A Report for Vision-based Navigation of UAVs

Set of instructions for formulating simulation model of landmark-based vision-guided navigation of UAVs.

May 26, 2016

A Report to DST Group

Author: Dr Aakash Dawadee

**Contents**

1. Introduction

1.1

1.2

1. Image Processing Algorithms

2.1 Introduction

2.2 Adaptive Histogram Equalization (AHE)

2.3 The Three Stage Landmark Detection Algorithm

2.3.1 First Stage

2.3.2 Second Stage

2.3.3 Third Stage

2.4 The Three Stage Landmark Detection Algorithm

3. 6DoF Kinematics of UAV motion

2.1 Motion in a Straight Line

2.2 Rotational Motion

2.3 Dead Reckoning

2.4 Drift Computation

2.5 Wind Prediction and Correction

1. Simulation with MATLAB

3.1

3.2

1. Hardware in the Loop Simulation

4.1

4.2

1. **Introduction**

This document presents overview of an approach for vision-guided navigation of UAV which relies on natural and/ or man-made landmarks in the ground.

1. **Image Processing Algorithms**

**2.1 Introduction**

In this in, image processing algorithms are presented which is suitable for the autonomous navigation of vision- based UAVs. First, the image is converted to grayscale and normalized using the Adaptive Histogram Equalization (AHE) to make it somewhat illumination-invariant. Then three stage object detection (landmark identification) technique is used. In the first step of landmark identification, the normalized image is adaptively binarized to extract candidate landmarks. In the second stage, landmarks are localized by referring to their scale and rotation invariant properties. Then a reference point is computed as the centroid of localized landmark centroids. A scaling factor is also computed using features of landmarks from the current image and those from the reference image. The current image is then scaled by this factor before further processing to make the algorithm scale-invariant. In the final stage, each candidate landmark is cropped around their centroids and a one-dimensional feature signature is generated from each of them. The feature signature of each candidate landmark is compared with those stored in the landmark library by computing a correlation coefficient. When a significant correlation is obtained, the candidate landmark is regarded as a detected landmark. The constellation of all detected landmarks is then used to form a waypoint. Finally, the waypoint features are compared with those in the database. The detected waypoint and its features are then forwarded to the UAV navigation system which determines true position of the UAV in space.

**2.2 Adaptive Histogram Equalization (AHE)**

The histogram equalization is a brightness normalization process. The pixel response range, [LP , HP ] of a grayscale image, g(x, y) is adjusted to a fixed interval [LR , HR ] to make the image somewhat invariant to illumination [166]. The fixed interval is evaluated from the reference image, gr (x, y) of the image scene. Let us define histogram of a grayscale image, g(x, y) as:

n(k), (2.1)

where

k = LP, LP +1, LP+2, ..., HP.

The histogram, n(k) is normalized (adjusted to fixed interval), [LR , HR ] using following formula:

(2.2)

where

x ∈ {1, 2, ..., M }, y ∈ {1, 2, ..., N },

M = number of row pixels on image g(x, y),

N = number of column pixels on image g(x, y).

**2.3 The Three Stage Landmark Detection Algorithm**

This sub-section describes the landmark detection algorithm which is achieved in three different stages.

**2.3.1 First Stage**

In the first stage of landmark recognition, the normalized image, gn (x, y) was adaptively binarized (Algorithm 1) with morphological operations to extract n or less significant objects. Morphological operations were used to remove objects outside a range [PL ,PH ] where PL and PH are lower and up- per thresholds of number of pixels to make an object. For all white objects in the 2D binary image, pixels with 8 neighboring connectivity were considered. The each iteration of *Algorithm 1* used this operation.

The binary thresholding was initialized with a low value at the start which was then gradually increased inside a “while loop” to detect a required number of objects. In the filtered image, each object is created by a group of neighboring bright pixels. The second condition of the “while loop” was used to prevent excessive white pixels for lower values of binary threshold where BT is the threshold which is computed from reference image such that:

NL < BT ≤ NT, (2.3)

where,

NT = Total number of pixels in the image,

NL = Total number of pixels covering all landmarks of the binary reference image.

BT depends on size of the objects of interest in the reference image and contrast of objects with reference to the background.

|  |
| --- |
| Algorithm-1: Adaptive binarization of a normalized grayscale image |
| T h ← Initial Value, where 0 ≤ T h ≤ 255  Define BT  ib (x, y) ← gn (x, y) > T h  ibm (x, y) ← ib (x, y)(H, L), where ibm (x, y) = Binary image after morphological operations on  ib (x, y) that removes size of objects outside low threshold L and high threshold H  Compute N (ibm (x, y)), the number of objects in the binary image, ibm (x, y)  Compute S = Sum of all pixels of ibm (x, y)  while (N (ibm (x, y)) > n & S > BT ) do  T h ← T h + 1  ib (x, y) ← g(x, y) > T h ibm (x, y) ← ib (x, y) Compute N (ibm (x, y)) Compute S  end while |

**2.3.2 Second Stage**

In the second stage (*Algorithm-2*), different groups were formed using the objects extracted in the first stage. These groups had all possible combinations with a minimum of three and maximum of Ndb objects in a group; where Ndb is the number of landmarks whose features were kept in the database. The scale and rotation invariant features of each object in each of these groups were compared with those stored in the database. The priority was to match the group with the higher number of objects. Once the match was found that group of objects was selected as candidate landmarks and the centroid of their centroids was selected as the reference point which was used for feature computation in the second and third stages of landmark matching.

