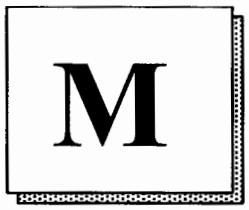
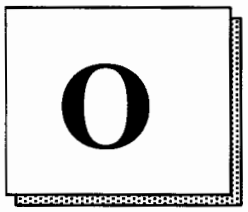
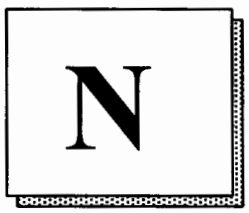
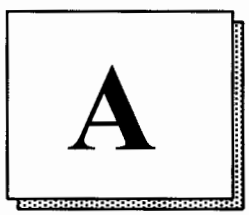
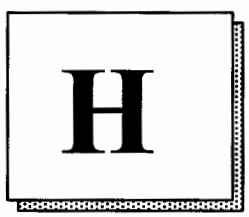










FOURTH YEAR ELECTRICAL AND COMPUTER SYSTEMS ENGINEERING
4th YEAR THESIS PROJECT ECE4912

UAV: Feature Based Navigation

By: Wang Yui Kong
ID number: 1810 2263

DEPARTMENT OF ELECTRICAL AND COMPUTER SYSTEMS ENGINEERING
MONASH UNIVERSITY, CLAYTON,
VICTORIA 3168, AUSTRALIA

Thesis Project

UAV: Feature Based Navigation

Thesis Report

By: Wang Yui Kong (1810 2263)

Acknowledgments

I wish to express my sincere thanks to my supervisor, Professor Gregory K. Egan, for his time, insight and fervour. I also wish to thank Mr. Terry Cornell for his insight and guidance and Mr. Ray Cooper for his help throughout this project.

Executive Summary

Most UAV (Unmanned Aerial Vehicles) today use GPS (Global Positioning System) as a navigation source. While GPS is an accurate and reliable method of navigation, it can be unavailable for various reasons. The aim of this project is to develop an alternative navigation method which can be used when GPS is unavailable.

The method suggested in this report is called Feature Based Navigation. It works by comparing two images, one taken by on board camera of the UAV and one extracted from an internal reference map based on current attribute of the UAV. The system then compares locations of identical features extracted from the images and hence calculating the location of UAV. This report outlines the basic theory and implementation of this method, as well as experimental results of it with real data taken from the data log of a UAV in flight. It explains why this method is viable and suggests improvement to the implementation of this system to improve its performance.

Table of Content

1 Introduction

UAV (Unmanned Aerial Vehicles) are basically unpiloted airborne vehicles that are designed to fly autonomously with little or no input from human. Tasks which can be performed by an UAV include search and rescue, unmanned reconnaissance and space exploration with missions such as exploration of Mar. Because it is unpiloted, UAV requires detail location of where it is as input into the navigation system to navigate.

Most UAV in service use GPS (Global Positioning System) as a primary mean of navigation. A GPS receiver relies on using information transmitted via radio frequency from at least three of the 24 satellites to calculate location of itself [1]. The problem with GPS is it can be unavailable for various reasons such as bad weather causing cloud cover thus disrupt reception of signals from satellites, or in time of conflict the source might be cut or jammed. The system is itself own by American, and until year 2000 only a less accurate version this system is available to general public [2], thus highlighting the sensitive natures of this system.

Aim of this project is to develop a system capable of providing periodical update of the location of a UAV flying at a reasonable altitude when GPS is unavailable. This system is to operate during mid-day in summer around 11am – 2pm, when shadows are not very noticeable, and in feature rich country side. This system is to operate on one of Monash University's UAV, and will have as its input, data from sensors of the UAV which will provide it with various attributes of the UAV, such as it altitude, pitch, roll, yaw and heading. This system will also be provided with images captured by on board camera.

A few alternatives navigation system for this UAV were researched; these alternatives are discussed in the "Background" section of this report. But due to restriction of equipments available on the UAV which this system intended to be operating on, a new method called Feature Based Navigation is proposed and tested.

Feature Based Navigation is based on comparing image captured by the UAV's onboard camera against an image of what the UAV's on board camera should be capturing as generated by the system and thus deriving the location of the UAV. The image of what the UAV's on board camera should be capturing is generated from a stored image of area which the UAV should be flying over (this image is refer to as the reference map in this document, in this project a large aerial photo generated by an aircraft for the area that it is flying over is used. Technically a reference map can also be generated by mosaicing photos taken by the UAV's onboard camera when it is flying over the area at a constant altitude, pitch and roll, but it is difficult for a UAV to fly so steadily for the duration of time required) and data including current orientation, altitude and the last known location of the UAV which is either where it loss connection with GPS or last result calculated by the system, and the optical properties and the mounting arrangement of the onboard camera. The system then attempt to extract and locate identical features from the images captured by the on board camera and that generated by the system. Then using the locations of these features on the images as well as attributes of the UAV; the system can attempt to aligned the features on the two images and thus deduce the location of the UAV.

One restriction of this system is that it requires the image taken by the onboard camera must consist entirely of the ground when it is operating. In another word if the pitch of the UAV is so large that the camera is facing either the sky

or the horizon, then the image taken can not be used by the system to calculate the location of the UAV. It also require the UAV to be flying above a certain altitude to be effective, otherwise the image captured by the UAV's onboard camera is unlikely to contain sufficient information for valid calculation. This system also must have a basic error checking system which is able to filter out results which are obviously wrong given the physical restriction imposed by specification of the UAV such as its maximum acceleration, velocity and turning ability. For example if the calculation shows a location which will require the UAV to travel at the speed of light to arrival there, then the system have to be able to identify and ignore the result.

This system is developed using Matlab's programming environment. Originally the system were to be tested with a pre-recorded sequence in a video file, which is taken by the UAV in flight, and the accompanied flight log which would give the status of the UAV every 200ms. But the optical properties of the camera when the video was taken are unknown, this means two of the crucial parameter; the view angles of the onboard camera are unknown and hence making it difficult to use that video sequence. Instead an artificial sequence of images is generated by using the same technique as extracting image from the reference map using the real longitude and latitude recorded from the flight log and a set of assumed view angles. This sequence of images should represent what the UAV onboard camera is capturing and was used to test this system. The main problem with this approach is the fact that longitude and latitude measurement was only measured in per second basis not every 200ms, so linear interpolation were used to obtain values in between to obtain the location of the UAV every 200ms.

2 Background

There are a few alternative navigation methods to GPS that are available. While they may not be exclusively design for UAV navigation, some of the concept used can be adapted and used by it. A few of such methods and why they are not suitable for this project are listed here, the reasons why feature based navigation is used is also stated.

Alternative Satellite Navigation System

There are other global satellite navigation networks which are design to perform similar task to GPS. These systems include the GLONASS (Global Navigation Satellite System) which was developed by former U.S.S.R and the Galileo project which is current being develop by European Space Agency (ESA). If multiple systems are implemented as mean of navigation source, it will be highly unlike that all such system are unavailable at the same time. However because of incompatibility of the systems, it is expensive to purchase systems which are able to utilize multiple navigation systems at the same time [2]. Also cloud covers which block signals from satellite during bad weather remains as an unresolved issue.

Celestial Navigation

One of the oldest navigation techniques available to mankind, and had been used by sailors as a primary source of navigation for centuries until the available of GPS. Celestial navigation is based on locating celestial bodies in combination with information such as exact time the measurement is taken, almanac which give angular schedules of celestial objects, and a chart of the region to be navigated, then using information available in sight reduction table the only mathematics required is addition and subtraction to deduced one's location [3]. The idea used here is similar to those employed by GPS, expect with GPS navigation satellites are used instead of celestial bodies.

There are however a few problems with this technique. Firstly, when used on a moving body such as UAV, celestial navigation has a practical accuracy of about 2.8 km [3], this range of error is already a significant portion of usual range of an UAV. Secondly celestial navigation requires some instruments that can measure the angular position of celestial bodies. This type of equipments are not available on typical UAV and one research paper quote one such star tracker produced by Lockheed as weighs about 4 kg with dimension of 15 cm × 15 cm × 30 cm [4]. This is quite impractical to put on a UAV. Lastly celestial navigation only works at night. This system is to work during the day, so this alternative is not feasible.

Geomorphometric Navigation

This method basically involves using radar or laser scanning system to scan the ground for terrain features, features may include hill, vegetation and building on the ground [5]. Once the locations for these features are obtained using the sensor, these information are then searched in the database for a match and thus locating the location of the UAV. Techniques which operate under similar idea are used in submarine and are called bathymetric navigation [4]. In this technique, echo-sounder in submarines are used to scan the bottom of the ocean to obtain terrain details, which are then

compared with digitally stored terrain data of the sea bottom being traversed.

This method required detail and up-to-date database of terrain detail of where the UAV is flying over and instrument such as laser/radar to scan the ground. Neither of these two resources is available thus making it difficult to implement for the purpose of this project.

Localisation Using Model Image Correspondence

This method is in a way similar to geomorphometric navigation, expect it uses features extracted from images from onboard camera instead of radar and laser. First a detail outline of the area to be navigated is store in some sort of digital model, for example a CAD model for indoor navigation or digital elevation map which store the topography of the area in a digital format (a data base which store the coordinates and numerical descriptions of altitude of each location) for the purpose of outdoor navigation. Then using information about features extracted from images taken by the on board camera an internal database is searched for corresponding feature hence achieving localization of vehicle [6]. Again this method requires detail model of the space the UAV is flying over, and because it is flying outdoor it is likely to be in the form of a digital elevation map. This type of information is not available for this project, thus making implementation of this technique difficult.

Feature Based Navigation

The technique purposed "Feature Based Navigation" is similar to the technique "Localisation Using Model Images Correspondence", but it does not require a database which store locations and properties of features. This method works by comparing location of features on two images, one of such images is taken by the on board camera of the UAV during flight, while the other image is extracted from a reference map base on attribute of the UAV such as its pitch, roll, yaw, heading, altitude at the instance of time when the image is captured by the on board camera and the location of the UAV at previous time step. The location of the UAV is calculated based on relative locations of identical features that are in on the two images. In a way the reference map replace the database and searching of the database is replace by searching through the features extracted from the image extracted from reference map and the image taken by onboard camera.

Feature based navigation is considered the most appropriate technique in this case, as it does not require additions of instruments or hardware to be installed on the UAV. The whole system can be implemented in software. Nor does it require detail modelling of the area which the UAV is flying over like Localisation Using Model Images Correspondence which maybe difficult to obtain. All that is required is a reference map in the form of an aerial photo and the longitude and latitude location of the four corners which are information that can be readily obtained by purchasing aerial photo as in the case of this project, or from mosaicing images taken by the onboard camera when UAV is flying at a constant altitude, pitch, roll and yaw.

3 Basic System Operation

This section serve as an overall introduction to the structure of the software system, it describing major tasks involved in feature based navigation and how they flow logically. It also gives a brief introduction into each of these tasks, while full description of how major components work and are implemented will be detailed later in this report. Figure 1 shows a flow chart which demonstrated the major tasks involved and the flow of the system; it also highlights major subsystems of the system.

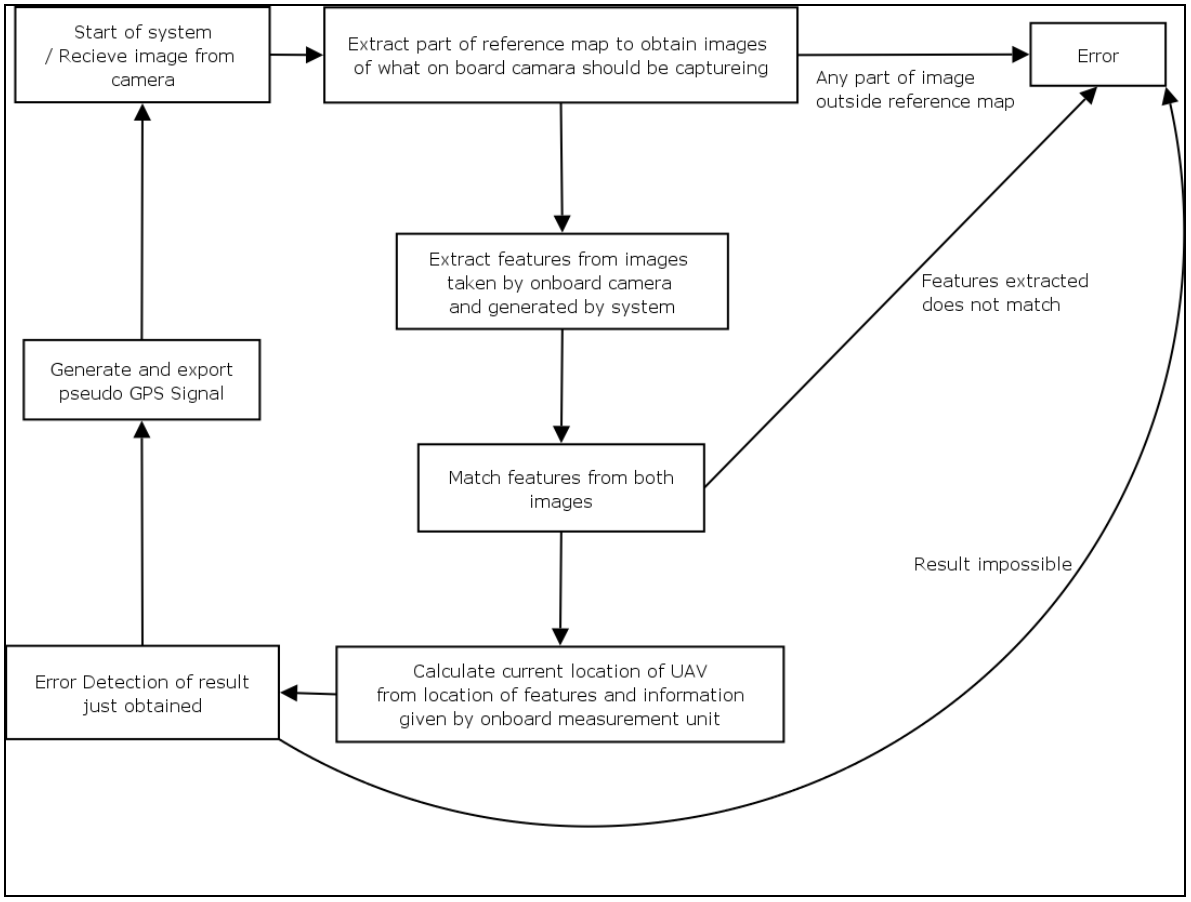


Figure 1: Flow chart of the system

From the above flow chart the following independent subsystems are identified and listed as follows:

Extract part of reference map to obtained images of what on board camera should be capturing

***Main task*:**

To generate an image of what the UAV's on board camera should be capturing

***Approach to task*:**

Using attribute of the UAV such as its pitch, roll, yaw, heading and altitude at the time when image from the on board camera is captured and the location of the UAV at the previous time step, an image of what the UAV's on board camera

should be capturing is generated. If during the calculation the system found that it was impossible to generate the image according to the information provided because either the image seem to be out of the bound of the reference map or the information indicates the camera capture a area which does not consist entirely of the ground it will flag an error and the system will enter the error state.

Extract features from image taken by onboard camera and image generated by system

Main task:

Generate a list of candidate features which is passed on to the features matching stage.

Approach to task:

The images are first transformed to grey level images, and then the images are passed through an edge detection algorithm to extract features for comparison. Characteristic of such as area, centroid, Euler number of each features are calculated. Lastly a number of largest of features are then selected as candidate features.

Match features on both images

Main task:

Find matching features on both images

Approach to task:

From the characteristics of features within the images, the system selects 3 matching features which are present in both images. If matching features cannot be found then an error is flagged.

Calculate the location of UAV:

Main task:

Calculate the location of the UAV based on the location of the matching features on the two images

Approach to task:

Using the different in relative location of matching features on both images the system calculate the location of the UAV. It then check the whether the results obtained is sensible by check it against the previous location of the UAV and properties of the UAV (such as maximum speed, roll, pitch and yaw), if it is a then the information is passed on to the external interface to generate pseudo GPS signal, if not error will be flagged.

Error handling

Purpose: To handle errors conditions during the course of the program

Description: This task is not really an independent task, but rather it is implemented within every subsystem to handle errors that occur during their operation. Different subsystems handle error slight differently, but the basic response of the system to error such as failure to match features and the error in calculation is to maintain the location output to that of the previous time step.

External interface

Main task:

Grabbing images off the video to enable comparison and generate pseudo GPS signal when the calculation for the location of the UAV is complete

Approach to task:

This module will regularly grab a frame of image from the video and pass it through the system. It is also be responsible for generation of pseudo GPS signal once the calculation is complete.

Each tasks stated above besides error handling are implemented as independent subsystems. Primary reason for implementing these tasks as independent subsystems is based on the software engineering principle of modularisation and reduction of dependency. This way of implementing the system enables easy replacement of subsystems which are performing unsatisfactorily. As long as a subsystem has the same interface (identical format of input and output parameters) as the previous implementations then there should be no problems in taking it out and plugging in the new one. This enables rapid prototyping to test out new theories and ideas without affecting the overall structure and stability of the system and makes the system easier debugging and maintain as system is broken into more manageable subsystems and error should be localised. It also enables the system to be implemented incrementally and enable testing of system without have the entire system built.

The theory and the implementation detail that are behind the first four modules listed above will be discussed in upcoming sections of this report.

