Contents

Vison based Line Tracking Algorithm

1 Introduction:

1.1 Objectives:

This project, A Vision-based Line Following Method for Micro Air Vehicles has two main objectives:

- First, colour tracking algorithms are developed in simulation using MATLAB/Simulink.
  To do so, different computer vision approaches will be applied to the image of the drone's down-facing camera.
- Second, proposed algorithms will be tested in the minidrone in a real setup scenario.

In order to meet the thesis objectives, the following aspects are investigated:

- Deploy the computer vision tool for line follower algorithm.
- Efficient algorithm for planning the path in the 3D space (smooth, time of flight).
- Understanding the structure of the Simulink Support Package for Parrot Minidrones.
- Perform the flight on the Parrot Mambo Fly in the real setup scenario.

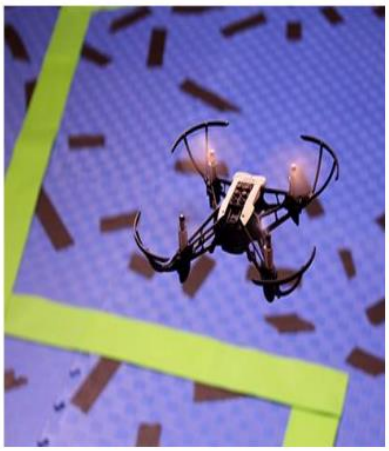

Figure 1 Minidrone and a coloured path.

1.2 Project specifications:

- The drone should follow a track laid on the arena and land on the circular marker in the shortest time, see Fig. 3. In Fig. 4, the landing on the circular marker is unsuccessful.
- The track will be divided into multiple sections. In addition, the track will have only straight lines and no smooth curves.
- The arena is a 4x4 meter space enclosed by nets on all sides.
- The lines will be 10 cm in width.
- The landing circular marker will have a diameter of 20 cm.
- The angle between any two track sections may be between 10 and 350 degrees.

• The track may have between 1 to 10 connected line segments. The initial position of the drone will always be at the start of the line. However, the front part of the drone may not always face the direction of the first line on the track.
• The distance from the end of the track to the centre of the circle will be 25 cm, see Fig. 2.
• The background on which the track will be laid may not be a single colour and will have texture.

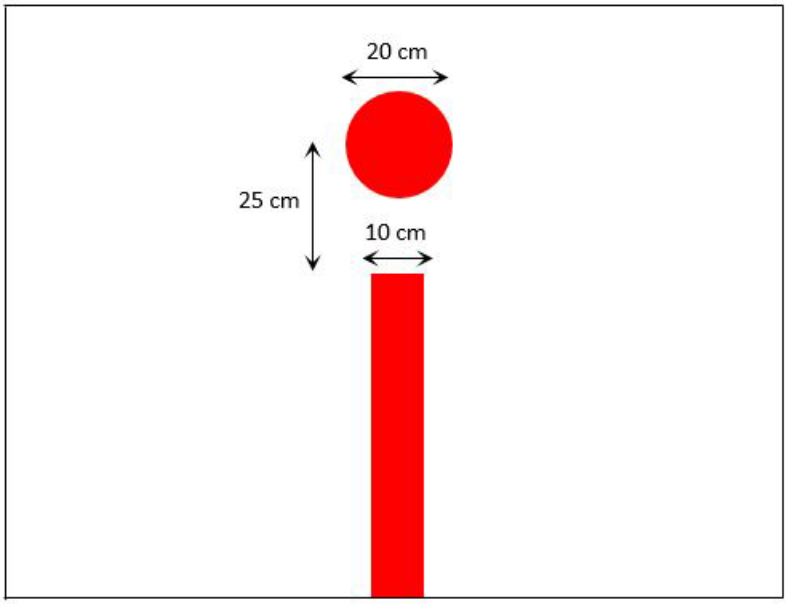

Figure 2: Arena details.

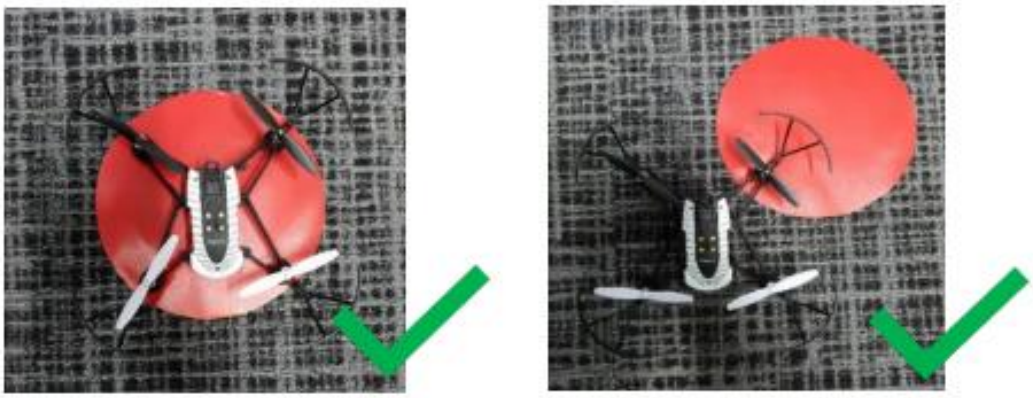

Figure 3: Successful landings in the circle. In the centre of the circle or by touching it.