|  |
| --- |
| Algorithm-2: Second stage landmark detection |
| Load Ndb = number of landmarks in the database, where Ndb ≥ 3  Load N = number of candidate landmarks extracted by the Algorithm-1  Compute Ccl = centroid of each of N candidate landmarks  Set break flag, bf = 0  if (N ≥ Ndb )  Nstart = Ndb  else if (3 ≤ N ≤ Ndb )  Nstart = N  else  Nstart = 0  Report Error: Not enough candidate landmarks were detected  end if  if (Nstart ! = 0)  for (n = Nstart : −1 : 3)  Load Adb (n) = angle matrix from database (Adb is function of n)  Compute a group matrix, G that contains centroid of landmarks with different combinations (C (N, n))  Compute Ng = number of different combinations in group matrix G Load SoSdb = sum of sides formed by n landmarks from database  for (i = 1 : Ng )  Compute C = centroid of G  for (j = 1 : n)  for (k = 1 : n)  if ( j ! = k)  Compute angle made by line connecting reference landmark j and centroid C with line connecting landmark k and centroid C, and store it in matrix, A  end if end for  end for  Algorithm-2: Second stage landmark detection (contd…) |
| Sort matrix A row-wise in ascending order  Compute Adif f = Adb (n) - A  if (Adif f < Angtol ), where Angtol = Matrix of tolerance angles  bf = 1  Break for loop  end if end for  if (bf = 1)  Compute SoSi = sum of sides for polygon formed by points in matrix G  Compute scaling factor,  Break for loop  end if end for  end if |

**2.3.3 Third Stage**

A novel approach to represent features of a landmark from a grayscale image as a one-dimensional vector is proposed which is used in the final stage of landmark recognition. A normalized grayscale image represented by a two-dimensional function, gn (x, y) was scaled by scaling factor ξ such that:

gns (x, y) = ξ gn (x, y), (2.3)

where

,

SoSdb = Sum of all sides of the polygon formed by joining centroids of landmarks of the reference image in the database,

SoSi = Sum of all sides of the polygon formed by joining centroids of candidate landmarks of an instantaneous image.

This step ensured that the third stage of the landmark detection algorithm was invariant to scale.

The scaled and normalized grayscale image gns (x, y) was mapped to a sub-space i(x, y) within the image,

gns (x, y) → i(x, y). (2.4)

The size of i(x, y) was chosen to cover the area of interest around the landmark. We choose i(x, y) as a square matrix of size s×s where ‘s’ is a positive odd integer. We choose value of ‘s’ as ‘101’ for all experiments in this thesis. Hence, all the landmarks were contained within 101×101 pixels. Now, the center pixe3l of i(x,y) is ). Fig. 2.1(a) shows an example of a 7×7 image. Lines are drawn from the center pixel ) to each pixel on the border of the image i(x,y) starting from pixel ) to the pixel above it in the clockwise direction as shown in Fig. 2.1(a).

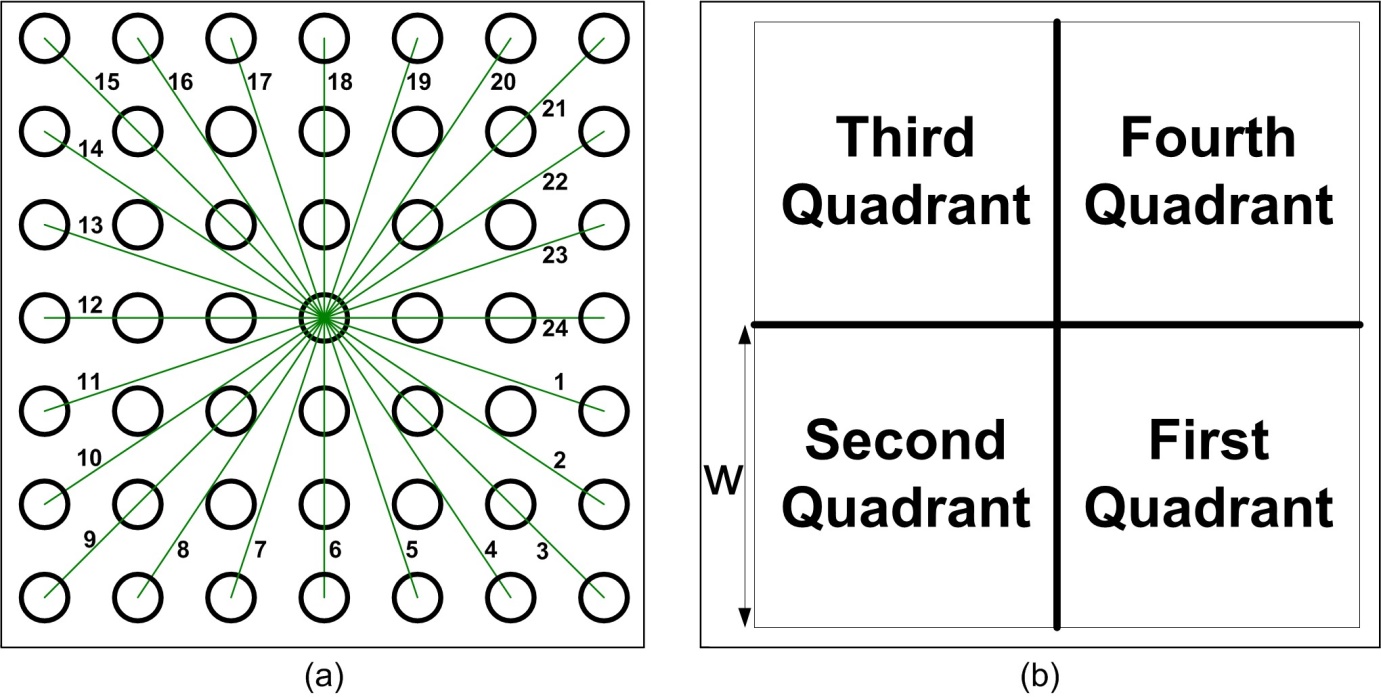


Figure 2.1: (a) An image with 7×7 pixels; (b) Four quadrants of an image.