4 Reference Map Extraction

As part of the feature matching process, the system has to be able to generate an image of what the camera should be capturing when the UAV is flying at a certain location on the reference map and having particular attributes. Before starting to discuss how this is achieved, it is worth summarizing what information are known at any instant of time when this process is needed.

The following is a list of data that are known at instances when there is a need to start extracting part of the reference map. These data can either be obtained from the flight log (on board sensor), pervious calculation, or from measurement of physical properties of camera and UAV:

- Optical properties of the camera, these include the view angles of the camera. The view angles refer to the angle from the centre of the camera to where it is able to captured horizontal and vertically.
    - These data can be obtained by measuring the view area of the on board camera at a known distance
- Altitude, pitch, roll, yaw and heading of the UAV when image is taken
    - These information can be obtained from the on board sensor, or from flight log of the UAV
- Location of the UAV (latitude and longitude location) at previous time step
    - This might be the previous result of the system or the last GPS coordinate before the GPS signal is lost and this system kick in
- Latitude and Longitude location for the four corners of the reference map, and its scale (i.e. what is physically dimension 1 pixel on the reference map equal to on the ground, for example in the reference map used in this project 1 pixel is equal to a 0.36m by 0.36m square on the ground)
- Mounting arrangement of the UAV, specifically the mounting angle of the camera with respect to vertically down as shown by the angle labelled ω in the diagram below. For example mount angle of the camera mounted on the UAV used in this project is 57°

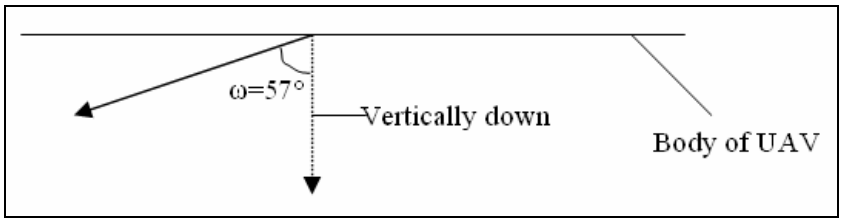


Figure 2: Camera mounting arrangement of UAV

- Resolution of the onboard camera

With the above information it is possible to generate an image of what the UAV camera should be viewing.

The main problem with the extraction of reference map is how to account for the effect of orientation of the UAV on the image the on board camera will capture. If the camera is pointing directly down towards the ground, then it is relative simple to calculate the view area of the camera on the ground by trigonometry. In this case, the image taken by the on board camera will simply be a rectangular section of the reference map directly under the location of the UAV, and the size of the section will be a function of the altitude of the UAV when the images is taken and the viewing angle of the

UAV. But as the UAV fly it has varying pitch, roll, yaw and heading, so it is almost impossible for the camera to be always pointing directly down at the ground during flight. This make the calculation for which section of the reference map the camera should be viewing a rather tricky task, as the view area will no longer be a nice and regular rectangular, but would be a four sided polygon of irregular shapes depending on the attribute of the UAV at the time. Thus it is crucial to find some way to represent how the orientation of the UAV will affect the view area.

4.1 Coordinate System Used

Before going into mathematics of how to find the view area, it is important to make note of the coordinate systems used.

There are two coordinate systems used. One of them is the global coordinate system or the coordinate system used to represent location on the reference map; in this coordinate system positive *x* is aligned with due north, positive *z* is due east and positive *y* is vertically up. Origin of this coordinate system is the bottom left hand corner of the reference map. This coordinate system is shown in Figure 3.

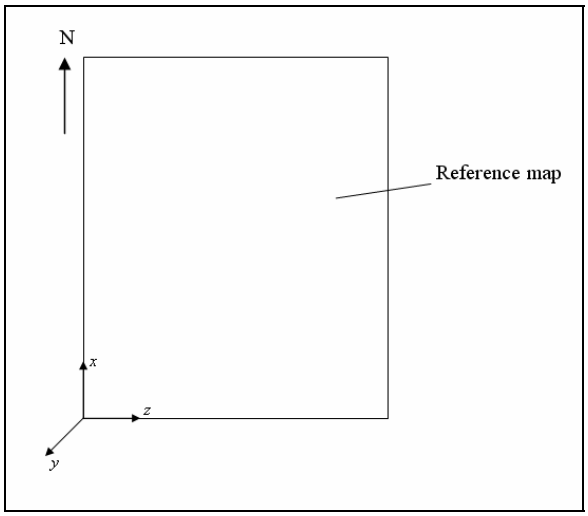


Figure 3: Global coordinate system

In the local coordinate system the UAV or the body coordinate of the UAV, positive *x* axis is towards the nose of the UAV, positive *z* axis is towards the right wing of the UAV, and positive *y* axis is towards the sky when the UAV is flying upright. Origin of this coordinate system is assumed to be at the location of the on board camera. It is also worth mentioning the notation for direction of pitch, roll, yaw and heading. Positive pitch is consider considered to be noise up, positive roll is consider as right wing down and positive yaw and heading is consider to be noise to the right. The body coordinate system of the UAV and the notation for pitch, roll and yaw are shown in Figure 4.

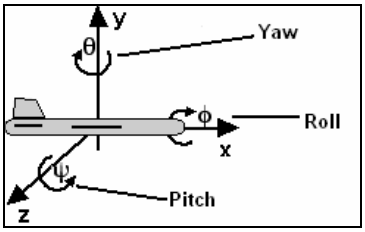


Figure 4: UAV coordinate system (modified from images from [7])

4.2 Euler Angle

Euler angle is a way to represent the rotational properties of a rigid body; it is specific by three angles along the three major axles. It can be used generate transformation matrix to rotate a vector with the required pitch, roll, and yaw angle to represent orientation of a rigid body. This is very useful for purpose of this project as the information about the view angles of the camera can be used to generate view vector in the body coordinate system of the UAV, then by rotating the view vector by the transformation matrix given by the Euler angle and translating it, the location of the where the camera is capturing at an instance can be known, thus allowing the calculation of the view area of the camera. The method involved in calculating the view area will be mentioned later, first the basics of Euler Angle is explained here.

Euler angle is defined of three angular quantities, the pitch, roll, and yaw of an object. Each of these angles represents a rotation along one of the three major axles (*x, y* and *z* axis). In the coordinate systems defined above, pitch (angle represented by symbol φ) represents rotation along the *z* axis, roll (angle represented by symbol ψ) is rotation along the *x* and yaw (angle represented by symbol θ) is rotation along the *y* axis. The rotation direction of each of these rotation is specific by the right hand rule (hold the hand such that the fingers are crooked and the thumb is point out, now with the thumb point out at the axis, the direction where the fingers are crooked is the positive direction of rotation).

Converting these rotation angle into matrix notation allow to easily transform a vector by a given angle. Here are the three transformation matrices which represent the three Euler angle:

Pitch(rotation along *z* axis):

$$R_Z(\varphi) = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Roll(rotation along *x* axis):

$$R_X(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & -\sin(\psi) \\ 0 & \sin(\psi) & \cos(\psi) \end{bmatrix}$$

Yaw(rotation along *y* axis):

$$R_Y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

Rotational matrix which are the result of rotation along the three major axles as stated [7], and [8]

These matrix will be able to calculate the require transformation matrix when only one of the three Euler angles is involved. But often all three Euler angles would be needed to correct represent a UAV orientation. Like most transformation, the effect of Euler angle can be combined to calculate the final transformation matrix which will in one step perform the required transformation. The problem here is the order in which the transformation should be

performed; as matrix multiplications are not commutative a wrong order can mean generating the wrong transformation matrix.

It turns out with Euler angle yaw should be applied first; follow by pitch and lastly by roll, this is to follow the NASA Standard Aeroplane format [7] [8]. Because we are only required to perform rotation on the UAV to represent the orientation of the UAV, and the UAV is defined in the global coordinate system, so the required transformation will be operating in global coordinate system. In global coordinate system the order of transformation is given as $R=R_1R_2R_3$, where R is the final combined transformation matrix and R_1, R_2 and R_3 are the first, second and third rotation required, therefore the order of transformation become the following:

$$R = R_X(\psi)R_Z(\varphi)R_Y(\theta)$$

Any vector in the global coordinate system that need to be transformed by pitch angle φ, roll angle ψ and yaw angle θ can then be transformed like the following:

$$v' = R_X(\psi)R_Z(\varphi)R_Y(\theta)v$$

where v is the original vector and v' is the transformed vector

4.3 Reference Map Extraction - Basic Concept

With Euler angle explained, it is now appropriate to explain the basic concept of how part of the reference map is extracted according to various attribute of the UAV.

With most digital camera the view area of it at a certain distance is rectangular. The size of the viewing rectangle depends on two things; the distance between the image and the camera, and the viewing angles of the camera. An image taken by a digital camera also has a certain resolution, which specific the amount of pixels vertically and horizontally on every image the camera takes. Using these two ideas and assuming pixels are equally spaced across an image; it is possible to calculate the physical location of what is captured by a pixel in an image when it is capturing an object at a certain distance away from the camera. The following example show how to calculate where the top left corner an image of a wall taken by a camera with view angles of 13° vertically and 16.9° horizontal at 1m from the wall:

Given the vertical view angle is 13°, the arc tangent of it multiple by 1 will give the distance vertically up that the pixel is capturing; this is demonstrated by the diagram below:

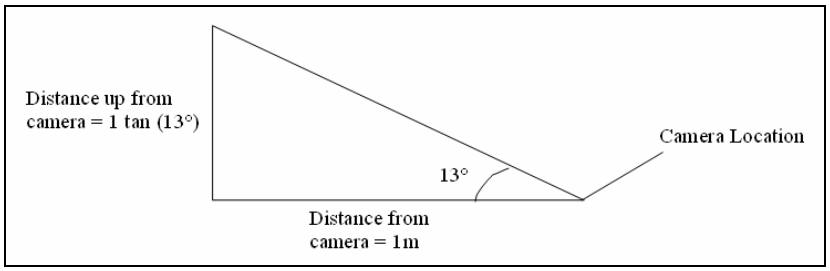


Figure 5: Vertical view distance of a camera at 1m away

The distance up will be 0.23m in this case. Similarly the horizontal distance is 1 tan(16.9°) = 0.304m

Assuming the camera is at the origin and pointing North, the top left corner will be at coordinate (1, 0.23, -0.304)

Using the same concept as above the four corners of the camera viewing rectangle at a certain distance can be works out easily. Once the location of the four corners are known, then using the idea that the pixels are placed at equal distance from each other, the physical location of where a pixel is capturing can be easily calculated as show in the following example:

Assuming a camera with the same optical properties as those given above, it has a resolution of 320 by 240 pixels (320 horizontal and 240 vertical) and is taking a photo of something one meter away. The location for what is captured by pixel on row 200, column 100 can be calculated can be calculated as follow. First calculate the 4 corners of the viewing rectangle at a given distance:

	x	*Y*	*z*
Top left	1	0.23	-0.304
Top right	1	0.23	0.304
Bottom left	1	-0.23	-0.304
Bottom right	1	-0.23	0.304

Row 200 is 5/6 of the way down the image from the top, so its y location is at:
$0.23 - 0.23 \times 2 \times 5/6 = -0.1533$
Similarly column 100 is 5/16 way across the image from the top, so its z location is at:
$-0.304 + 0.304 \times 2 \times 5/16 = -0.114$
So the location of this pixel is at (-0.1533, 1, -0.114) in local coordinate system of the UAV, and in global coordinate system if the camera is assumed to be at the origin of the global coordinate system.

Using the method describe above the physical location of object captured by each pixel when the object is at a certain distance from the camera can be calculated.

This method effective gives a vector from the camera location (assumed to be at origin in above case) to any pixel on the viewing rectangle of a camera at a certain distance. This vector can be transformed with the Euler angle

transformation matrix stated above to represent the effect on of pitch, roll, and yaw when the UAV is flying. Furthermore this vector can be translated into place in the global coordinate system and linear equation can be used to calculate the point on the ground which this pixel will capture. More about this will be discussed in implementation section.

4.4 Reference Map Extraction – Implementation

The tasks involved in implementing the above algorithm include the following steps, and these steps have to be conducted for every pixel in the camera's view rectangle:

1. Calculate the vector between the camera and what is captured physically at a know distance for the pixel in question, this is done by applying the transformation matrix which is calculated according to the orientation of the UAV at the instance when the image is taken by the on board camera to the viewing vector of the pixel
2. Translate the vector to where the UAV is suppose to, so the vector now give the physical location of the what the pixel should be capturing at a certain distance away from the UAV camera when the UAV is in the longitude and latitude location it is at the previous time step, but at the altitude of the current time step
3. Using the location of the UAV and the vector from the previous step to calculate the location of what this pixel should be capturing on the ground (i.e. the pixel location on the reference map this pixel should be capturing).
4. Use the information provide by the previous step to fill the pixel with the correct information

Each of these steps will now be explained in detail.

Step 1

In the system the initial position of the camera is assumed to be at the origin of the world coordinate system, point directly toward the ground. This has no real physical meaning, but just made it easier to conduct the sequence calculation required. The system stored an array of vector, one for each pixel in the image, these vectors contain the physical location of what a pixel should be capturing at a given distance from the camera when it is pointing directly down towards the ground and placed at the origin of the global coordinate system. As the resolution for the camera used in this project is 320 by 240 pixels, there are in total 320×240 such vectors. This is done primarily to save repeated calculation. To process the calculation for a pixel, the system first finds the correct vector by using the row and column information to access the array. Then this vector is rotated along the z axis by the angle which is the mounting angle of the camera (ω as stated above) in the UAV. This step is necessary as it compensate for the fact that the camera is not pointing vertical down, but is mounted at an angle. This angle cannot be treated as an additional pitch as it is an inherit properties of the camera, not part of the UAV orientation. This effectively gives the view vector for the pixel without taking into account of the effect of pitch, roll, yaw and heading of the UAV.

The next step involved calculating the necessary transformation matrix R based on the orientation of the UAV. The yaw angle also includes the heading of the UAV, so the yaw angle is actually the sum of heading and yaw of the UAV. Once this matrix is calculated it can then be apply to the vector thus giving a vector which represent the location of what the

pixel should be capturing when it is places at the origin but orientated at the correct orientation.

So the overall mathematics involved in this step is listed as the following:

$$v' = R(\psi,\varphi,\theta)R_X(\omega)v = R_X(\psi)R_Z(\varphi)R_Y(\theta)R_Z(\omega)v$$

where *v'* is the final transformed vector, and *v* is the original unaltered vector taken from array

Step 2

This step involves using the location of the UAV at previous time step and the current altitude of the UAV to calculate a translation vector which will translate the UAV from origin in the world coordinate to where it is longitude and latitude wise at the previous time step but at the altitude of the current time step. The *x* and *z* magnitude of the translation vector is calculated by using the relative location between the last GPS coordinate (either from last result of system or last known GPS coordinate), the longitude and latitude location of the bottom left corner of the reference map, and scale and size of the reference map. The *y* magnitude is calculated by dividing the altitude of the UAV with the scale of the reference map. The translation vector found using this method state where the UAV is in the global coordinate system, by adding this vector to the vector obtained in the previous step the physical location of where the pixel should be capturing at a certain distance in world coordinate is obtained.

So the overall mathematics involved in this step is listed as the following:

$$tv = v' + TL$$

where *tv* is the final translated view vector, and *TL* is the translation vector and where the UAV is in the world coordinate system

The vector *tv* will give the location of what a pixel should be capturing in the global coordinate system at the location of the UAV when the camera is capturing objects at a certain distance away from it.

Step 3

This step involves the calculation of what the pixel is going to capture on the ground, which are basically the coordinate where a line which start from the camera location, passing through the point specific by the vector *tv* as calculated in the previous step will hit the ground. This can be done by using 3-D linear equation.

Assume a line which is show as show in Figure 6:

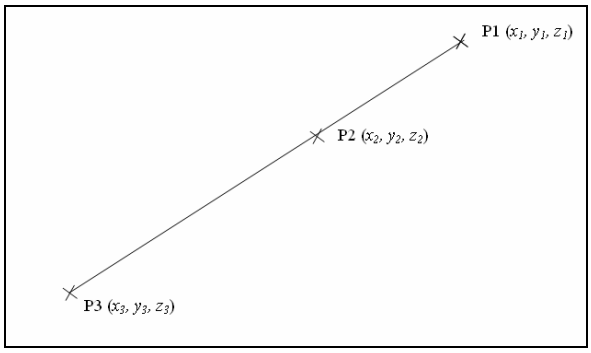


Figure 6: A 3-D line which passes through three points

In this case items that needed to be solved are the values of *x3* and z_3. The all values of P1 are know, they are simply the value of the translation vector *TL*, similarly values of P2 are that of viewing vector *tv*. The value of $y_3 = 0$ as it is always on the ground.

Values of *x3* and z_3 can be found by the following equations [9]:

$$m = \frac{y_3 - y_1}{y_2 - y_1}$$
$$x_3 = m(x_2 - x_1) + x_1$$
$$z_3 = m(z_2 - z_1) + z_1$$

Step 4

The last step is quite trivial, once the values of *x3* and z_3 are found; they had to be converted to row and column number in the reference image. Once they are converted the RGB value of the pixel can be read and stored. Then the system can move on with calculation of the next pixel

The diagrams on the next page show the reference map and the longitude and latitude measurement for the four corners of it. It also show some of the images extracted from the reference map according to data from the flight log.

Longitude and Latitudes positions for the four corners of the reference map:

Corners	Longitude (Degree South)	Latitudes (Degree East)
Top Left	37.87267	145.19927
Top Right	37.87272	145.21598
Bottom Left	37.88889	145.19934
Bottom Right	37.88893	145.2161

The above figures were taken by taking measure of the actual location of these points with a hand held GPS unit. The accuracy of the measurement was said to be 4m.