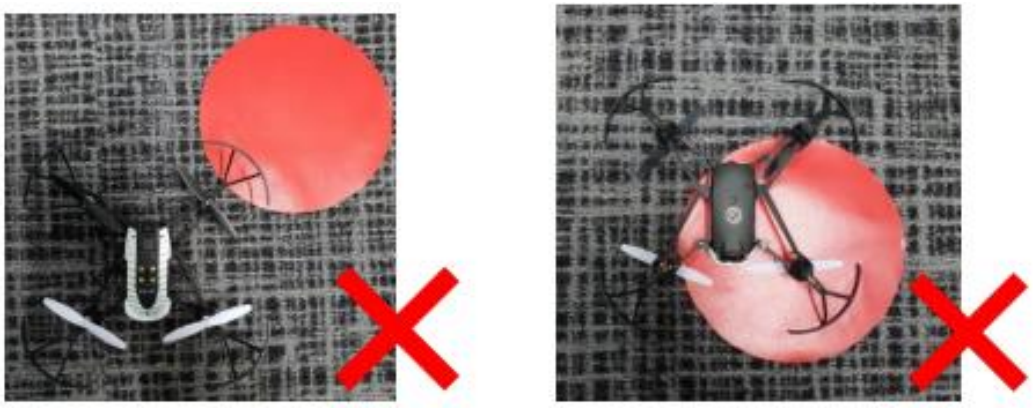

Figure 4: Unsuccessful landings in the circle. Landing out of the circle or upside down.

1.3 About Simulink Support Package for Parrot Minidrones:

It is a closed looped feedback system where the plant (Drone) receives the commands from the controller (Software Algorithm). The feedback is in the form of sensor data.

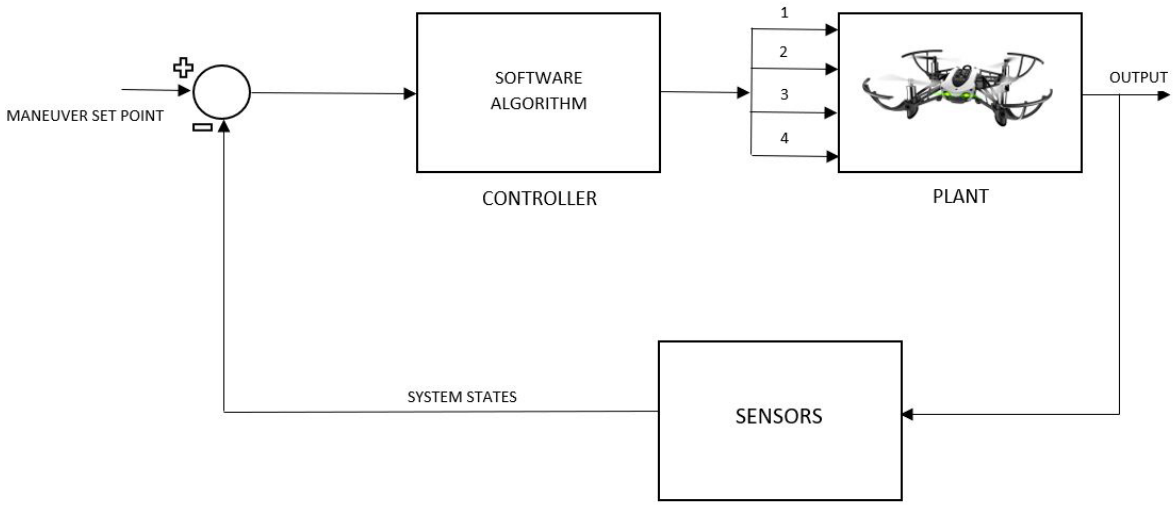

Figure 5 Working of the Simulink Support Package for Parrot Minidrone.

As illustrated in the Figure 5, the drone receives command signals for each of the four motors of the Quadcopter.

The motors in turn control the degrees of freedom namely:

1. Pitch
2. Yaw
3. Roll

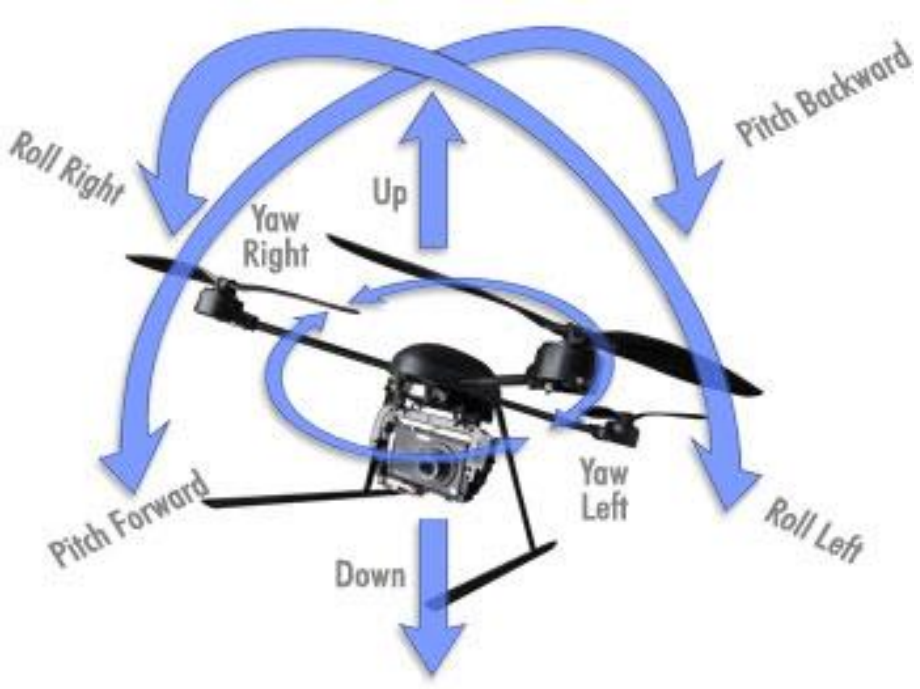

Figure 6 Degrees of Freedom of a Quadcopter

The software Controller is provided by MathWorks in the form of Support Package for Parrot Minidrones [1]. This toolbox lets us build and deploy the flight control algorithms on Parrot Minidrones.

