Proceedings of the Development of Computer Applications Class Project
ME701
December 19, 2014, Manhattan, Kansas, USA

paper number??

AUTOMATIC GENERATION OF WAYPOINT NAVIGATION FILE FOR USE IN MISSION PLANNER

Sydney Schinstock
Mechanical Engineering
Undergraduate Student, Kansas
State University
Manhattan, Kansas, USA

ABSTRACT

This paper describes a flight path generating software for use with the ground control station Mission Planner and the Ardupilot autopilot. It is for use in the KSU AIAA Unmanned Aerial Systems design competition and is intended to streamline the flight path generation process during competition. The software consists of a latitude and longitude conversion from degrees-minutes-seconds to decimal, a simple algorithm for waypoint generation, a means to save those waypoints in a format readable by Mission Planner, and a user-friendly GUI for quick and simple operation. The use of this software with greatly decrease the time dedicated to the pop-up target task and will allow for more mission time to be able to be given to other tasks.

INTRODUCTION

The KSU AIAA Student Unmanned Aerial Systems design team has been using Ardupilot as their autopilot firmware for 3 years now. This autopilot communicates with an open-source ground control station (GCS) called Mission Planner. Each year the team attends a competition that involves an autonomous flight mission. The mission flight time for each team is only 30 minutes. The team must accomplish numerous tasks within this period. With this small time frame, speedy execution of tasks is imperative to complete the mission. One of these tasks is a pop-up target, where GPS coordinates of a search area are given in degrees-minutes-seconds (DMS) to survey in the middle of the mission. The autopilot takes coordinates in decimal form, not DMS, and so the team must quickly convert those coordinates. In the past, a team member has used an excel file to calculate the conversion, which is then copy-pasted to a text file (with a .poly extension and the proper delimiters so Mission Planner can read it), uploaded to Mission Planner, and then a flight-path is generated in Mission Planner. This is hastily done, with lots of intermediate steps and room for error.

Alongside the quick-calculated pop-up target, team members must generate a large search area flight path the night before the mission. Mission Planner does have an auto-grid capability, but it does not factor in the turn radius of the team's plane and often creates a flight path that is impossible for the plane to fly. Consequently, once the waypoints are generated, a team member must rearrange the grid in an "alternating lawnmower" fashion (with the use of excel). This takes about 15 minutes, which is not a large amount of time in relation to an evening, but automating the process would allow for more time spent on something else, and an altogether more convenient process. This paper will describe software to automate the flight path generating process. All of the software described in this paper is written in the Python programming language.

First, the autopilot system will be described to provide proper background information. Second, a simple algorithm for creating the alternating lawnmower path will be discussed. The second section will describe the software used to create a flight plan readable by Mission Planner. The final section will describe a GUI to create more user friendly version of the software. So that with a few inputs and the click of a button, a ready to go flight plan will be created.

NOMENCLATURE

Θ: Angle describing rotation from x-axis (latitude)
UAS: Unmanned Aerial System
AIAA: American Institute for Aeronautics and Astronautics
d: Distance between each pass of a flight path
DMS: Degrees-Minutes-Seconds
GCS: Ground Control Station
GUI: Graphical User Interface
MHz: Megahertz
UAS: Unmanned Aerial System

THE SYSTEM

The main components of the UAS used in competition consist of the airframe, the autopilot, and the camera. The airframe is the physical fuselage, wings, tail, and payload. The autopilot is a small microcontroller board mounted in the

airframe. It contains an output rail for control of the airframe's control surfaces and throttle as well as many sensors (accelerometers, GPS, barometer) used for autonomous flight. The autopilot communicates with the GCS over 900 MHz radio. The Mission Planner GCS is on a laptop computer and is manned by the autopilot operator. It is the interface through which commands are sent to the autopilot during flight (via a communication protocol known as MAVLINK). Also onboard the airframe is a mirrorless digital camera pointing down at the ground for survey of an area, specifically to identify targets. The camera sends photos to a computer on the ground while in flight.

Mission Planner is also used in pre-flight planning and mid-flight retasking. It handles the creation of flight paths and sends them to the autopilot. Flight paths consist of waypoints, latitude-longitude positions with a user designated acceptance radius around them. These waypoints are enumerated and the UAS navigates them in numerical order, moving to the next waypoint after crossing the acceptance radius of the current waypoint. Flight paths in mission planner are generated and saved on the laptop in the form of a text file. A waypoint file could be generated without the use of mission planner if the proper parsing and headers were present.