Figure 7: Reference map used in this project

Figure 8: Two images extracted from the reference map. On the left the UAV is at 145.2130° East, 37.8774° South, with altitude of 176.29m, pitch angle of 1.5667°, roll angle of -20.8144°, and heading angle of -55.5053°. On the right the UAV is at 145.2082° East, and 37.8770° South, altitude of 171.37m, with pitch angle of -0.8952°, roll angle of -0.5036°, and heading angle of -107.7676°

Figure 9: An image show where the two extracted images are on the reference map, the blue line show the location of the image on the left, while the red line show the location of the image on the right

4.5 Performance of this sub system

Because 76800 pixels have to be processed each time an image is created, the process of extracting an image is quite calculation intensive and therefore time consuming. There might be scope for improvement by just running the process for just the four corners of the image, and use some sort of linear interpolation algorithm to calculate the pixels in between, this system is intended as a prove of concept system these optimization were not considered.

5 Feature Extraction and Matching

Two of the most important task in this project is extraction of features from the images extracted from the reference map using the longitude and latitude location of the pervious time step and attribute of UAV the at the current time step, and the images taken by the on board camera. Then once the features are extract finding matching features on the images. This section describes how those two tasks are done.

5.1 How Digital Images Are Stored

Before discussing about those two tasks, the basic of how images are stored digitally is hereby discussed. In digital imagery, one of the ways the colour of a coloured pixel can be represented is as by a set of three numbers ranging from zero to a maximum number (usually 255). One of these numbers is used to represent the intensity of red in that pixel, and the other two representing the intensity of blue and green in that pixel. For example a red pixel can be represented as having a red intensity value at maximum, while zero in both green and blue intensity value. This set of numbers is the RGB value of a pixel and is the way that colour of a pixel is stored in this project. With black and white images, the intensity of a pixel can be represented with a single value ranging from zero to a maximum number (again usually 255) which represent the brightness of a pixel ranging from completely black which has a value of 0 to white which has the value of the maximum number in the range. This value is known as intensity or grey level of a pixel. As a side note a black and white image is sometimes refer to as grey level image.

Using the above concept, an image is can be represented as a function of pixel intensity in different location of the image [10]. Essentially the intensity of pixels at different location of an image can be represented by a function of two variables; one which specific the row location and the other which specific the column location of a pixel. As an example, the intensity value of a pixel on a black and white image can be represented as follow:

$$f(x,y) = I_{x,y}$$

where $f(x,y)$ representing a function of pixel intensity at location x,y and $I_{x,y}$ is the pixel intensity of the image at location x,y

5.2 Choosing Feature Type of Features to be Matched

In this project it was decided that edges in images are used as features to be identified and match and used enable calculation for the location of the UAV. Edges are used because they are features that are not as easily affected by changes in the lighting condition when the images are taken. Originally to obtain candidates for matching features the images were thresholded by an intensity value and then find a number of largest pixel groups and used them as features. This is shown in Figure 10 below. Then by calculating different properties such as centroid, area and Euler number of each extracted candidates an algorithm can be used to match these identical candidates on both images. This

information can then be use later to calculate the location of the UAV. The major problem with this approach is if the images (reference map and those taken by the on board camera) are taken at significantly different lighting conditions then it is likely that shadows will change the shapes and sizes of the features. As the reference images might be taken months prior to the actual flight in different time of the day and possibly in different season, the lighting conditions are almost certainly different. This makes the method quite unreliable and not as robust as initially through.

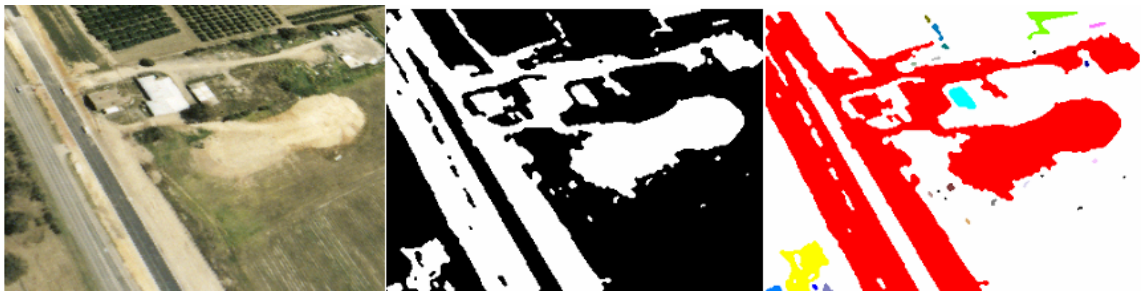


Figure 10: From left to right Original images, thresholded image, and thresholded with pixels group highlighted in different colour

To counter the problem above the features that are used as candidates must be invariant to lighting conditions, in another word they must remain more or less the same when lighting conditions change. One of such feature are edges in an image another is corners in an image. Different lighting conditions are likely to introduce or eliminate certain edges and corners in an image, but some of these features will always remain in place. This property allows us to use them as tracking features to calculate the location of the UAV. For this project edges was chosen to be the tracking feature.

Edges are used, as existing corner detectors are in general not as accurate and fast as edge detectors. Corner detectors such as Harris/Plessey Operator [11] and Trajkovic Operator [12] which are anisotropic (they exhibiting properties with different values when measured in different directions) and therefore not rotationally invariant, meaning if an image is rotated the corners detection ability of these operators are likely to suffer. This effect is demonstrated in Figure 11, it shows the Plessey operator operating on two identically images but one rotated at an angle. As there are likely to be slight different in rotation between the image generated by on board camera and the one generated by extracting from reference map, corner detector is not suitable for this purpose.

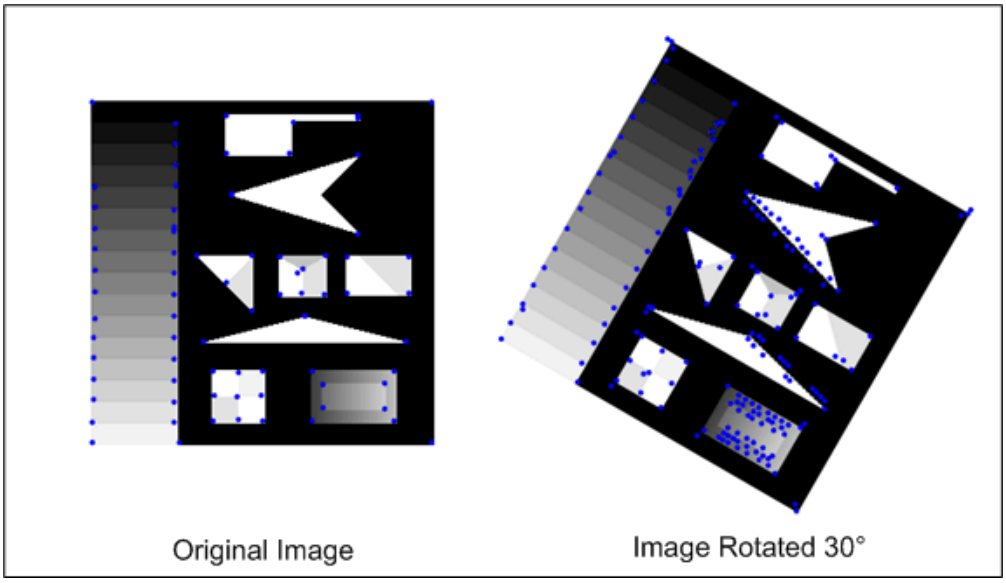


Figure 11: Rotational instability of Plessey operator due to anisotropic response [11]

Therefore edges were chosen as the features used for tracking purpose.

5.3 Basic Concept of Edge Detection

Edges on images are regions between two different intensity values, in another words they are where discontinuity of intensity values of pixels in an image exists [10]. In pure signal processing terms an edge are places where the spatial frequency is high. Therefore edges can be detected by finding first or zero crossing of second order derivatives of pixels intensity of an image.

Concept of edge detection by first derivative

Because an image is in 2-D, therefore gradient of intensity values is used to calculate the first derivative of it [10]. The gradient of a 2-D function, *f(x,y)* is defined as the vector:

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

The magnitude of this vector is:

$$\nabla f = mag(\nabla f) = [G_x^2 + G_y^2]^{1/2} = \left[\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \right]^{1/2} \approx G_x^2 + G_y^2 \approx |G_x| + |G_y|$$

The magnitude of the gradient function will be large in area of changing pixel intensity, and in region of constant intensity the magnitude of the gradient function will be small. Therefore using the above function an edge can be defined as regions where the magnitude of the gradient function is larger then a specific threshold.

Concept of edge detection by second derivative

The centre of an edge should be where the maximum of the first derivative occurs; therefore at these points the second derivative will be zero. Under these assumptions, edges are places where the second derivate of the intensity has a zero crossing.

A general way to compute the second order derivates is to compute the Laplacian of a 2-D function [10], as it is formed from second order derivatives as follow:

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

Commonly used digital approximations of second derivatives are:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y)$$

and

$$\frac{\partial^2 f}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y)$$

so that

$$\nabla^2 f(x,y) = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] - 4f(x,y)$$

However the Laplacian is in practice not used as an edge detection technique by itself. As a second derivates, it is very sensitive to noise, and will give double edges.

There are quite a few implementation of the basic theory into actual filter, in here a few of these will be listed and discussed:

- Sobel Edge Detector and Prewitt Edge Detector [10]

  These two edge detectors are based on the theory of first derivative stated above. It approximates the first derivative by calculating the G_x and G_y component like the following:

  Assuming the pixel which we want to find the derivative to is labelled Z_5 below:

Z_1	Z_2	Z_3
Z_4	Z_5	Z_6
Z_7	Z_8	Z_9

  Sobel approximate G_x and G_y as the following

$$G_x = (Z_7 + 2Z_8 + Z_9) - (Z_1 + 2Z_2 + Z_3)$$
$$G_y = (Z_3 + 2Z_6 + Z_9) - (Z_1 + 2Z_4 + Z_7)$$

  Prewitt approximate G_x and G_y as the following:

$$G_x = (Z_7 + 2Z_8 + Z_9) - (Z_1 + 2Z_2 + Z_3)$$
$$G_y = (Z_3 + 2Z_6 + Z_9) - (Z_1 + 2Z_4 + Z_7)$$

  With G_x and G_y calculated the magnitude of the first derivative at the point can be calculated by equation stated above and based on the magnitude of the result the pixel can be classified as part of an edge or not.

- Laplacian of a Gaussian (LoG) Detector [10]

  With this detector, two actions are performed on the image. First the image is smoothed or blurred by a Gaussian filter, where the image is blurred by convolving the following Gaussian function with an image:

  $$h(r) = -e^{-\frac{r^2}{2\sigma^2}}$$

  where $r^2 = x^2 + y^2$ and σ^2 = standard deviation, this variable affect the degree of bluring by this filter

  The Laplacian of this function (second derivate with respect to r) is:

  $$\nabla^2 h(r) = -\left[\frac{r^2 - \sigma^2}{\sigma^4}\right] e^{-\frac{r^2}{2\sigma^2}}$$

  By convoluting this function with the images two things are achieved. First it smoothes the image and at the same time create a double-edged images. Edges of the original images can then be located by finding the zero crossing between the double edges.

For this project the edge detector used is called the 'Canny Edge Detector'. Widely regarded as one of the most powerful edge detector [10] [13] it was developed by John Canny did for his Masters degree at MIT in 1983.

5.4 Canny Edge Detector

The Canny Edge Detector needs three parameters, sigma σ and two threshold values, lower threshold T1 and higher threshold T2. It works according to the following steps [10] [13]:

1. The image is first passed through a Gaussian filter by convolving it with a function like the Gaussian function described above. This step act as a noise suppression stage for this detector. The σ value is the standard deviation of the Gaussian filter and will affect the degree of blurring. The larger the sigma the more blur the image and the better the ability for the filter to suppress noise and unwanted edges.
2. The local derivative (gradient) of the filtered images is then calculated. This can be achieved by using either the Sobel or Prewitt edge detector described above. This step gives a 2-D array which contains the magnitude of the intensity gradient at each point of the smoothed image.
3. Step two will results in ridges in the gradient magnitude array. The algorithm then track along the top of these ridges and set to zero the pixels which are not actually on top of the ridges. This process is called nonmaximal suppression and will results in thin line in the output. The ridges pixels are then thresholded using the two thresholds T1 and T2. Ridges pixels with magnitude bigger then T2 is said to be "strong" edge pixels, while ridge pixels with values between T1 and T2 are said to be "weak" edge pixels. Tracking can only begin in "strong" edge pixels and continues in both directions out from that point until the height of the ridge falls below T2. This helps to ensure noisy ridges are not broken up into multiple fragments.
4. Finally the edges are linked together. In this step "weak" pixels are 8-connected to the strong pixels to from the output.

In theory an edge detector is essentially a high pass filter while blurring filter such as mean filter and Gaussian filter are low pass filter, so it seem rather illogical to first pass the image through a low pass filter before trying to obtain its high frequency component of it(edges). This issue highlight the major difference between other low pass filter such as mean filter and Gaussian filter employed here. With most low pass filter such as mean filter, the frequency response of such filter is often unknown and often non-uniform. But with Gaussian filter the frequency response is well known and uniform, it fact if is like half of a Gaussian curve. This is shown in plot of frequency response of a mean filter and a Gaussian filter in Figure 12[14]. The frequency responses of a Gaussian filter means by choosing appropriate value of σ; the filter can very effectively filter out anything that is above a certain frequency. This is very favourable in this application as it allow precise selection of the amount of detail which remain in the filter image and thus to a certain degree allowing for the selection of what edges should be leave in the edge detection scheme.

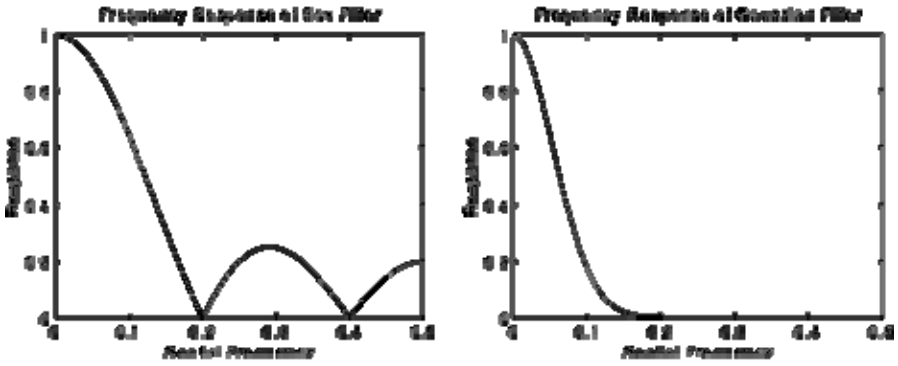


Figure 12: Frequency responses of Box (i.e. mean) filter (width 7 pixels) and Gaussian filter (σ = 3 pixels). The spatial frequency axis is marked in cycles per pixel, and hence no value above 0.5 has a real meaning [14].

The Gaussian filter is implemented as an image filter of variable size depending on the size of σ. Technically Gaussian filter should be infinitely large as it will never reach zero anywhere as shown in diagram below which plot the Gaussian distribution with σ = 1. But in practice it is effectively zero about 3σ away from mean as shown in Figure 13, so the filter can be truncated at this point.

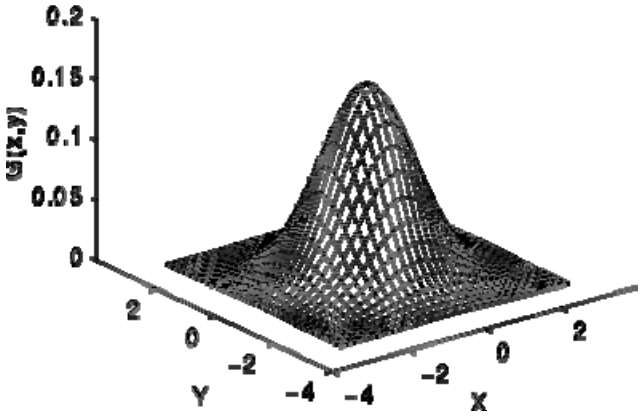


Figure 13: 2-D Gaussian distribution with mean (0,0) and σ=1

So a filter with σ of 1.4 can be implemented as the following filter [14]:

1/115

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2

The choosing of threshold also values also required some experimentation. As they have to be high threshold T2, has to be low enough to allow the required features to pass through, but at the same time cannot be set too low, otherwise there will be too much unwanted information in the image. The low threshold T1 also has to be selected such that there is no significant break up in edges captured.

5.5 Feature Matching

Because the time between each flight log entry is only 0.2 second and the speed which it is flying at is no more then 30 m/s with cruising speed of roughly 20m/s, therefore a feature across two frames should be located at very similar location. This enable a simple matching algorithm is used. A feature captured on two images is considered the same if certain characteristics of these two features are the same. These criteria will be discussed in more detail later.

5.6 Implementation

Before passing the images into the edge detector the images are first transformed into grey level (black and white or intensity) image. This is because the edge detecting algorithm will only take grey level images as inputs.

Edge Detector

Because of the aims to keep the number of candidates features to be matched to a minimum, thus reducing the chance of mismatch features, when implementing the edge detector cares were taken to select parameters that will achieve this aim. One way to achieve this aim is to use a reasonable large σ. As stated before, by increasing the size of σ the filtering effect of the Gaussian filter become more aggressive. This help to suppress the amount of edges pick up by the edge detector. Diagrams below show the effect of using different σ values in the detector on the output of the edge detector that is applied to the same image.