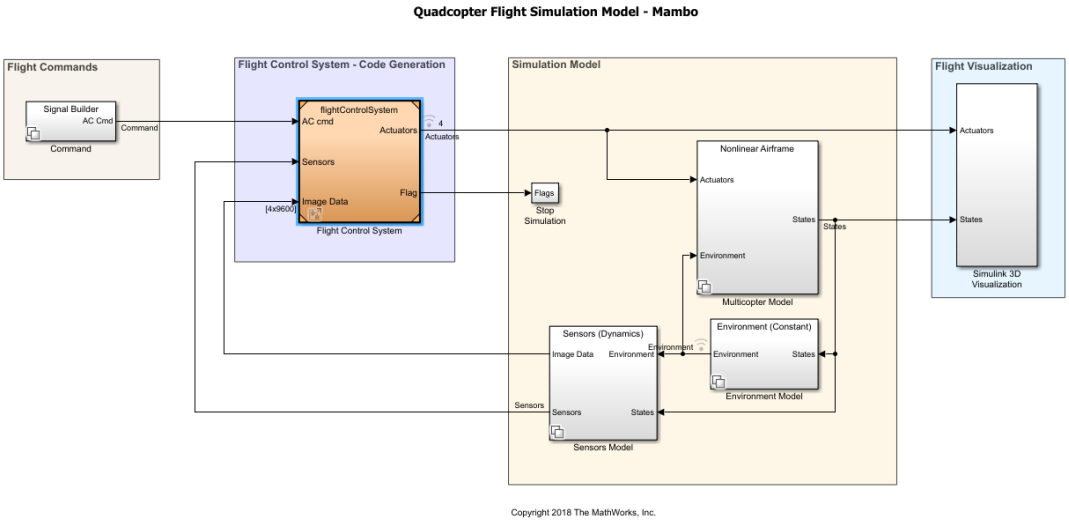

Figure 7 The Image illustrates the Simulink Support Package for Parrot Minidrone provided by MathWorks.

This program provides a simulation model that takes into account

1. Airframe: Mathematical model of the Quadcopter
2. Environment: Controls the environmental parameters like Gravity, Pressure, etc.
3. Sensors: Processing of obtained sensor data
4. Flight Commands: Dynamic commands from Algorithm or static commands from Signal builder, joystick, .mat data or spreadsheet data.
5. Flight Control System: The Image Processing System and the Path Planning block must be designed and implemented inside the Flight Control System.

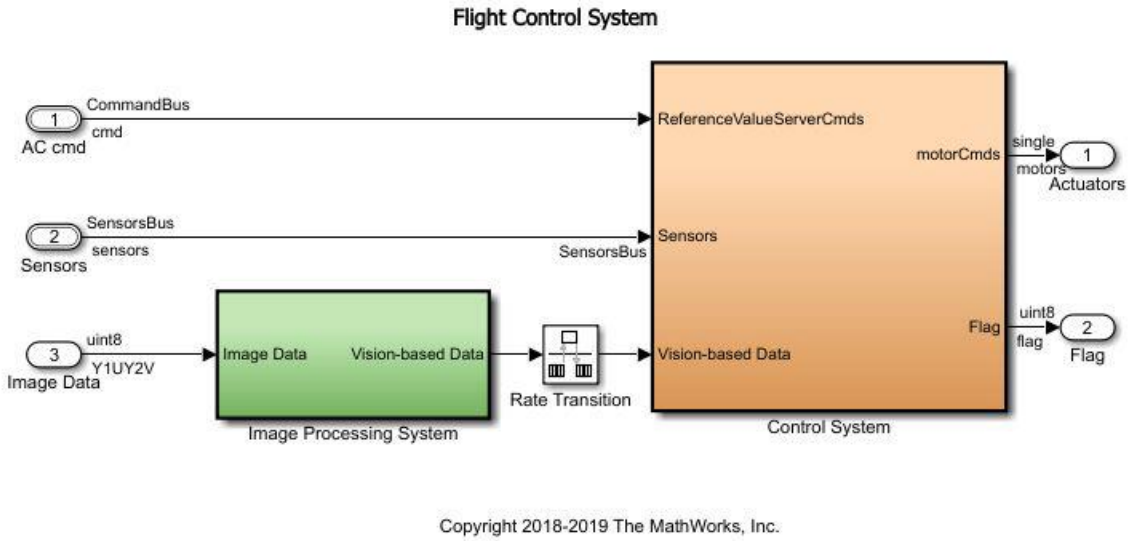

Figure 8 The Flight Control System Module containing the Image Processing and Path Planning Algorithms

The Model receives inputs as:

1. Image from Onboard Camera
2. Coordinates of the Position (x, y, z)

1.3 Work Plan

The work plan consists of 5 Phases:

1. Getting Started: Phase 1 is concerned with getting acquainted with the software environment that will be used during the competition.
2. Literature Review: In this phase, theoretical concepts and previous works were inspected.
3. Development 1: Develop and test the coloured line following method using computer vision.

4. Development 2: Develop and Test the Path Planning algorithm, Integrate the algorithms that were developed in Live Script into Simulink Environment and debug the MATLAB code was debugged for the C++ code generation.
5. Testing: Perform simulations and experiments by combining the path planning algorithm with the existing controller in the Simulink package.

