers,Pja,Pjb,Pjc,dir,type] = CurveFcn_withTurnType(R,theta_i,theta_f,theta_D,D)
D = D/R; %normalize distances
j = sqrt(-1);
s = [-1 -1 1 1]; f = [-1 1 -1 1];

% Find 2-Turn 1-Segment solutions:
XA = exp(j*(theta_i + s*pi/2)); % location of the center of the first turn
XC = D*exp(j*(theta_D)) + exp(j*(theta_f + f*pi/2)); % location of the center of the final turn
CA = XC - XA;% distance between centers
index = 1;
for i = 1:4
    j = sqrt(-1);
    if s(i) ~= f(i)
        SneF(i) = 1;
    else
        SneF(i) = 0;
    end
    if abs(CA(i))>2 || SneF(i)==0
        % if dist btwn centers is > 2 or the initial and final turns are in the same direction
        L(i) = CA(i) + 2*SneF(i)*exp(j*(angle(CA(i)) - s(i)*acos(2/abs(CA(i))) + pi)) ;
        thetaAseg(i) = ProperRange(s(i)*(angle(L(i)) - theta_i));
        thetaCseg(i) = ProperRange(f(i)*(theta_f - angle(L(i))));
        thetaTseg(i) = thetaAseg(i) + thetaCseg(i) + abs(L(i));
    end
end

% Figure out the 3-Turn Solutions:
i = 0:1:3; S = 2*floor(i/2)-1; PM = 2*mod(i,2)-1;
G = .5*((D*exp(j*(theta_D-theta_i+S*pi/2)) - exp(j*(theta_f - theta_i)))+1);
Gi = abs(G); theta_Gi = angle(G); ac = acos(Gi/2);
a = PM.*ac; k = 1;
for i=1:1:4
    if imag(a(i)) == 0
        anew(k) = a(i);
        k = k+1;
    end
end
end
if D*R > 4*R; % a three-turn solution doesn't exist.
    % find fastest 2-turn solution
```

```

[thetaT,index] = min(thetaTseg);      thetaA = thetaAseg(index);
thetaB = L(index);                    thetaC = thetaCseg(index);
thetaT = thetaTseg(index);
XA = exp(j*(theta_i + s(index)*pi/2));% location of the center of the first turn
XC = D*exp(j*(theta_D)) + exp(j*(theta_f + f(index)*pi/2)); % center of the final turn
CA = XC - XA ; % distance between centers
k = 0:0.01:(thetaA + thetaC);      ka = 0:0.01:(thetaA);
kc = thetaA:0.01:(thetaA + thetaC);
Pja = (XA + exp(j*(ka*s(index) + theta_i - s(index)*pi/2)))*R;
Pjc = (XC + exp(j*(f(index)*(kc - thetaA - pi/2) + angle(L(index)))))*R;
[r,c] = size(Pja);                  Pjb = [Pja(c) Pjc(1)];
Thetas = [thetaA thetaB thetaC];
Centers = [XA NaN XC]*R;            dir = S(index);
if SneF(index) == 1 % initial and final turns in opposite directions
    type = '2Topp';
elseif SneF(index) == 0 % initial and final turns in same direction
    type = '2Turn';
end
return
else
a = anew;
[r,num_sol] = size(a); % num_sol is number of valid solutions
for i = 1:num_sol
    Theta_A(i) = ProperRange(S(i)*(a(i)+theta_Gi(i)));
    Theta_B(i) = ProperRange(S(i)*(2*a(i) - pi));
    Theta_C(i) = ProperRange(S(i)*(theta_f - theta_i + a(i) - pi - theta_Gi(i)));
    Theta_T(i) = Theta_A(i) + Theta_B(i) + Theta_C(i);
end
end

% All possible combinations calculated. Find the fastest:
[ThetaT3,index3] = min(Theta_T);    m = 1;
for k = 1:4
    if thetaTseg(k) ~= 0
        dummyT(m) = thetaTseg(k);  dummyA(m) = thetaAseg(k);  dummyC(m) = thetaCseg(k);
        dummyL(m) = L(k);          dummyXC(m) = XC(k);      dummyXA(m) = XA(k);
        dummydir(m) = s(k);        dummyf(m) = f(k);      dummysnef(m) = SneF(k);
        m = m+1;
    end
end
thetaTseg = dummyT;  thetaAseg = dummyA;  thetaCseg = dummyC;
XC = dummyXC;      XA = dummyXA;      dir = dummydir;
SneF = dummysnef;  f = dummyf;      L = dummyL;
[ThetaT2,index2] = min(thetaTseg);

```

```

if ThetaT3 < ThetaT2 || ThetaT3==ThetaT2% the three turn solution is faster (or they're the same)

    thetaA = Theta_A(index3);  thetaB = Theta_B(index3);
    thetaC = Theta_C(index3);  thetaT = Theta_T(index3);
    dir = S(index3)            j = 0:.01:thetaT;            jA = 0:.01:thetaA;
    [r,c]=size(jA);           jB = jA(c):.01:(thetaA+thetaB);
    [r,c]=size(jB);          jC = jB(c):.01:thetaT;        i = sqrt(-1);

    % the centers of the turns:
    CA = exp(i*(theta_i+S(index3)*pi/2));
    CB = CA + 2*exp(i*(S(index3)*thetaA + angle(CA) + pi));
    CC = CB + 2*exp(i*(-S(index3)*thetaB + angle(CB-CA) + pi));
    Centers = [CA CB CC]*R;

    Pja = (CA + exp(i*(jA*S(index3) + theta_i - S(index3)*pi/2)))*R;
    Pjb = (CB + exp(i*(-S(index3)*(jB - 2*thetaA) + theta_i - S(index3)*pi/2 + pi)))*R;
    Pjc = (CC + exp(i*(S(index3)*(jC - 2*thetaB) + theta_i - S(index3)*pi/2)))*R;

    type = '3Turn';

elseif ThetaT2 < ThetaT3 % the two-turn solution is faster

    thetaA = thetaAseg(index2);  thetaB = L(index2);
    thetaC = thetaCseg(index2);  thetaT = thetaTseg(index2);
    dir = dir(index2);           XA = XA(index2);
    XC = XC(index2);           Centers = [XA NaN XC]*R;

    CA = XC - XA  ;% distance between centers

    k = 0:0.01:(thetaA + thetaC);
    ka = 0:0.01:(thetaA);       kc = thetaA:0.01:(thetaA + thetaC);
    Pja = (XA + exp(j*(ka*s(index2) + theta_i - s(index2)*pi/2)))*R;
    Pjc = (XC + exp(j*(f(index2)*(kc - thetaA - pi/2) + angle(L(index2)) )))*R;
    [r,c] = size(Pja);  Pjb = [Pja(c) Pjc(1)];

    if SneF(index2) == 0 % if the initial and final turns of a two-turn maneuver are in same directions
        type = '2Turn';
    elseif SneF(index2) == 1 % initial and final turns in opposite direction
        type = '2Topp';
    end

end

Thetas = [thetaA thetaB thetaC];

```

0/123456789