Wind Estimation and Effects of Wind on Waypoint Navigation of UAVs

by

Anandrao Shesherao Biradar

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2014 by the
Graduate Supervisory Committee:

Srikanth Saripalli, Co-Chair
Spring Berman, Co-Chair
Jekan Thanga

ARIZONA STATE UNIVERSITY

May 2014

ABSTRACT

The presented work in this report is about Real time Estimation of wind and analyzing current wind correction algorithm in commercial off the shelf Autopilot board. The open source ArduPilot Mega 2.5 (APM 2.5) board manufactured by 3D Robotics is used. Currently there is lot of development being done in the field of Unmanned Aerial Systems (UAVs), various aerial platforms and corresponding; autonomous systems for them. This technology has advanced to such a stage that UAVs can be used for specific designed missions and deployed with reliability. But in some areas like missions requiring high maneuverability with greater efficiency is still under research area. This would help in increasing reliability and augmenting range of UAVs significantly.

One of the problems addressed through this thesis work is, current autopilot systems have algorithm that handles wind by attitude correction with appropriate Crab angle. But the real time wind vector (direction) and its calculated velocity is based on geometrical and algebraic transformation between ground speed and air speed vectors. This method of wind estimation and prediction, many a times leads to inaccuracy in attitude correction. The same has been proved in the following report with simulation and actual field testing. In later part, new ways to tackle while flying windy conditions have been proposed.

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Prof. Srikanth Saripalli for the continuous support in my Masters Study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me throughout research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Masters research.

I would also like to thank Prof. Spring Berman for support and guidance throughout the thesis. She helped me improve different aspect of research with their different point of view. I am thankful to Prof Jekan Thanga for reviewing my thesis and helping improve it in various areas.

I can’t be thankful enough to my friend Mr. Sai Vemperala for setting up software platforms; and giving continuous help whenever needed. I appreciate the help in setting up Wind Sensor hardware by Alexander Kafka. I am also thankful to my friend Prasanna for being continuously involved in discussion on different aspects of thesis which generated various ideas.

I would also like to thank my parents, and younger sister for their endless support. They always encouraged me with their best wishes.

I always got many ideas during various discussions with my colleagues and friends. They were always available to help me in the software issues and helped me a lot in field testing. Finally I am thankful to various student organizations I was involved with. It helped me a lot to learn various things, which I would never be able to learn through classes.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

UAVs : Unmanned Aerial Vehicle

UAS : Unmanned Aerial Systems

SITL : Software in the Loop Simulation

FDM : Flight Dynamics Model

AS : Airspeed

GS : Ground Speed

HH:MM:SS : Hours : Minutes : Seconds

MM:SS : Minutes : Seconds

LIST OF FIGURES

FIGURES Page

LIST OF TABLES

1. INTRODUCTION

- **Motivation**

Current advancements in UAV technologies such as hardware reliability, controls and open source autopilot systems (APM 2.5) has made it possible by just using Google Maps point and click features, plan your mission, specify altitude, toggle between different flight modes. It also helped setup a good platform for using UAVs for maneuvering missions and long duration flights in surveillance. One of the well-known and growing interest applications of UAVs is 3D Terrain mapping. Many commercial image processing tools are available, which will generate the requirements for frontal and sideways overlap for capturing images. This gives a good idea about what altitude and latitude-longitude the images have to be taken. The images are triggered at time intervals by setting up waypoints in Lawn Mower Pattern. But there are nonlinear natural conditions always present to challenge these systems such as winds. The control algorithms do compensate for this wind disturbance, but there are no real time wind speed and directions sensors onboard. The absence of direct wind measurements that can update correct wind data lead to inaccuracy in attitude corrections applied by autopilot board. Because of this error in Wind estimation, UAVs drift a lot from their course and they tend to miss the waypoint and come back again to pass through the waypoints.

The drift in waypoint following results in reduced endurance and less area coverage in mapping missions. This report will bring out these differences and open up area for newer algorithm development.

- **Problem Statement**

1) Using Software in Loop Simulations show the effects of wind from various directions with different speeds on selected UAV model. Show how it affects the waypoint achievement and drifting from designated Path following.
2) Through actual flying of UAV, with the wind sensor on board, show the error in wind direction and speed calculated in APM 2.5 with independent wind direction and speed sensor.

- **Overview of the Thesis**
    - The current work done in this area
    - Current methods of wind handling in UAVs
    - Sensors currently used for wind measurements in Aircrafts
    - Experimental setup, discussing different instruments used
    - Simulation environment, about Software in the loop simulations
    - Flight testing characteristics
    - Current algorithm used for Wind Estimation
    - Flight Dynamics Model used in SITL
    - Effects of Wind from various directions and with different speeds
    - Discussion of actual field testing data and showing differences in wind velocity estimations
    - Suggested correction methods.

2. CURRENT SCENARIOS

- **Background**

Unmanned Aerial Vehicles (UAVs) have lots of applications in various mission scenarios. The development in various areas has increased their endurance, payload capacity, speed, reduced mission costs.

There are many defense applications such as,

- Various surveillance scenarios
- Target mission, reaching particular location in munitions applications
- Following a moving target, and many more.

Figure 1, US Reaper UAV in combat

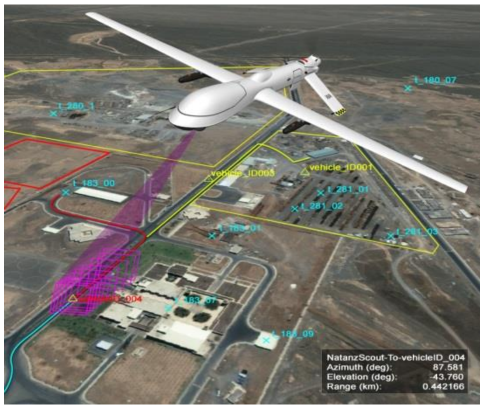


Figure 2, UAV in surveillance

There are many civil applications as mentioned below,

- 3D mapping
- Search and rescue
- Weather monitoring and many more

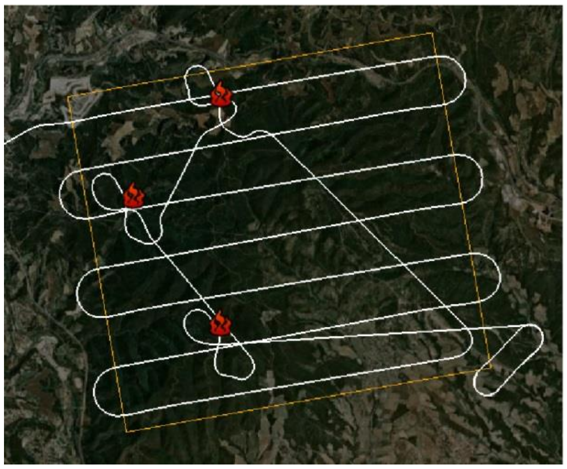


Figure 3, Mapping Using Lawn Mower Pattern

Among the uses mentioned above, the current work is more focused on performance of UAVs in missions which require lot of maneuvering while navigating through waypoints. UAV maneuvering becomes more challenging in conditions where it needs to perform a minimum radius turns. The important condition when UAV perform minimum radius turn is UAV needs to follow a coordinated turn. Aircraft is said to be following a coordinated turn when it doesn't have any side sleep in turn.

The path planning of maneuvering missions depends on various parameters of UAV, some of them are;

- UAV dynamics
- Maneuvering capabilities
- Range
- Cruise speed
- Wind conditions

Most of these parameters are known or we can have a good estimation based on the UAV design before the flight in mission planning phase. But accurate local wind conditions are difficult to calculate and predict, that’s why it’s estimated onboard and control actions are applied accordingly. The methods and algorithms used for estimation of wind plays very important role in robust wind tackling algorithms.

- **Some Earlier methods**

  Wind conditions were used in various ways in earlier UAV applications.

1. Wind soaring technique:

   Wind soaring techniques were most popular ways to reduce energy consumption onboard by flying in existing wind directions. Soaring is the process of gaining energy from the atmosphere in-flight using an aerodynamic free-flying platform. Dynamic soaring utilizes the energy available in vertical wind gradients; this allows UAVs to spend less energy to generate lift. [Lawrence et al, 2009]

2. Boeing Condor UAV:

   “Fly in backwards” this concept was implemented in Boeing condor UAV. The airfoil with additional control techniques was designed in such a way that allows UAV to hover or fly backwards avoiding stall conditions. Boeing Condor set several records for piston-powered aircraft by reaching a top altitude of 67,028 feet and staying aloft for nearly two and one-half days [http://www.boeing.com/boeing/history/boeing/condor]

Figure 4, Boeing Condor Unmanned Aerial Vehicle

- Significance of wind in earlier methods:

  These techniques were more focused on sustaining and changing current flight course to use windy conditions rather than maneuvering through wind to follow preplanned mission. Preplanned missions will require UAVs to follow waypoints in particular sequential order irrespective of wind conditions. This makes it challenging to fly with varying attitudes towards with respect to average wind. UAV needs to turn in different direction based on the waypoint order. Turning of UAV changes the apparent wind conditions with respect to UAV frame. The accurate wind estimation becomes more significant in turning conditions in comparison to techniques discussed above.

- **Current work in Wind Estimation**

  Recently, there has been significant work done in the area of path planning of UAVs with consideration of Wind effects by Air Force Institute of Technology in Ohio, Aerospace Control Laboratory at MIT, and University of Washington.

But most of the work is focused more on wind prediction techniques than direct wind measurement.

"An aircraft in flight is airborne and subject to the movement of the air mass in relation to the surface; i.e. the wind. The relatively low cruising speed of light aircraft makes them particularly affected by wind velocity. Consequently the calculation of the wind effect on aircraft movement relative to the ground is a major part of light aircraft flight planning and navigation." [John Brandon, 2013]
This quote explains the need of direct wind vector calculation with respect to earth frame on light aircrafts i.e. UAVs.

- Current Methods

There are different kinds of wind estimation methods used currently. Some of the methods are discussed below,

1. Using GPS Velocity

Lagelaan et al. [13] elaborates this method as, GPS provides a direct measurement of velocity with respect to the Earth, accurate to approximately 0.1 m/s. Carrier phase differential GPS allows velocity measurements accurate to the order of millimeters per second) [16]. Thus, local air mass velocity components can be obtained directly from vehicle kinematics and the GPS velocity.

$$\begin{bmatrix} W_x \\ W_y \\ W_z \end{bmatrix} = \begin{bmatrix} S_x \\ S_y \\ S_z \end{bmatrix} - \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix}$$

W = Wind speed vector over ground

S = Ground speed (GPS Velocity)

V = Air speed vector in earth frame

The method is based on autopilot module providing estimated components of airspeed and Euler angles. This method doesn't consider the changes in magnetic heading and turn rate while predicting the wind vector. This leads to inaccuracy in wind direction and velocity prediction.

2. Computing Wind from Vehicle Response [13]

This method is based on comparison of measurements of aircraft motion with respect to the Earth (GPS measurements) with predictions of aircraft motion obtained from UAV dynamic model (based on aircraft acceleration with respect to surrounding air).

This method doesn't produce good results due to GPS uncertainty is of the order of 3 m. The carrier phase differential GPS does improve position accuracy to the order of centimeters but this is still noisier than GPS velocity estimates. The aircraft dynamic model takes into account aerodynamic forces, which depends on wind gradients. If the wind gradients are large, then predicted forces will be significantly different from actual forces, resulting an error in UAV dynamic model.

3. Wind Triangle method

Wind triangle method for wind estimation [6] works on GPS ground track vector and true airspeed vector and considers turning rate (yaw rate) of aircraft. As shown in figure (5), this method works on predicting wind vector (V_w) based on geometric transformation between true airspeed vector (V_a) and ground speed vector(V_g).

The algorithm based on this method is discussed in detail in Chapter 4. The wind triangle method is used in Wind estimation algorithm in Ardupilot Mega 2.5 autopilot board used in this work.

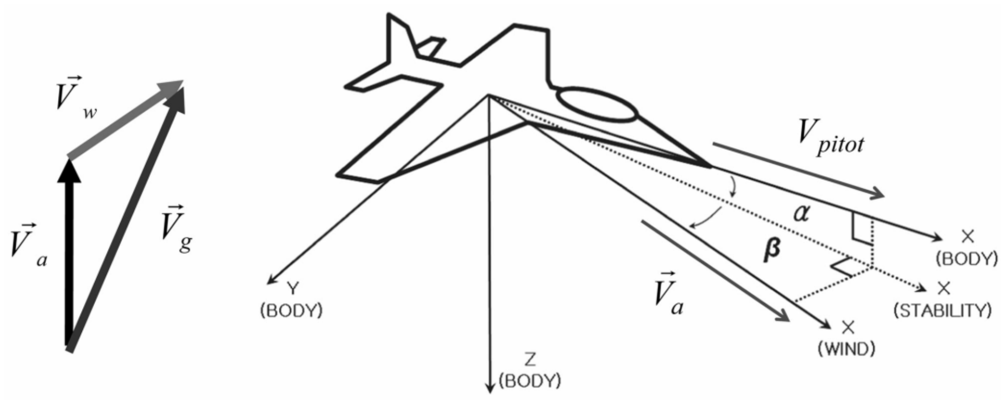


Figure 5, Wind triangle method for wind estimation

- **Summarizing current methods**

The various methods discussed above are mainly prediction based method rather than direct measurement. These methods predict wind speed and direction based on airspeed sensor and GPS measurements. These methods don’t use direct wind measurement sensor to calculate wind speed and direction.

The current work has shown the inaccuracy of prediction based wind estimation techniques used in current commercial of the shelf autopilot boards. The estimated wind is compared with the independent wind sensor mounted onboard of UAV. This independent wind sensor gives true wind speed and direction data of actual flying field environment. The prediction based techniques works very well in simple waypoint following mission. But these estimations won’t be optimal, leading to significant cross-track error in maneuvering missions. Maneuvering missions involves challenging waypoints navigation which requires more dynamic and real time wind estimation in UAVs.

- **Flying in windy conditions**

The best way to fly in windy conditions is flying into direction of wind, i.e. when the fuselage is parallel to the wind. Flying into the wind will allow maximum wind speed on leading edge of wing airfoil, as they are designed in such a way. But this isn’t always possible as the path planned may have orientation either away from or at an angle with respect to the wind direction with no choice. In such cases the aircrafts are flown with their nose pointing towards the wind instead of directly towards planned path. The angle between Aircraft heading and desired course is known as “Crab angle”. The calculation of crab angle depends on various factors such as wind direction, magnitude, aircraft velocity and the course it needs to follow. The current algorithms are designed in such a way that they calculate the crab based on the predicted wind direction and speed. In following figure (6) [Britannica] the first picture shows a drift in wind without applying any attitude correction and second picture shows applied attitude correction to maintain the flight path by turning the nose towards the wind.

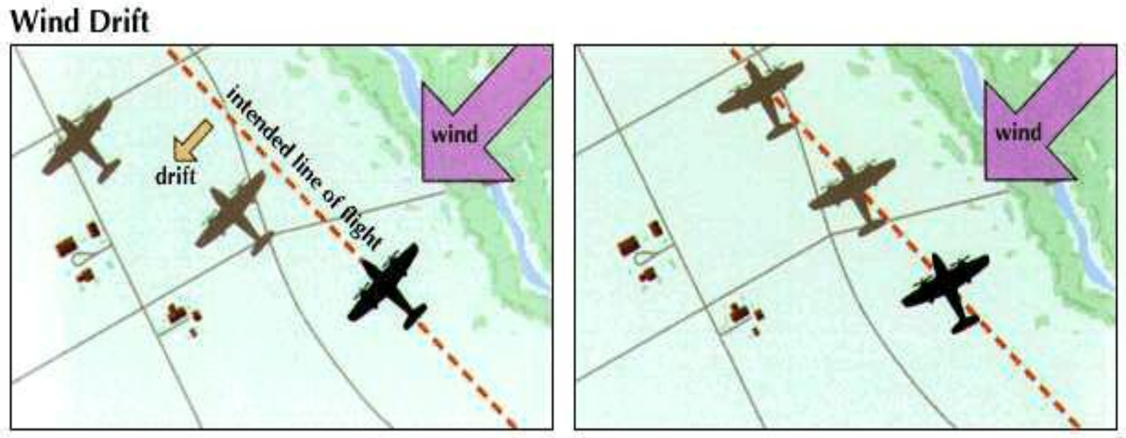


Figure 6, Wind Drift

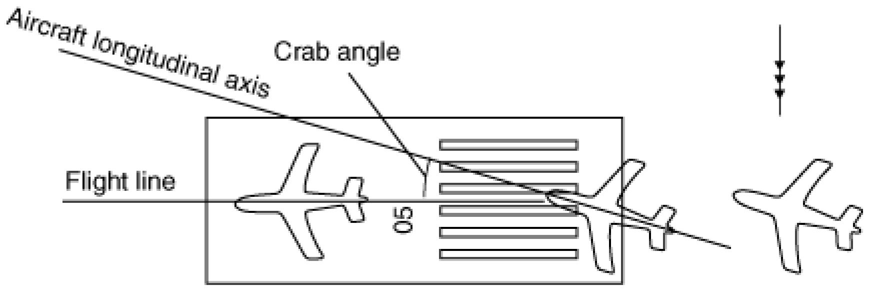


Figure 7, Crab Angle

3. EXPERIMENTAL SETUPS

This section contains description of most of the hardware used in this experiment.