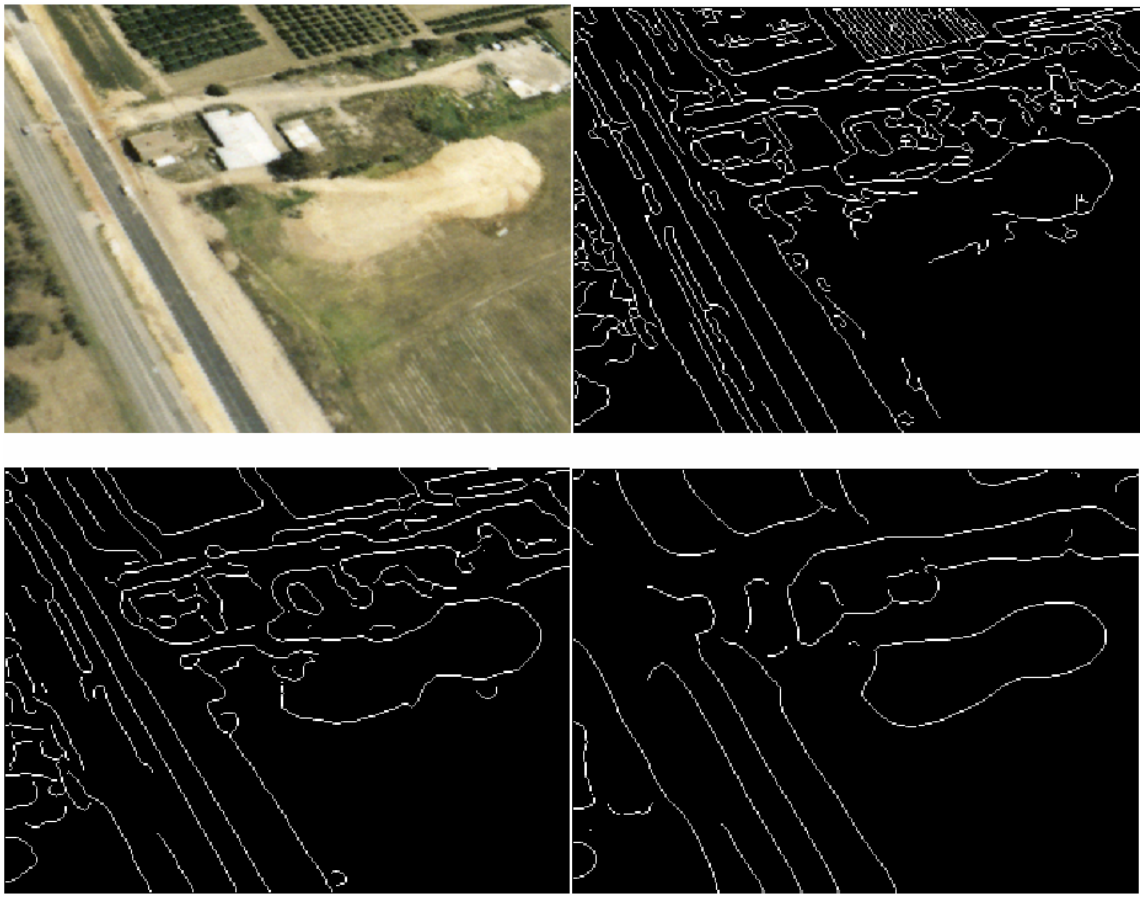


Figure 14: Canny filter with different σ. Top left: original image. Top right: σ = 1. Bottom left σ = 2. Bottom right σ = 6

After some experimentation it was decided that σ value of roughly five is the right value for this purpose as it is able to suppress majority of the unwanted features leaving only major edges.

In addition to using a σ value of roughly five, the threshold values were also chosen as the follow: T1 = 0.05 and T2 = 0.1. This set of threshold values are rather low, but as the Gaussian has already suppressed most of the unwanted features, it was decided that the nonmaximal suppression should not try to take away anymore of the features, but rather try to keep fragmentation of edges to a minimum. The diagrams below show a two outputs of the edge detector when different threshold values are used.

Figure 15: Canny filter with different threshold. From left, original image, T1 = 0.05 T2 = 0.10, T1 = 0.2 T2 = 0.3

Feature Matching

In design of the feature matching module a few factors are considered:

- Flight log entry is available every 0.2 seconds
- The flying speed of the UAV during cruising is roughly 20m/s, so the distance travel by UAV in between 2 log entries is roughly 5 meters
- Change in pitches, row, yaw and heading between 2 logs points are generally no more then 1-2 degree between log entries

Therefore if this system is to replace GPS on board it can at maximum be able to calculate the location of the UAV every 0.2 seconds, or as often as now information about the UAV is provided.

Under these assumptions, it can be assumed that features capture by the UAV should also have moved and distorted by only a small amount during the time interval between 2 log entries.

The feature matching module takes a list of candidate features and their properties from two images. The two images are from the on board camera and that extracted from reference map based on attributes of the UAV at the instance when the on board camera take the image. The module then attempt to find the largest 3 matching features, this list of matching features are then passed on to another module to calculate location of UAV.

The subsystem matching features based an algorithm which is developed using the above assumptions. The algorithm attempts to find features on both images which have similar physical properties. Properties such as the length of the edge, the location for the centroid of the edge in the image, Euler number of the area encircled by an edge, and the solidity of an edge (the amount of area in percentage that the edge will occupy if in a convex polygon that is to contain the entire edge) are all taken in to account. A feature on one image is considered to have a match on the other image if there is a feature which has similar properties to itself on the other image. The properties of the images will not be exactly the same, but they should be similar assuming the assumptions above are met.

The variation allowed for each property are adjusted base on what it is measuring and the accuracy it needs to be and are therefore slight different. For example the location for the centroid and area of the edge is allow to shift by ±15%, because it is expected that the captured feature will translate and deform by a slight amount between the time interval due to movement of the UAV, while Euler number of the area encircled by an edge is allow to vary by one, because the edge detector might accidentally close a curve hence altering the Euler number, but if the variation is more then one then the two feature in question might not be matching feature.

The images on the next page show a work example of different stages involve in the edge matching algorithm working on a set of images as discussed above.

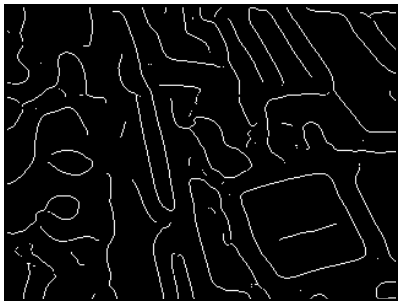
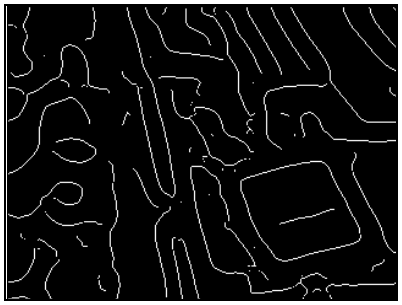
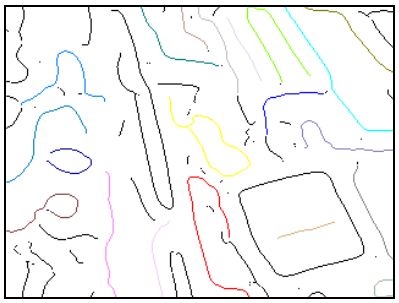
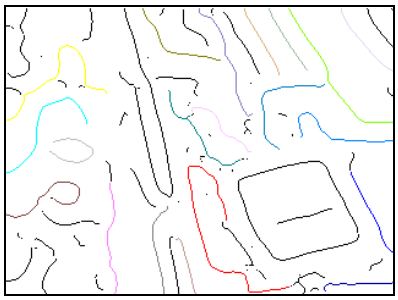
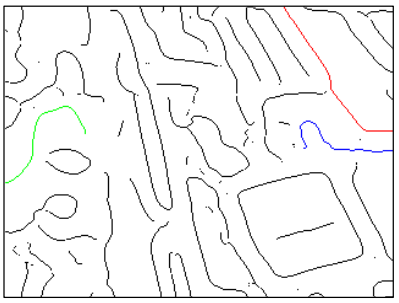
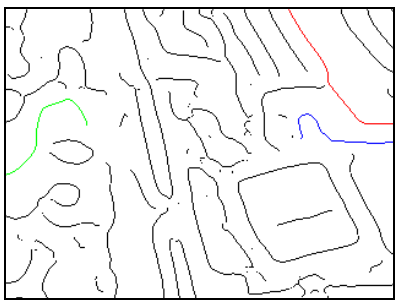
Stages	On board	Extraction of Reference map
Original images		
Images after passing through a edge detection filter		
Images with feature candidates highlighted in different colour		
Identical features on both images as identified by the system. (They are highlighted in the same colour on the images shown)		

Table 1: a work example of different stages involve in the edge matching algorithm working on a set of images as discussed above.

6 Calculating Location of UAV

This process is rather like running the step 3 of the reference map extraction process backward. In this case centroid for a feature is known on both images, the task here is to calculate a translation vector for the location of the UAV when the image from the on board camera is taken, which would explain the drift between the two.

The first step involves establishing where the centroid of the feature concerned is on the world coordinate system. This step is relatively easy as the pixel location for the centroid of the feature in the image which is generated by extracting part of the reference map is known, and the transformation matrix and translation matrix which are used to extract this image are also known. Therefore using the method describe in previous section for calculating the area which is captured by a pixel in the camera, the location of the feature on the reference map can be easily found.

The next step involves calculating the location of the UAV based on the location of the feature captured on the image captured by the on board camera. As physical location for the centroid of this feature in global coordinate is known from previous calculation, and from the flight log the transformation matrix *R* and translation vector *TL* can be calculated, this mean the viewing vector *v'* is also known. Then it becomes a straight forward linear equation to solve. This is shown in the diagram below.

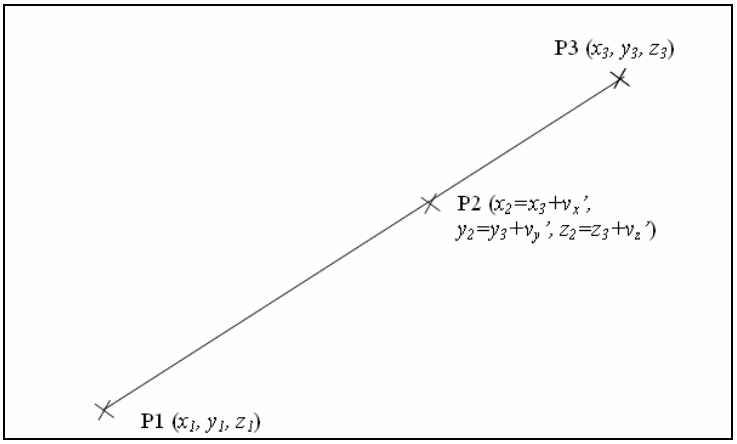


Figure 16: Line passing from the feature to the camera

In the diagram above, the location of P1 is known based on calculation for the location of the feature in global coordinate described above. The value of y_3 which is based on the altitude of the UAV is known, and because the view vector *v'* which is based on the location of the feature on the image captured by the UAV's on board camera and the transformation matrix *R* is known the value for v_x', v_y' and v_z' are also known. The aim is to solve for x_3 and z_3, which is the location of the camera.

Location of can be solved by the following equations:

$$m = \frac{y_3 - y_1}{y_3 + v_y'} = \frac{y_3}{y_3 + v_y'}$$

$$z_3 = \frac{\left(mv_z' + z_1 - mz_1\right)}{(1-m)}$$

$$x_3 = \frac{\left(mv_x' + x_1 - mx_1\right)}{(1-m)}$$

With the values of x_3 and z_3 known, it is a simply matter of converting them to their equivalent longitude and latitude to locate the UAV.

Technically only one feature is needed to calculate the value x_3 and z_3, but in the actual implementation the x_3 and z_3 for three matching features were calculated and average to improve the accuracy of the system.

7 Experimental results and discussion

As stated in the introduction the tested were conducted with a sequence of images generated from using exact location of the UAV on flight log as the flight sequence was found to be unusable. However because GPS location is only available on per second basis in flight log and the precise GPS location is need every 200ms in order to generate the images sequence, the GPS coordinates between valid GPS signals are generated by linearly interpolating between two GPS coordinates.

There are two tests conducted to test the performance of this system. The first test involve using the system to calculate the location of the UAV in next time step for 125 entries which are recorded in the flight log. These 125 entries are basically represents 25 seconds of flight time. This test is kind of like an “open loop” test, aiming to see how with accurate information the system will be able to perform when different data is fed into it. The next test is a "closed loop" test and involves using the result of previous calculation as the input for the next calculation. This is what would happen when GPS signal is lost; the input to the system will depends on the measurement taken by on board sensor as well as the location of the previous time step. Again this test was conducted for 125 entries of the flight log which represent 25 seconds of flight time. The flight log sequences used in both tests were identical.

Result of two tests done on earlier revision of the system where the location of the UAV is calculated on every second of flight time instead of every 200ms is also included. In this test 30 flight log entries were used, each of these entries are spaced in 1 seconds of flight time, this essential mean the system is supplied with the status of the UAV and image taken by on board camera every second, and is expected to calculated the location of the UAV based on location of the UAV in the previous second and status of the UAV in the current time. The starting point for these two tests was about seven seconds earlier in flight time then that of the previous two tests.

7.1 Test results for calculation done every 0.2 second

Results for "open loop" test

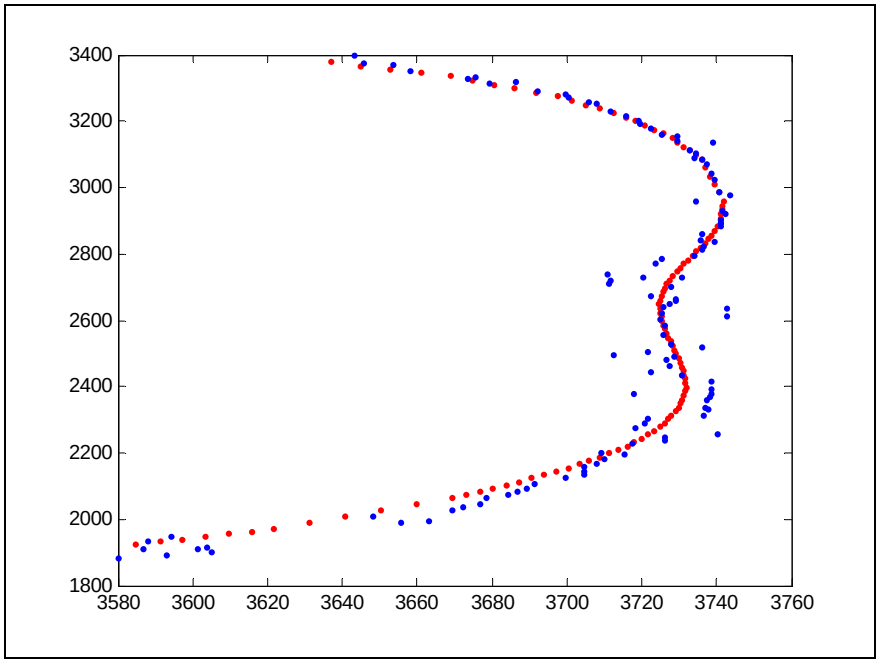


Figure 17: Result of "open loop" test, the blue points are location of the UAV as calcalated by the system, while the red points are location of the UAV which are recored in the flight log

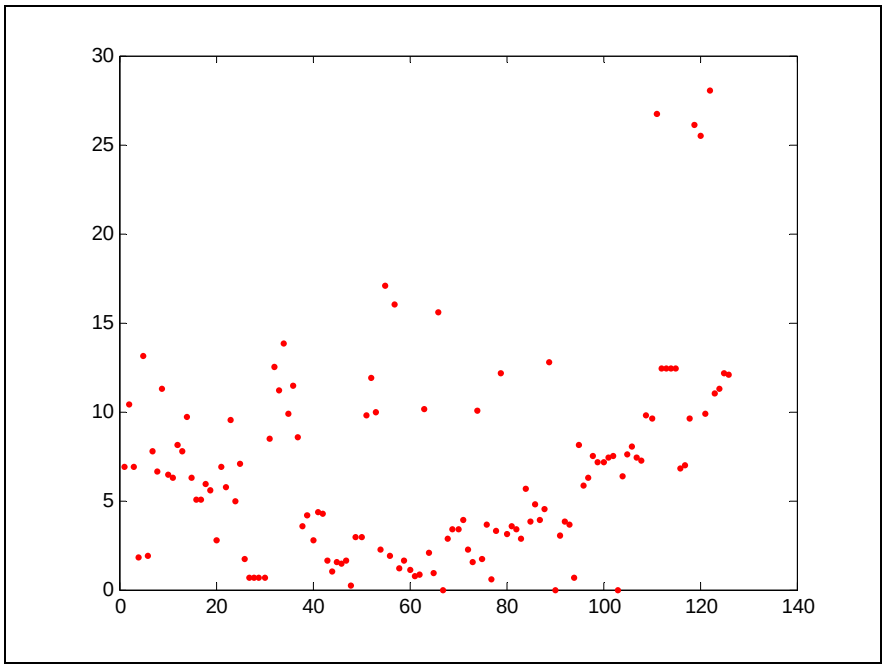


Figure 18: Error of the system, this measures the different in meter between the results calculated by the system and that recorded in the flight log

The mean error in the open loop test is 6.68 m.