The image sub-space, i(x,y) is divided into four quadrants as shown in Fig. 2.1(b), where the size of each quadrant is . It can be observed that there are 2w lines in each quadrant. For each quadrant a set of equations was formulated to compute the angle made by lines at horizontal or vertical axes.

1. Angles (in degrees) made by ‘w’ lines with the positive horizontal axis at the center pixel for the first quadrant:

, i ∈ (1, 2, …, w) (2.5)

1. Angles (in degrees) made by remaining ‘w’ lines with the negative vertical axis at the center pixel for the first quadrant:

, i ∈ (1, 2, …, w) (2.6)

1. Angles (in degrees) made by w lines with the negative vertical axis at the center pixel for the second quadrant:

, i ∈ (1, 2, …, w) (2.7)

1. Angles (in degrees) made by remaining w lines with the negative horizontal axis at the center pixel for the second quadrant:

, i ∈ (1, 2, …, w) (2.8)

1. Angles (in degrees) made by w lines with the negative horizontal axis at the center pixel for the third quadrant:

, i ∈ (1, 2, …, w) (2.9)

1. Angles (in degrees) made by remaining w lines with the positive vertical axis at the center pixel for the third quadrant:

, i ∈ (1, 2, …, w) (2.10)

1. Angles (in degrees) made by w lines with the positive vertical axis at the center pixel for the fourth quadrant:

, i ∈ (1, 2, …, w) (2.11)

1. Angles (in degrees) made by remaining w lines with the positive horizontal axis at the center pixel for the fourth quadrant:

, i ∈ (1, 2, …, w) (2.12)

As an example, θ2 and φ2 for the first quadrant are shown in Fig. 2.2(a). It can be observed that some lines do not pass through the center of pixels. These lines were approximated to the nearest pixels. We calculated lengths of perpendicular segments, Pij from the horizontal or vertical axis. A typical example is shown in Fig. 4.2(b) for lines 2 and 5. Lengths of these perpendicular segments were given by the following set of formulae:

1. First Quadrant (j = 1,2,…, w),

(2.13)

1. Second Quadrant (j = 1,2,…, w),

(2.14)

1. Third Quadrant (j = 1,2,…, w),

(2.15)

1. Second Quadrant (j = 1,2,…, w),

(2.16)