- **Unmanned Aerial Vehicle**

  We used an RC aircraft known as Skywalker. It is a ready to fly preassembled kit airplane from Event 38 Unmanned Systems. It's a remote controlled electric powered plane and provides good platform for surveillance purposes. It is pusher propeller configuration which gives more space in forward area of fuselage for more equipment installation. It requires 30-35% of maximum throttle for cruise conditions and flies with cruise speed of 12 m/s. This cruise speed is slow enough for image capturing missions. These images would be later used in software called Agisoft for generating 3D map of whole terrain.

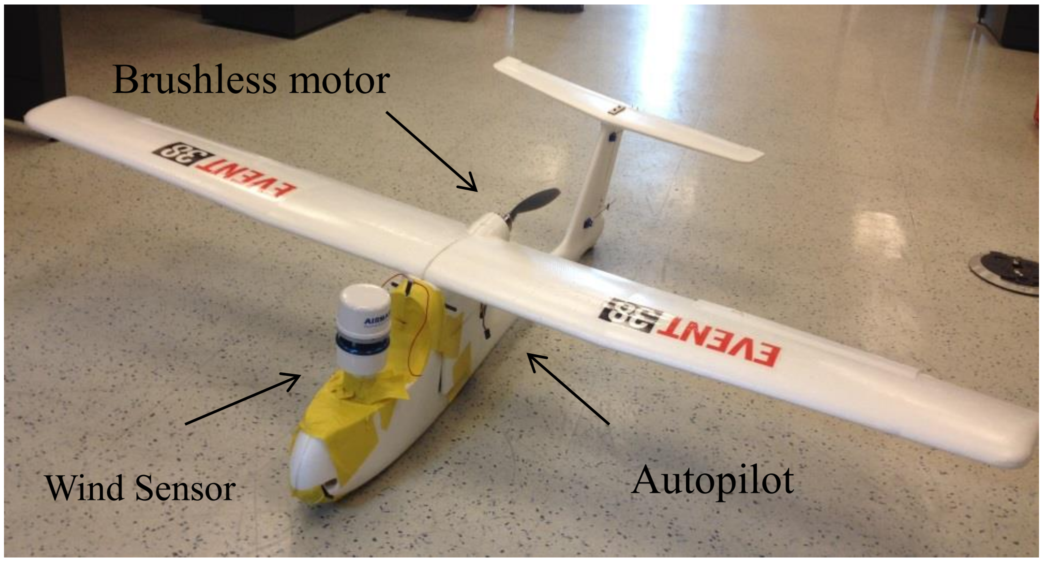


Figure 8, Skywalker UAV with AIRMAR Wind Sensor

- Skywalker Characteristics:

Table 1

Skywalker Airplane Characteristics

Wing Span	1800 mm
Length	1300 mm
Cruise Speed	12 m/s
Max Speed	42 m/s
Normal flight time	30-50 min 20-25 min with 1.2kg payload

- **ArduPilot Mega 2.5**

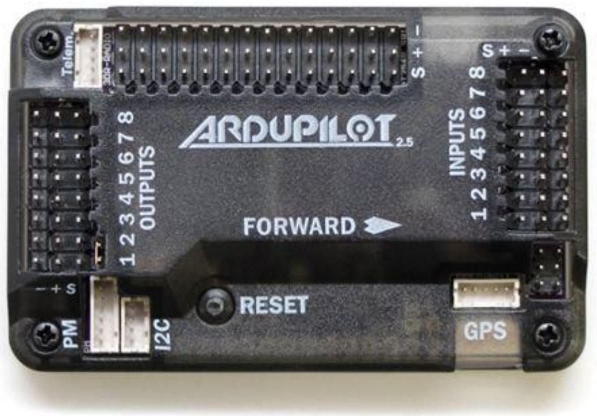


Figure 9, Ardupilot Mega 2.5 board

The heart of the control system used in UAV is open source developed ArduPilot Mega version 2.5 board, popularly known as APM 2.5 purchased from 3DRobotics Inc. Its Arduino compatible, Atmel's ATMEGA2560 based board with following features:

- 3-axis gyro, accelerometer and magnetometer, along with a high-performance barometer
- Digital compass powered by Honeywell's HMC5883L-TR chip
- Mediatek MT3329 GPS Module.
- It uses Invensense 6 DoF Accelerometer/Gyro MPU-6000.
- Barometric pressure sensor MS5611, MEAS High Resolution Altimeter.

- **Mission Planner**

  Mission Planner is open source Ground Control Station software with following features:

  - Point-and-click waypoint entry, using Google Maps
  - Select mission commands from drop-down menus
  - Download mission log files and analyze them, covert them to KMZ for Google earth data processing
  - Configure APM settings for airframe
  - See the output from APM’s serial terminal sent over telemetry

Mission Planner was also used to setup Software in the Loop simulation. Mission Planner was really important part of simulation to control the flight conditions, to verify conditions setup in the simulation script in an effort to simulate as close to real time scenarios as possible.

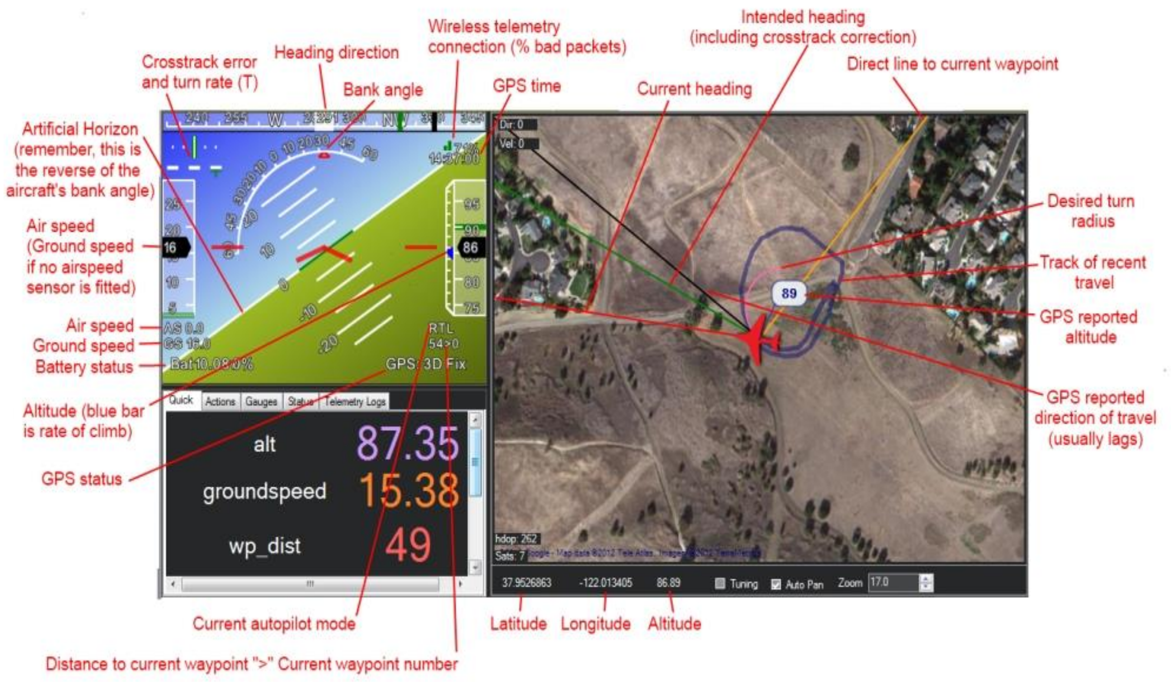

Crosstrack error and turn rate (T)
Heading direction
Wireless telemetry connection (% bad packets)
Intended heading (including crosstrack correction)
Direct line to current waypoint
Bank angle
GPS time
Current heading
Artificial Horizon (remember, this is the reverse of the aircraft's bank angle)
Air speed (Ground speed if no airspeed sensor is fitted)
Desired turn radius
Track of recent travel
GPS reported altitude
Air speed
Ground speed
Battery status
Altitude (blue bar is rate of climb)
GPS reported direction of travel (usually lags)
GPS status
alt 87.35
groundspeed 15.38
wp_dist 49
Current autopilot mode
Latitude
Longitude
Altitude
Distance to current waypoint ">" Current waypoint number

Figure 10, Mission Planner Layout, from DIY Drones

- **Airspeed Sensor**

  An airspeed sensor has been used to calculate airspeed with Pitot tube setup; this data was used in conjunction with Ground Speed calculated by GPS to predict wind speed and its direction.

Features:

- ➢ Measure pressure up to 7kPa through each port for pressure sensing but also for vacuum sensing.
- ➢ -2 to 2 kPa (-0.3 to 0.3 psi).

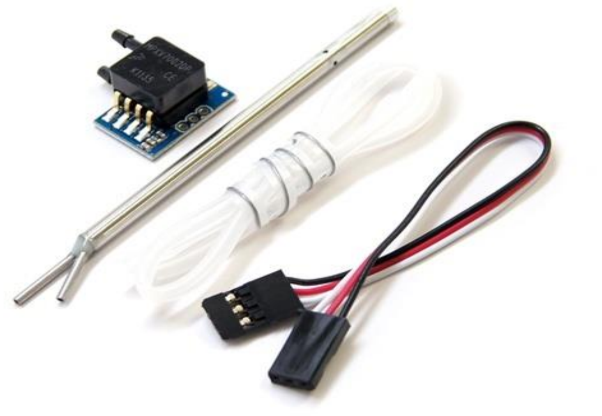


Figure 11, Airspeed Sensor, 3DRobotics

- **AIRMAR Weather sensor**

  AIRMAR PB200 weather sensor has been used to collect real time wind data. The AIRMAR Weather Station Instrument is the only all-in-one weather sensor that calculates apparent wind speed and direction, barometric pressure, air temperature, and wind chill temperature. With the internal compass and WAAS/EGNOS GPS, true wind speed and direction can also be calculated.

The AIRMAR wind sensor was really helpful in getting actual wind data such as true wind speed with respect to ground fame, apparent wind speed, which is corresponding to airspeed of the airplane. It also gives these data with the true magnetic heading which was used to compare the wind direction predicted by APM 2.5.

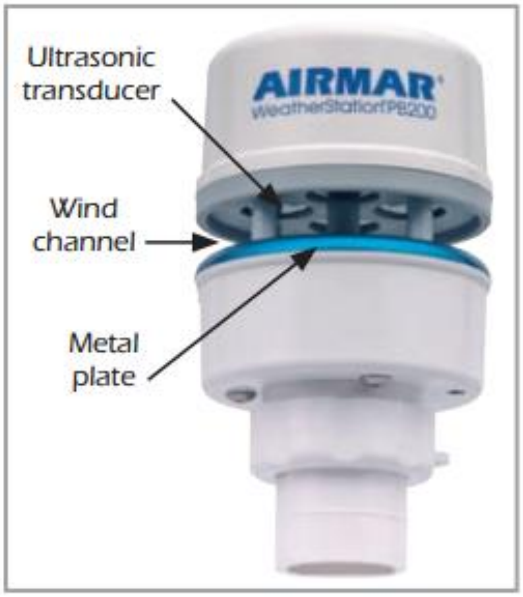


Figure 12, AIRMAR Sensor

- Wind Speed Range: 0 knots to 80 knots (0 MPH to 92 MPH)
- Wind Speed Resolution: 0.1 knots (0.1 MPH)
- Wind Direction Range: 0° to 360°
- Wind Direction Resolution: 0.1°

- **Field Conditions at flight testing field** ; W Fry Rd, Chandler

The temperature ranges from 50°F to 80°F, with wind speeds from 2 mph to 20 mph. The area shown approx. 2000*2000 ft.

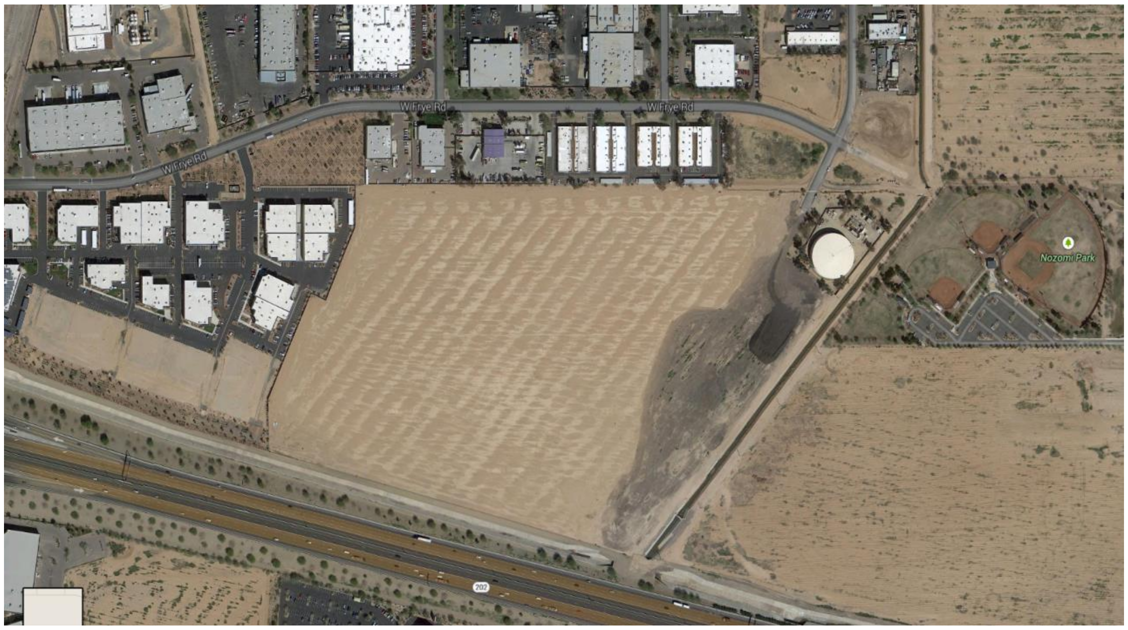


Figure 13, Google maps Field view of Chandler flight test location.

4. SIMULATIONS

- ## Overview of SITL:

Software in the Loop modeling and simulation (SITL) setup has been used for all simulation. The advantages are described as follows by Special Interest Group on Simulation and Modeling:

- Software-in-the-loop Modeling & Simulation can be viewed as Simulation-based Software Evaluation.
- A software system can be executed under simulated input conditions for the purpose of evaluating how well the software system functions under such input conditions.
- Software-in-the-loop Modeling & Simulation is a cost-effective method for evaluating a developed, mission-critical software system before it is used in the real world.

In current scenario, the SITL was used to fly a particular model to follow generated a flight path from Mission Planner software and simulate the wind conditions from various directions. The effects of wind on UAV with different directions and various speeds were observed with SITL simulation.

In Figure (14) shows how SITL architecture works [DIY Drones community].