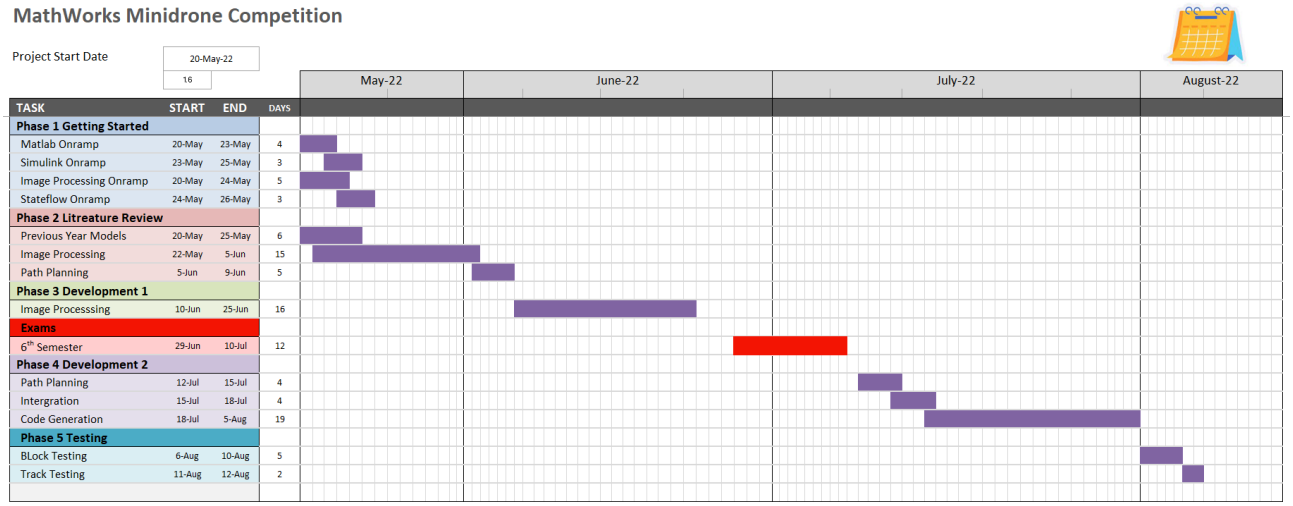

Figure 9 Gantt Chart

2 Methodology

The Algorithm detects the corner points from the binarized images. The points are then filtered, sorted and stored. These points then are fed to template which in turn generates motor commands.

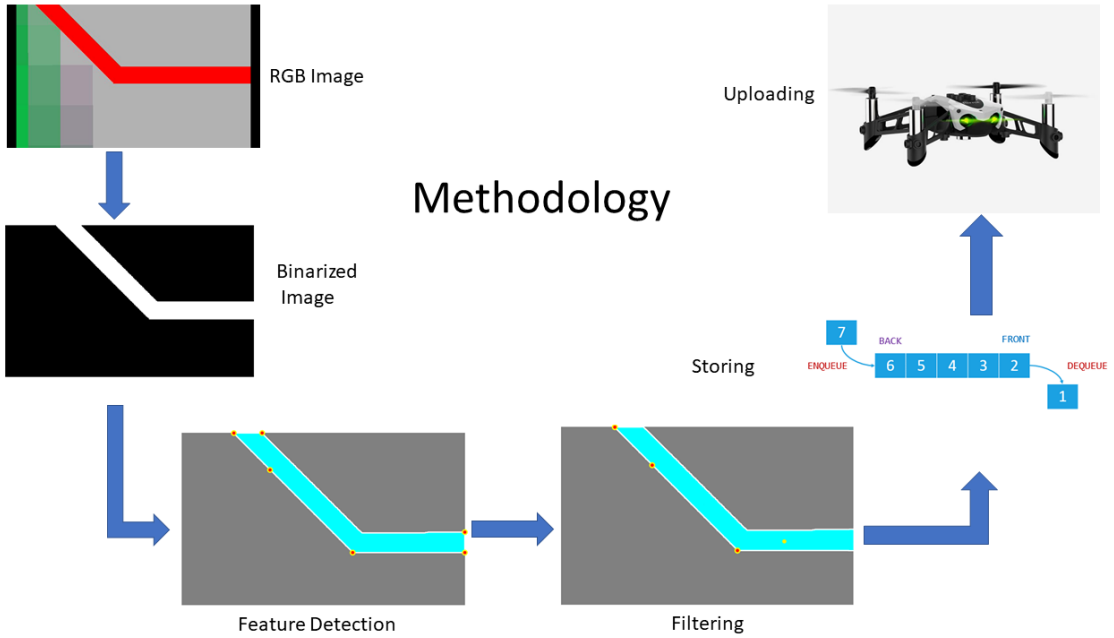

Figure 10 Methodology for the Algorithm Development

2.1 Image Processing

The onboard camera has resolution of 438×720p. This resolution is resized to resolution the of 120×160 by the Simulink Support Package. As the algorithm can work efficiently even in low resolution. The image is further reduced to 60×80-pixel resolution for improved performance.

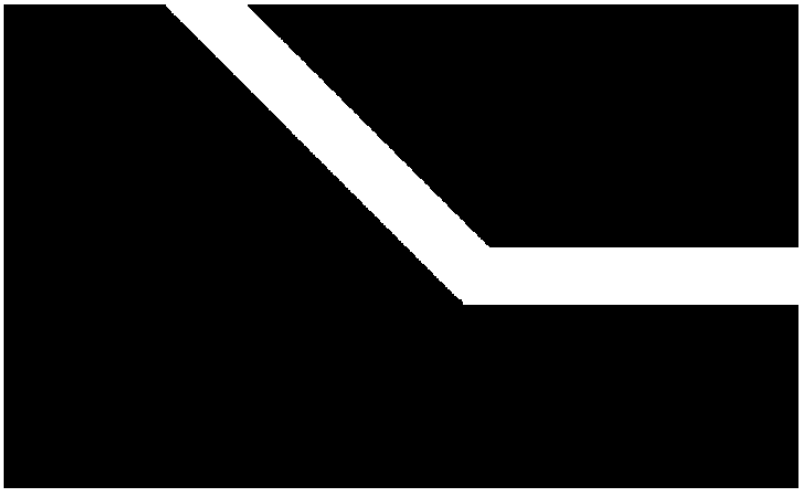

Figure 11 The Image illustrates binary image of on-board camera image.