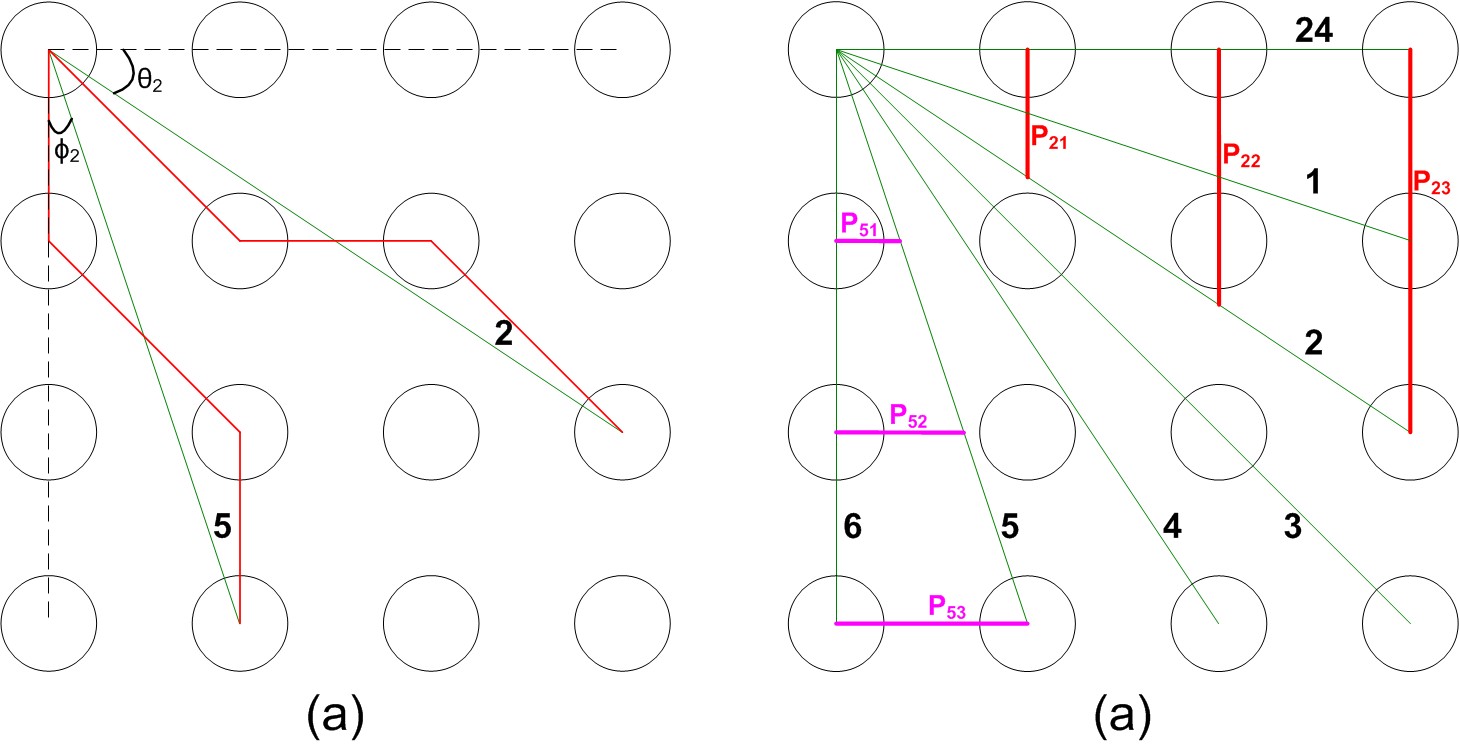


Figure 2.2: (a) Lines passing through center of pixels; (b) Perpendicular sections of lines 2 and 5.

All perpendicular length sections were then rounded to the nearest integer to create new lines that pass through the center of the nearest pixels. Fig. 2.2(a) shows the example with the new lines 2 and 5. The pixels through which lines pass are:

(2.17)

The “Feature Signature” *F(i)* is defined as the sum of all pixel values along these lines.

(2.18)

These feature signatures of landmarks of interest are computationally inexpensive compared to other popular template matching approaches in the literature which we will present in Chapter 6. Moreover, when feature signatures are used with a constellation of landmarks, they are scale and rotation invariant.

If feature signatures F1, F2 were to be compared, we compared them using 2D correlation coefficient given by:

, (2.19)

where

Mean of vector ,

Mean of vector ,

**2.4 Waypoint Detection Algorithm**

The waypoint polygon was constructed by joining centroids of the detected landmarks. The internal angles, αi of the polygon are invariant to scale and rotation. Moreover, ratios of each side of the polygon to the sum of all sides, Ri are also invariant to scale and rotation. Matching of waypoint features, αi and Ri between an instantaneous image and the reference image was regarded as successful detection of a waypoint.

Let us consider a waypoint polygon from the reference scene shown in Fig. 2.3(a). With i = 1, 2, 3, 4, αRi are internal angles of this polygon and dRi are its sides. If the waypoint scene was captured from closer to the scene, we obtain a waypoint polygon as shown in Fig. 2.3(b) with αLi and dLi as internal angles and sides respectively. On the other hand, if the same waypoint scene was captured from further away from the scene, we obtain waypoint polygon as shown in Fig. 2.3(c) with αSi and dSi as internal angles and sides respectively. As all of these are similar polygons, we can write:

αRi = αLi = αSi (2.20)

(2.21)

It can be observed that features represented by Eqs. 4.20 and 4.21 are invariant to scale, rotation and translation. Our results in the next section will demonstrate these properties.

**2.5 Landmark and Waypoint Library**

For each waypoint scene, a set of landmark and waypoint features was stored in a landmark and waypoint library. We picked lower and higher threshold [LR , HR ] of Adaptive Histogram Equalization (AHE) and stored in the database. Then each waypoint scene was adaptively binarized and a number of objects were chosen as landmarks. A centroid of centroids of these landmarks was chosen as a reference point. Now angles made by each landmark centroid at the reference point with remaining landmark centroids was computed and stored in the database. The sum of sides of the polygon formed by joining the centroid of landmarks was stored in the database and used to make the third stage of the landmark recognition invariant to scale. One-dimensional feature signatures defined by Eq. 4.18 was computed for each landmark and kept in the database. Features of the waypoint polygon were also computed and stored in the library. We kept all combinations of landmark and waypoint features made by at least three landmarks in the database so that even when fewer landmarks were detected, their features could be compared with the database for landmark and waypoint feature matching. This ensures that the algorithm works in the case of partial occlusion or failure to detect some landmarks. Our results with the computer generated images address partial occlusion and results with the UAV flight trial images show the case of fewer landmarks detection.