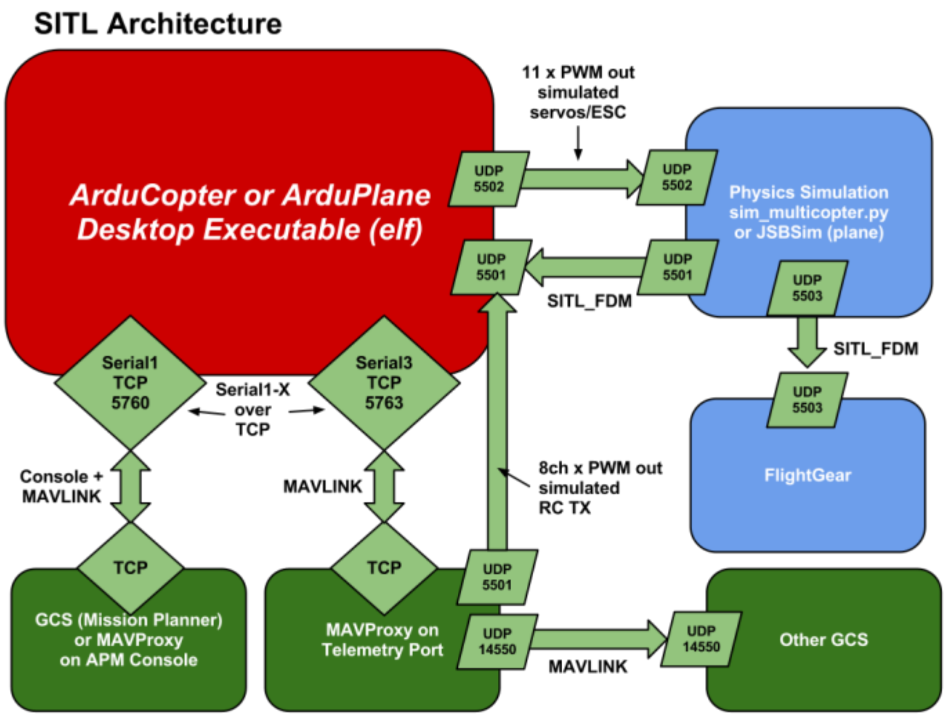

SITL Architecture
ArduCopter or ArduPlane Desktop Executable (elf)
11 x PWM out simulated servos/ESC
UDP 5502
UDP 5502
Physics Simulation sim_multicopter.py or JSBSim (plane)
UDP 5501
UDP 5501
SITL_FDM
UDP 5503
SITL_FDM
UDP 5503
FlightGear
Serial1 TCP 5760
Serial1-X over TCP
Serial3 TCP 5763
Console + MAVLINK
MAVLINK
8ch x PWM out simulated RC TX
TCP
TCP
GCS (Mission Planner) or MAVProxy on APM Console
MAVProxy on Telemetry Port
UDP 5501
UDP 14550
MAVLINK
UDP 14550
Other GCS

Figure 14, SITL Architecture

How it works:

- The ArduPlane.elf will be built initially which will have the firmware. The firmware defines most flight parameters. The build contains most important part i.e. control algorithms designed to fly ArduPlane.
- The flight script ArduPlane.py shown in later section will be written considering all mission requirement to be tested. ArduPlane.py is also connected to runsim.py which calls for Rascal FDM from library.
- Then fly.ArduPlane command will start running which runs ArduPlane.py.

- The output of this flight will be displayed on Ground Control Software (GCS) Mission Planner. The mission can be changed and other parameters such as speed, modes can be changed real time via Mission Planner.

The part of SITL implemented in current simulations is shown in figure (15).

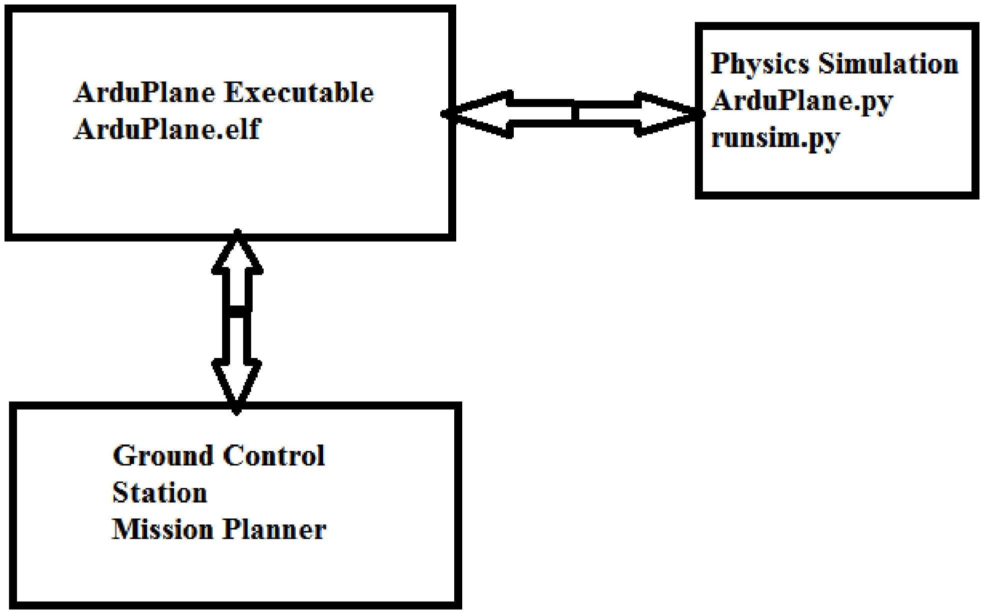


Figure 15, Part of SITL Architecture used

- **Parts of SITL**:

The following areas were important parts of Software in the loop simulation and modeling, it will give more idea about each component and the specifications used:

1) Flight Dynamics Model used from JSBSim
2) Flight Parameters and Mission Script
3) Simulating Wind conditions
4) Ground station - Mission Planner

Discussing each component in detail:

1) Flight Dynamics Model (FDM):

The SITL simulations results can be closest to actual aircraft by selecting closest flight dynamics model to actual test flown aircraft (in this work the test platform would be Skywalker). For current simulations Rascal 110 Flight dynamics model has been used and it's compiled by JSBSim (an open source flight dynamics model library). FDM defines all physics and dynamics associated with aircraft. FDM consists of all data such as dimension, weight, CG, moment of Inertia's etc. The whole FDM is the foundation to run the simulation for a particular model.

Rascal 110 FDM has following characteristics:

Table 2

Rascal 110 FDM Characteristics

Max weight	6 kg.
Wing span	2.8 m.
Length	1.92 m.
Wing Area	0.98 m^2
Cruise Speed	30.87 m/s
Maximum Speed	44 m/s

2) **Flight parameters-Simulation parameters**:

➢ Cruise speed:

The ArduPlane control algorithm tries to maintain the preplanned cruise speed. It also compensates for windy conditions such as headwind, side wind, which changes the aircraft speed accordingly and tries to bring back to preplanned cruise speed. The cruise speed was set to 20 m/s with considering reduction in throttle to better observe wind effects on actual path following than planned path.

➢ The PID gains:

The ArduPlane control algorithm works on PID gains. The simulation environments allows to tune PID gains for executing better flight controls and is important while comparing parameters to follow the planned path as accurately as possible.

3) Various wind conditions:

It is a well-known fact that planes fly best with wind velocities not exceeding their maximum achievable velocities. Based on this and as mentioned earlier the cruise velocity of Rascal FDM is 30.87 m/s, so we have set the maximum velocity of wind in simulation to be 25 m/s. The simulation input parameters are combination of parameters from column 1 and column 2 as stated in following table.

Table 3

Simulation Parameters

Wind Velocities	Direction
5 m/s	0°
10 m/s	90°
15 m/s	180°
20 m/s	270°

These simulated conditions will show the drift or path swayed away from actual planned path of UAV. The drift would change based on the direction and magnitude of wind speed vector.

4) **Flight script used in the simulations :**

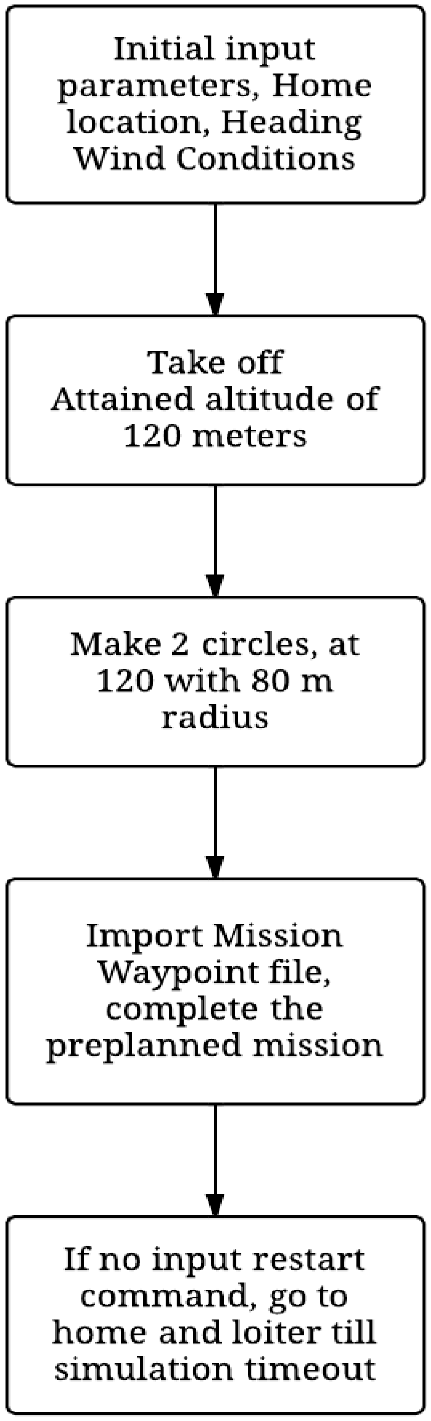

Initial input parameters, Home location, Heading Wind Conditions
Take off Attained altitude of 120 meters
Make 2 circles, at 120 with 80 m radius
Import Mission Waypoint file, complete the preplanned mission
If no input restart command, go to home and loiter till simulation timeout

Figure 17, Flow chart of the python based flight script.

This is the modified flight script and tailored to simulate mission for wind estimation and observing its effects. The original script was referred from GitHub directory of Ardupilot. (https://github.com/diydrones/ardupilot). The python code is shown in Appendix I.

5) Mission design:

The waypoints are setup in such way that UAV will follow a figure of 8. This configuration will orient UAV heading towards all directions.

The waypoint radius: The perimeter around the waypoint is defined so that UAV can have certain tolerance while passing through waypoint instead of just defining a single point in 3D space. The radius has to be set in such a way that it is not too small so that UAV will miss it even with a small wind or other disturbance. The radius shouldn't be too big so that it would drift from center of waypoint instead of passing through it. This is also basis for testing Lawn mower patterns for 3D mapping of terrain. The figure (18) shows the waypoint radius was defined bigger than maximum.

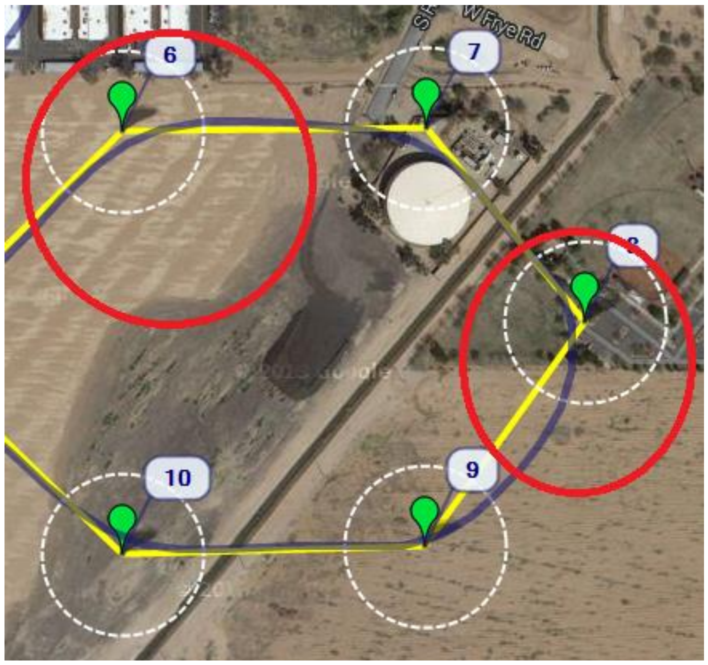


Figure 18, Defining waypoint radius

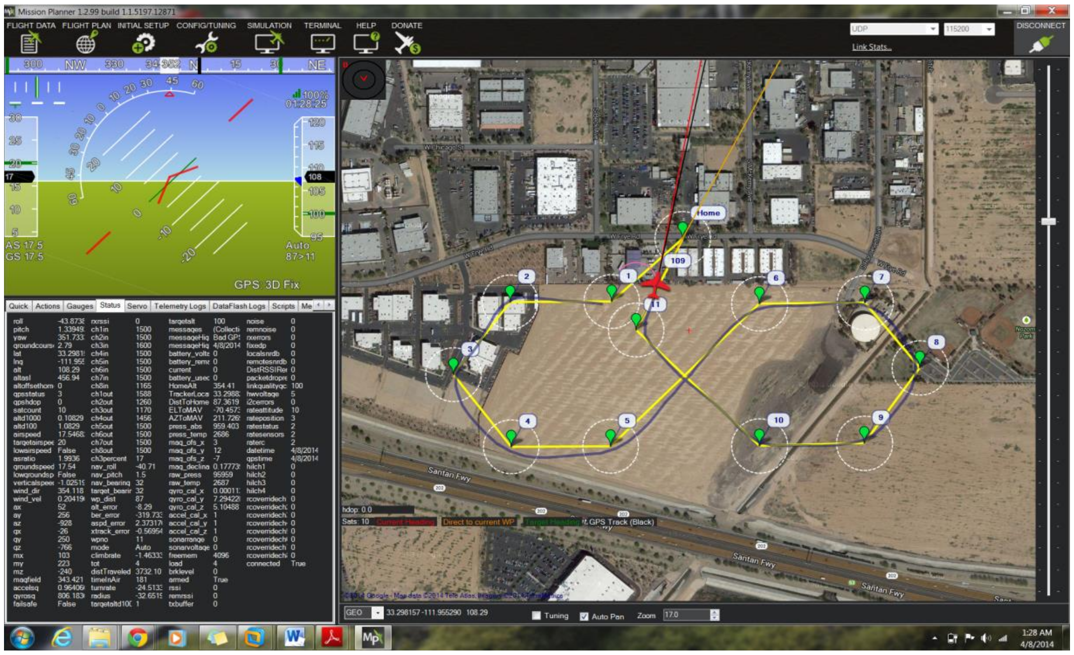
- **Simulations:**

The Software in The Loop simulation was performed with all combinations of input parameters mentioned in the table 3. The effects of the wind from different directions and magnitude can be seen by looking over the blue colored track.

Yellow track:	designated path generated by predefined missions containing Waypoints.
Blue Track:	Path followed by UAV

The dotted circles around waypoints: It shows the predefined waypoint radius.