2.2 Feature Detection:

2.2. bwboundaries ():

bwboundaries () [2] function is used to identify the blobs in the images and trace the boundaries of the blobs. It also efficiently detects the noisy blobs. The redundant blobs are filtered. Detected points are white. The same function is used to detect circular blob (Landing Marker).

Figure 12 Detected Points are displayed in white colour.

2.2.b cMin Max:

cMin Max [3] [4] algorithm is based on projecting the boundary points on the axes and extracting the points which subtend minimum and maximum intercept value. These points represent the corners of the figure. The image is rotated after each search to get the remaining points.
The Advantage of this Algorithm are:

1. Accurate and Precise Corner Detection.
2. Simple and Reliable.
3. Image processing like Adjusting, Dilatation, Eroding **not** required.
4. High noise filtration.
5. Works efficiently on Low Resolution Images.

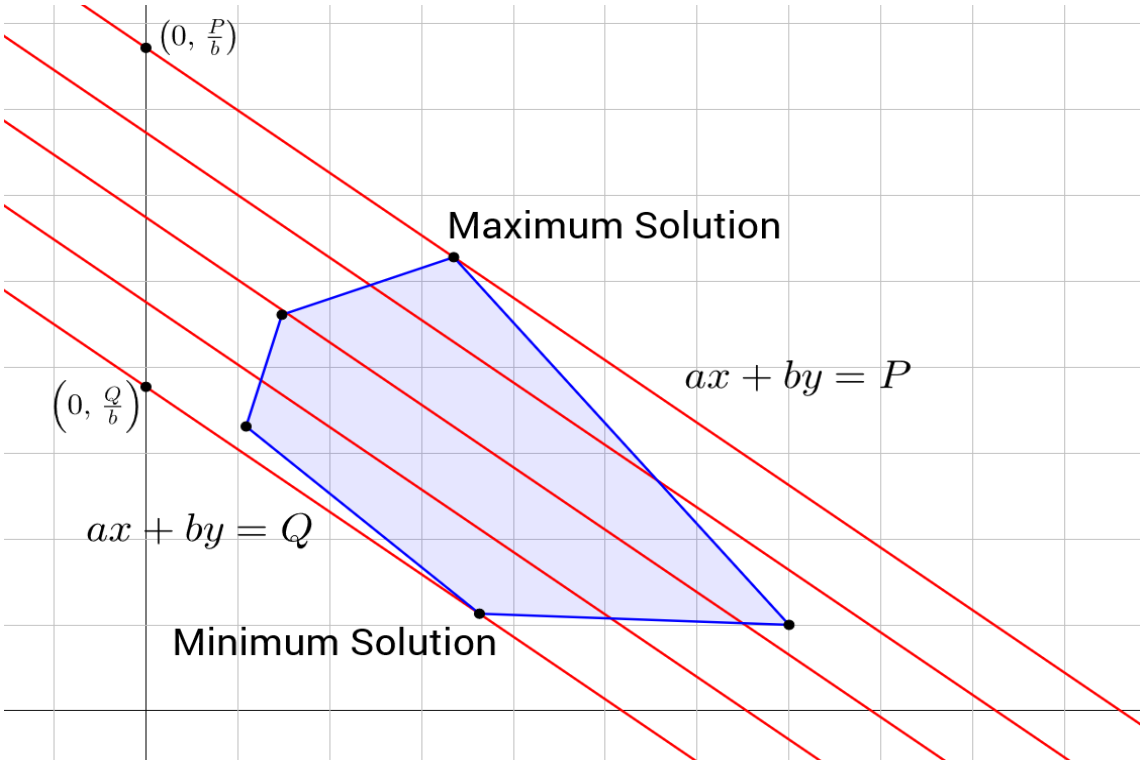

Figure 13 The corner points produce the minimum and maximum intercept on the axes

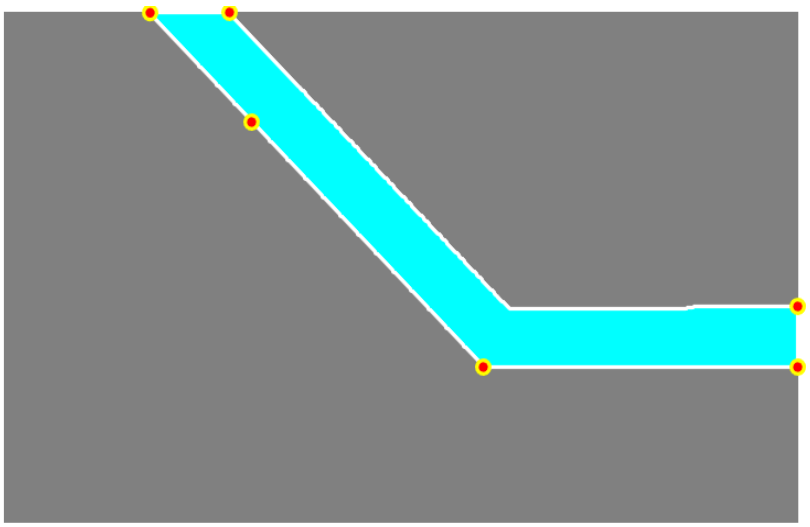

Figure 14 Highlighted points are detected corners whereas white coloured entities are the boundaries of the polygon.