Results for "close loop" test

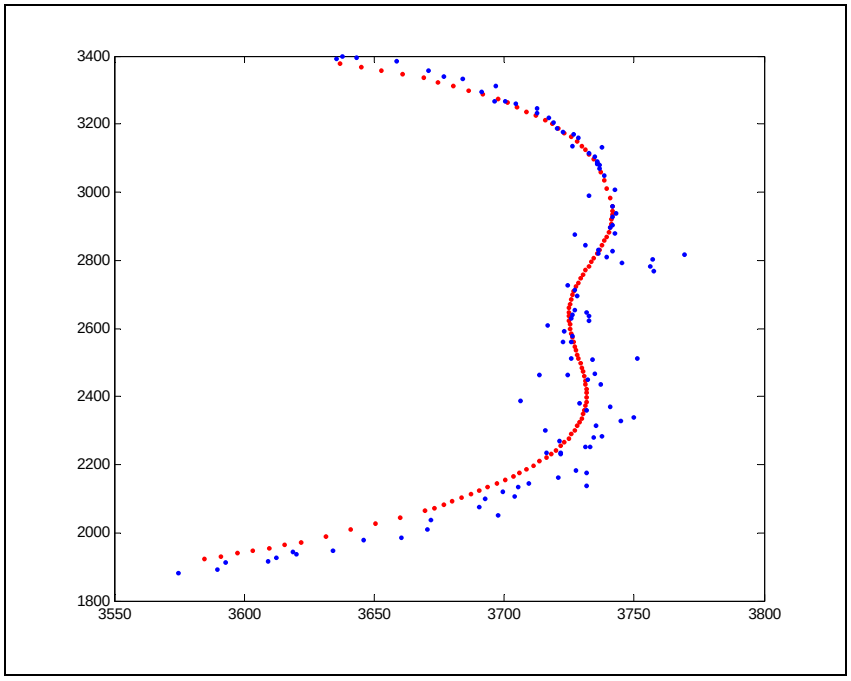


Figure 19: Result of "open loop" test, the blue points are location of the UAV as calcalated by the system, while the red points are location of the UAV which are recored in the flight log

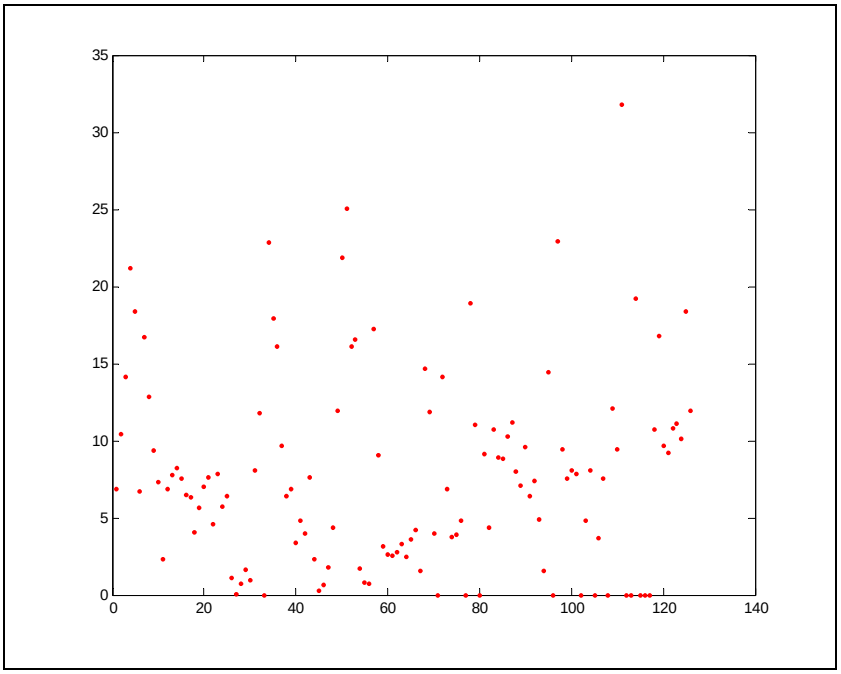


Figure 20: Error of the system, this measures the different in meter between the results calculated by the system and that recorded in the flight log

The mean error in the close loop test is 7.75 m.

7.2 Test results for calculation done every 1 second

Results for "open loop" test

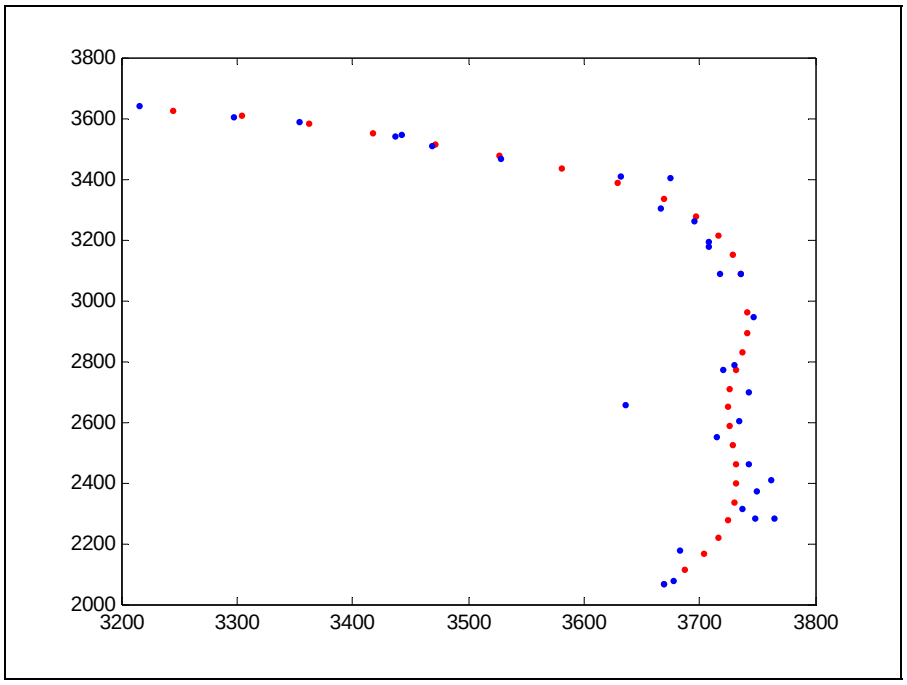


Figure 21: Result of "open loop" test, the blue points are location of the UAV as calcalated by the system, while the red points are location of the UAV which are recored in the flight log

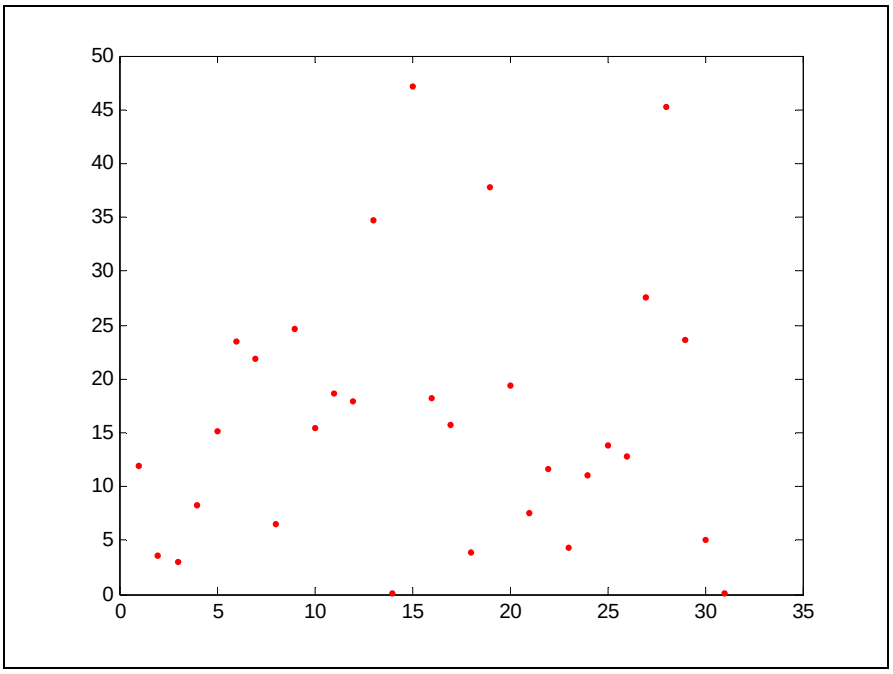


Figure 22: Error of the system, this measures the different in meter between the results calculated by the system and that recorded in the flight log

The mean error in the open loop test is 16.4348m.

Result of "close loop" test:

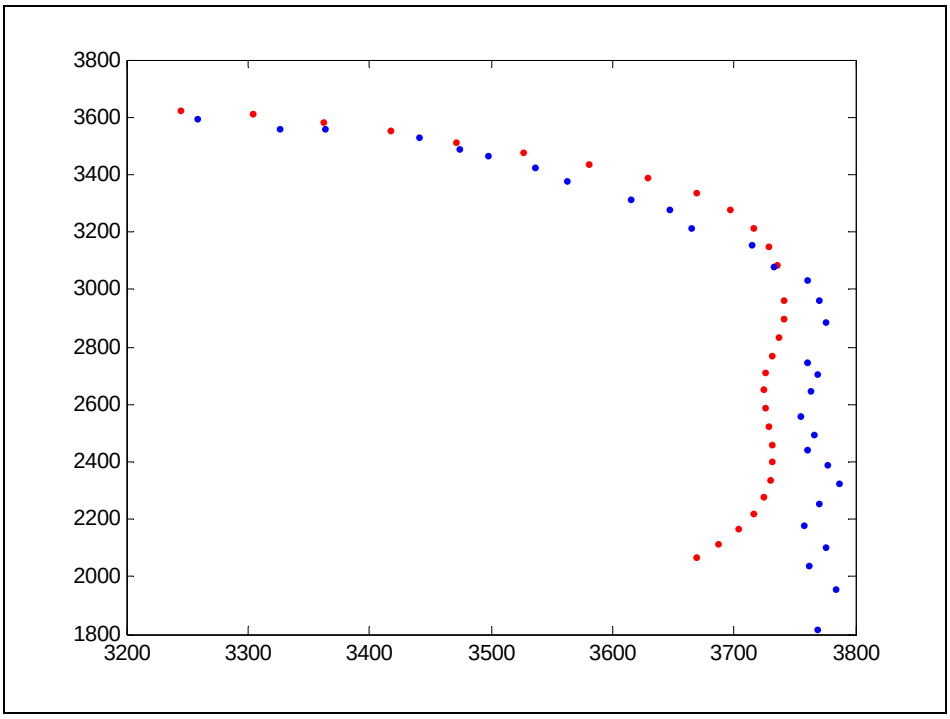


Figure 23: Result of "open loop" test, the blue points are location of the UAV as calcalated by the system, while the red points are location of the UAV which are recored in the flight log

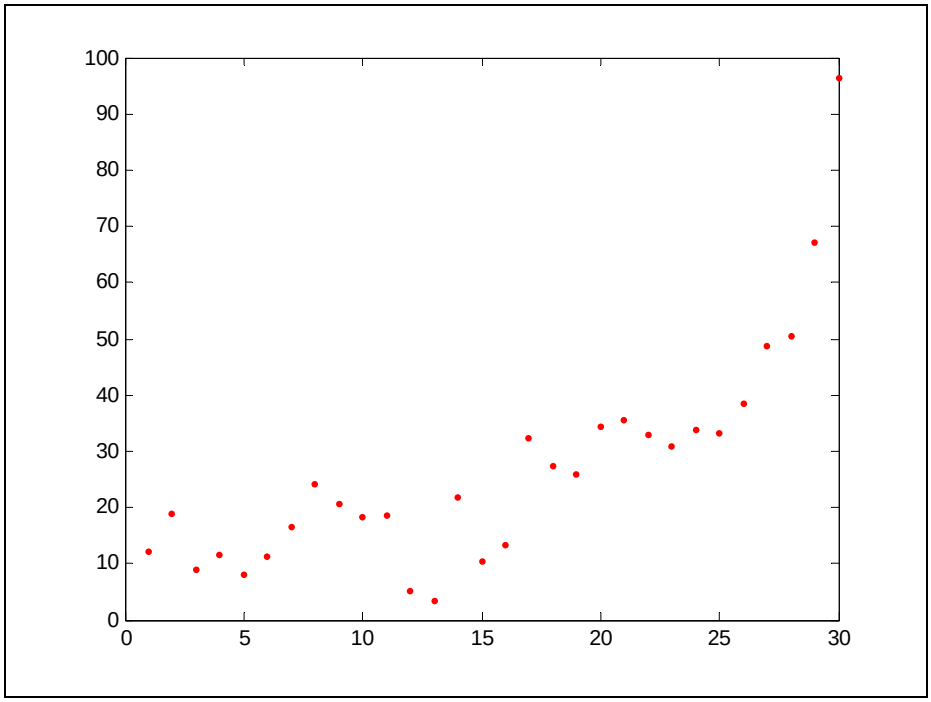


Figure 24: Error of the system in "close loop", this measures the different in meter between the results calculated by the system and that recorded in the flight log

The mean error in the close loop test is 26.9895m.

7.3 Discussion

The "open loop" test results for the system running every one second and 0.2 second had demonstrated that the system with a few exceptions is capable of calculating the location of the UAV with error of less then 25 meters for the test sequences. The system is able to on average calculate the location of the UAV with an error of 16.4m when it is operating with accurate location information in the previous input which is 1 second before the current point, and with an error of 6.68m when it is operating with accurate information which is 0.2 second before the current point.

There is always likely to be some error due to inaccuracy in the raw data such as the measurement of location by the on board GPS system, linear estimation of longitude and latitude location, and inaccuracy in measurement for the corners of the map. So an average error of 6.68m and 16.4 m is quite acceptable, and considering most commercial GPS system has an error range of 20-30 meters, this system is probably at least on par with commercial system. The large errors in selected points are likely to be due to the simple feature matching algorithm used to determine if two features are matching. It seem that at some points there are too many features in the filtered images and it caused the matching algorithm to register two features which are mismatched to be matching features. This causes the system to treats the features as shifted by a large amount thus causing an error in the calculation for the location of the UAV. It is shown that by reducing the time span between each calculation the average errors and the frequency of points with large errors decrease, which mean when the system is operating at 0.2 second per calculation the matching algorithm is performing much better and frequency of such mismatch decrease. This is because the features are no longer shifted as much and the stresses place on the matching algorithm has lessen, but still at some points the matching algorithm do fail, and this can only be fixed by implementing a more sophisticated matching algorithm.

The "close loop" test results also show some promising results with the system being able to track the actual location of the UAV with an error of less then 20 meters for most of the duration of the test when it is operating at 0.2 second per calculation, and for less then 30 meters for the first 20 seconds when it is operating at 1 second per calculation. Average error for 0.2 second case is 7.75m while the 1 second case is 27m. Those two figures are both within the limits of the accuracy of a commercial unit and are therefore quite acceptable. But what is worrying about the result is the fact that the error of the system seems to be increasing as more points are processed. This means in the current state this system is only valid as a navigation tools for about half a minute, beyond that it is not like that it will be able to provide accurate information about the location of the UAV.

To a certain degree this is to be expected, as each result will inevitably contain within it some amount of error, therefore using the previous result to calculate the location of the current time step is effectively like adding a random translation factor to the input thus causing stress on the feature matching algorithm. As the amount of input error accumulate with more points process, the output also shows an increase amount of error, until at certain points the system compute a totally unreasonable result which knock out calculation causing the whole algorithm to fail, and at the causing the break down of the tracking mechanism because the feature matching mechanism will by then constantly fail to find matching features or will return mismatching features as results.

Again a more sophisticated feature matching algorithm is needed to fix this problem. At the moment it is too rigid and not accommodating, more about this will be discussed in section on “Future Work” in this report. The result can also be

improved by applying an algorithm to dynamically adjust the threshold of the parameters the for Canny edge detection algorithm. As even though the algorithm does a rather good job to filter out major edges and keeping them intact, there are still instances where edges are broken up in places where they should not be, and in some cases it had caused problem of mismatching edges. This is likely to be due to problems with threshold values and value of σ selected, and if they are slightly adjusted based on content of the image, the matching algorithm should work better. There might be also needs to find better parameters to match two features instead of the simple one listed above.

The time it takes to process each log entry is roughly 8 second. As expect the most time consuming component is the extraction of reference map using known information. This makes the method describe here not entirely suitable for real time application. But with improvement of this subsystem the method described in this document should be applicable in real time as well.

In summary the method of feature based navigation described in this document should be a viable alternate to GPS in real time, provided that the time interval between input points are shorter then what it currently is, this is shown by its ability to calculate a the correct location of the UAV with a tolerable error in the "open loop" test and optimisation are made in the extraction of reference map based on known information. But at its current state the system is only able to provide accurate location information for 20 seconds at maximum.

8 Future work

The are a two major issues in this system that are identified in the discussion section which will need to be rectify before this system can be used as a viable alternatives for navigation of a UAV in real time. They are the issues with feature matching and the issue of time needed to extract part of the reference map, possible solutions for these two issues will be proposed and discussed here.

Real time operation

There are two possible solutions to this problem, the first involve implementing the current algorithm more efficiently. Arguably the current algorithm just involves repeated multiplication of a floating point vector to a floating point matrix, then follow by an operation which involve a few multiplication, addition and division to solve a linear equation, and lastly operation to access an array which will write the result to another. These are all very repetitive operations, and can be optimised by making all of the arithmetic operation integer operations by scaling them and possibly implementing the whole operation in a FPGA. This is not as flexible as a software solution, but it should not be very often that parameters such as viewing angles and resolution of the camera use will change, so it is a possible solution.