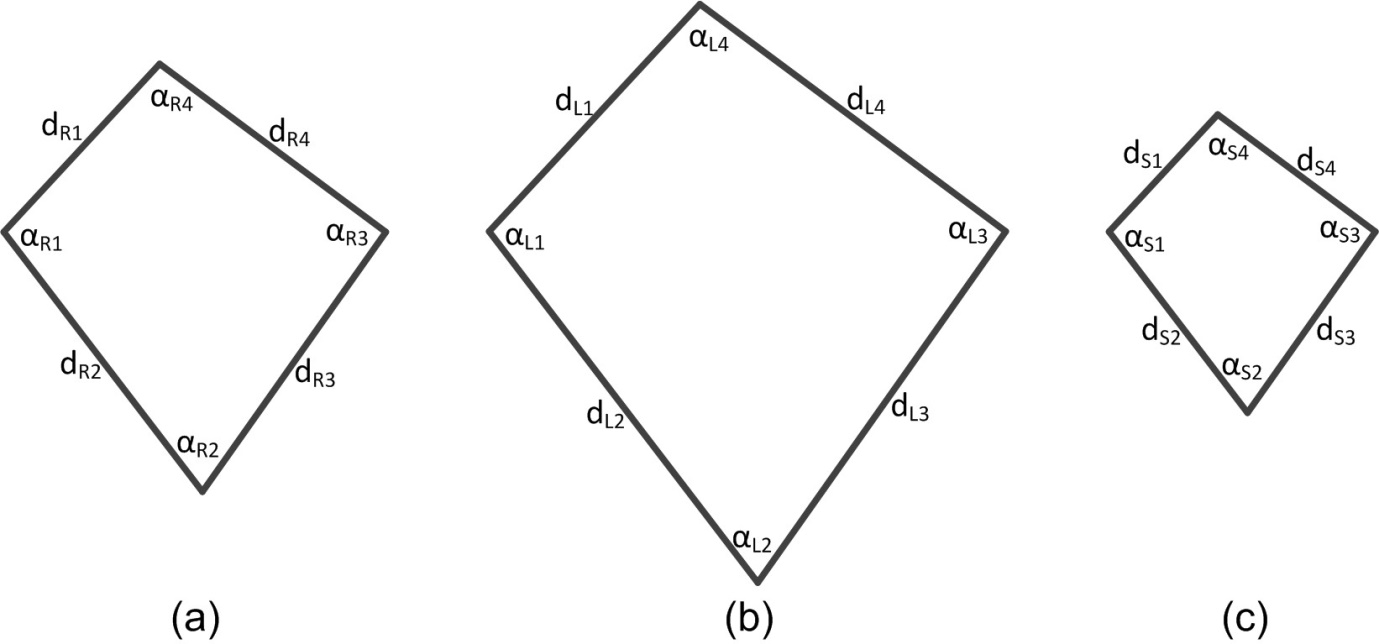


Figure 2.3: Example of a waypoint polygon: (a) Waypoint polygon of reference image; (b) Waypoint polygon when image was captured from closer to the scene; (c) Waypoint polygon when image was captured from further away from the scene.

1. **Kinematics of UAV Motion and Vision-based Drift Computation**

This section describes basic Kinematics of UAV motion which is used in simulation models presented in sections 3 and 4.

Let us consider the wind bias (Vb ) and wind noise (Vn ) as additive vectors in the XY-plane to the velocity of the UAV in the heading direction (Vu).

The instantaneous value of total wind velocity vector is written as:

Vw = Vb + Vn. (3.1)

The velocity of the UAV in the yaw-direction (heading direction) is:

Vu = |Vu|∠φ. (3.2)

The above expression represents the polar representation of the vector Vu with its magnitude and direction. We use this representation throughout this chapter and thesis.

The overall velocity (the course velocity) of the UAV is written as:

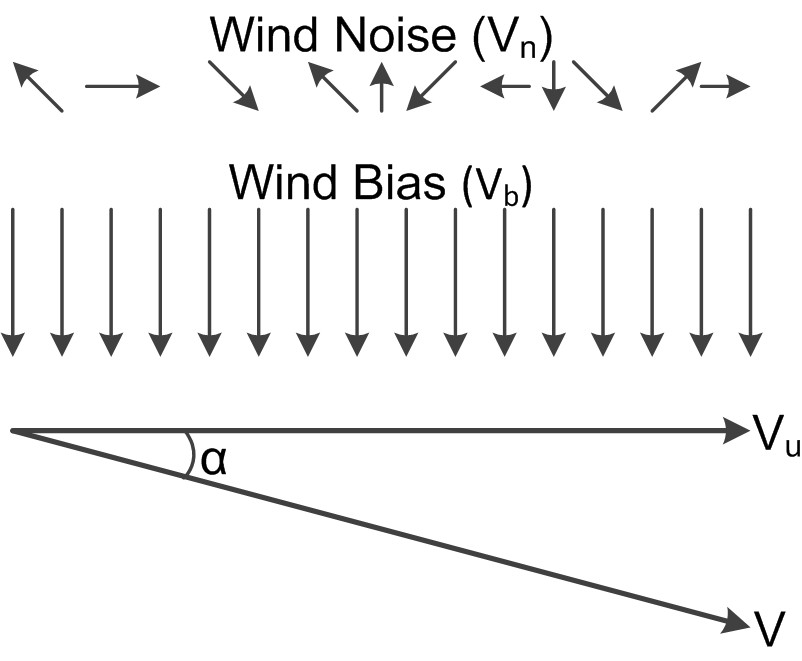
V = Vu + Vw. (3.3)

V = |V |∠α, (3.4)

where

α = The course angle of the UAV.

Fig. 3.1 shows an example of a Phasor diagram showing different velocity vectors. The wind bias is assumed to be at an angle of −900 and random wind noise is shown at the top with random magnitude and directions. The yaw angle is assumed to be 00, hence heading angle is 00. Due to wind, the UAV moves in the direction of the course angle α with magnitude |V |.

  
Figure 3.1: A Phasor diagram showing different velocity components of wind.

Furthermore, let θ be the instantaneous pitch of the UAV. Then the course velocity in space (XYZ- direction) is written as:

(3.5)

**3.1 Motion in a Straight Line**

The motion of the UAV in a straight line is due to the course velocity. Instantaneous position of the UAV while travelling in the straight line is written as:

P (t + ∆t) = P (t) + [Vxyz]∆t, (3.6)

where

Change of position in time, ∆t can be written as:

∆P = [Vxyz] ∆t. (3.7)

, (3.8)

where

.