2.3 Filtering:

All the points detected need not be sent to quadcopter to follow as some points are:

1. Redundant.
2. Already passed by the Quadcopter.

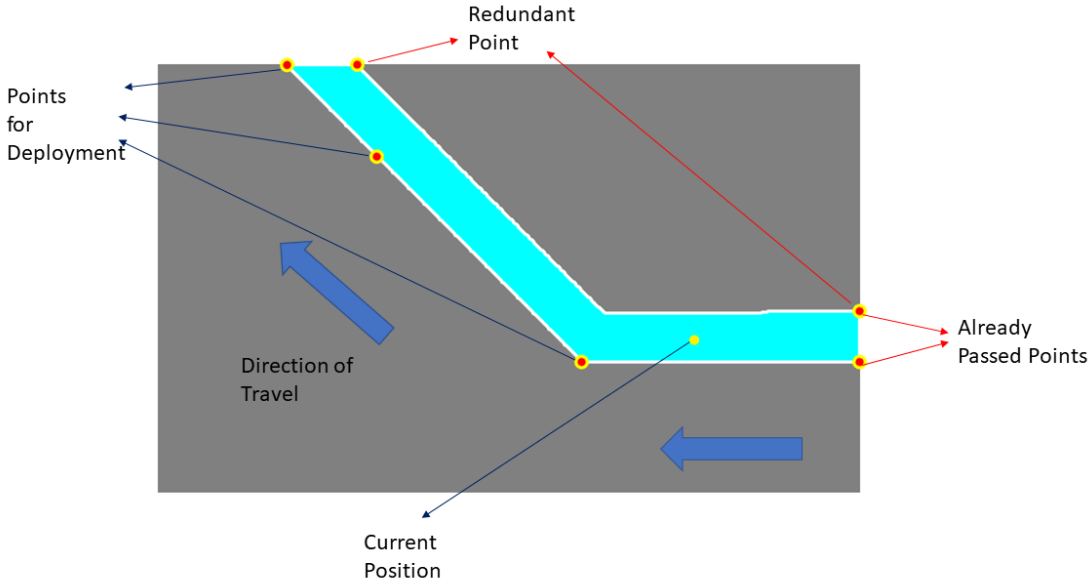

Figure 15 Illustration of the points to be filtered

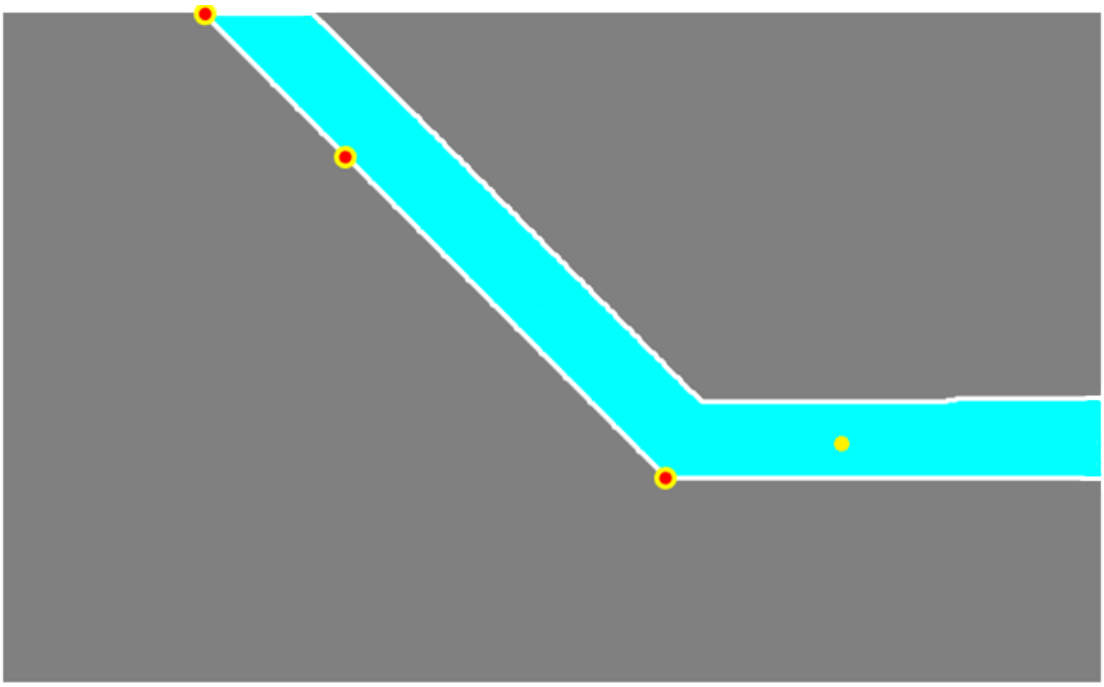

Figure 16 Filtered Points

2.4 Storing and uploading

1. The filtered points are uploaded to Quadcopter to follow. These points can change according to the states. Each state outputs the global coordinates (x_plan, y_plan) to follow.
2. Error between current position and planned points is monitored. If error reaches certain threshold, points are updated.
3. If all the points are exhausted, a fresh array of points is supplied by the Image Processing System.

The x_plan, y_plan, z_plan differs for each state like Hovering, Tracking and Landing. This variation is facilitated by the MATLAB's Stateflow. The states used are:

1. Gain_altitude: This state sets the height of 1.1m.
2. Tracking_2_corner: In this state, the 'previous_positions' are created.
3. Tracking_4_corners: This is the state where the quadcopter will operate most of the time.
4. Landing: This state is activated after the circle is detected. The quadcopter lands at the centre of the circle.