The second solution is instead of calculating the exact location of the ground captured by a pixel, just calculate the location which the four corners of the image will be capturing and linearly interpolate the location captured by remaining pixels. This way instead of performing the current calculation 320×240 times, it only needed to be performed four times and a simpler operation will linearly interpolate the rest of the pixels. This should improve the performance significantly if this is done right.

But at the end with Feature Based Navigation there is a need to generate an image for internal purpose by extracting it from a reference map. This process is always likely to be expensive, as there are at least $320 \times 240 \times 3$ assignments (or 320×240 if the reference map is first converted to grey level image) required to generate this image. This means Feature Based Navigation might never be fast enough for real time application.

Feature matching

As stated in the discussion section, one of the major issues is the mismatching of edges due to inaccuracy. One way to solve this problem might be to change the matching method from the idea that two features are matching if they pass a list of criteria, to two features are matching if they are found to be closest matching by a scoring system and at the same time pass a certain criteria by having for example scoring more then certain amount of points in a scoring system design to measure the similarity between two edges.

At the moment two features are considered matching if they pass a list of criteria. For example they must be within a certain distance from each other, if they are then they must be of similar length, and if those two criteria are passed then if they have Euler numbers which are offset by less then one, and if all these criteria are passed then these two features must be the same features. Instead using this system, maybe a scoring system which allow different emphasis to be to

be place on different criteria might be more appropriate, then two features are consider the same if they have similarity which result in a score higher then a certain threshold and at the same time are pairs which results in highest matching scores will be used as matching feature.

Using this method the amount of mismatching should decrease thus making the result must more accurate.

Once these two improvements are made to the system, the method of Feature Based Navigation as discussed in this report should be a viable alternative to GPS for navigation of a UAV.

9 Conclusion

The aim of this project is to provide alternative navigation method to an UAV when GPS is unavailable. A few alternatives methods were explored and due to limitation of equipments on the UAV and available resource, a method called Feature Based Navigation is purposed and tested. This method works by comparing location of features on two images, one of such images is taken by the on board camera of the UAV during flight, while the other images is extracted from a reference map base on attribute of the at the instance of time when the image is captured by the on board camera and the location of the UAV at previous time step. The location of the UAV is calculated based on locations of identical features that are in on the two images.

As seen from this report, Feature Based Navigation is a viable navigation method which can be used to replace GPS in a UAV. While the current implementation as shown in this report is unsuitable for real time application and because of issues with the feature matching algorithm, making it unable to provide accurate navigation information for a long duration of time, the experimental results included in this report had show that there are promising results which prove that the method described is able to provide location of the UAV to within the standard of commercial unit in “open loop” and had successfully track the location of the UAV in close loop for at least 20 seconds without significant errors. Once the improvement suggested in the “Future Work” section of this report are implemented the method of Feature Based Navigation should be a viable alternative to GPS in providing a UAV with periodical update to its location in flight.

10 References

[1] *Global Positioning System*, http://en.wikipedia.org/wiki/GPS, 2005

[2] Andrzej Michalski, *The Accuracy of Global Positioning Systems*, IEEE Instrumentation & Measurement Magazine, March 2004, p. 56 – 61

[3] *Celestial Navigation*, http://en.wikipedia.org/wiki/Celestial_navigation, 2005

[4] F. Pappalardi, S.J. Dunham, M.E. LeBlang, T.E. Jones, J. Bangert and Kaplan, *Alternative to GPS*, Proceeding of OCEANS 97 Conference, 1997.

[5] Andy Shaw, Dave Barnes and Phil Summers, *Landmark recognition for localization and navigation of aerial vehicles,* International Conference on Intelligent Robots and Systems 2003, 2003

[6] Raj Talluri, *Mobile Robot Self-Location Using Model-Image Feature Correspondence*, IEEE Transactions on Robotics and Automation, 1996

[7] Martin John Baker, *Maths - Euler Angles*, http://www.euclideanspace.com/maths/geometry/rotations/euler/, 2005

[8] Bernard Ekin, *Dynamics of flight: Stability and Control,* John Wiley & Sons, Inc., 1959

[9] Priya Asokarathinam, *Math Basic – Line Equation,* http://www.cs.fit.edu/wds/classes/cse5255/thesis/lineEqn/lineEqn.html, 2005

[10] Rafael C. Gonzalez, Richard E. Woods and Steven L. Eddins, *Digital Images Processing Using MATLAB,* Peason Education Asia Limited, 2004

[11] Donovan Parks and Jean-Philippe Gravel, *Coroner Detection - Harris/Plessey Operator,* http://www.cim.mcgill.ca/~dparks/harris.htm, 2005

[12] Donovan Parks and Jean-Philippe Gravel, *Coroner Detection - Trajkovic Operator (8-Neighbours),* http://www.cim.mcgill.ca/~dparks/trajkovic8.htm, 2005

[13] Bob Fisher, Simon Perkins, Ashley Walker and Erik Wolfart, *Feature Detector - Canny Edge Detector,* http://www.cee.hw.ac.uk/hipr/html/canny.html, 1994

[14] Bob Fisher, Simon Perkins, Ashley Walker and Erik Wolfart, *Spatial Filter – Gaussian Smoothing,,* http://www.cee.hw.ac.uk/hipr/html/gsmooth.html, 1994

Reference map used in this project is provided by Qasco Victoria Pty Ltd.

11 Appendices

11.1 Appendix A – List of program files

The following is a list of files which are part of this program, and description of what each of them do:

File name	File Type	Description
rotate_z.m	MATLAB function	Function prototype: *mat = rotate_z(a)* This function return a rotation matrix *mat* which will rotate a 3-D vector along the *z* axis by a radian *a*
rotate_x.m	MATLAB function	Function prototype: *mat = rotate_x(a)* This function return a rotation matrix *mat* which will rotate a 3-D vector along the *x* axis by a radian *a*
rotate_y.m	MATLAB function	Function prototype: *mat = rotate_y(a)* This function return a rotation matrix *mat* which will rotate a 3-D vector along the *y* axis by a radian *a*
initialize.m	MATLAB script	Contain definition of constants and data structure used by this system. Must be called before the system is to be used.
air_ptxz.m	MATLAB function	Function prototype: air = air_ptrx(gnd, row, col, height) Calculate a 3-D vector which specific the location of the camera, based on mathematics specific by the section "Calculcate UAV location" of the report. Return 'air' is a 3-D vector. Parameter 'gnd' specific the location of point P1, 'row' and 'col' specific the location of the point in an image, and 'height' specific the altitude of the UAV.
find_ground_row_col.m	MATLAB function	Function prototype: gnd = find_ground_row_col(row, col, TL) Calculate the point on the reference map captured by a pixel on an image with location as specific by parameters 'row' and 'col'.

		Parameter 'TL' is a 3-D vector which specific the current location of the UAV
find_match.m	MATLAB function	Function prototype: output = find_match(data_ref, labelref, data_other, labelother, points) Based two set of data output by the function 'largestnObj_edges.m' when it operate on the extraction of reference map and the on board camera image. The 'output' give the data of three matching features on those two images.
gnd_ptxz.m	MATLAB function	Function prototype: gnd = gnd_ptx(cam, fra) Calculate the location where a pixel is supposes to be on the ground. Mathematics of this function is specific in the section "Reference Map Extraction – Implementation" in this report. The parameter 'cam' is a 3-D vector which specific the location of the camera, 'fra' is a 3-D vectoric which specific the location of the view vector, and the output 'gnd' is also a 3-D vector which specific the location which a pixel is suppose to capture on the reference map.
largestnObj_edge.m	MATLAB function	Function prototype: [output labeledRef threshold] = largestnObj_mod(ref_im, threshold, n) Find the largest number of objects (number of edges, number specific by parameter 'n') within an image and return the data about the objects found in 'output' with an images which these features labeled in 'labeledRef'. The parameter 'ref_im' is the image to operate on, while the parameter threshold is a 2-D array specifying the threshold value T1 and T2 to be used by the canny operator.
largestnObj.m	MATLAB function	Function prototype: [output labeledRef] = largestnObj_mod(ref_im, , n) Earlier attempts at features candidate extraction. Find the largest number of objects (largest group of pixels, number specific by parameter 'n') within an image and return the data about the objects found in 'output' with an images which these features labeled in 'labeledRef'. The parameter 'ref_im' is the image to operate on.
main_closeloop200.m	MATLAB script	Script to start the entire system. This script is for 'close loop' test of the system with UAV location

		interpolated from the flight log to test the system with a data input of 0.2sec per calculation instead of every 1 second
main_openloop200.m	MATLAB script	Script to start the entire system. This script is for 'open loop' test of the system with UAV location interpolated from the flight log to test the system with a data input of 0.2sec per calculation instead of every 1 second
lat_long2tl.m	MATLAB function	Function prototype: TL = lat_long2tl(lat_long) This function takes a 2-D row vector 'lat_long', with first parameter specifying the latitude and second parameter specifying the longitude of the UAV. It then calculate a 3-D vector 'TL' which specific the *x* and z location of the UAV in global coordinate system.
match_critial.m	MATLAB function	Function prototype: condition = match_critial(datapoint1, datapoint2) This function is used by the function ' find_match' to determine if two features from two images are the same feature. If it is 'condition' = true, else 'condition' = false
read_log.m	MATLAB function	Function prototype: read_log This function read information from an unformatted flight log file outputted by the UAV into a global variable which can then be accessed by the system. It also performs such task as data conversion to put the data into useable form.
refmap_extract.m	MATLAB function	Function prototype: refmap_extract(LEN, mode, lat, long) Extract part of the reference map based on the latitude and longitude location supplied and the information of the UAV specified in the flight log entry specify by the number LEN. The 'mode' parameter specifics which global variable the result should be sent to. If 'mode' = BY_LAT_LONG, then the result will be send to global variable 'refmap_ext', else it would be send to global variable 'avi_image'. This enable the same function to generate both the extraction of reference map as needed and the sequence of images which will represent the images taken by the UAV camera during flight. The function will alter the global variables 'cur_TL' and 'current_R' which will record the translation vector used and the transformation

		matrix used respectively. The detail mathematics behind this function is the section "Reference Map Extraction – Implementation" of this report.
tl2lat_long.m	MATLAB function	Function prototype: lat_long = tl2lat_long(TL) This function calculates the latitude and longitude location of a point in the global coordinate system as specific by the parameter TL. This function return a row vector containing with it the latitude follows by the longitude of the point.
flight_video.avi	Video file	Contain a part of the sequence of video captured by the UAV's on board camera. Because of uncertainty in viewing angle of the camera when the video is taken, it is not used in this project, but this file must be present as it still specific the dimension of the images taken by the on board camera
log_test.txt	Flight log	A text file which contain a part of the original flight log. This file is used to test the system.
refmap.jpg	Jpg images	This file is the images file which contains the reference map used by this system.

Notes

- All of the files listed above must be in a folder within the path of MATLAB for this system to work.
- The system can be started by calling either the script file 'main_closeloop200.m' or 'main_openloop200.m'.
- MATLAB files listed above will be listed later in this report.

11.2 Appendix B - Listing of Source code

This section list all of the MATLAB source code used in this project.

rotate_z.m

```
function mat = rotate_z(a)
    mat = [cos(a) -sin(a) 0;
        sin(a) cos(a) 0;
        0 0 1];
end
```

rotate_x.m

```
function mat = rotate_x(a)
    mat = [1 0 0;
        0 cos(a) -sin(a);
        0 sin(a) cos(a)];
end
```

rotate_y.m

```
function mat = rotate_y(a)
    mat = [cos(a) 0 sin(a);
        0 1 0;
        -sin(a) 0 cos(a)];
end
```

initialize.m

```
%Initialization script.  Run before running program.

status = 'Start initialzation';
disp(status)

global log_file_name;
global log_file_section;  %define the revelant section of the log file to
                          %read and the order to read them
global log_entry_scale;
global log_file_delim;
global flight_log;        %global variable containing flight log of avi
global log_offset;     %the first vaild log entry

global refmap_scale;
global refmap
global refmap_file_name;
global refmap_size;
global refmap_TL;
global refmap_TR;
global refmap_BL;
global refmap_BR;
global refmap_delta_lat;
```

```
global refmap_delta_long;
global pitch_cam;
global refmap_ext;
global refmap_ext_ok ;

global avi_file_name;
global avi_info;
global avi_image;
global avi_ext_ok;

%virtual frame setting
global vert_angle;
global horiz_angle;
global virt_dist_from_cam;
global virt_dist_vert;
global virt_dist_horiz;
global current_R;       %Current rotational matrix
global virt_frame;

%Extraction from refmap mode setting
global BY_LOG;
global BY_LAT_LONG;
BY_LOG = 1;
BY_LAT_LONG =2;

%Results
%global cur_lat_long;           %Current latitude and logtitude as calculated
by system
global cur_TL;
global crusing_speed;
global speed_factor;
global correction_factor;


%Use to filter out results which are too far out.
crusing_speed = 20;
speed_factor = 10;
correction_factor = 0.0004;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%VIDEO FILE SETTING%%%%%%%%%%%%%%%%%%%%%%%%%%%
avi_file_name = 'flight_video.avi';
avi_info = aviinfo(avi_file_name);
avi_ext_ok = false;
%Allocate and initialize image array
avi_image( avi_info.Height, avi_info.Width,3) = uint8(0);
avi_image = uint8(avi_image);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%SCALE FACTOR FOR GLOBAL MAP%%%%%%%%%%%%%%%%%%%%%%%%
refmap_scale = 0.36;
refmap_file_name = 'refmap.jpg';
refmap = imread(refmap_file_name);
refmap_size = size(refmap);     %row * col * 3

%GPS coordinates for corners of map
%first coordinate is latitude, follow by longitude
refmap_TL = [-37.8726667 145.1992667];   %Top left corner
refmap_TR = [-37.87271667 145.2159833];  %Top right corner
refmap_BL = [-37.88888336 145.1993434];  %Bottom left corner
refmap_BR = [-37.88893333 145.2161];     %Bottom right corner
```

```
%Change in longitude and latitude of the refmap map, assume the change in
%latitude and longtitude is roughly in straight line (for refmap used in
%this program error is approximate 0.4%)
refmap_delta_lat = abs((abs(refmap_TL(1))-abs(refmap_BL(1)))+(abs(refmap_TR(1))-
abs(refmap_BR(1))))/2;
refmap_delta_long = abs((abs(refmap_TL(2))-
abs(refmap_TR(2)))+(abs(refmap_BL(2))-abs(refmap_BR(2))))/2;

%Pitch angle correction factor of camera, this compensate for the fact that
%the camera is not mounted vertically down towards the ground.  This
%specific the pitch angle needed if the virtual viewing frame is under the
%camera.
pitch_cam = deg2rad(57);

%Allocating an array to store image generate by extract_refmap.m
refmap_ext( avi_info.Height, avi_info.Width,3) = uint8(0);
refmap_ext = uint8(refmap_ext);

refmap_ext_ok = false;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%LOG FILE SETTING%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
log_file_name = 'log_test.txt';
%Order of data in global variable log:
%Current Pitch(rad), Current Roll(rad), Current Yaw(rad), Current
%Altitude(m), Current Speed(m/s), Heading(rad), x_acc(m/s2), y_dot(m/s2)
%pitch dot(deg/s), roll dot(deg/s), yaw dot(deg/s), GPS_POS_E(deg),
%GPS_POS_N(deg), GPS_POS_U(m), GPS_VEL_U(m/s), gpsSpsed(m/s), LocationE(m),
%LocationN(m)
log_file_section = [1 2 37 29 27 6 35 4 43 10 11 25 26 33 34 10 7 8 ];
%Scale from raw data in log file to format as specified above
log_entry_scale = [1/1024 1/1024 1/1024 -0.0381 0.3048 1/10*pi/180 9.8/100 ...
    9.8/100 1/375 1/375 1/375 1 1 1 0.3048 0.3048 0.3048 0.3048];
log_file_delim = '\t';      %specific the demlimiter used in CSV of flight log
read_log;    %initialize function to read the flight log into global variable
log_offset = 66;    %first log entry which is vaild
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%virtual frame setting%%%%%%%%%%%%%%%%%%%%%%%%%%
%vert_angle = atan(2760/2/3830);
%horiz_angle = atan(2745/2/2666);
vert_angle = deg2rad(15);
horiz_angle = deg2rad(19.5);
%Virtual distance from camera to virtual viewing frame in meters
virt_dist_from_cam = 1*refmap_scale;
%Vertical height of virtual viewing frame
virt_dist_vert= virt_dist_from_cam * tan(vert_angle);
%Horizontal weight of virtual viewing frame
virt_dist_horiz= virt_dist_from_cam * tan(horiz_angle);

%Initializing initial (not transformed) rotational frame
for y = 1:avi_info.Height
    for x = 1:avi_info.Width
        virt_frame(y,x,1) = (virt_dist_vert - virt_dist_vert *2 /avi_info.Height
* (y-1));
        virt_frame(y,x,2) = -1*virt_dist_from_cam;
        virt_frame(y,x,3) = -1*virt_dist_horiz + virt_dist_horiz *2 /
avi_info.Width *(x-1);
    end
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

status = 'Initialization complete';
disp(status)
```