The following results will show the drift from the preplanned path by UAV. The following image is a sample screen shot of Mission Planner displaying output of SITL simulation. It can be compared with the path followed by UAV in no input wind conditions (i.e. velocity 0 m/s) shown in figure 20.



```
roll            -43.873E  rxrssi          0          targetalt       100        noise           0
pitch           1.33949   ch1in           1500       messages        (Collecti  remnoise        0
yaw             351.733   ch2in           1500       messageHiq      Bad GP!    rxerrors        0
groundcours     2.79      ch3in           1600       messageHiq      4/8/2014   fixedp          0
lat             33.2981!  ch4in           1500       battery_volt    0          localsnrdb      0
lng             -111.95E  ch5in           1500       battery_rem     0          remotesnrdb     0
alt             108.29    ch6in           1500       current         0          DistRSSIRe      0
altasl          456.94    ch7in           1500       battery_usec    0          packetdropre    0
altoffsethom    0         ch8in           1165       HomeAlt         354.41     linkqualityqc   100
qpsstatus       3         ch1out          1588       TrackerLoca     33.2988    hwvoltage       5
qpshdop         0         ch2out          1260       DistToHome      87.3619    i2cerrors       0
satcount        10        ch3out          1170       ELToMAV         -70.457    rateattitude    10
altd1000        0.10829   ch4out          1456       AZToMAV         211.726    rateposition    3
altd100         1.0829    ch5out          1500       press_abs       959.403    ratestatus      2
airspeed        17.5468   ch6out          1500       press_temp      2686       ratesensors     2
targetairspee   20        ch7out          1500       maq_ofs_x       3          raterc          2
lowairspeed     False     ch8out          1500       maq_ofs_y       12         datetime        4/8/2014
asratio         1.9936    ch3percent      17         maq_ofs_z       -7         qpstime         4/8/2014
groundspeed     17.54     nav_roll        -40.71     maq_declina     0.17773    hilch1          0
lowqroundsp     False     nav_pitch       1.5        raw_press       95959      hilch2          0
verticalspee    -1.0251   nav_bearinq     32         raw_temp        2687       hilch3          0
wind_dir        354.118   target_bearir   32         qyro_cal_x      0.00011    hilch4          0
wind_vel        0.20419   wp_dist         87         qyro_cal_y      7.29422    rcoverridech    0
ax              52        alt_error       -8.29      qyro_cal_z      5.10488    rcoverridech    0
ay              256       ber_error       -319.73    accel_cal_x     1          rcoverridech    0
az              -928      aspd_error      2.37317    accel_cal_y     1          rcoverridech    0
qx              -26       xtrack_error    -0.56954   accel_cal_z     1          rcoverridech    0
qy              250       wpno            11         sonarrange      0          rcoverridech    0
qz              -766      mode            Auto       sonarvoltage    0          rcoverridech    0
mx              103       climbrate       -1.4633    freemem         4096       rcoverridech    0
my              223       tot             4          load            4          connected       True
mz              -240      distTraveled    3732.10    brklevel        0
maqfield        343.421   timeInAir       181        armed           True
accelsq         0.96406   turnrate        -24.513    rssi            0
qyrosq          806.183   radius          -32.651    remrssi         0
failsafe        False     targetaltd10(   1          txbuffer        0
```

Figure 19, Sample screen shot of Mission Planner

All the images shown are with standard upward north convention.

1) Figure 20 shows the simulation with following input conditions:

Wind speed: 0 m/s

Wind Direction: 0°

The shown output of simulation can be presumed as ideal flight conditions because of no input wind conditions. This would be benchmarking case for testing how the Rascal 110 flight model flies with no wind conditions.

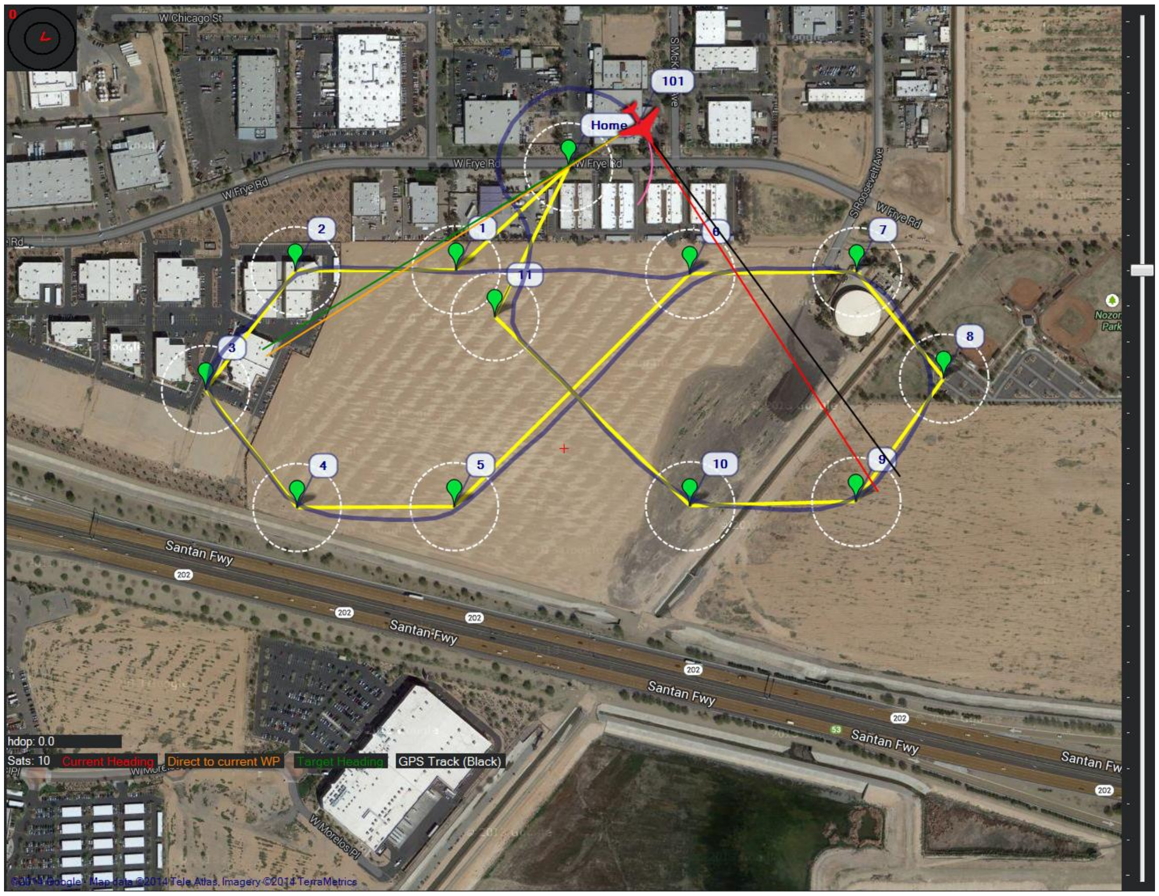


Figure 20, Wind Speed 0 m/s

Figure 20 shows near ideal situation with minimal natural disturbance and following almost close to planned yellow track.

2) The next result shows the effects with:

Wind Direction: 0°, from north to south

Wind Speed: 5 m/s

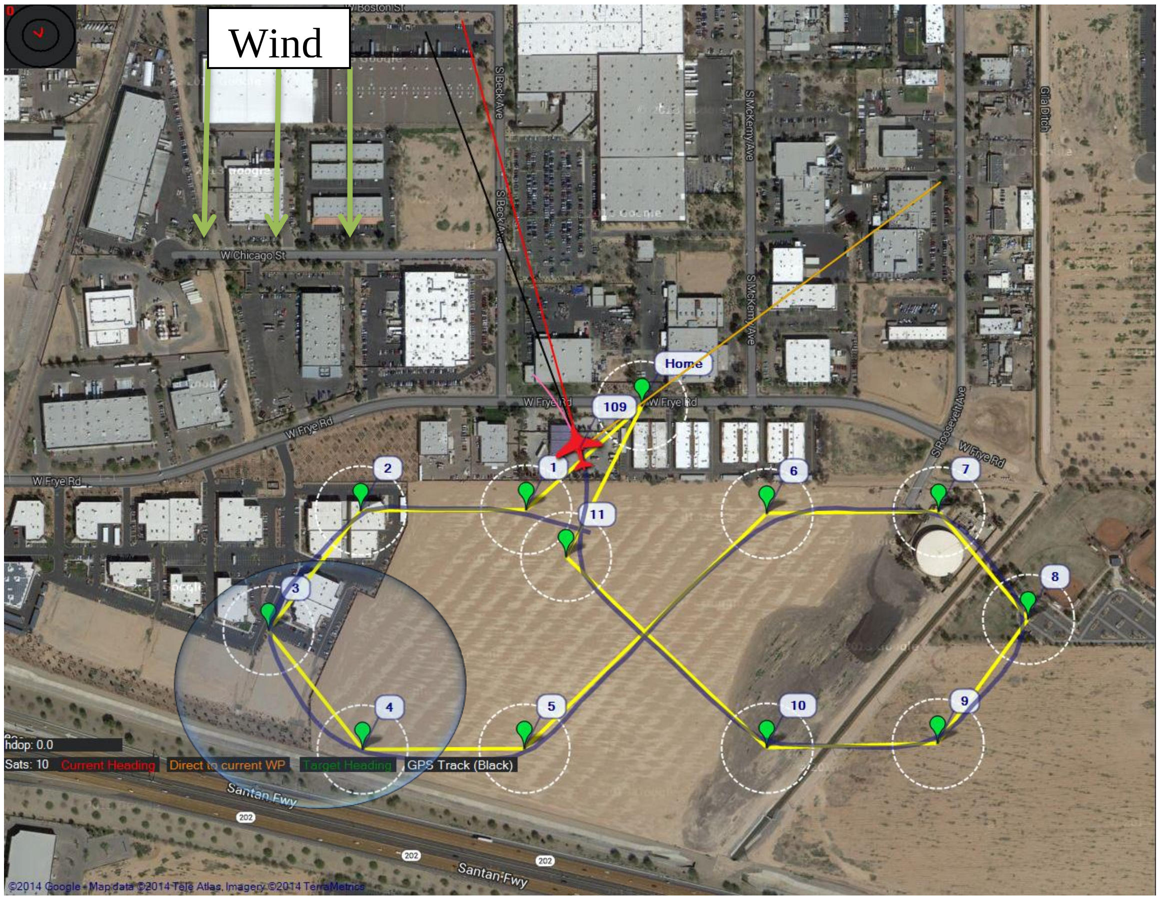


Figure 21, Wind speed 5 m/s

The drifted blue line track between waypoint 3 and 4 can be seen in circle marked. Wind speed of 5 m/s would be normal environmental conditions in Arizona, USA.

3) The next result shows the effects with:

   Wind Direction: 0°, from north to south

   Wind Speed: 10 m/s

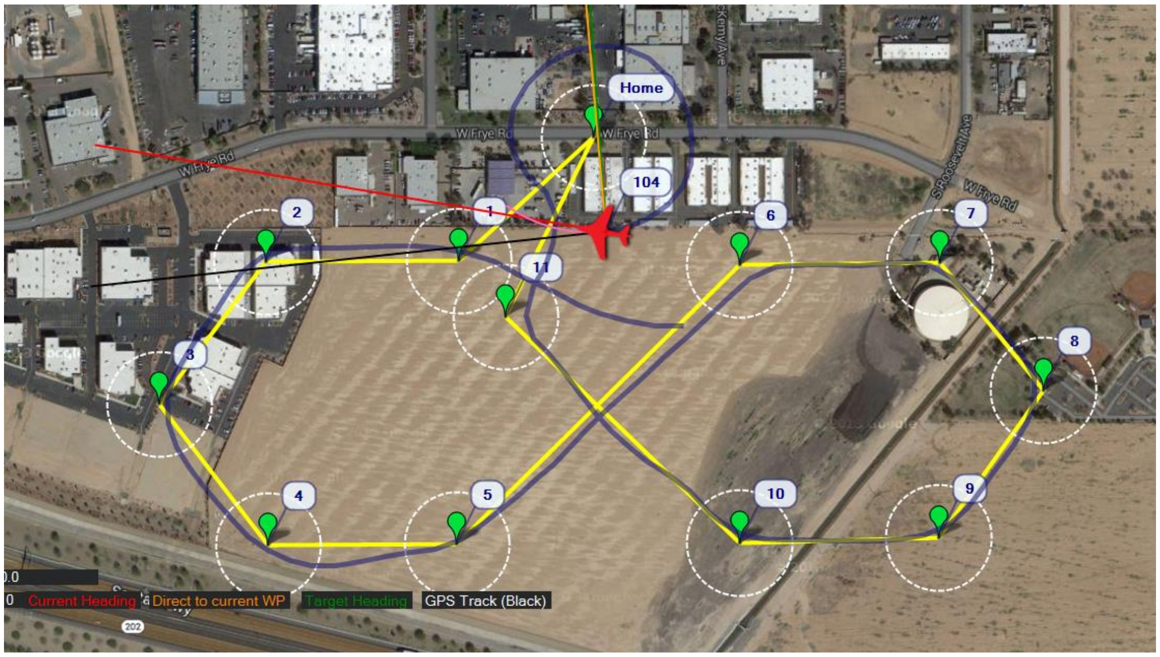


Figure 22, Wind speed 10 m/s

It can be noticed that the flight path has drifted more compared to Figure 22 between waypoint 3 to 4 and 4 to 5 also towards waypoint 6. In later part after 6 there isn’t significant cross track error. The significant error in this case would be a cross track error of more than 2 meters from the yellow planned path. Here is very important point to note when comparing the waypoint followed from 3 – 4 – 5, in this case UAV was experiencing a tail wind conditions and while approaching waypoint 6 it experiences headwind conditions. Applying attitude corrections in case of headwind conditions is faster than tailwind case as there would be better control in headwind conditions.

4) The next result shows the effects with:

Wind Direction: 0°, from north to south

Wind Speed: 20 m/s

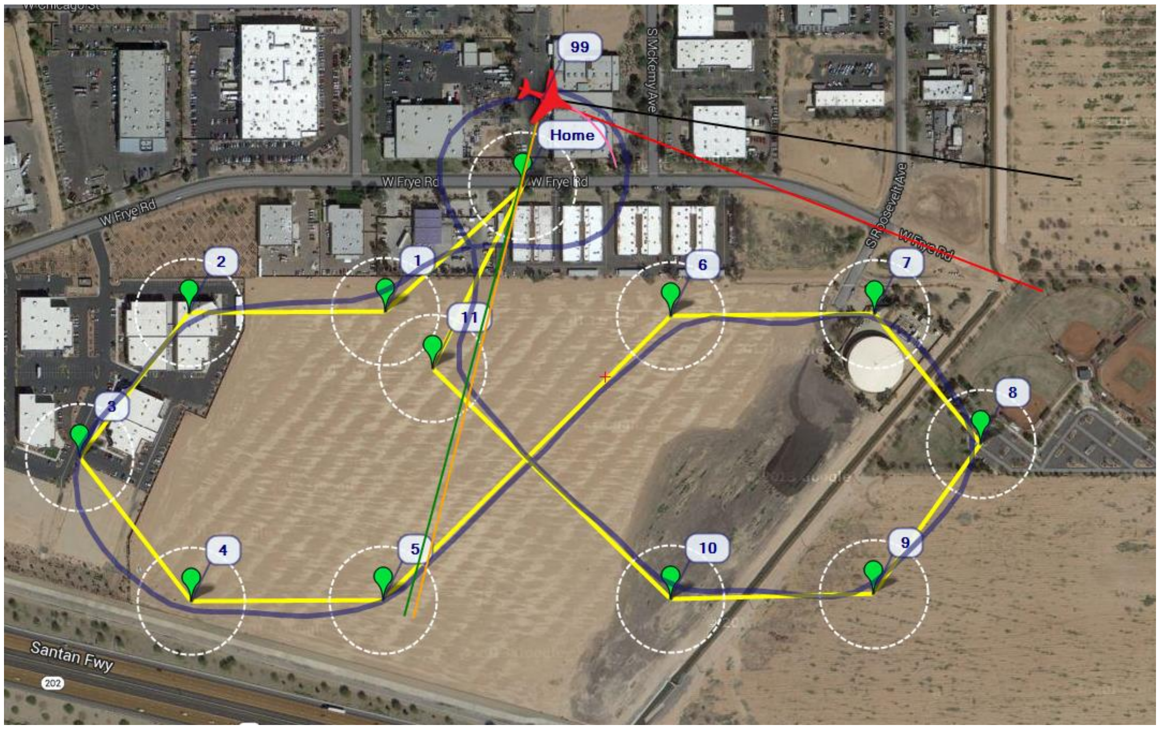


Figure 23, Wind speed 20 m/s

The maximum drift can be observed with 20 m/s wind from north to south. Wind velocity of 20 m/s being maximum velocity conditions for Skywalker. The drift is more between waypoint 3 and 4 as compare to 4 and 5. It is because Ardupilot doesn’t predict the wind direction and velocity in advance, before reaching to waypoint 3. This causes the UAV to drift more but after waypoint 4, Ardupilot predicts approximately the wind direction which helps to obtain good attitude correction i.e. turning the nose towards wind (into the wind direction) with appropriate crab angle. However this stabilization time won’t be enough when the UAV is taking a U turn with turning radius close minimum turning radius, in this case it need better wind estimation in advance.