**3.2 Rotational Motion**

At each waypoint, banked turn was used to achieve a heading angle that took the UAV to the next waypoint. For this, the UAV was rolled by constantly increasing roll angle to a maximum roll angle threshold, ±ψmax which was maintained until achieving the desired yaw angle φD . Meanwhile, the change in yaw angle while increasing/decreasing roll angle from zero to ±ψmax is calculated which is used to update desired yaw angle (φD). The roll angle was gradually decreased to zero once updated yaw angle is achieved.

Let φi be the initial yaw angle of the UAV before turn and φ is the instantaneous (current) value of the yaw angle. Then we compute change in yaw angle, ±φc while increasing/decreasing roll angle from zero to ±ψmax as follows:

(3.9)

Then desired yaw angle φD is updated as:

φD = φD − (±φc). (3.10)

The turn radius of the UAV while rolling is given as:

, (3.11)



where

V = Velocity of the UAV,

g = Acceleration due to gravity.

Now, change in yaw, ∆φ in time ∆t is computed by calculating its displacement, D along the perimeter, P of the circular path formed by radius, R.

UAV displacement, D in time ∆t is:

D = |V|∆t. (3.12)

Perimeter, P of the circle formed by radius, R is:

P = 2πR. (3.13)

Now change in the yaw angle in degrees, ∆φ is:

(3.14)

While advancing in time, the yaw angle, φ is updated as follows:

φ(t + ∆t) = φ(t) + ∆φ. (3.15)

At the end of the banked turn, a constant pitch rate was used to achieve a climb that would take the UAV to the next waypoint altitude.

**3.3 Dead Reckoning**

The dead-reckoning is a basic navigation method where the current position is calculated from previously determined positions and velocity measurements [93]. We used dead-reckoning to navigate a UAV between two waypoints. Let vi be the instantaneous velocity of the UAV, φT be the true heading in the XY-plane and θ be the pitch of the UAV. Then velocities in space are given by:

. (3.16)

The instantaneous distance travelled by the UAV during time interval t is given by:

. (3.17)

where

x − x0 = distance travelled in the x-direction during measurement interval t,

y − y0 = distance travelled in the y-direction during measurement interval t,

z − z0 = distance travelled in the z-direction during measurement interval t.

**3.4 Drift Computation**

Consider a reference image RI and its rotation/ scale invariant features, FRI located at position, PRI (x, y) in the XY-plane. Let us consider that another image I that has same features, FRI at position, PI (x, y). The position drift of I with reference to RI in terms of number of pixels is given by:

DP (x, y) = PI (x, y) − PRI (x, y). (3.18)

Let the image I is captured from an aircraft with heading angle of φ degrees. Then the position drift in the XY-plane can be computed using the transformation matrix, T P representing the transformation of the drift in terms of image pixels to the drift in the XY-plane.

I

. (3.19)

where

r = The resolution of each pixel in meters while capturing an image from the reference altitude.

Then the drift in the XY-plane in meters, DXY is computed as:

(3.20)

where

ξ = Scaling factor computed in the second stage of landmark recognition,

Waypoint prediction has an error associated with it. The total error, ε is written as:

ε = εT + εA, (3.21)

where

εT = Error in the transformation,

εA = Error due to the attitude bias.

The error, εT depends on the resolution of the image, higher the resolution, smaller the potential error. However larger resolution demands high capacity hardware and longer processing time.

Drift in the Z-axis (i.e. the altitude drift) is computed as:

∆ψ = (ξ − 1) × Zref, (3.22)

where

Zref = The reference altitude,

ξ = Scaling factor computed at landmark detection.

**3.5 Wind Prediction and Correction**

At each waypoint, the wind velocity could be predicted from the calculated drift.

, (3.23)

where

i = Waypoint number,

= Time of flight between *(i – 1 )th* waypoint to *ith* waypoint.

The velocity to be commanded to the UAV in the XY-plane to compensate Vw is:

, (3.24)

where

= Body velocity of the UAV in XY-axes.

As we are using a kinematic model, instead of updating the required velocity, we update the heading angle and time of flight to the next waypoint.

Wind compensation angle is given as:

, (3.25)

where

= Direction of vector Vreq in polar representation,

= Direction of vector Vxy in polar representation.

If φT is the heading angle calculated by dead-reckoning to take the UAV to the next waypoint, we update the heading angle as follows:

φu = φT + . (3.26)

Time of flight, tf between ith waypoint and (i + 1)th waypoint is updated as:

, (3.27)

where

Dn= Distance to the next waypoint.

1. **Vision-based UAV Navigation in Linux Environment**

This section presents implementation of vision-based navigation in Ubuntu Linux environment. Ubuntu 16.04 was chosen as operating system in PC. However, compiled programs would run in any Debian based Linux environments such as Raspbian OS of Raspberry Pi.

Following environments and packages should be installed in Linux environment:

1. Python 3.0
2. OpenCV Version 2.4.13
3. numpy package
4. scipy package
5. pyplotlib package

Once these environments and packages are set-up, programs can be executed directly from the terminal. Overall vision-based navigation is divided into two sections: Image Processing and Navigation.

**4.1 Image Processing**

Image processing is integral part of overall vision-based navigation system. Image processing includes generating data, feature extraction and matching of landmark and waypoint features.