FLIGHT PATH GENERATION

At the competition, search areas are given in the form of a convex polygon, with a latitude and longitude describing the location of each point of the polygon. The position of these points are in units of DMS, and so the first step of the flight path generator is to convert the positions to decimal notation. This is a simple calculation, minutes are divided by 60, seconds are divided by 3600, and they are both added to the degree number. The software here uses the numpy library to create a 3 dimensional array containing degrees, minutes, seconds for the 0, 2, 3 columns respectively. Each even indexed row is a latitude point, each odd indexed row is a longitude point. The software loops through the numpy array and stores the new decimal locations in a new 1 dimensional numpy array.

After converting the coordinates, a new function takes them to create the flight path. This function also takes user inputs of distance, d, and angle, Θ, from 0 to 90 that the flight path will be rotated from the x axis (latitude). In order to properly cover an area, the team's UAS must pass over that area multiple times, with a certain distance between each pass. This distance is dependent on the UAS's altitude and the onboard camera's field of view and orientation. The distance is a number precalculated by the Image Analysis team and is in units of feet. To work with the proper coordinates, two conversion constants were calculated: the difference in latitude per foot and difference in longitude per foot. Because the distance between latitudes and longitudes is not constant over the earth, the constant was calculated specifically for the competition area location. This was achieved by recording horizontal distances in feet on a map of the competition area and recording the difference in longitude between them. The latitude conversion factor was calculated in the same way, only with vertical distances being used. The conversion constants for a location with coordinates of (38.145827, -26.425533) are as follows:

$$349832.083\,\frac{ft}{deg.\,latitude}$$

$$302846.386\,\frac{ft}{deg.\,longitude}$$

This conversion is also applied to the aircraft's turn radius, a hard-coded parameter that has been determined by the design team as the minimum safe radius that the aircraft can turn to change direction by 180º.

Next, the user specified angle is taken into account. Currently, only 0º (latitudinal navigation) and 90º (longitudinal navigation) can be inputted. This is partially because the distance conversion is different for latitude than for longitude, and simple rotation of a path may not cover the right amount of area. Second, the use of the rotation matrix [1] yields latitude and longitudes nowhere near the specified search area, since the rotation is about the point (0, 0). However, more research may be done at a later date (i.e. finals week) and this may be resolved. As of right now, however, the angle of 0 or 90 is used in deciding how to increment within the search area polygon. If Θ is specified as 0º, a 3 dimensional numpy array is created that is twice the length of the maximum latitude and the minimum latitude divided by d. If Θ is specified as 90º, a numpy array is created similarly but with the length determined using longitude instead of latitude. This array is used to store the generated waypoints and contains all zeros at the start. The 3 columns are to contain a waypoint index, latitude, and longitude.

After creating the array it is populated with a flight path. First, the latitude or longitude (depending on Θ, 0º- latitude, 90º-longitude) is incremented by subtracting d from the maximum value of the latitude or longitude of the polygon and storing it in the proper column until the minimum latitude or longitude is reached. Each leg has two points assigned to it so that the path goes straight from one side of the area to the other. After populating the lat/long column, the array is looped through, checking if the incremental difference between two points is greater than the turn diameter. If it is not, the next line is stepped down to. Once the distance is greater than the turn diameter, the maximum or minimum lat/long (if Θ=0º, the longitude, if Θ=90º, the latitude) is stored in the array. An index number (enumeration for waypoint number) is also stored in the array. This creates a rectangle that lays over the search area. Once the array is looped through, it is looped back up, checking for zeros in the index column, and filling them with indices and lat/longs if they are far enough away from the previous ones. A final pass is taken through the array, filling in any last missing legs. After the array is full, it is then sorted by the indices in descending order and graphed so that the user can check the validity of the flight plan.

A graph of a generated path can be seen in Figure 1.