5) The next result shows the effects with:

Wind Direction: 90°, from east to west

Wind Speed: 15 m/s

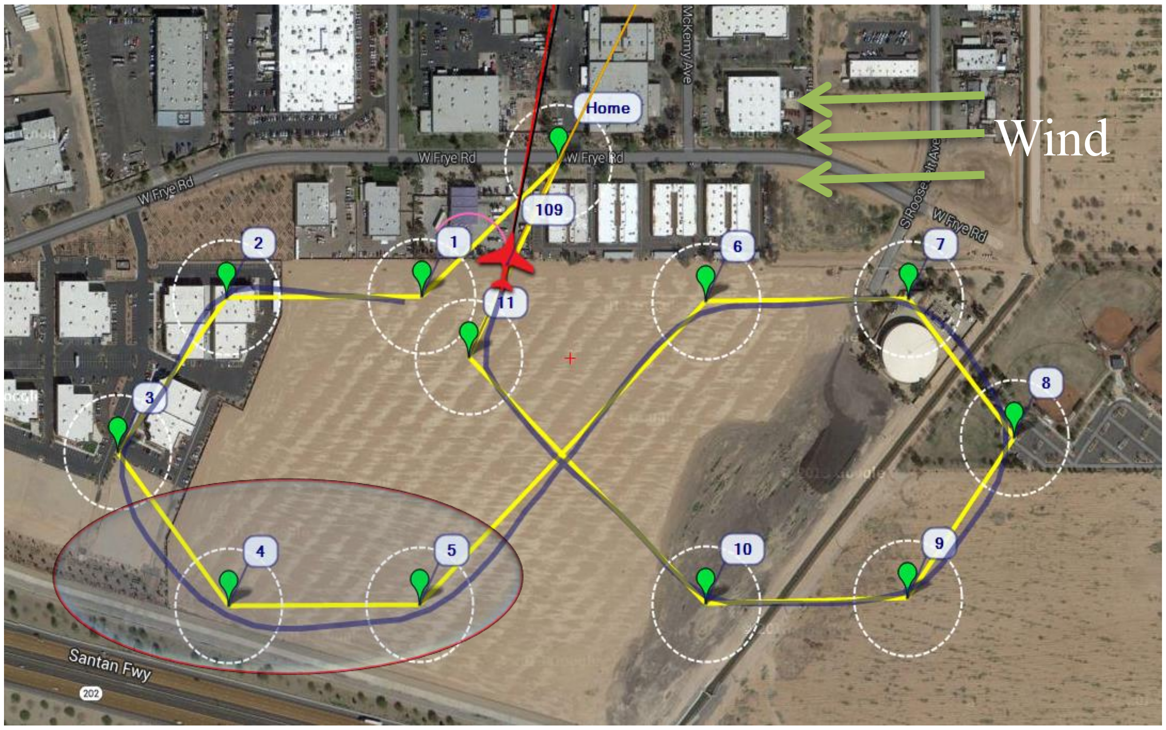


Figure 24, Wind speed 15 m/s from 90°

The drift can be seen in the oval, but there isn't any drift inwards while the UAV is flying from waypoint 7 to 8. This can be explained with UAV flying from waypoint 6 towards 7 it flies with heading into the wind and it doesn't predict wind in advance and enter the right turn while passing through waypoint 7, this is also responsible for overshooting the path between waypoint 7 and 8.

- **Working of wind estimation algorithm:**

  The whole wind estimation method in ArduPlane code is based on William Premerlani's Algorithm. This algorithm works on geometric transformation between Air Speed and Ground Speed. The figure (25) [free online private pilot], gives a detail about the airspeed vector (V) and groundspeed vector (S) used for wind vector prediction (W). [William Premerlani, 2009]

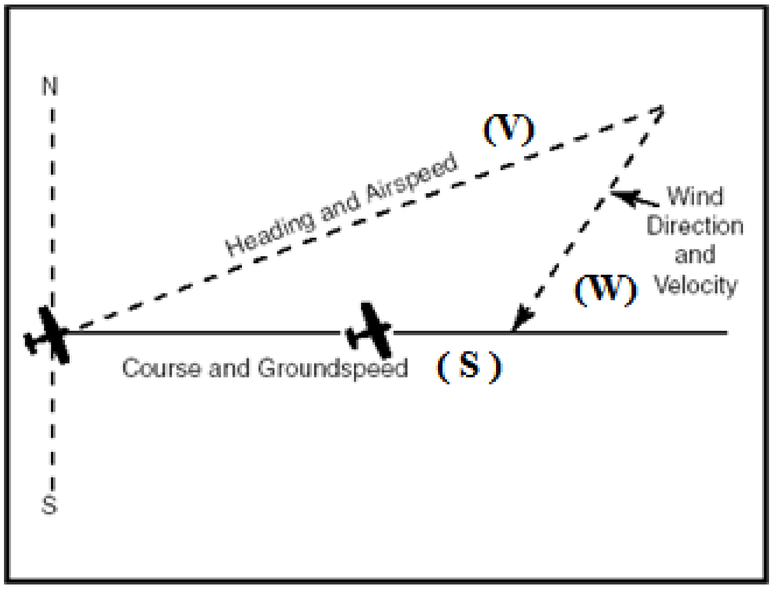


Figure 25, Wind triangle method

$$S = V + W \tag{1}$$

$$S = \begin{bmatrix} S_x \\ S_y \\ S_z \end{bmatrix} = speed\ over\ ground\ vector\ in\ earth\ frame$$

$$V = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = air\ speed\ vector\ in\ earth\ frame$$

$$W = \begin{bmatrix} W_x \\ W_y \\ W_z \end{bmatrix} = wind\ speed\ vector\ in\ earth\ frame$$

$$S_2 - S_1 = V_2 - V_1 \qquad (2)$$

$$W = \frac{(S_2 - S_1) - (V_2 + V_1)}{2} \qquad (3)$$

$$\boldsymbol{V} \approx V. \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot F \qquad (4)$$

V = magnitude of the airspeed

θ= residual yaw error in the direction cosine matrix

F=column of the DCM that represents the fuselage

From equation 4 and 2

$$S_2 - S_1 \approx V. \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} . (F_2 - F_1) \qquad (5)$$

Equation 5 leads to the following equation for computing the airspeed:

$$V \approx \frac{|S_2 - S_1|}{|F_2 - F_1|} \qquad (6)$$

Equation 6 is used in to compute airspeed in case there isn't airspeed sensor mounted onboard. But a vital step in understanding how the transformation between AS and GS has been worked out to predict Wind direction and speed as shown in further part.

$$s_{21} = \frac{S_2 - S_1}{|S_2 - S_1|}$$

$$f_{21} = \frac{F_2 - F_1}{|F_2 - F_1|} \qquad (7)$$

$$\begin{bmatrix} s_{21x} \\ s_{21y} \\ s_{21z} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} f_{21x} \\ f_{21y} \\ f_{21z} \end{bmatrix} \qquad (8)$$

$$\begin{bmatrix} s_{21x} \\ s_{21y} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} . \begin{bmatrix} f_{21x} \\ f_{21y} \end{bmatrix} \qquad (9)$$

$$W_x = \frac{S_{1x} + S_{2x} - V.(\cos(\theta).(F_{1x} + F_{2x}) - \sin(\theta) \cdot (F_{1y} + F_{2y}))}{2}$$

$$W_y = \frac{S_{1y} + S_{2y} - V.(\sin(\theta).(F_{1x} + F_{2x}) + \cos(\theta) \cdot (F_{1y} + F_{2y}))}{2}$$

$$W_z = \frac{S_{1z} + S_{2z} - V.(F_{1z} + F_{2z})}{2}$$

The above 3 equations explains the 3 components of wind vector estimated by ArduPlane.

Note: The direction cosine matrices are additional data added in wind prediction algorithm to take into account the changes in magnetic heading due to yaw rates in

UAVs. The graphical explanation of Direction Cosine Matrix implemented is shown in appendix II.

- **Error with the prediction based algorithm**

  As shown above the wind estimation algorithm used by ArduPlane code is mainly a prediction based algorithm instead of direct wind measurements. The absence of direct measurement techniques leads to inaccurate wind speed and direction calculation. This is very important in maneuvering missions. Due to use of prediction based algorithm there are cross track errors as shown in simulation results earlier. The simulations are performed with constant wind speed and direction. The effects of varying wind conditions can be seen well in actual field testing data shown in Chapter 5.

5. FIELD TESTING

The flight testing was done with Skywalker UAV and AIRMAR sensor onboard. The following results will show the difference between true wind speed with respect to ground and the wind speed calculated by APM 2.5.

Table (4) is a reference table for details about flight tests performed on the flying field.

Table 4,

Flight tests details

Test No.	Flight Starting Time HH:MM:SS	Total Flight Time (MM:SS)	Average Wind Conditions Speed (m/s) & Direction	Waypoint Mission
Flight 1	17:27:10	7:30	16 m/s from 225°	Rectangular
Flight 2	18:10:00	10:10	16 m/s from 225°	Lawn Mower
Flight 3	9:33:45	11:25	10 m/s from 52°	Rectangular & Figure of 8
Flight 4	9:59:25	6:5	12 m/s from 325°	Lawn Mower

Figure 25, shows the path followed by Skywalker in Flight 1 (Table 4) with 4 waypoints, the maximum cross track error can be seen between waypoint 3 and 4.

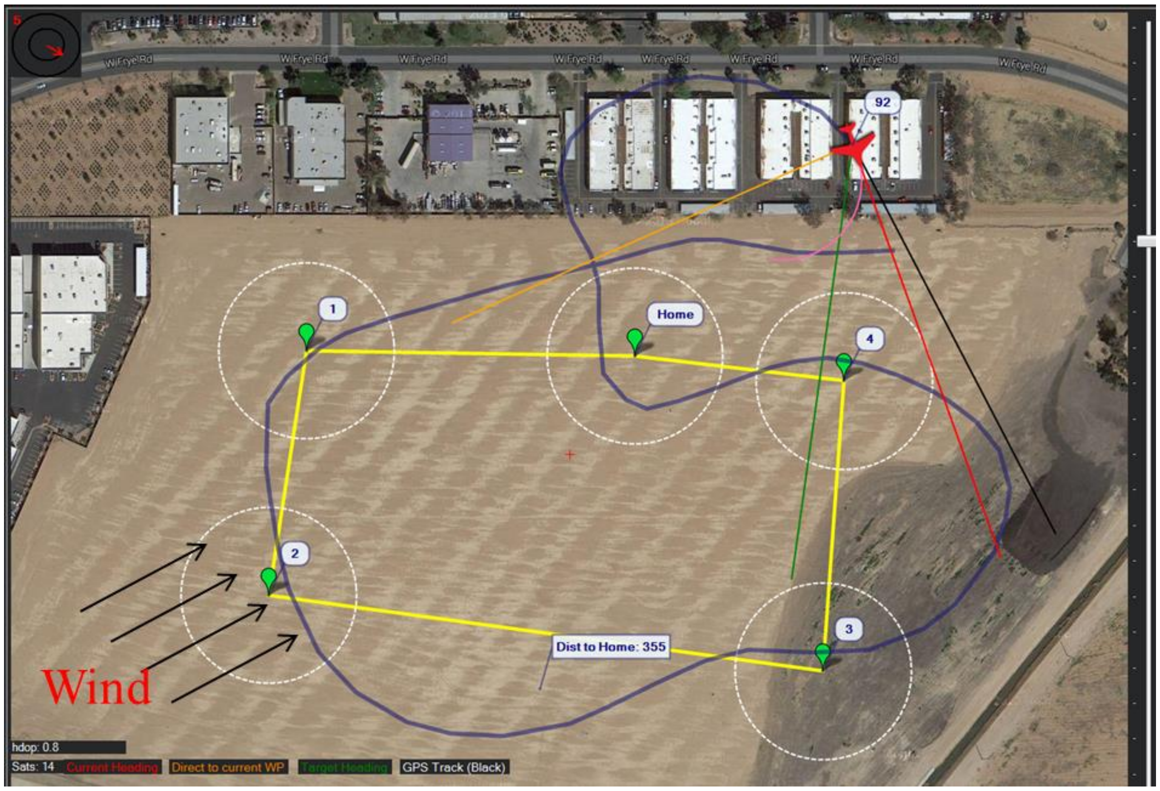


Figure 26, Flight with 4 waypoints with simple rectangular path mission.

Wind Speed 16 m/s, the average in Chandler, Arizona USA is considered to be lot higher than average of 5 m/s. This can be considered as high and gusty windy conditions.

- Software in the loop simulation:

The software in the loop simulation was performed with the same waypoints, wind conditions were similar as actual field tests and the result is shown in Figure (20). The cross track error is lesser than the actual flight test (figure 19). The reason behind that is wind conditions in simulation were constant throughout the flight, whereas in actual flight test the wind conditions don’t remain constant. The wind prediction stabilizes after sometime and remains almost constant. That’s why; cross track error in simulation is less than cross track error in actual flight.

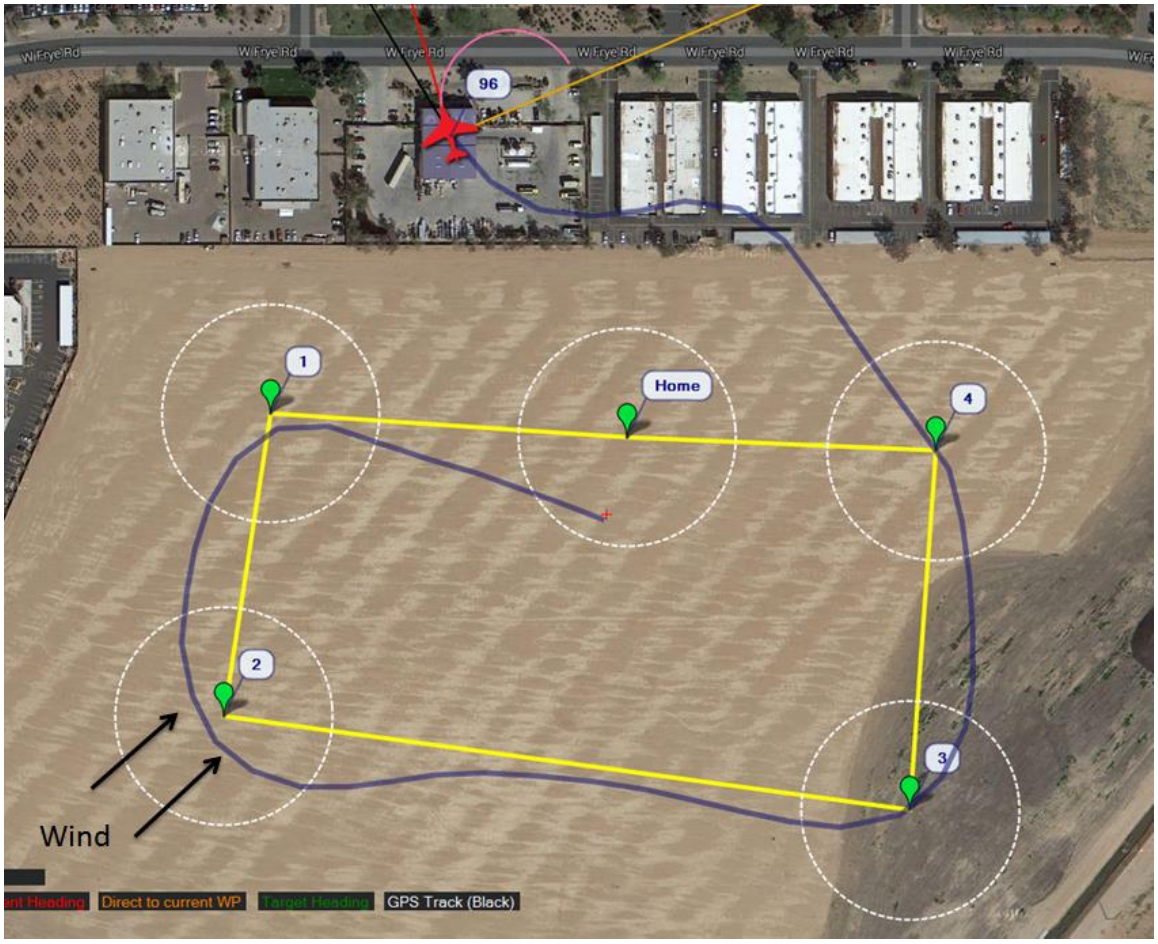


Figure 27, Software in loop simulation of Flight 1.