air_ptxz.m

```
function air = air_ptxz(gnd, row, col, height)
    %Declearing global variables used
    global current_R;       %Current rotational matrix
    global virt_frame;

    pti(1,1) = virt_frame(row, col, 1);
    pti(2,1) = virt_frame(row, col, 2);
    pti(3,1) = virt_frame(row, col, 3);
    pti = current_R * pti;
    m = height / (pti(2)+height);
    %specifing y
    air(2,1) = height;
    %specifing x
    air(1,1) = (m*pti(1) + gnd(1,1) - m*gnd(1,1))/(1-m);
    %specifing z
    air(3,1) = (m*pti(3) + gnd(3,1) - m*gnd(3,1))/(1-m);
end
```

find_gound_row_col.m

```
%Use linear equation to find out where this pixel hit the
%ground.  This function include operation which rotate the
%virtual view frame and translating it in place.  It then
%calculate the place where a specific pixel (given by row
%and col) in the view screen should hit the ground

function gnd = find_ground_row_col(row, col, TL)
    %Declearing global variables used in this function
    global refmap_scale;
    global current_R;       %Current rotational matrix
    global virt_frame;
    global cur_TL;

    pt(3,1) = virt_frame(row,col,3);
    pt(2,1) = virt_frame(row,col,2);
    pt(1,1) = virt_frame(row,col,1);

    %Rotating the view vector and translating it in place so that
    %it is pointing from the point where the camera is suppose to
    %be to where the view area should be
    pt = current_R*pt;
    if (TL ==0)
        pt = pt + cur_TL;
        gnd = gnd_ptxz(cur_TL, pt);
    else
        pt = pt + TL;
        gnd = gnd_ptxz(TL, pt);
    end
```

```
    %Use linear equation to find out where this pixel hit the
    %ground

end
```

find_match.m

```
function output = find_match(data_ref, labelref, data_other, labelother, points)
    %The data variable contain data on each of the objects found
    %and the label variable contain a array which specific which pixel
    %belong to which object.  The number of entry in each of the data
    %varilabe is specific by the parameters points.
    %The variables 'data_ref' and 'labelref' contain data for one set of
    %image, and the variables 'data_other' and 'labelothter' contain
    %information related to another images
    %Parameter 'points' specific the number of points in each of the
    %data_ref and data_other parameters.

    %Number of data points each data varible in the parameters contained
    n = points;
    pairs_needed = 3;
    col_no_label = 4;
    found = 1;
    %Allocating variable 'pairs', varible 'pairs' contain within it:
    %1st col: The label for the refim in the pair
    %2nd col: The label for the otherim in the pair
    %3rd col: The nth largest object this object is in the refim
    %4th col: The nth largest object this object is in the otherim
    pairs(pairs_needed,4) = 0;

    for x = 1:1:n
        for y = 1:1:n
            if(match_critial(data_ref(x,:), data_other(y,:)))
                %Pair found based on matching criterial, record the data
                pairs(found, 1) = data_ref(x, col_no_label);
                pairs(found, 2) = data_other(y, col_no_label);
                pairs(found, 3) = x;
                pairs(found, 4) = y;
                found = found +1;

                if(found > pairs_needed)
                    %Drawing function
                    numregion_ref = max(labelref(:));
                    numregion_other = max(labelother(:));
                    [row,col] = size(labelref);
                    RGB_ref(col, row, 3) = uint8(0);
                    RGB_other(col, row, 3) = uint8(0);

                    map_ref(numregion_ref, 3)  = uint8(0);
                    map_other(numregion_other, 3)  = uint8(0);

                    for i = 1:1:pairs_needed
                        %generate map for label2rgb function calls
                        map_ref(pairs(i, 1), i) = 1;
                        map_other(pairs(i, 2), i) = 1;

                    end
                    %RGB_ref = label2rgb(labelref,map_ref, 'w') ;
                    %RGB_other = label2rgb(labelother,map_other, 'w') ;
```

```
                    %show(RGB_ref);
                    %show(RGB_other);
                    output = pairs;
                    return;
                end
                break;
            end
        end
    end
    if(found <= pairs_needed)
        output = false ;
        return;
    end

end
```

gnd_ptxz.m

```
%Calculate the location which a pixel is suppose to be on the ground, the
%mathematics is specific in the report.
%Parameter 'cam' 3-D vector which specific the location of the camera,
%'fra' is a 3-D vector which specific the location of the view vector, the
%output 'gnd' is a 3-D vector which specific the location which the pixel
%will be capturing on the reference map (in global coordinate system)

function [gnd] = gnd_ptxz(cam, fra)
    m = (0 - cam(2,1))/(fra(2,1) - cam(2,1));
    gnd(1,1) = m*(fra(1,1) - cam(1,1)) + cam(1,1);
    gnd(3,1) = m*(fra(3,1) - cam(3,1)) + cam(3,1);
end
```

largestnObj_edge.m

```
%Return the largest n objects area and radian location

function [output labeledRef threshold]= largestnObj_mod(ref_im, threshold,n)

    %Converting input image to gray scale (intensity) image
    ref_g = rgb2gray(ref_im);
    %Using Canny filter, let system decide threshold, but set standard
    %deviation of Gaussian filter to larger then default value.  This
    %should increase the bluring effect and reduce the noise present in the
    %photo, hence also reducing number of candidites.
    %Need to fine tune this number as it affect the finess of the abilty of
    %edge detector to detect edges.
    %Need to find tune the blurring factor

    [bwref, threshold]= edge(ref_g, 'canny',  threshold, 5);

       % bwref = edge(ref_g, 'sobel');


    %show(bwref)
    %Grouping near by pixels together.  The connection is specific as 8
    %connected meaning that a neighbour is any of the 8 pixels surrounding
    %a pixel.  The variable labeledRef return an image with the different
    %labeled area specificing different object the system pick up, the
    %numObjectsRef is the number of objects the system pick up.
```

```
    [labeledRef, numObjectsRef] = bwlabel(bwref, 8);
    %dataRef contain the data of each object identified by the system in
    %the previous command
    dataRef = regionprops(labeledRef, 'Area', 'Centroid',
'EulerNumber','Solidity');

    %Allocating array which will contain the sorted items
    refArray(numObjectsRef, 2) = 0;
    for i = 1:1:numObjectsRef
        refArray(i, 1) = dataRef(i).Area;
        refArray(i, 2) = i;         %contain which index in the array this area
corresponds to, ie label on the map
    end
    refArray = sortrows(refArray,1);   %Sort by area assending order ie from
small to large
    output(n,3) = 0;

    [row, col] = size(labeledRef);
    for i = 1:1:n
        %item of largest area are in the bottom of array
        %list the largest n item ignoring the first 3 largest item, as they
        %are likely that of the border of the image.
        n_ref = refArray(numObjectsRef-i-1, 2);
        row_coordinate = dataRef(n_ref).Centroid(1);
        col_coordinate = dataRef(n_ref).Centroid(2);

        %Filling the data table
        %1st column of table specific the area object
        %2nd and 3rd column specific the coordinate for the centroid of the
        %object, col 2 specific row, col 3 specific col
        %Coordinate in row and col format as used in array.  i.e. row count
        %increase from top to bottom and column count increase from left to
        %right
        %4th column specific the label which specific the object in the
        %labeledRef created.
        output(i, 1) = dataRef(n_ref).Area;
        output(i, 2) = row_coordinate;
        output(i, 3) = col_coordinate;
        output(i, 4) = n_ref;
        output(i, 5) = dataRef(n_ref).EulerNumber;
        output(i, 6) = dataRef(n_ref).Solidity;
    end

    RGB(col, row, 3) = uint8(0);


    map(numObjectsRef, 3)  = 0;
baseColourMap = [255 0 0; 255 255 0; 128 255 0; 0 255 255; 0 128 255; 0 128 192;
128 128 192;...
    255 128 255; 128 64 64; 0 128 128; 128 128 0; 128 128 128; 192 192 192; 0 0
255; 0 0 160; ...
    249 206 255; 223 219 242; 222 177 118;191 154 149; 155 185 157; 121 219
187  ];
baseColourMap = baseColourMap / 256;
    %Generate colour map
    for i = 1:1:n

        map(output(i, 4), 1) = baseColourMap(i, 1);
        map(output(i, 4), 2) = baseColourMap(i, 2);
        map(output(i, 4), 3) = baseColourMap(i, 3);
    end
```

```
    %RGB = label2rgb(labeledRef,map, 'w') ;

    %show(RGB)
end
```

largestnObj.m

```
%Return the largest n objects area and radian location

function output = largestnObj(ref_im, n)

    ref_g = rgb2gray(ref_im);

    %Code for double medium filter both images
 ref_filt = medfilt2(medfilt2(ref_g));


    %code for double mean filter both images
    %MF = 1/9 * [1 1 1; 1 1 1; 1 1 1]        %Mean filter for 9 block
    %ref_filt = imfilter(imfilter(ref_g, MF), MF);

    level = graythresh(ref_filt);
    bwref = im2bw(ref_filt, level);
    show(bwref)
    [labeledRef, numObjectsRef] = bwlabel(bwref, 8);
    dataRef = regionprops(labeledRef, 'Area', 'Centroid');

    %Allocating array
    refArray(numObjectsRef, 2) = 0;

    for i = 1:1:numObjectsRef
        refArray(i, 1) = dataRef(i).Area;
        refArray(i, 2) = i;
    end
    refArray = sortrows(refArray,1);   %Sort by assending order ie from small to
large
    output(n,3) = 0;
    numObjectsRef
 [col, row] = size(labeledRef);
    for i = 1:1:n
        n_ref = refArray(numObjectsRef+1-i, 2);
        [TH,R] = CART2POL(dataRef(n_ref).Centroid(1),
dataRef(n_ref).Centroid(2)) ;
        output(i, 1) = dataRef(n_ref).Area / (row * col);
        output(i, 2) = TH;
        output(i, 3) = R;
        output(i, 4) = n_ref;
    end

    RGB(col, row, 3) = uint8(0);

    numregion = max(labeledRef(:))
    map(numregion, 3)  = uint8(0);
    g = 0;
    for i = 1:1:3
        g = g +1;
        map(output(i, 4), i) = 1;
  %      map(output(i, 4), 2) = i * 0.1;
```

```
%        map(output(i, 4), 3) = i * 0.1;
    end
    g
    RGB = label2rgb(labeledRef,map, 'w') ;

    show(RGB)
end
```

main_closeloop200.m

```
%GPS entry of flight log update every 5 entry (1 second)

%Declearing global variables used
global refmap_scale;
global refmap
global refmap_file_name;
global refmap_size;
global refmap_TL;
global refmap_TR;
global refmap_BL;
global refmap_BR;
global refmap_delta_lat;
global refmap_delta_long;
global pitch_cam;
global refmap_ext;
global refmap_ext_ok ;

%virtual frame setting
global vert_angle;
global horiz_angle;
global virt_dist_from_cam;
global virt_dist_vert;
global virt_dist_horiz;
global current_R;       %Current rotational matrix


global virt_frame;

%Extraction from refmap mode setting
global BY_LOG;
global BY_LAT_LONG;
BY_LOG = 1;
BY_LAT_LONG =2;

%Results
%global cur_lat_long;            %Current latitude and logtitude as calculated
by system
global cur_TL;
global crusing_speed;
global speed_factor;

n = 20;
g = 1;
clear graph_real_x;
clear graph_real_y;
clear graph_sys_x;
clear graph_sys_y;
clear error_x;
clear error_y;
```

```
points  = 125;
%Allocation space to hold results
graph_real_x(points) = 0;
graph_real_y(points) = 0;
graph_sys_x(points) = 0;
graph_sys_y(points) = 0;

error_x(points) = 0;
error_y(points) = 0;
%Start from this point on the log
startpt = 102;
%Starting latitude and longitude of system
cur_lat_long = [flight_log(startpt, 13), flight_log(startpt, 12)];

inc = 1;
shift_factor(3,3)=0;
too_much = 0;
cannot = 0;
msg = sprintf('Close loop 0.2sec');
disp(msg)
for LEN =startpt:1:startpt+points

    %Every 5 frames addjust the change in average factor
    if (inc ==1)
        inc_lat_long = ([flight_log(LEN+5,13) flight_log(LEN+5, 12)] -
[flight_log(LEN,13) flight_log(LEN, 12)])/5;
        real_lat_long = [flight_log(LEN,13) flight_log(LEN, 12)];
    end
    msg = sprintf('Log entry: %d\n', LEN);
    disp(msg)
    %Calculate where the UAV when the avi images is taken should be
    next_lat_long = real_lat_long + inc_lat_long *(inc);
    msg = sprintf('Current location of the UAV according to flight log is:
\n%0.5f North and %0.5f East\n', next_lat_long(1), next_lat_long(2));
    disp(msg)
    %Generating AVI image
    refmap_extract(LEN+1, BY_LOG, next_lat_long(1), next_lat_long(2));
    graph_real_y(g) = cur_TL(3,1);
    graph_real_x(g) = cur_TL(1,1);
    %Calculate where the UAV when the reference images is taken should be
    cur = real_lat_long + inc_lat_long *(inc-1);
    %Generating refmap extraction image
    refmap_extract(LEN+1, BY_LAT_LONG,cur_lat_long(1) , cur_lat_long(2) );
    %show(avi_image)
    %show(refmap_ext)

    [avi_data, avi_label] = largestnObj_edge(avi_image, [0.05 0.1],n);
    [ref_data, ref_label] = largestnObj_edge(refmap_ext, [0.05 0.1],n);

    %avi_data
    %ref_data
    clear matched_data;
    matched_data = find_match(avi_data, avi_label, ref_data, ref_label, 10);
    matched_data;
    shift_factor(:,:) = 0;

    if (matched_data == false)
        %do nothing and let later clause catch the error
    else
    %There are three matched points, for each of the matched points
```

```
    %calculate the translation factor TL.
        for x = 1:3
           avi_pt_no = matched_data(x,3);
           ref_pt_no = matched_data(x,4);
           avi_pt = round([avi_data(avi_pt_no,2) , avi_data(avi_pt_no,3)]);
           ref_pt = round([ref_data(ref_pt_no,2) , ref_data(ref_pt_no,3)]);
           gnd_ref = find_ground_row_col(ref_pt(2), ref_pt(1), 0);
           avi_TL = air_ptxz(gnd_ref, avi_pt(2), avi_pt(1), cur_TL(2,1));
           shift_factor(1,x) = avi_TL(1,1);
           shift_factor(2,x) = avi_TL(2,1);
           shift_factor(3,x) = avi_TL(3,1);
        end
    end
    %Calculate the mean shift factor (TL)
    mT = mean(shift_factor, 2);
    if (matched_data == false)
      %if there is an error then don't touch anything just continue
      graph_sys_x(g) = graph_sys_x(g-1);
      graph_sys_y(g) = graph_sys_y(g-1);
      cannot = cannot+1;
      msg = sprintf('Cannot calculate GPS, number of this type of errors = %d\n',
cannot);
      disp(msg)
    else
      %Calculate the longtitude and latitude coordinate
      if (g > 1)
        cur_lat_long_new = tl2lat_long(mT);
        mT = lat_long2tl(cur_lat_long);
        mTdif = mT_old - mT;
        dist = (mTdif(1)^2 + mTdif(3)^2)^0.5;
        %To guard against too much change in latitude and longtitude
        if(dist > 0.2*crusing_speed*speed_factor/refmap_scale)
            graph_sys_x(g) = graph_sys_x(g-1);
            graph_sys_y(g) = graph_sys_y(g-1);
            error_x(g) = abs(graph_real_x(g) - graph_sys_x(g));
            error_y(g) = abs(graph_real_y(g) - graph_sys_y(g));
            too_much = too_much+1;
            msg = sprintf('Too far from previous points, number of similar
errors = %d\n', too_much);
            disp(msg)
            mT = mT_old;
        else
            graph_sys_y(g) = mT(3,1);
            graph_sys_x(g) = mT(1,1);
            error_x(g) = abs(graph_real_x(g) - graph_sys_x(g));
            error_y(g) = abs(graph_real_y(g) - graph_sys_y(g));
            cur_lat_long = cur_lat_long_new;
            mT_old = mT;
        end
      else
          cur_lat_long = tl2lat_long(mT);
          mT = lat_long2tl(cur_lat_long);
          graph_sys_y(g) = mT(3,1);
          graph_sys_x(g) = mT(1,1);
          error_x(g) = abs(graph_real_x(g) - graph_sys_x(g));
          error_y(g) = abs(graph_real_y(g) - graph_sys_y(g));
          mT_old = mT;
      end
    end

    %5 frames has passed
```

```
    if (inc == change)
        inc = 0;
    end

    g = g+1;
    inc = inc+1;
    %Output location of system
    msg =sprintf('Current location of the UAV according to system is: \n%0.5f
North and %0.5f East\n', cur_lat_long(1), cur_lat_long(2));
    disp(msg)
    %dividing line
    msg = sprintf('-------------------------------------------------\n\n');
    disp(msg)

end

%Plotting the results
%Plot longtitude and latitude
figure
plot(graph_real_x, graph_real_y,'r.')
hold on;
plot(graph_sys_x, graph_sys_y,'b.')

%figure
%plot(error_x*refmap_scale, 'r.')
%figure
%plot(error_y*refmap_scale,'b.')

%Plot error on each point
figure
error_oa = (error_x.^2 + error_y.^2).^0.5;
plot(error_oa*refmap_scale, 'r.')
%Work out mean errors
meanx = mean(error_x*refmap_scale);
meany = mean(error_y*refmap_scale);
meanoa = mean(error_oa * refmap_scale);
msg = sprintf('Mean error is: %0.2f m\n', meanoa);
disp(msg)
msg = sprintf('Cannnot calculate location in %d cases.\n', cannot + too_much);
disp(msg)
msg = sprintf('Close loop 0.2sec');
disp(msg)
```