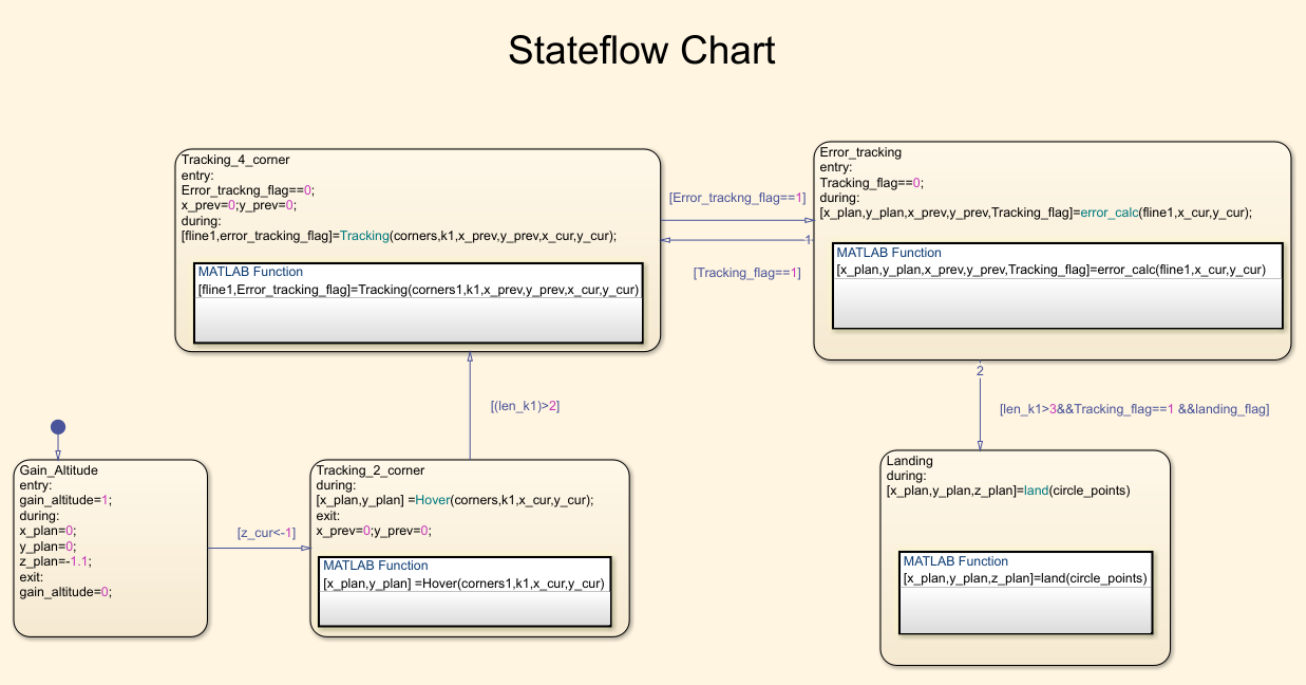

Figure 17 Stateflow Chart

3 Conclusion

An efficient and low computational powered solution is set to generate the spatial coordinates of the target Points to fulfil the competition requirements successfully. The Algorithm can work efficiently with noisy data and low-resolution images which in turn drastically decreases processing time.

4 Future Work

The Algorithm can store the detected points in the first attempt and then use it to as pre-set target points eliminating the Image Processing block for the consecutive attempts. Bypassing the Image Processing block will further decrease the computational which in turn make improve the Completion time.

5 References

[1] Simulink Support Package- https://in.mathworks.com/hardware-support/parrot-drone-matlab.html

[2] bwboundaries() - https://in.mathworks.com/help/images/ref/bwboundaries.html

[3] cMin Max - https://arxiv.org/abs/2011.14035

[4] cMin Max- https://in.mathworks.com/matlabcentral/fileexchange/74181-find-vertices-in-image-of-convex-polygon