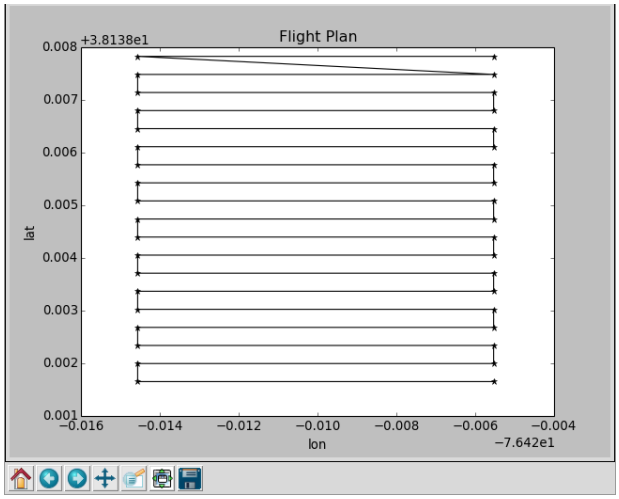

Figure 1 Generated Flight Plan with Angle = 0

WAYPOINT FILE CREATION

In order for Mission Planner to receive the flight path that was just created, the waypoint file to load must be properly formatted (word choice?). A waypoint file is a text file that contains 12 columns. The first column contains the waypoint number, the order in which the waypoints should be flown. The second columns is specifically for setting the home waypoint, the waypoint that the UAS would return to if radio signal was lost or in the event of some other emergency. This should only be an integer 1 in the first waypoint row, where the index is 0. The next number has to do with whether or not the waypoint is commanded. Every waypoint but the home waypoint is commanded and so contain an integer 3. The fourth column contains a waypoint type enumeration (another integer), these numbers indicate if the waypoint is a typical navigation unlimited loiter waypoint, 17. The next four columns are reserved for other special waypoints, they contain various parameters for use with certain types and are floats with 6 decimals. After the parameters, a latitude column and longitude column contain the location of the waypoint. The next column contains the altitude in meters. The latitude, longitude, and altitude columns are floats with 6 decimal points. The final column is a Boolean for whether or not the waypoint is an active command, all waypoints should contain a 1 [2]. In addition to the 12 columns, a header of "QGC WPL 110" must be included at the top of the text file for mission planner to recognize the text file as waypoint parameters. Each column must also be separated by a tab. The flight path software is able to create a text file with the proper numbers, text, and tabs for quick transfer to mission planner.

The file writing function takes the generated flight path numpy array and a user specified altitude to write a new numpy array with 12 columns. A home waypoint is added to the top of the file, with the latitude-longitude taken from waypoint 1. The proper header is also added to the top of the file. Each subsequent waypoint is assigned a 16 in column 4. The last waypoint in the file is given a 17 to indicate that it is an unlimited loiter waypoint. The UAS will circle that waypoint until it is commanded otherwise. A generated sample file is shown in Figure 2 and corresponds with the waypoints generated in Figure 1. This file was directly loaded into Mission Planner and created the desired flight plan. This is pictured in Figure 3. Figure 4 is another generated flight plan with a larger distance, d, that better shows the alternating lawnmower capabilities of the software.

```
# QGC WPL 110
0   1   0   16  0.000000    0.000000    0.000000    0.000000    38.145827   -76.425533  76.200000   1
1   0   3   16  0.000000    0.000000    0.000000    0.000000    38.145827   -76.425533  76.200000   1
2   0   3   16  0.000000    0.000000    0.000000    0.000000    38.145827   -76.434567  76.200000   1
3   0   3   16  0.000000    0.000000    0.000000    0.000000    38.145484   -76.425533  76.200000   1
4   0   3   16  0.000000    0.000000    0.000000    0.000000    38.145484   -76.434567  76.200000   1
5   0   3   16  0.000000    0.000000    0.000000    0.000000    38.145141   -76.434567  76.200000   1
6   0   3   16  0.000000    0.000000    0.000000    0.000000    38.145141   -76.425533  76.200000   1
7   0   3   16  0.000000    0.000000    0.000000    0.000000    38.144798   -76.425533  76.200000   1
8   0   3   16  0.000000    0.000000    0.000000    0.000000    38.144798   -76.434567  76.200000   1
9   0   3   16  0.000000    0.000000    0.000000    0.000000    38.144455   -76.434567  76.200000   1
10  0   3   16  0.000000    0.000000    0.000000    0.000000    38.144455   -76.425533  76.200000   1
11  0   3   16  0.000000    0.000000    0.000000    0.000000    38.144112   -76.425533  76.200000   1
12  0   3   16  0.000000    0.000000    0.000000    0.000000    38.144112   -76.434567  76.200000   1
13  0   3   16  0.000000    0.000000    0.000000    0.000000    38.143769   -76.434567  76.200000   1
14  0   3   16  0.000000    0.000000    0.000000    0.000000    38.143769   -76.425533  76.200000   1
15  0   3   16  0.000000    0.000000    0.000000    0.000000    38.143426   -76.425533  76.200000   1
16  0   3   16  0.000000    0.000000    0.000000    0.000000    38.143426   -76.434567  76.200000   1
17  0   3   16  0.000000    0.000000    0.000000    0.000000    38.143083   -76.434567  76.200000   1
18  0   3   16  0.000000    0.000000    0.000000    0.000000    38.143083   -76.425533  76.200000   1
19  0   3   16  0.000000    0.000000    0.000000    0.000000    38.142740   -76.425533  76.200000   1
20  0   3   16  0.000000    0.000000    0.000000    0.000000    38.142740   -76.434567  76.200000   1
21  0   3   16  0.000000    0.000000    0.000000    0.000000    38.142397   -76.434567  76.200000   1
```