main_openloop200.m

```
%GPS entry of flight log update every 5 entry (1 second)

%Declearing global variables used
global refmap_scale;
global refmap
global refmap_file_name;
global refmap_size;
global refmap_TL;
global refmap_TR;
global refmap_BL;
global refmap_BR;
global refmap_delta_lat;
global refmap_delta_long;
global pitch_cam;
```

```
global refmap_ext;
global refmap_ext_ok ;

%virtual frame setting
global vert_angle;
global horiz_angle;
global virt_dist_from_cam;
global virt_dist_vert;
global virt_dist_horiz;
global current_R;       %Current rotational matrix


global virt_frame;

%Extraction from refmap mode setting
global BY_LOG;
global BY_LAT_LONG;
BY_LOG = 1;
BY_LAT_LONG =2;

%Results
%global cur_lat_long;            %Current latitude and logtitude as calculated
by system
global cur_TL;
global crusing_speed;
global speed_factor;

n = 20;
g = 1;
clear graph_real_x;
clear graph_real_y;
clear graph_sys_x;
clear graph_sys_y;
clear error_x;
clear error_y;

points  = 150;
%Allocation space to hold results
graph_real_x(points) = 0;
graph_real_y(points) = 0;
graph_sys_x(points) = 0;
graph_sys_y(points) = 0;

error_x(points) = 0;
error_y(points) = 0;
%Start from this point on the log
startpt = 66;
%Starting latitude and longitude of system
cur_lat_long = [flight_log(startpt, 13), flight_log(startpt, 12)];

inc = 1;
shift_factor(3,3)=0;
too_much = 0;
cannot = 0;
msg = sprintf('Open loop 0.2sec');
disp(msg)
for LEN =startpt:1:startpt+points

    %Every 5 frames addjust the change in average factor
    if (inc ==1)
        inc_lat_long = ([flight_log(LEN+5,13) flight_log(LEN+5, 12)] -
```

```
[flight_log(LEN,13) flight_log(LEN, 12)])/5;
        real_lat_long = [flight_log(LEN,13) flight_log(LEN, 12)];
    end
    msg = sprintf('Log entry: %d\n', LEN);
    disp(msg)
    %Calculate where the UAV when the avi images is taken should be
    next_lat_long = real_lat_long + inc_lat_long *(inc);
    msg = sprintf('Current location of the UAV according to flight log is:
\n%0.5f North and %0.5f East\n', next_lat_long(1), next_lat_long(2));
    disp(msg)
    %Generating AVI image
    refmap_extract(LEN+1, BY_LOG, next_lat_long(1), next_lat_long(2));
    graph_real_y(g) = cur_TL(3,1);
    graph_real_x(g) = cur_TL(1,1);
    %Calculate where the UAV when the reference images is taken should be
    cur = real_lat_long + inc_lat_long *(inc-1);
    %Generating refmap extraction image
    refmap_extract(LEN+1, BY_LAT_LONG,cur(1) , cur(2) );
    %show(avi_image)
    %show(refmap_ext)

    [avi_data, avi_label] = largestnObj_edge(avi_image, [0.05 0.1],n);
    [ref_data, ref_label] = largestnObj_edge(refmap_ext, [0.05 0.1],n);

    %avi_data
    %ref_data
    clear matched_data;
    matched_data = find_match(avi_data, avi_label, ref_data, ref_label, 10);
    matched_data;
    shift_factor(:,:) = 0;

    if (matched_data == false)
        %do nothing and let later clause catch the error
    else
    %There are three matched points, for each of the matched points
    %calculate the translation factor TL.
        for x = 1:3
           avi_pt_no = matched_data(x,3);
           ref_pt_no = matched_data(x,4);
           avi_pt = round([avi_data(avi_pt_no,2) , avi_data(avi_pt_no,3)]);
           ref_pt = round([ref_data(ref_pt_no,2) , ref_data(ref_pt_no,3)]);
           gnd_ref = find_ground_row_col(ref_pt(2), ref_pt(1), 0);
           avi_TL = air_ptxz(gnd_ref, avi_pt(2), avi_pt(1), cur_TL(2,1));
           shift_factor(1,x) = avi_TL(1,1);
           shift_factor(2,x) = avi_TL(2,1);
           shift_factor(3,x) = avi_TL(3,1);
        end
    end
    %Calculate the mean shift factor (TL)
    mT = mean(shift_factor, 2);
    if (matched_data == false)
      %if there is an error then don't touch anything just continue
      graph_sys_x(g) = graph_sys_x(g-1);
      graph_sys_y(g) = graph_sys_y(g-1);
      cannot = cannot+1;
      msg = sprintf('Cannot calculate GPS, number of this type of errors = %d\n',
cannot);
      disp(msg)
    else
      %Calculate the longtitude and latitude coordinate
      if (g > 1)
```

```
        cur_lat_long_new = tl2lat_long(mT);
        mT = lat_long2tl(cur_lat_long);
        mTdif = mT_old - mT;
        dist = (mTdif(1)^2 + mTdif(3)^2)^0.5;
        %To guard against too much change in latitude and longtitude
        if(dist > 0.2*crusing_speed*speed_factor/refmap_scale)
            graph_sys_x(g) = graph_sys_x(g-1);
            graph_sys_y(g) = graph_sys_y(g-1);
            error_x(g) = abs(graph_real_x(g) - graph_sys_x(g));
            error_y(g) = abs(graph_real_y(g) - graph_sys_y(g));
            too_much = too_much+1;
            msg = sprintf('Too far from previous points, number of similar
errors = %d\n', too_much);
            disp(msg)
            mT = mT_old;
        else
            graph_sys_y(g) = mT(3,1);
            graph_sys_x(g) = mT(1,1);
            error_x(g) = abs(graph_real_x(g) - graph_sys_x(g));
            error_y(g) = abs(graph_real_y(g) - graph_sys_y(g));
            cur_lat_long = cur_lat_long_new;
            mT_old = mT;
        end
      else
          cur_lat_long = tl2lat_long(mT);
          mT = lat_long2tl(cur_lat_long);
          graph_sys_y(g) = mT(3,1);
          graph_sys_x(g) = mT(1,1);
          error_x(g) = abs(graph_real_x(g) - graph_sys_x(g));
          error_y(g) = abs(graph_real_y(g) - graph_sys_y(g));
          mT_old = mT;
      end
    end

    %5 frames has passed
    if (inc == change)
        inc = 0;
    end

    g = g+1;
    inc = inc+1;
    %Output location of system
    msg =sprintf('Current location of the UAV according to system is: \n%0.5f
North and %0.5f East\n', cur_lat_long(1), cur_lat_long(2));
    disp(msg)
    %dividing line
    msg = sprintf('-------------------------------------------------\n\n');
    disp(msg)

end

%Plotting the results
%Plot longtitude and latitude
figure
plot(graph_real_x, graph_real_y,'r.')
hold on;
plot(graph_sys_x, graph_sys_y,'b.')

%figure
%plot(error_x*refmap_scale, 'r.')
%figure
```

```
%plot(error_y*refmap_scale,'b.')

%Plot error on each point
figure
error_oa = (error_x.^2 + error_y.^2).^0.5;
plot(error_oa*refmap_scale, 'r.')
%Work out mean errors
meanx = mean(error_x*refmap_scale);
meany = mean(error_y*refmap_scale);
meanoa = mean(error_oa * refmap_scale);
msg = sprintf('Mean error is: %0.2f m\n', meanoa);
disp(msg)
msg = sprintf('Cannnot calculate location in %d cases.\n', cannot + too_much);
disp(msg)

msg = sprintf('Open loop 0.2sec');
disp(msg)
```

lat_long2tl.m

```
function TL = lat_long2tl(lat_long)
    global refmap_BL;
    global refmap_delta_lat;
    global refmap_delta_long;
    global refmap_size;

    TL = [abs(abs(lat_long(1)) -
abs(refmap_BL(1)))/refmap_delta_lat*refmap_size(1);
        0;
        abs(abs(lat_long(2)) -
abs(refmap_BL(2)))/refmap_delta_lat*refmap_size(2)];
end
```

match_critial.m

```
function condition = match_critial(datapoint1, datapoint2)
    global avi_info;
    DF = datapoint1(1:3)-datapoint2(1:3);
    area_tol = 0.15;             %Tolorance for error in area, radian and theta
    xy_tol = 0.10;                 %Percentage Tolorance for difference x and y
cordinate
    eulerNo_tol = 1;
    solitidy_tol = 0.03;
    DF(1) = DF(1)./datapoint1( 1);
    DF = abs(DF);
    if(abs(datapoint1(5) - datapoint2(5) )< eulerNo_tol)
        if((DF(1) < area_tol) == true)        %if the area is no more different
then 10%
            if(abs(datapoint1(5)-datapoint2(5)) < solitidy_tol)
                if((DF(2)/avi_info.Width) < xy_tol)
                    if((DF(3)/avi_info.Width) < xy_tol)

                        condition = true;
                        return;
                    end
                end
            end
```

```
        end
    end
    condition = false;
    return;
end
```

read_log.m

```
%read section of flight log file as specific in the variable 'log_file_name'
%in the file initialize.m.  Store result in global variable 'flight_log'
%Also scale the data to data which make sense

function read_log
    %Declearing all the global variables that are used;
    global log_file_section;
    global log_file_name;
    global flight_log;
    global log_entry_scale;
    global log_file_delim;

    %reading raw data
    raw_log = dlmread(log_file_name,log_file_delim);
    [x no_col_selected] =size(log_file_section);
    raw_log_size = size(raw_log);
    flight_log(raw_log_size(1),no_col_selected) = 0;
    for n = 1:no_col_selected
        flight_log(:,n) = raw_log(:, log_file_section(n))*log_entry_scale(n);

    end
end
```

refmap_extract.m

```
%Coordinate system use:
%x is north, z is east, y is up
%Head of the UAV is facing north (positive x), right wing of UAV is point
%east, top of UAV is point towards sky
function refmap_extract(LEN, mode, lat, long)
    %The variable 'mode' let the function know whether it is reading the
    %latitude and longtitude information from log file or the function is
    %to use a manual input as specific by the variable lat and long.

    %After reading the latitude and longtitude, this function generate the
    %image according to attribute of the UAV specific in the 'LEN' entry of
    %the flight log.
    %If 'mode' == BY_LOG then the image is stored in the global variable
    %avi_image, else the images is stored in the global variable
    %refmap_ext.

    %This function also store its translation factor and transformation
    %matrix in the global variables 'cur_TL' and 'current_R'.

    %Declearing global variables used in this function
    global flight_log;        %global variable containing flight log of avi
    global refmap_scale;
    global refmap;
    global refmap_size;
```

```
    global refmap_TL;
    global refmap_TR;
    global refmap_BL;
    global refmap_BR;
    global refmap_delta_lat;
    global refmap_delta_long;
    global pitch_cam;
    global refmap_ext;
    global refmap_ext_ok ;

    global virt_dist_from_cam;
    global virt_dist_vert;
    global virt_dist_horiz;
    global virt_frame;

    global avi_info;
    global avi_image;
    global avi_ext_ok;

    global BY_LOG;
    global BY_LAT_LONG;
    global cur_lat_long;            %Current latitude and logtitude as
calculated by system
    %The translation matrix which specific where the camera is on the
    %reference map
    global cur_TL;
    global current_R;

    %Function is to use the flight log latitude and longtitude, read them
    %from the global variable.
    %if(mode == BY_LOG)
     %   lat = flight_log(LEN,13);
    %    long = flight_log(LEN, 12);
    %end

    %position according to GPS
    TL = [abs(abs(lat) - abs(refmap_BL(1)))/refmap_delta_lat*refmap_size(1);
        flight_log(LEN, 4) / refmap_scale;
        abs(abs(long) - abs(refmap_BL(2)))/refmap_delta_lat*refmap_size(2)];
    cur_TL = TL;
    %Obtain pitch(a), roll(b) and yaw(h) from flight log
    a = flight_log(LEN, 1);
    b = flight_log(LEN, 2);
    h = flight_log(LEN, 3)*-1;

    %setting up virtual frame, DEBUG PURPOSE
    BLC = [-1*virt_dist_vert -1*virt_dist_from_cam -1*virt_dist_horiz]' /
refmap_scale;
    TLC = [ 1*virt_dist_vert -1*virt_dist_from_cam -1*virt_dist_horiz]' /
refmap_scale;
    BRC = [-1*virt_dist_vert -1*virt_dist_from_cam  1*virt_dist_horiz]' /
refmap_scale;
    TRC = [ 1*virt_dist_vert -1*virt_dist_from_cam  1*virt_dist_horiz]' /
refmap_scale;


    %Sequence of rotation:
    %1. Pitch angle correction for camera to compensate for mount location
    %of camera on UAV
    %2. Apply yaw rotation (along
    %3. Apply pitch
```

```
%Taken from www.euclideanspace.com/maths/geometry/rotations/euler/

%Rotation direction used in the series of rotation function is as
%given by right hand rule.  i.e. Thumb point at the axis direction and
%the curl of finger give the direction of rotation.
%In log:
%Positive angle of roll is right wing down.
%Positive angle of pitch is nose up.
%Positive angle of yaw is nose right.
%This mean the yaw angle recorded in log is in the opposite direction
%to that given by the rotational direction, hence it is multipled by
%negative above to correct the rotational direction.

%Final rotational matrix
R = rotate_x(b)*rotate_z(a)*rotate_y(h)*rotate_z(pitch_cam);
current_R = R;

%Start Debug display
%BLCR = R*BLC
%TLCR = R*TLC
%BRCR = R*BRC
%TRCR = R*TRC

%BLCRT = BLCR+TL
%TLCRT = TLCR+TL
%BRCRT = BRCR+TL
%TRCRT = TRCR+TL

%BLCG = gnd_ptxz(TL, BLCRT)
%TLCG = gnd_ptxz(TL, TLCRT)
%BRCG = gnd_ptxz(TL, BRCRT)
%TRCG = gnd_ptxz(TL, TRCRT)
%End debug display

%Change in distance with each pixel in the generated image.
vert_dist_in = virt_dist_vert *2 /avi_info.Height;
horiz_dist_in = virt_dist_horiz *2 / avi_info.Width;

%Correction factor used to convert cartersian coordinate to pixels on
%the image.
PCFT = [refmap_size(1); 0 ; 0];
PCCF = [1; 0; -1];

%Loop to extract image from reference map.
%Work by generating a vector from camera to every pixel in the virtual
%frame.  This vector is then rotated by the rotational matrix R, and
%translated to place in the correct location by adding the the vector
%TL which is the location of the UAV.  Then using the location of the
%camera and the location of the rotated virtual frame, linear equations
%is used to find the location where the ground is for this pixel.
for y = 1:avi_info.Height
    for x = 1:avi_info.Width

        %Use linear equation to find out where this pixel hit the
        %ground.  This function include operation which rotate the
        %virtual view frame and translating it in place.  It then
        %calculate the place where a specific pixel in the view screen
        %should hit the ground
        pt_gnd = find_ground_row_col(y,x,0);
        %Coverting the ground coordinate (i.e. x-y coordinate) to pixel
        %coordinate (row and column of image)
```

```
            pt_photo = round((PCFT - pt_gnd) .* PCCF);
            %Error checking to see if the coordinate generated is out of
            %bound, if out of bound set refmap_ext_ok flag to false and
            %exit, else generate the image and set refmap_ext_ok flag to
            %true and continue
            %Checking row number within bound
            if ((pt_photo(1,1) < 1) | (pt_photo(1,1) > refmap_size(1)))
                if(mode == BY_LOG)
                    avi_ext_ok = false;
                else
                    refmap_ext_ok = false;
                end
                'error'
                return;
            end
            %Checking column number within bound
            if ((pt_photo(3,1) < 1) | (pt_photo(3,1) > refmap_size(2)))
                if(mode == BY_LOG)
                    avi_ext_ok = false;
                else
                    refmap_ext_ok = false;
                end
                return;
            end
            %Filling the image.
            if(mode == BY_LOG)
                avi_image(y, x,1) = refmap(pt_photo(1,1), pt_photo(3,1), 1);
                avi_image(y, x,2) = refmap(pt_photo(1,1), pt_photo(3,1), 2);
                avi_image(y, x,3) = refmap(pt_photo(1,1), pt_photo(3,1), 3);
                avi_ext_ok = true;
            else
                refmap_ext(y, x,1) = refmap(pt_photo(1,1), pt_photo(3,1), 1);
                refmap_ext(y, x,2) = refmap(pt_photo(1,1), pt_photo(3,1), 2);
                refmap_ext(y, x,3) = refmap(pt_photo(1,1), pt_photo(3,1), 3);
                refmap_ext_ok = true;
            end
        end
    end
   % show(refmap_ext)
```

tl2lat_long.m

```
%Calculating the location of the UAV from the translation factor TL.  Using
%the approximation that the longitude and latitude of the earth can be
%calculated like a square grid.  The approximation is true as long as the
%UAV is flying over a small distance and the reference map is no bigger
%then about 10km square.  For this sample data both of the above is true.

function lat_long = tl2lat_long(TL)
    %Importing global variables used.
    global refmap_BL;
    global refmap_delta_lat;
    global refmap_delta_long;
    global refmap_size;
    global correction_factor;


    lat_long(1) = refmap_BL(1) +(TL(1,1))*refmap_delta_lat/refmap_size(1);
    lat_long(2) = refmap_BL(2) +(TL(3,1))*refmap_delta_long/refmap_size(2)-
```

```
correction_factor;
end
```