Field testing data from sensors:

1. **Wind direction data:**

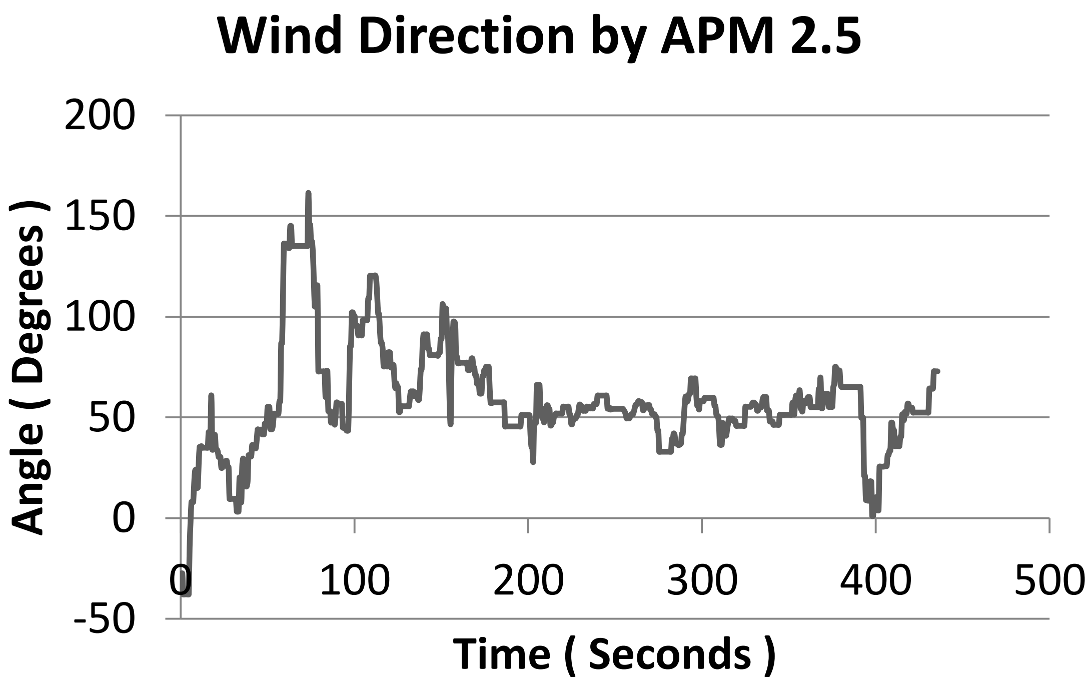


Figure 28, Wind direction predicted by APM 2.5

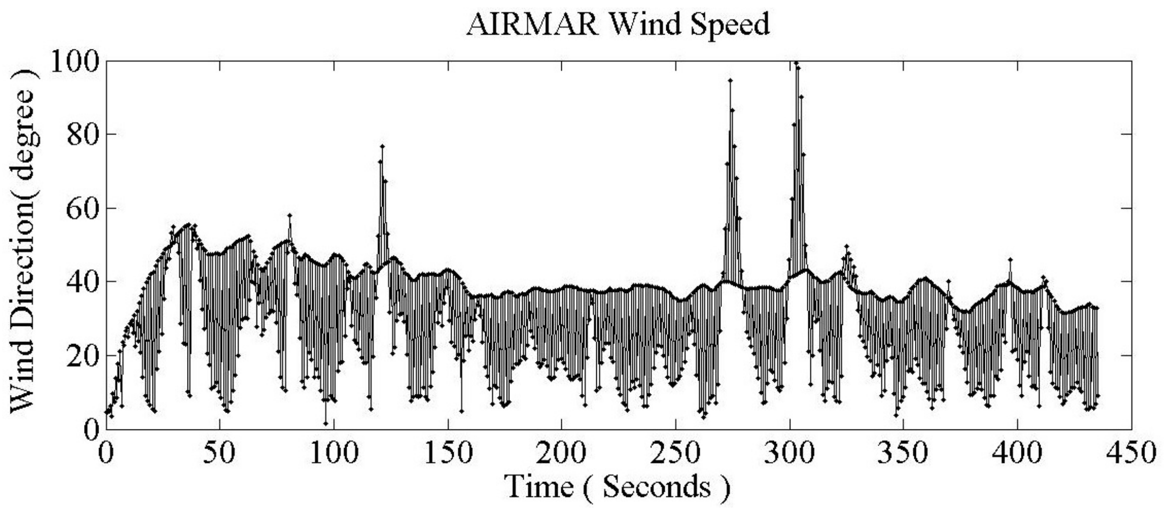


Figure 29, Wind direction by AIRMAR Wind Sensor

Graph Comparison:

Comparing figure 28, showing wind direction predicted by APM 2.5 with figure 29, showing wind direction by AIRMAR Wind Sensor (true wind direction) with the local values of angles (shows wind direction), it can be clearly seen the error in wind prediction by APM 2.5. This error leads to wrong crab angle calculation for attitude correction. This overall results into cross track errors, as the correct attitude correction was not applied.

2. **Wind Speed data:**

    Referring to the graphs shown on next page, figure (30) shows the graph of wind speed by predicted by APM 2.5 with **average of 4 m/s**. Figure (31), shows true wind speed by AIRMAR Wind Sensor with average of **10 m/s**. comparing local values at different time instances shown in figure (30) and figure (31), it can be clearly seen the inaccuracy of wind speed prediction by APM 2.5 and the actual wind speed. This will result in less power supplied to main brushless motor propelling the UAV, which will make it fly slower or not able to fly in windy conditions.

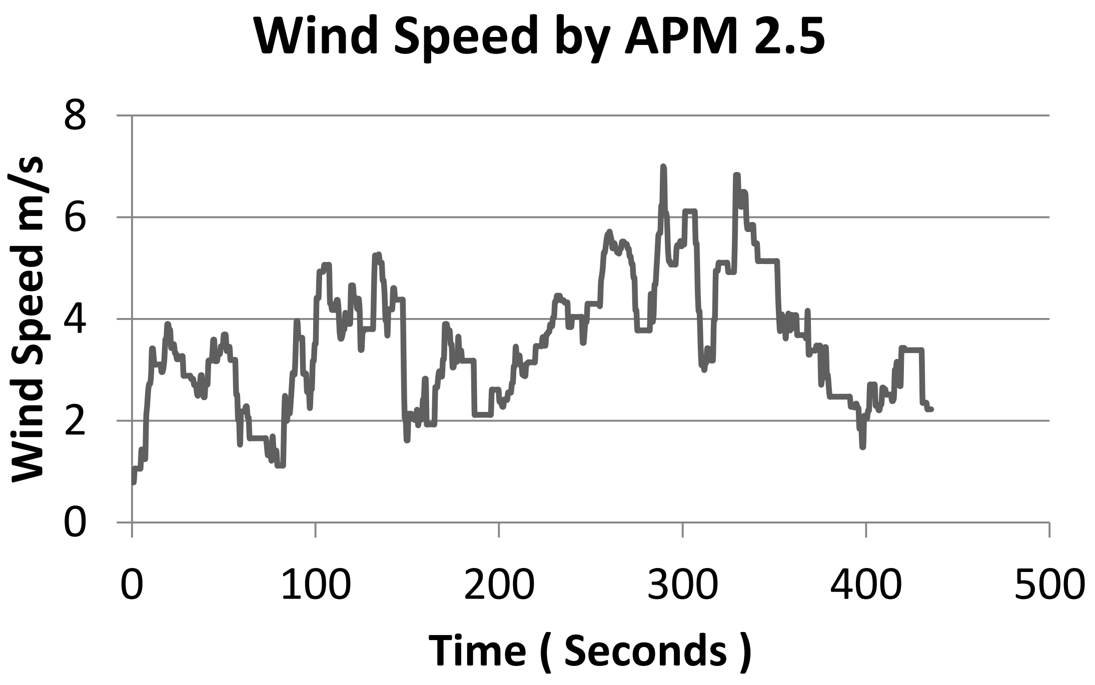


Figure 30, Wind Speed predicted by APM 2.5

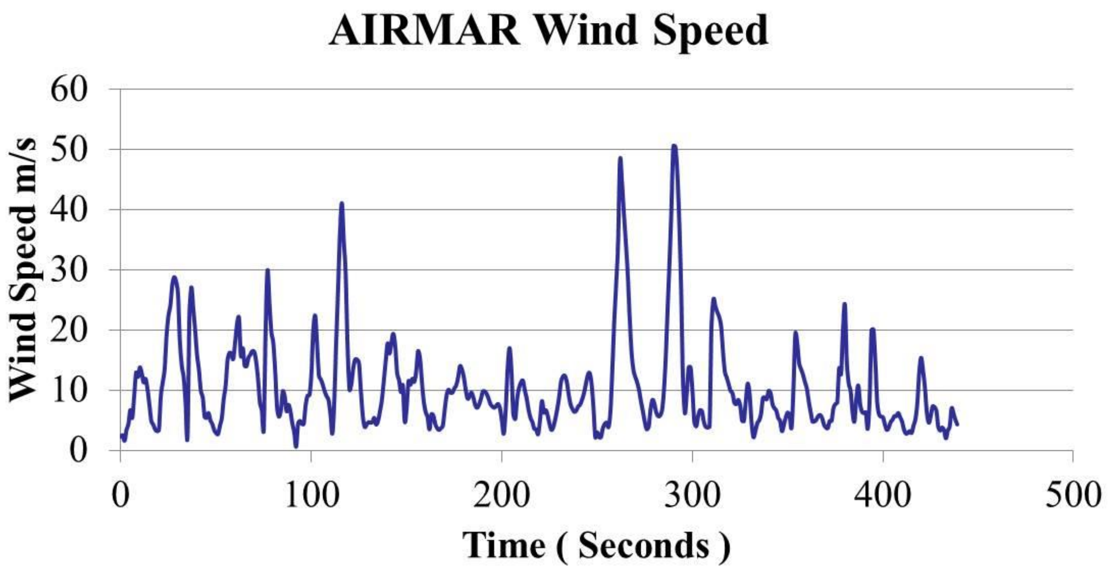


Figure 31, Wind speed by AIRMAR

- **Actual flight test with lawn mower pattern**

The actual test of the wind prediction algorithm was performed by flying UAV with through lawn mower pattern mission. Flight 2 from Table (4) explains the flight details. The lawn mower pattern involved turns, (referring turn between waypoint 2 and 3) with more than minimum radius.

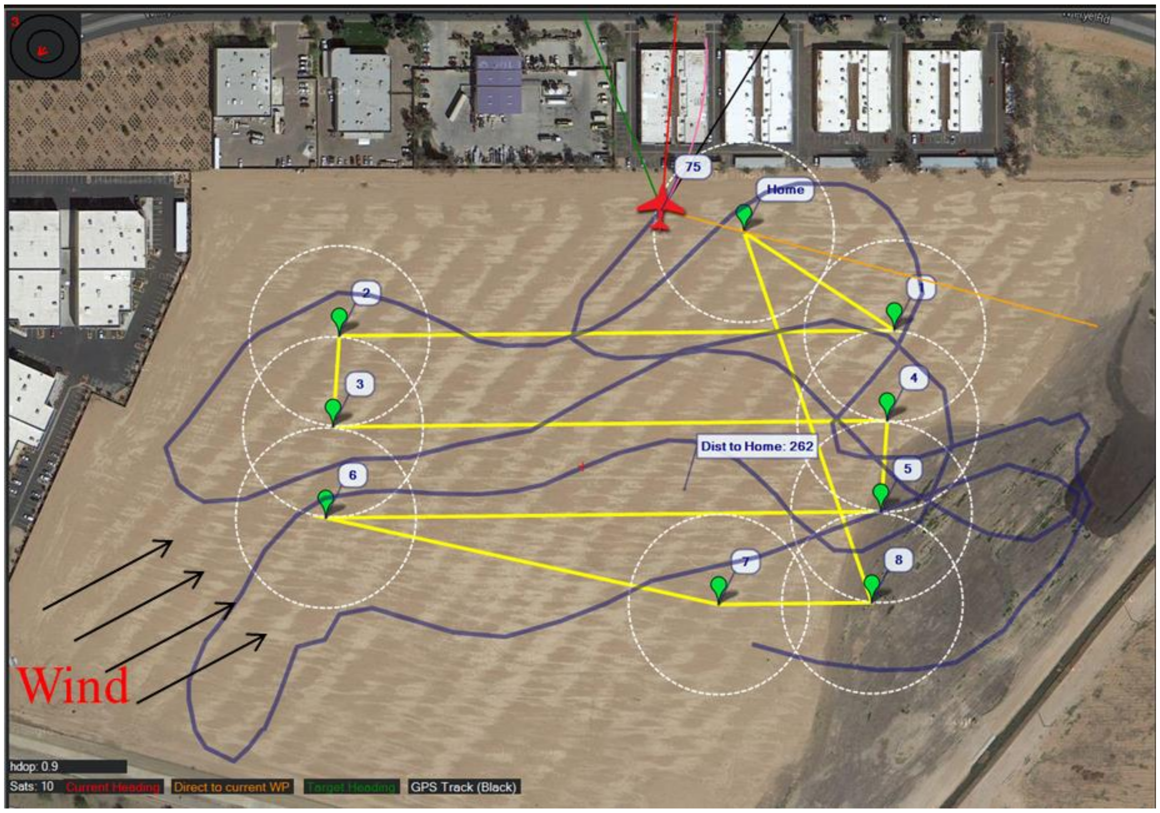


Figure 32, Lawn mower pattern

Figure (32) shows the blue track way off from the preplanned path (shown by yellow track). The maximum cross track error can be noted between waypoint 6 and 7. This is very bad scenario in lawn mower pattern following. The UAV spent nearly 50% more distance than required planned path. This leads to UAV spending more mission time and more battery power. This is because the way Ardupilot calculates wind with the geometrical transformation based prediction method, discussed in Chapter 4.

- Software in the loop simulation:

Figure (33) shows software in the loop simulation of same lawn mower pattern mission with simulating the flight conditions mentioned in flight 2 (Table 4), the drift in actual flight path can be seen with the maximum drift between waypoint 5 and 6.

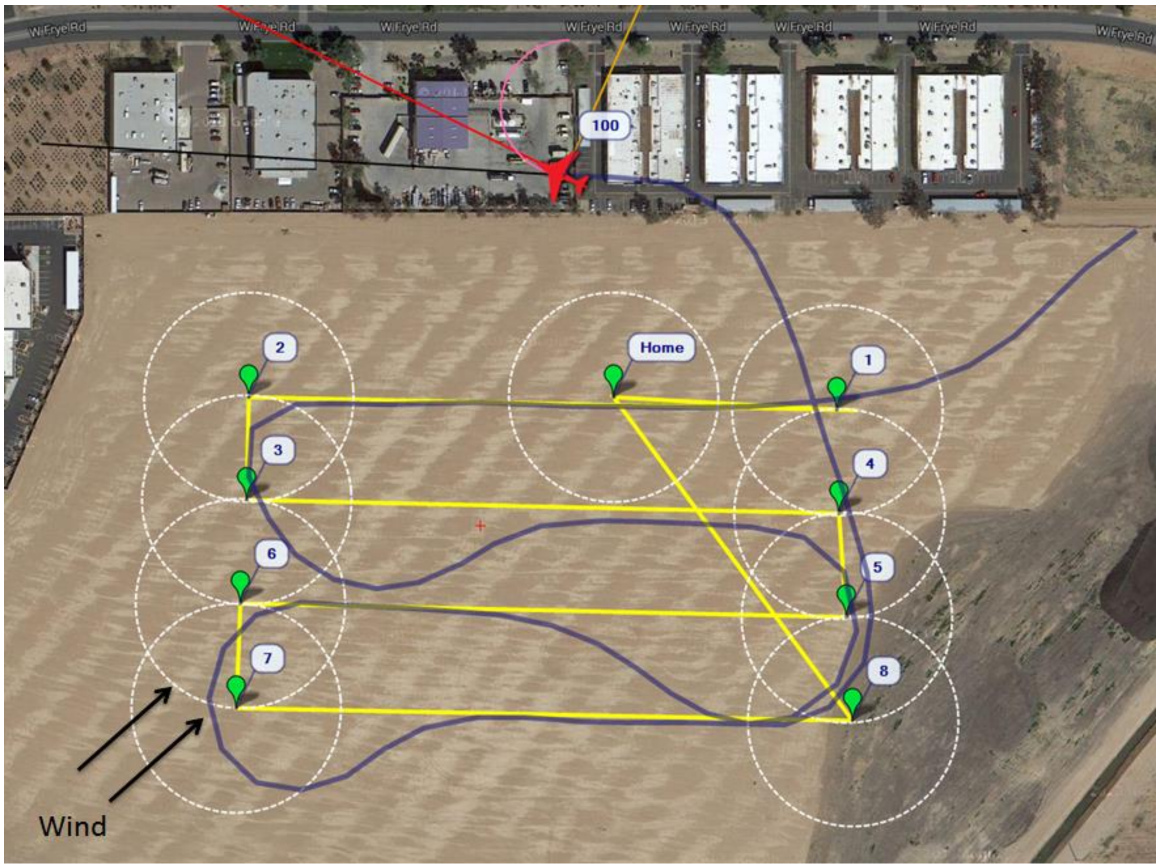


Figure 33, SITL Simulations of Lawn mower pattern

- **Additional flight tests:**

Some more flights tests were performed to test the UAV in different conditions.