Figure 2 Generated Waypoint File

way]

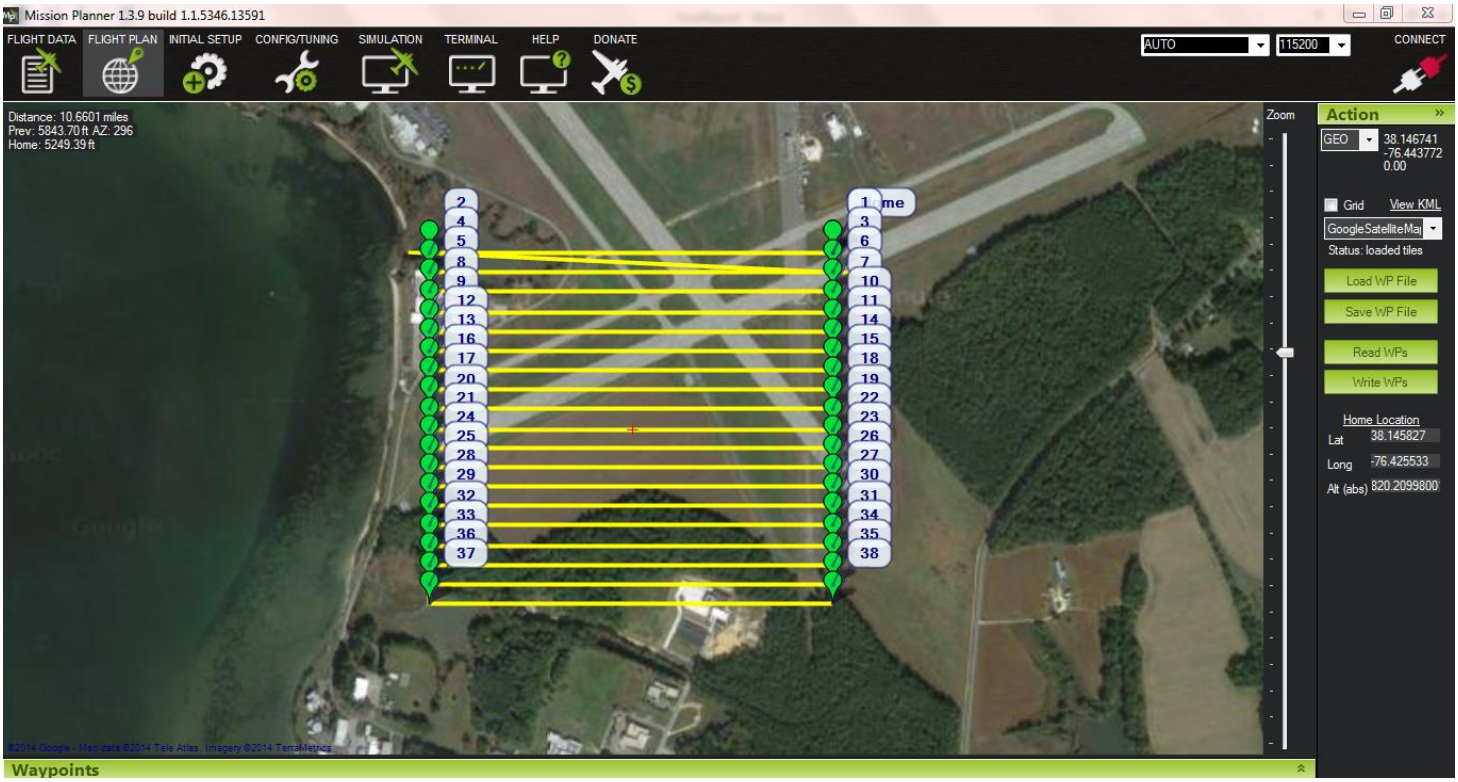

Figure 3 Loaded Flight Plan

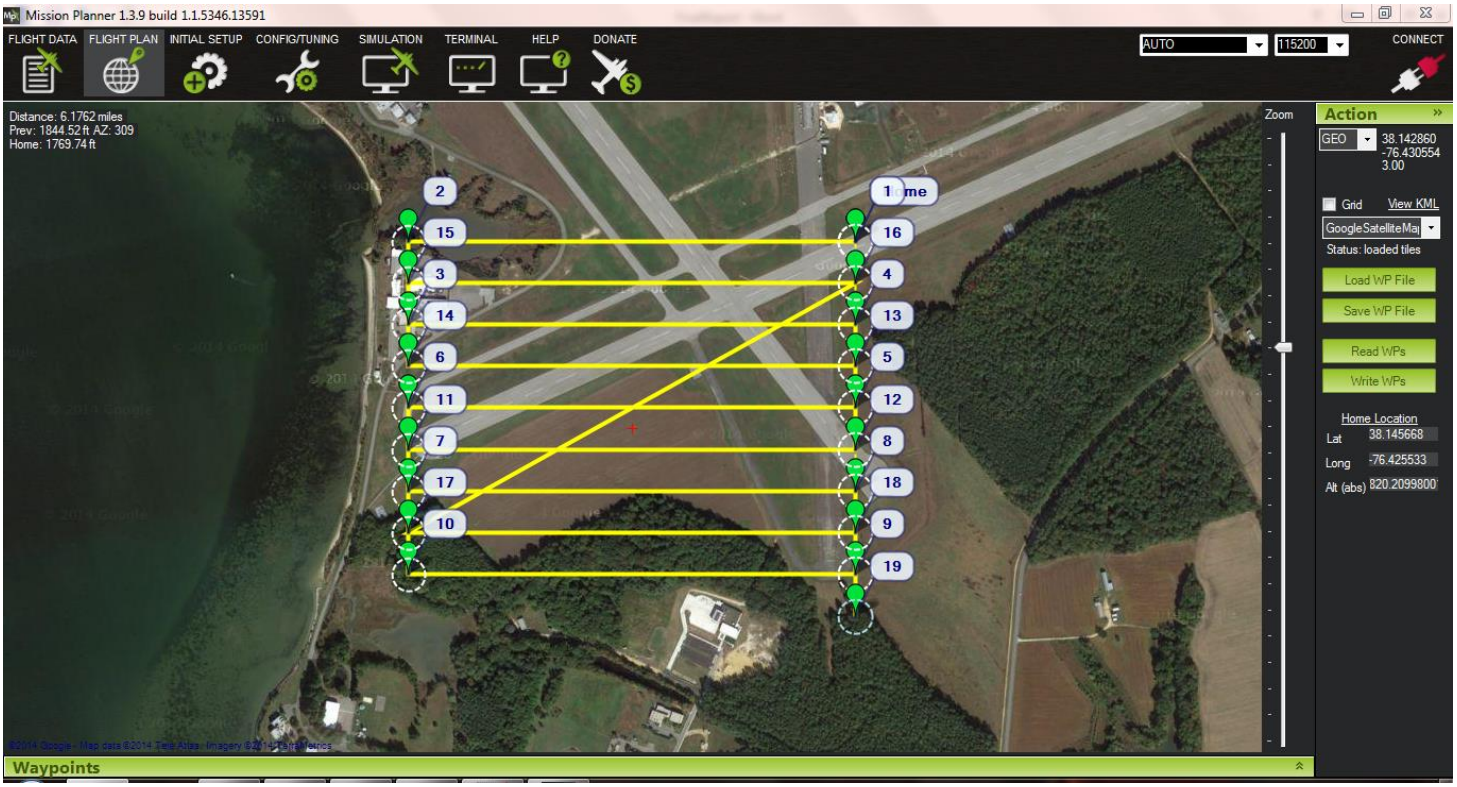

Figure 4 Loaded Flight Plan with Alternating Lawnmower

GUI

To speed up and simplify the use of this software, a GUI has been developed using PyQt4 to make use of all of the previously described functions and create a flight path with the click of a button.