**4.1.1 Generating Thresholds, Landmark and Waypoint Data**

For each waypoint, Pre-flight image and some navigation data will be required to generate image processing thresholds, landmark data and waypoint data.

**Requirements**

From the flight data following parameters shall be collected.

1. **Waypoint image:** with minimum of THREE visual landmarks. Ideally FOUR landmarks are preferred.

2. **Altitude (in meters):** This is reference altitude which can be obtained by looking at navigation data when image was captured.

3. **Yaw angle (in degrees)**: Yaw angle can also be obtained by looking at navigation data when image was captured.

Note: **Altitude** and **Yaw Angle** are obtained from the flight data (explained in later section) for **HiL** and **Flight Tests**. These parameters are not required in **Simulation** mode.

**Pre-Processing: Rename Source Image and Generate Sub-images**

Following process shall be completed for each waypoint image.

All source image and sub-images should be placed in folder **Database** inside work folder.

**Case 1: Waypoint Image with FOUR landmarks**

First, landmarks are virtually marked as 0, 1, 2, and 3 (Note: Python indexing starts from 0. Numbers were chosen in accordance with Python indexing convention)

Landmark that has highest centroid y-value in the image plane (i.e. the landmark on the bottom most position) is indexed with 0 and indexing is increased as centroid y-value de creases. Fig. 4.1 shows an example of waypoint with four landmarks marked with landmark numbers. Note that landmark number increases from bottom to top of the image.

 Fig. 4.1: An example of waypoint image with four landmarks.

Each waypoint image should be named with following naming convention:

waypoint\_**WaypointNumber**\_**parLmrkNos**.jpg

where

**WaypointNumber** = waypoint number in navigation sequence

**parLdmrkNos** = different landmark combination sets with THREE landmarks out of FOUR landmarks

**parLdmrkNos** field shall be set to **0** while generating data with all FOUR landmarks.

**parLdmrkNos** field shall be set to 012, 013, 023, 123 for landmark combinations different landmark combinations. Fig. 4.2 shows images with partial landmark conditions for waypoint image shown in Fig. 4.1.

Example: for waypoint\_number = 1:

Waypoint image with all four landmarks shall be named as **waypoint\_1\_0.jpg**

Waypoint image with landmarks 0, 1 and 2 shall be named as **waypoint\_1\_012.jpg**

Waypoint image with landmarks 0, 1 and 3 shall be named as **waypoint\_1\_013.jpg**

Waypoint image with landmarks 0, 2 and 3 shall be named as **waypoint\_1\_023.jpg**

Waypoint image with landmarks 1, 2 and 3 shall be named as **waypoint\_1\_123.jpg**

Four image files with partial landmarks (parLdmrkNos != 0) shall be edited to remove one landmark that does not appear in **parLdmrkNos** field. For example, for **waypoint\_1\_012.jpg**, landmark 3 should be removed from image file. This can be done by using simple image editor such as Paint (in Windows) or Pinta (in Ubuntu). For this, nearby background should be cropped and covered the landmark.

Remainder of pre-processing shall be completed by changing parameters in Python file *“ImProTest.py"*, in the main repository.

**Case 2: Waypoint with THREE landmarks**

When THREE landmarks are selected, only one set of data with all three landmarks are kept in the database. Hence, only one image should be created with following naming convention.

waypoint\_**WaypointNumber**\_0.jpg

where

**WaypointNumber** = waypoint number in navigation sequence

1. (b)

(c) (d)

Fig. 4.2: Partial landmarks of waypoint image of Fig. 4.1 with waypoint combinations (a) ‘012’; (b) ‘013’; (c) ‘023’; (d) ‘123’.

**4.1.2 Threshold adjustments:**

Thresholds are defined in following Python file: *ImProTest.py*. Refer to Python file for more description of these parameters.

**Case 1: Waypoint with FOUR landmarks**

Following thresholds shall be fixed in file *ImProTest.py:*

* NLdmrk = [4]

Following parameters shall be adjusted as per requirement (In first instance, try with default values)

* ImageTh: default value = 0.004
* BinaryTh: default = 110
* BinaryThIncDec: default = 5
* BwAreaTh: default = [200,2000]
* LdmrkAngTh: default = 5
* ImCropTh: default = [100,36]
* WpAngTh: default = 5
* WpSiRatTh: default = 0.1

Following parameters should be copied from navigation data:

* AltRef: Altitude of UAV when reference image was captured.
* YawRef: Yaw angle of UAV when reference image was captured.

**Case 3: Waypoint with THREE landmarks**

In the case where only THREE landmarks are chosen to be kept in the database, following parameter should be fixed:

* NLdmrk = [3]

All other parameters should be treated as it was in case-1 (with FOUR landmarks).

**4.1.3 Generating Data:**

Following parameters in *ImProTest.py* file shall be set for generating data:

* GenData = True
* parLdmrakNos = ‘0’

Run ImProTest.py file and observe if program runs successfully and produces following two lines of output at the end:

Data Successfully written to Database

IpCalDrift = [0.0, 0.0]

A number of binary images will be seen when image is being processed.

A numpy file with following naming convention will be saved in **Database** sub-folder in main repository:

Wp\_**WaypointNumber**\_Data.npy

where

**WaypointNumber** = waypoint number in navigation sequence

This process will generate data with all FOUR landmarks in the image.

Once the data is generated with all FOUR landmarks, data shall be generated for four different cases of partial landmarks (**parLdmrkNos** = ‘012’, ‘013’, ‘023’ and ‘123’).