1. Rectangular Pattern: Figure (34) shows snapshot of a mission data with Flight 3 (table 4). The maximum cross track error can be noted between waypoint 2 and 3. The wind direction is opposite as compared to Flight 1 shown in Figure (27).

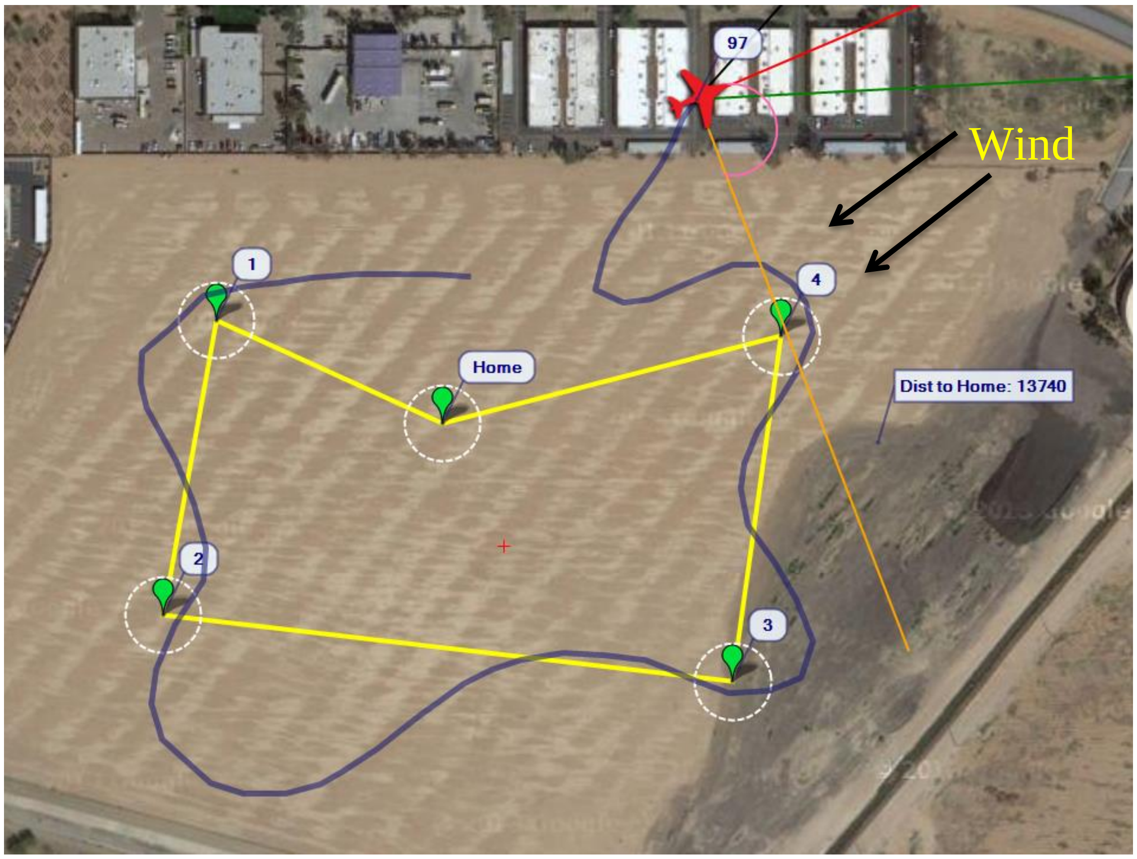


Figure 34, Rectangular pattern mission with Flight 3 conditions

2. ‘Figure of 8 Pattern’: This pattern is used in mostly all software in loop simulations (figure 35). This pattern was tested with same flight 3 conditions (table 4). Average 15 meters of cross track error was noted with maximum being 31 meters between waypoint 6 and 7. The cross track error is less than cross track error in rectangular pattern shown in figure (34). This is mainly due to turning angle is around 120° in ‘figure of 8’ loop as compared to turning angle of around 90° in rectangular. The rectangular pattern involve more sharper turns which require better wind estimation to reduce cross track error. The earlier graph comparison of wind speed and direction data showed the inaccuracy on ArduPlane code.

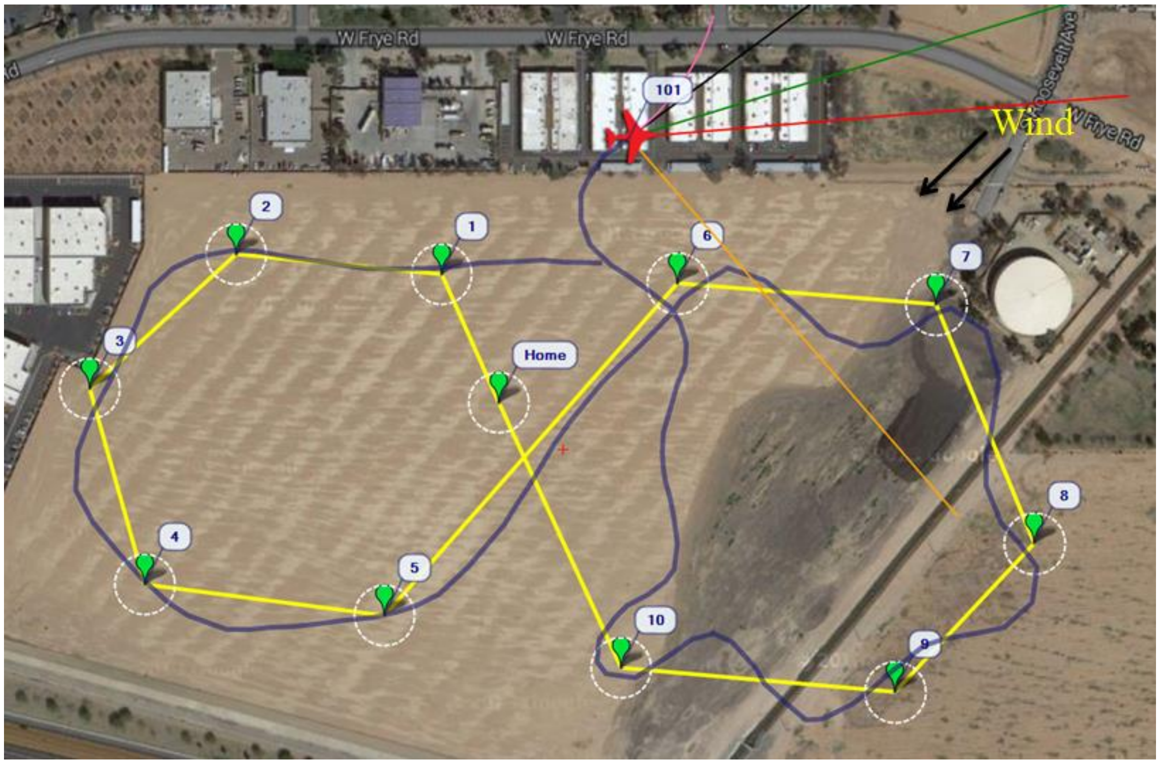


Figure 35, ‘Figure of 8’ waypoint pattern mission

3. Lawn Mower Pattern: The lawn mower pattern showed in figure (36) shows the flight track under different wind condition than previous lawn mower pattern mission (figure 36). It can be seen from the flight track the drift due to wind after waypoint 2. The drift increased more and more due to inaccurate wind predictions before reaching back to track between 3 and 4.

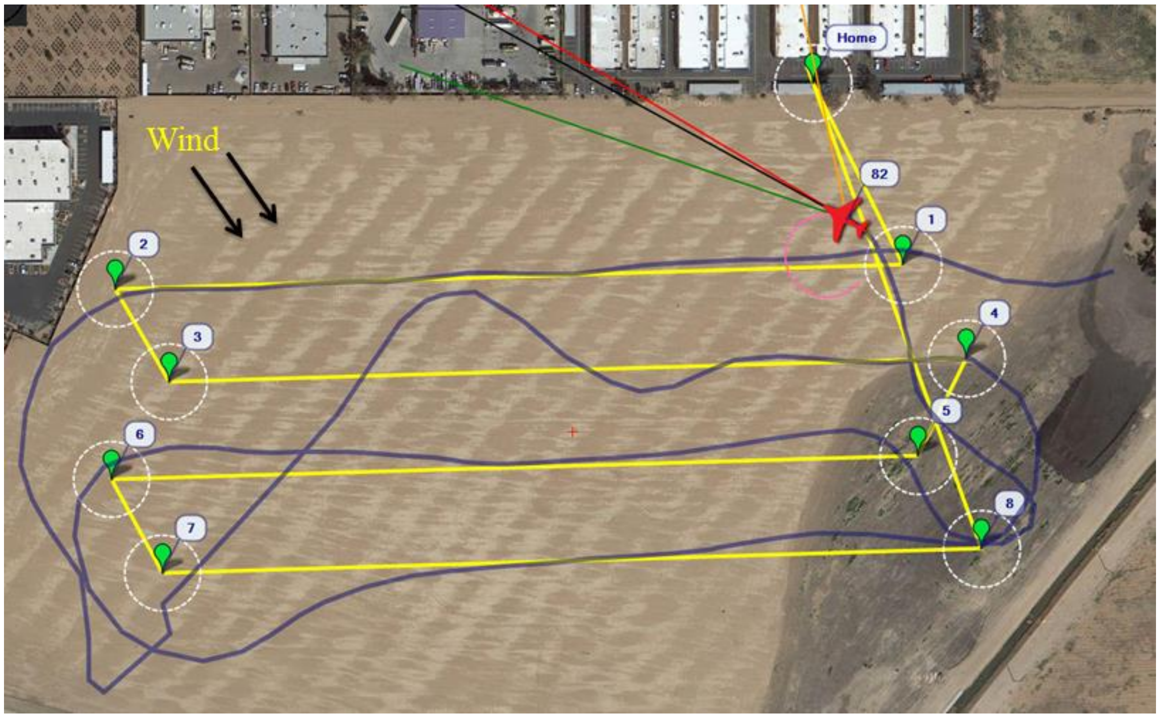


Figure 36, Lawn mower pattern mission with different wind condition

- **Summarizing cross track error and wind speed**:

The following graph shows the maximum cross track error in each of missions discussed earlier with wind velocity. The graph explains the variation in cross track error depending on different missions and wind conditions. The maximum cross track errors can be seen in lawn mower pattern missions having more turns.

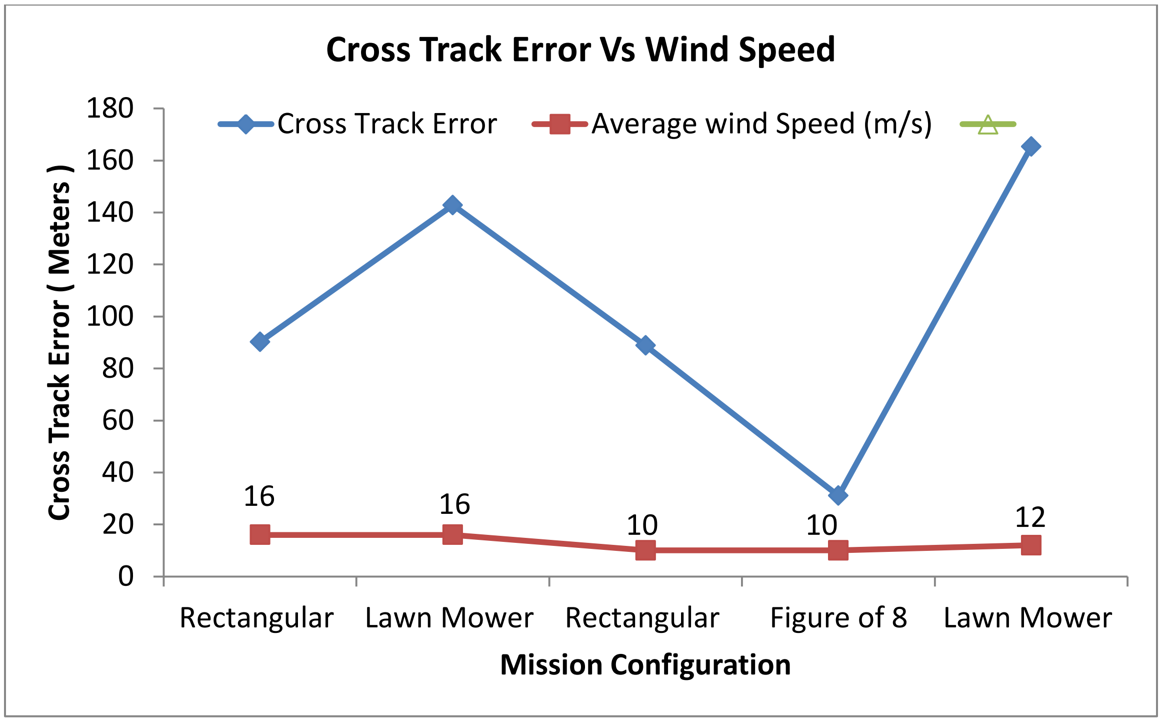


Figure 37, Cross Track Error vs. Wind Speed

6. CONCLUSIONS

The current method of wind estimation and prediction doesn’t predict true wind speed and direction with respect to earth frame. The predicted wind speed and direction are more than 60% off from the actual wind condition. The inaccuracy in real time updating of wind data leads to error in attitude correction applied by autopilot board. The effects of this method are shown with Software in The Loop simulations (with the constant wind conditions) and actual field testing.

The field testing confirms the difference between wind speed and direction predicted by autopilot board (APM 2.5) and the true wind speed, direction (actual field conditions) by AIRMAR wind sensor. This difference is significantly large which leads to cross track error. In some cases the maximum cross track error occurred while flying between two waypoints is larger than the distance between them. The larger cross track error, increases mission completion time with more power consumption than estimated.

The current method of wind estimation and prediction is not accurate enough to perform minimum radius turns for waypoints navigation in highly maneuvering missions such as lawn mower patterns.

Future Work

Mapping missions requires capturing images at predefined waypoints with many waypoint and turns. The UAV needs to fly with different heading depending on the turns they are flying through. Considering one of the following cases which involve high wind conditions and maneuvering. One of the difficult parts in the planned path is achieving waypoint 2, as shown wind direction would be 45° to path between waypoint 1 and 2. The UAV would drift from the waypoint 2 while on the way to waypoint 3. To overcome this problem following solution can be implemented.