The GUI is based off of a previously developed GUI in the ME701 class. It contains input boxes for polygon points to be input in DMS, an altitude, distance, and angle. The polygon points must be input in a 3 dimensional numpy array type with the first entry being degrees, the second being minutes, and the third being seconds. The input must also be latitude first, longitude second. If there is an uneven number of entries, the GUI will return an error message. An example of how the input should look is below:

$$[[D_{lat}\ ,\ M_{\ lat}\ ,S_{lat}],[D_{long}\ ,\ M_{\ long}\ ,S_{long}]]$$

When the "Create Flight Path" button is pushed, the flight path is graphed on the screen. If the user approves of the plan, they may go to the File > Save As menu and save the generated plan as a waypoint file for use in Mission Planner. If for some reason the user decides they don't want to use that feature, the GUI also outputs the converted polygon points to the screen. These could easily be copied into a text file and uploaded to Mission Planner as a .poly file for use in the Mission Planner flight path generating. Two GUI windows can be viewed in Figures 5 & 6, they contain a latitudinal flight plan and a longitudinal flight plan, respectively.

[NOTE: The save as button does not currently work. I am trying to get it to work right now. If I get it to work, I may add a function to save a .poly file too.]

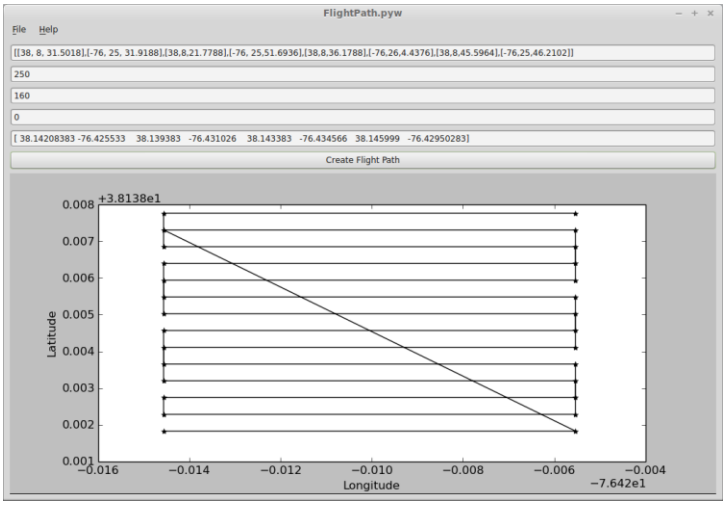

Figure 5 Latitudinal Flight Plan

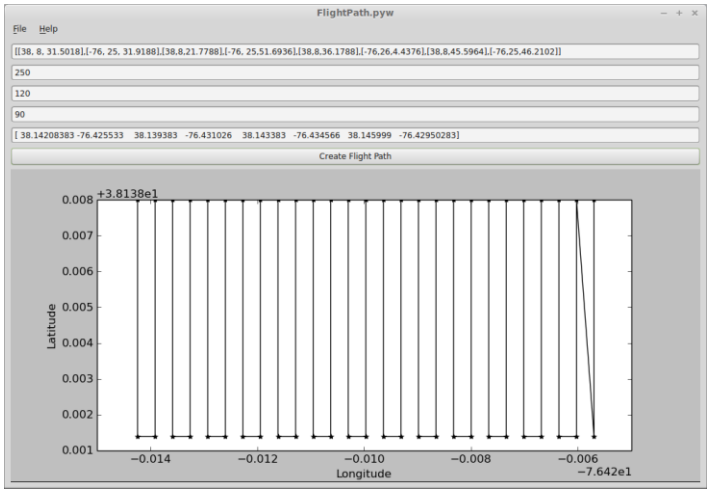

Figure 6 Longitudinal Flight Plan

CONCLUSION

The use of this automatically generated flight plan software will greatly speed up the pop up target process on the flight line during competition. Although no angled flight plan is currently calculable, the pop up search area is small and can easily navigated either east-west or north-south. The use of a GUI makes the flight plan process much more smooth and user friendly.

REFERENCES

[1] Weisstein, Eric W., 1999, "Rotation Matrix." from http://mathworld.wolfram.com/RotationMatrix.html

[2] Ippolito, Andrew et al., 2012, "APM Mission Planner 1.2.1 –R2 Final." from http://www.quadforge.net/docs/G03-APM%20mission%20planner%201.2.1%20-R2%20Final.pdf