As all data for all cases will be saved in same data file, data for all FOUR landmarks should always be generated first. This initializes fields for saving data for partial landmarks. Next, data for partial landmarks should be generated. Data for FOUR landmarks should not be generated generated again as this will reset data for partial landmark cases. It is recommended to change value of **parLdmrkNos** one at a time.

Once these steps are completed data will be saved in the following location: **main-repo/Database**.

**4.2 Navigation with Image Processing**

Once landmark and waypoint data are saved in correct location, navigation can be performed such that the UAV navigates over visual waypoints. Python navigation code can be run under three different scenarios:

* Simulation
* Hardware in the Loop (HiL)
* Flight Test

*VisNav.py* Python file in the main repository shall be used to perform all navigation tasks. All three cases can be attained by changing parameters in *VisNav.py* file. Brief description of these parameters can be found in the Python file itself.

In following subsections, different navigation scenarios will be described in further detail.

**4.2.1 Simulation**

**Parameter Setting**

Refer to *VisNav.py* Python file for more description about these parameters.

|  |  |
| --- | --- |
| **totTrials** = **1** | This means ONE experiment is conducted in per program run. |
| **useRPi** = **False** | This means Raspberry Pi is not used during experiments (i.e. program is executed in **Simulation** mode) |
| **HiL** = **False** | This parameter has no effect in when **useRPi** flag is set to **False**. |
| **ImgPro** = **False** | When set to **False**, drift calculated by image processing will not be used. In open loop simulation drift cannot be calculated. |
| **AutoWpTh** = **10** | This is set to 10 for **Simulation** experiment. This parameter is useful while conducting **HiL** or **Flight Test** experiments in collecting images from pre-flight tests. |
| **RollCdMax** = **0** | Not used in **Simulation**. |
| **TiOutTh** = **2** | Not used in **Simulation**. |
| **AutoWps** = **[ ]** | Not used in **Simulation**. Set to empty matrix ([ ]) |
| **VisualWpsSim**  example:  [[0,50],[50,50],[50,0]] | 2D matrix containting XY co-ordinate position of visual waypoints used in simulation. These waypoint coordinates are relative to starting point of simulation. Starting point of simulation is always [0,0] |
| **VisualWps** = **[]** | Not used in **Simulation**. For **HiL** and **Flight Tests** GPS coordinates (latitude, longitude) of visual waypoint to 7 decimal places |

**Test Image**

Once parameters are set, for each landmark one test image will be required. For the purpose of simulation, images already used to generate landmark and waypoint data could be used. However, they should be named with following naming convention:

waypoint\_**waypoint\_number**\_test.jpg

where,

**waypoint\_number** = waypoint number in navigation sequence

These files should then be placed in following directory: **main-repo/FlightData/Simulation**.

**Running the Program**

Simulation is performed by running *VisNav.py* Python file.

In HiL and flight tests, after initial launch, the UAV is flown in Auto mode before switching to fly by wire mode for autonomous vision-based navigation. In case of simulation, there is no such switching is possible to simulate. Hence in the simulation mode, program is wait to make user input as keyboard ‘Enter’. This action is similar to the switching performed in HiL and flight tests. Position of the UAV is set to [0, 0] in XY co-ordinate when **Enter** button is pressed in the keyboard and other visual waypoints co-ordinates are relative to this.

**Program Output**

The console show up a number of messages including outputs generated from by Image processing algorithm. As the program enters in two different threads while performing image processing, output from image processing thread is mixed with navigation thread in the console.

Following output will show up in the final line of the console if everything goes wright with the program:

Navigation has been successfully accomplished. Data log files have been saved.

In case there are problems in image processing section following error message will be displayed in the final line of the console:

IMAGE PROCESSING ERROR: Navigation has been aborted.

In case if there are problems in some other places in the code following error message will be displayed in the console.

NAVIGATION INCOMPLETE ERROR: Did not pass through all wps.

1. Image Processing Functions

This section describes image processing functions which is used to for implementing landmark detection algorithms.

4.1 MATLAB Image Processing functions

In this subsection, MATLAB functions for image processing algorithms are described.

4.1.1 ImageProcessing.m

This is the main image processing function. In other words, it is the entry point to the image processing functions.

1. Main thresholds of image processing are defined at the beginning of this function.
2. The **LandmarkDetection** function is called which performs all landmark detection operations, and then returns detected landmark locations and reference point.
3. The **WaypointDeteciton** function is called which checks polygonal waypoint features against those in the waypoint library.
4. Finally position drift is computed based on computed reference point and the reference point stored in the database.

4.1.2 LandmarkDetection.m

This function performs landmark detection operations.

1. Firstly, adaptive binarization is performed which extracts a number of candidate landmarks.
2. **LandmarkLocalization** function is called which uses geometrical features to match landmark features. This function returns a group of landmarks.
3. Group of landmarks returned by **LandmarkLocalization** function is tested against feature signatures stored in database by using **FeatureSignature** function.

1. Simulation with MATLAB and POVRay

5.1 Introduction

This section describes procedure for obtaining a closed-loop simulation of vision-guided UAV navigation in the presence of wind drift and noise. Persistence of Vision Raytracer (POVRay) is used to render natural looking terrain which also simulates camera. POVRay is then integrated inside 6DoF Kinematic MATLAB code to formulate closed-loop simulation model. First, MATLAB image processing functions will be described.