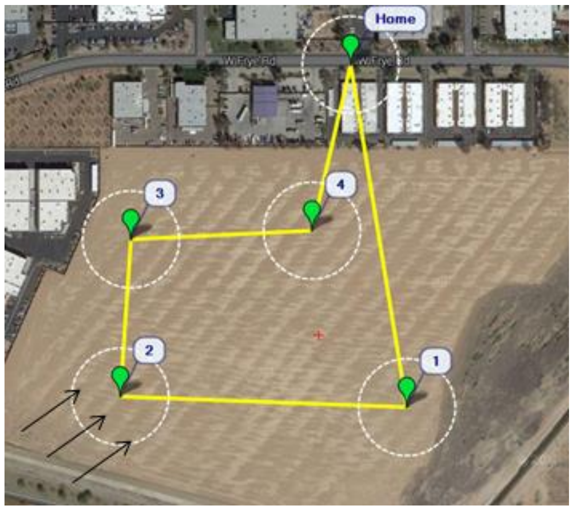


Figure 38, Future development in algorithm

- Designing an algorithm which will use the real time wind data and will increase the waypoint radius real-time, so the UAV can still achieve the waypoint.
- But the radius cannot be increased more than maximum allowable tolerance so that it will affect the frontal and sideways overlap of image to be captured by camera while passing through the

waypoint. The next part of algorithm would be, if the UAV drifts more than maximum limit of the radius then it should skip the waypoint and going towards the next one. It should come back later while flying into the wind to achieve missed waypoint.

References

1. John Osborne and Rolf Rysdyky “Waypoint Guidance for Small UAVs in Wind” Journal of Guidance, Control and Dynamics, July-August 2011.

2. Brent K. Robinson a thesis report on “an investigation into robust wind correction algorithms for off-the-shelf unmanned aerial vehicle autopilots” Air Force Institute of Technology, Ohio.

3. DIY Drones, UAV development community.

4. William Premerlani “IMU Wind estimation theory” 12/12/2009.

5. Jan Petrich, Kamesh Subbarao “On-Board Wind Speed Estimation for UAVs” Nextgen Aeronautics and the University of Texas Arlington.

6. Am Cho, Jihoon Kim, Sanghyo Lee, Changdon Kee “Wind Estimation and Airspeed Calibration using a UAV with a single –Antenna GPS Receiver and Pitot Tube” IEEE Transactions on Aerospace and Electronic System, Vol 47 No 1, January 2011.

7. Adrian Stanley “Flight Path Deconfliction of Autonomous UAVs” QuinetiQ, Thurleigh, Bedfordshire, UK.

8. Duncan Miller “Autonomous Vehicle Laboratory for Sense and Avoid Research and Hardware-in-the-Loop” University of Michigan, Ann Arbor.

9. Hollister, W. M., Bradford, E. R., and Welch, J. D. “ Using aircraft radar tracks to estimate winds aloft ” MIT Lincoln Laboratory Journal, 2 (1989), 555—565.

10. Khelif, D., Burns, S. P., and Friehe, C. A. “Improved wind measurements on research aircraft” Journal of Atmospheric and Oceanic Technology, (1998)

11. Meir Pachter, Nicola Ceccarelli and Philip R. Chandler “Estimating MAV’s Heading and the Wind Speed and Direction Using GPS, Inertial, and Air Speed Measurements” Air Force Institute of Technology Wright-Patterson Air Force Base, OH 45433.

12. GitHub for Ardupilot “ https://github.com/diydrones/ardupilot ”

13. Jack W. Langelaan, Nicholas Alley and James Neidhofer “Wind Field Estimation for Small Unmanned Aerial Vehicles”, Journal of Guidance, Control and Dynamics, July-August 2011.

14. N. Ceccarelli, J.J. Enright, E. Frazzoli, S.J. Rasmussen and C.J. Schumacher, "Micro UAV Path Planning for Reconnaissance in Wind", American Control Conference.

15. Lawrance, N.R.J, Sukkarieh, S. "A guidance and control strategy for dynamic soaring with a gliding UAV" Robotics and Automation, 2009. ICRA.

16. Mishra, P. and Enge, P. "Gloabal Positioning System: Signals, Measurements and Performance, 2nd Ed", Lincoln, MA, 2006.

17. Rysdyk, "Unmanned Aerial Vehicle Path Following for Target Observation in Wind", Journal of Guidance, Control and Dynamics, September-October 2006.

18. Jonathan How, Ellis King, Yoshiaki Kuwata, "Flight Demonstrations of Cooperative Control", Unmanned Unlimited, September 2004, Chicago, Illinois.

19. K.P.A. Lievins, J. A. Mulder and P. Chu, "Single GPS antenna attitude determination of a fixed wind aircraft aided with aircraft aerodynamics", AIAA Guidance, Navigation and Control Conference and Exhibit, August 2005, San Francisco, California.

APPENDIX I

FLIGHT SCRIPT

The following python code explains the flight script used to perform Software in the loop simulations. The script contains home location, wind parameter input and the way to import preplanned mission.

```
# fly ArduPlane in SIL
import util, pexpect, sys, time, math, shutil, os
from common import *
from pymavlink import mavutil
import  random

# get location of scripts
testdir=os.path.dirname(os.path.realpath(__file__))

HOME_LOCATION='33.298825,-111.954793,354,180'  # Setting up home GPS Co-
ordinates
WIND="30,90,0.2" # speed,direction,variance  # Specifying wind directions
homeloc = None
```

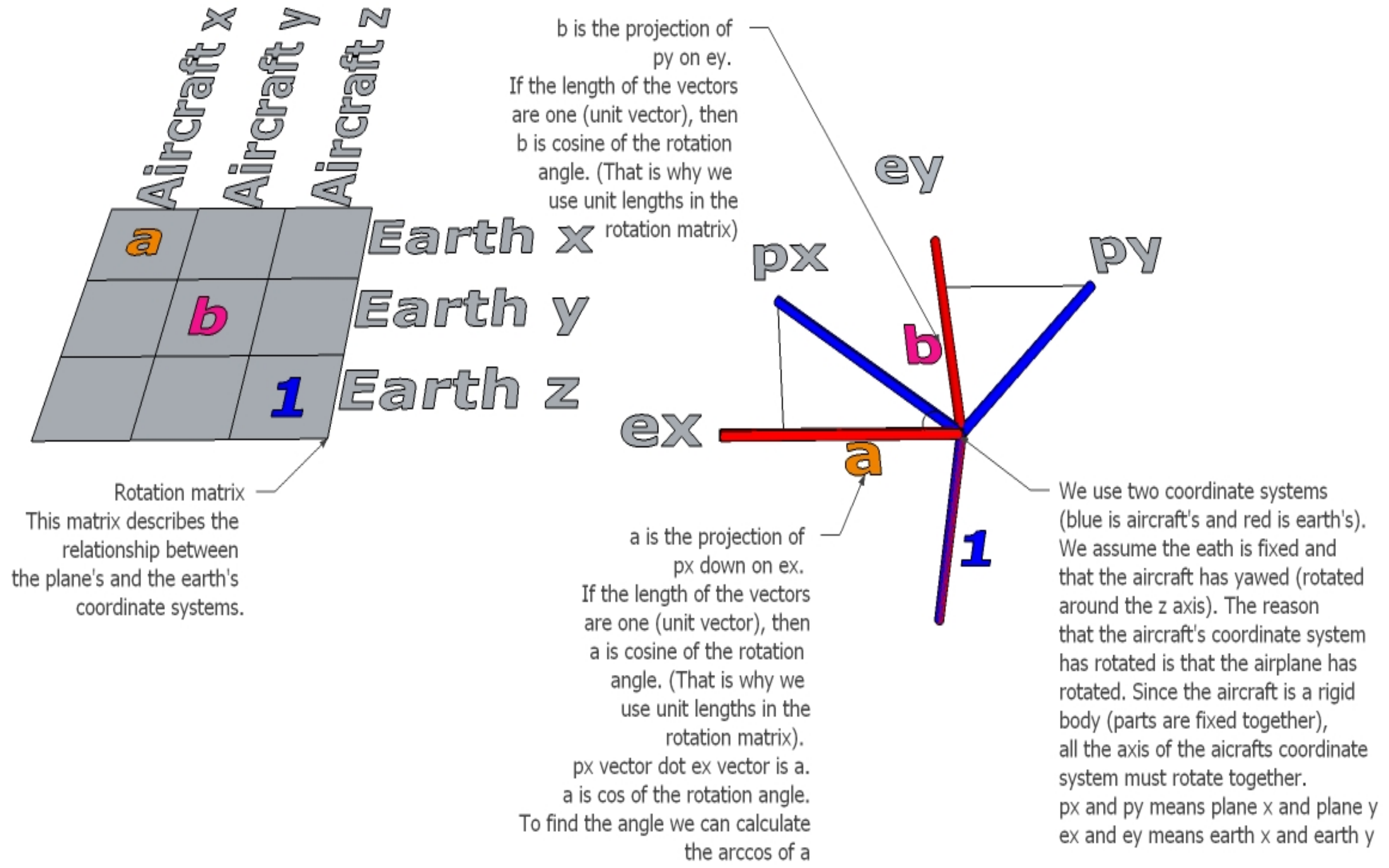
## The following first 3 blocks of code specifies the Takeoff, Fly in circuit, then Return to Launch to start Auto mode This allows UAV to start a waypoint mission

```
def takeoff(mavproxy, mav):
   '''takeoff get to 30m altitude'''
   mavproxy.send('switch 4\n')
   wait_mode(mav, 'FBWA')

   # gain a bit of altitude
   if not wait_altitude(mav, homeloc.alt+50, homeloc.alt+80, timeout=30):    # Changed
from 150 and 180
       return False

   # level off
   mavproxy.send('rc 2 1500\n')

   print("TAKEOFF COMPLETE")
   return True


def fly_left_circuit(mavproxy, mav):
   '''fly a left circuit, 200m on a side'''
   mavproxy.send('switch 4\n')
   wait_mode(mav, 'FBWA')
   mavproxy.send('rc 3 1600\n')              # changed fro 2000
   '''if not wait_level_flight(mavproxy, mav):
        return False'''

    print("Flying left circuit")
    # do 4 turns
    for i in range(0,4):
        # hard left
        print("Starting turn %u" % i)
        mavproxy.send('rc 1 1000\n')
        if not wait_heading(mav, 270 - (90*i), accuracy=10):
            return False
        mavproxy.send('rc 1 1500\n')
        print("Starting leg %u" % i)
        if not wait_distance(mav, 100, accuracy=20):
            return False
    print("Circuit complete")
    return True

def fly_RTL(mavproxy, mav):
    '''fly to home'''
    print("Flying home in RTL")
    mavproxy.send('switch 2\n')
    wait_mode(mav, 'RTL')
    if not wait_location(mav, homeloc, accuracy=120,
                         target_altitude=homeloc.alt+80, height_accuracy=20,
                         timeout=180):
        return False
    print("RTL Complete")
    return True

def setup_rc(mavproxy):
    '''setup RC override control'''
    for chan in [1,2,4,5,6,7]:
        mavproxy.send('rc %u 1500\n' % chan)
    mavproxy.send('rc 3 1000\n')
    mavproxy.send('rc 8 1800\n')

# The mission script is loaded here

def fly_mission(mavproxy, mav, filename, height_accuracy=-1, target_altitude=None):
    '''fly a mission from a file'''
    global homeloc
    print("Flying mission %s" % filename)
    mavproxy.send('wp load %s\n' % filename)
    mavproxy.expect('flight plan received')
    mavproxy.send('wp list\n')
    mavproxy.expect('Requesting [0-9]+ waypoints')
    mavproxy.send('switch 1\n') # auto mode
    wait_mode(mav, 'AUTO')
    if not wait_waypoint(mav, 1, 7, max_dist=60):
        return False
    if not wait_groundspeed(mav, 0, 0.5, timeout=60):
        return False
    print("Mission OK")
    return True
```

```
def fly_ArduPlane(viewerip=None, map=False):
    '''fly ArduPlane in SIL

    you can pass viewerip as an IP address to optionally send fg and
    mavproxy packets too for local viewing of the flight in real time
    '''
    global homeloc

    options = '--sitl=127.0.0.1:5501 --out=127.0.0.1:19550 --streamrate=10'
    print "Getting ready to send"
    if viewerip:
        options += " --out=%s:14550" % viewerip
            print "Sending on %st port 14550" % viewerip
    if map:
        options += ' --map'

    sil = util.start_SIL('ArduPlane', wipe=True)
    mavproxy = util.start_MAVProxy_SIL('ArduPlane', options=options)
    mavproxy.expect('Received [0-9]+ parameters')

    # setup test parameters
    mavproxy.send("param load %s/ArduPlane.parm\n" % testdir)
    mavproxy.expect('Loaded [0-9]+ parameters')

    mavproxy.send("param fetch\n")

    # restart with new parms
    util.pexpect_close(mavproxy)
    util.pexpect_close(sil)

    cmd = util.reltopdir("Tools/autotest/jsbsim/runsim.py")
    cmd += " --home=%s --wind=%s" % (HOME_LOCATION, WIND)
    if viewerip:
        cmd += " --fgout=%s:5503" % viewerip

    runsim = pexpect.spawn(cmd, logfile=sys.stdout, timeout=10)
    runsim.delaybeforesend = 0
    util.pexpect_autoclose(runsim)
    runsim.expect('Simulator ready to fly')

    sil = util.start_SIL('ArduPlane')
    mavproxy = util.start_MAVProxy_SIL('ArduPlane', options=options)
    mavproxy.expect('Logging to (\S+)')
    logfile = mavproxy.match.group(1)
    print("LOGFILE %s" % logfile)

    buildlog = util.reltopdir("../buildlogs/ArduPlane-test.tlog")
    print("buildlog=%s" % buildlog)
    if os.path.exists(buildlog):
        os.unlink(buildlog)
    try:
        os.link(logfile, buildlog)
    except Exception:
        pass

    mavproxy.expect('Received [0-9]+ parameters')
    util.expect_setup_callback(mavproxy, expect_callback)

    expect_list_clear()
    expect_list_extend([runsim, sil, mavproxy])

    print("Started simulator")

    # get a mavlink connection going
    try:
        mav = mavutil.mavlink_connection('127.0.0.1:19550', robust_parsing=True)
    except Exception, msg:
        print("Failed to start mavlink connection on 127.0.0.1:19550" % msg)
        raise
    mav.message_hooks.append(message_hook)
    mav.idle_hooks.append(idle_hook)

    failed = False
    e = 'None'
    try:
        if not fly_mission(mavproxy, mav, os.path.join(testdir, "ap1.txt"),
        height_accuracy = 10,
                        target_altitude=homeloc.alt+100):
            print("Failed mission")
            failed = True
        if not log_download(mavproxy, mav, util.reltopdir("../buildlogs/ArduPlane-
            log.bin")):
            print("Failed log download")
            failed = True
    except pexpect.TIMEOUT, e:
        print("Failed with timeout")
        failed = True

    mav.close()
    util.pexpect_close(mavproxy)
    util.pexpect_close(sil)
    util.pexpect_close(runsim)

    if os.path.exists('ArduPlane-valgrind.log'):
        os.chmod('ArduPlane-valgrind.log', 0644)
        shutil.copy("ArduPlane-valgrind.log", util.reltopdir("../buildlogs/ArduPlane-
valgrind.log"))

    if failed:
        print("FAILED: %s" % e)
        return False
    return True
```

APENDIX II

GRAPHICAL ILLUSTRATION FOR DIRECTION COSINE MATRICES



Rotation matrix
This matrix describes the relationship between the plane's and the earth's coordinate systems.

b is the projection of py on ey.
If the length of the vectors are one (unit vector), then b is cosine of the rotation angle. (That is why we use unit lengths in the rotation matrix)

a is the projection of px down on ex.
If the length of the vectors are one (unit vector), then a is cosine of the rotation angle. (That is why we use unit lengths in the rotation matrix).
px vector dot ex vector is a.
a is cos of the rotation angle.
To find the angle we can calculate the arccos of a

We use two coordinate systems (blue is aircraft's and red is earth's).
We assume the eath is fixed and that the aircraft has yawed (rotated around the z axis). The reason that the aircraft's coordinate system has rotated is that the airplane has rotated. Since the aircraft is a rigid body (parts are fixed together),
all the axis of the aicrafts coordinate system must rotate together.
px and py means plane x and plane y
ex and ey means earth x and earth y

Figure 39, Graphical representation of how the Direction Cosine Matrices are calculated.