6 Appendix

6.1 Simulink Model

6.1.1 Image Processing

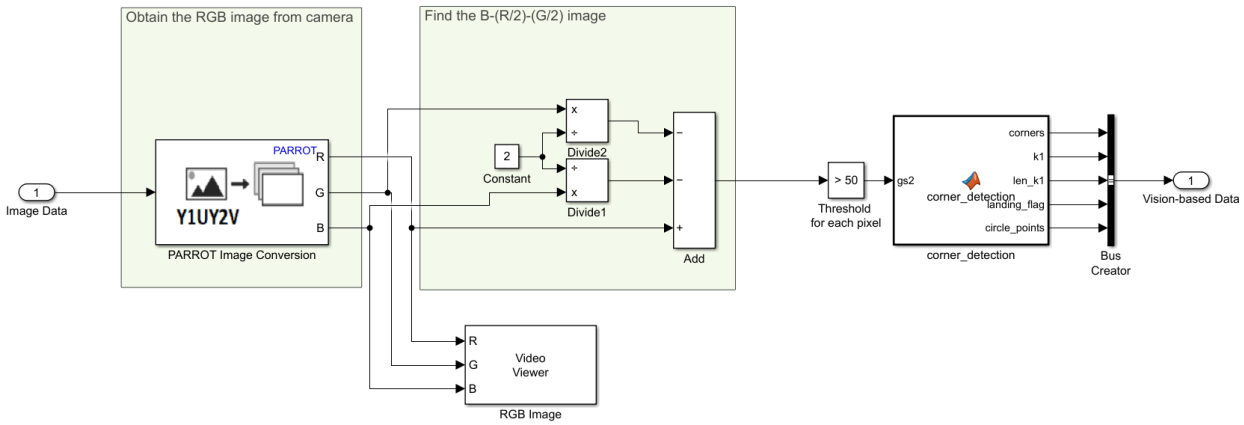

Figure 18 Image Processing Block

6.1.2 Path Planning

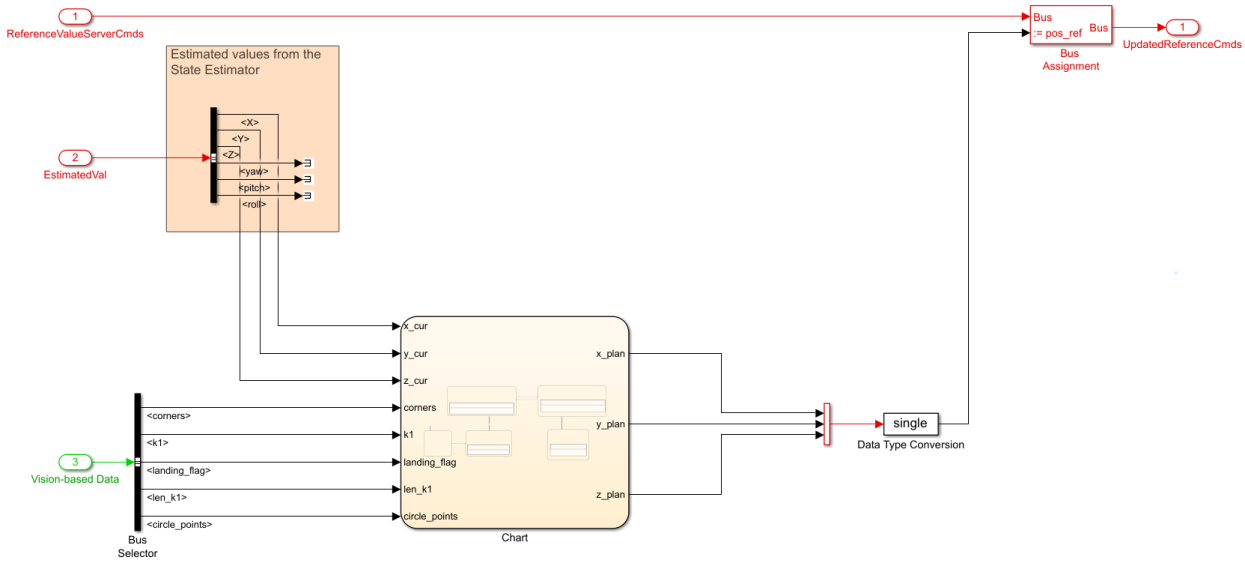

Figure 19 Path Planning Block

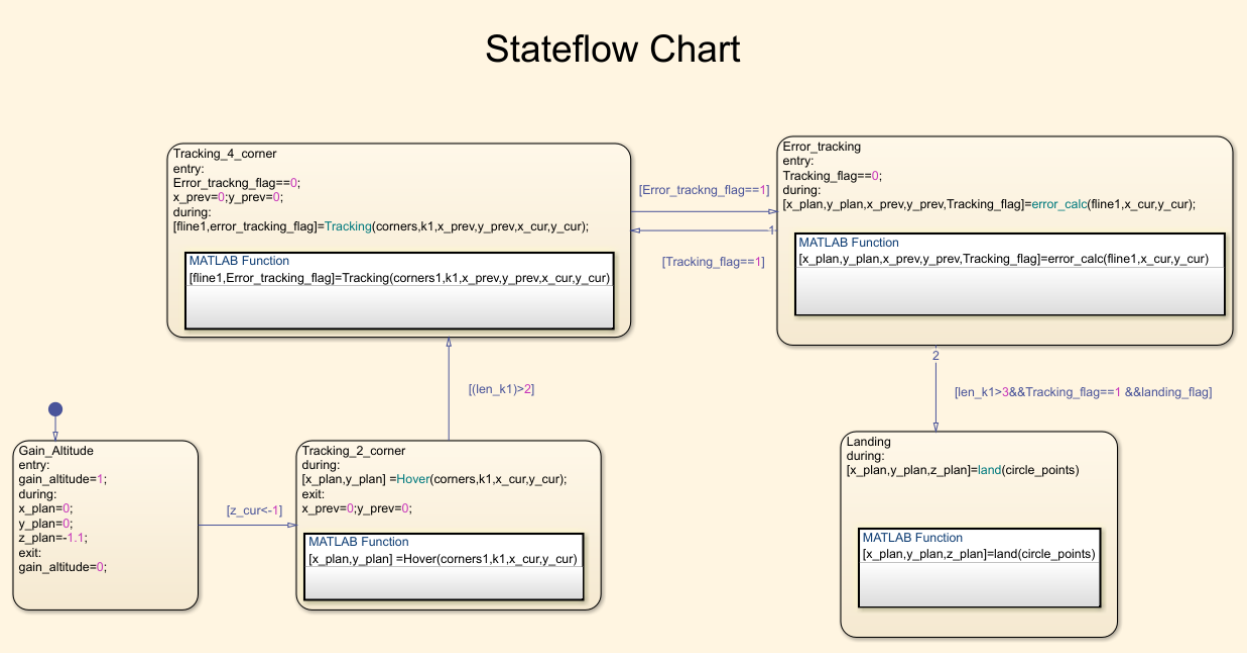

Figure 20 Stateflow Chart

6.2 MATLAB Functions

6.2.1 cMin Max Function:

```
theta=linspace(0,360,N+1); theta(end)=[];
c=nan(size(theta));


for i=1:N
    [~,c(i)]=max(IJ*[cosd(theta(i));sind(theta(i))]);
end


Ih=IJ(c,1); Jh=IJ(c,2);


[H,~,~,binX,binY]=histcounts2(Ih,Jh,k);
 bin=sub2ind([k,k],binX,binY);


[~,binmax] = maxk(H(:),k);


[tf,loc]=ismember(bin,binmax);


IJh=zeros(360,2);
IJh(:,1)=Ih(tf);
```

```
IJh(:,2)=Jh(tf);
G=loc(tf);

C = zeros(max(G),2);
for ii = 1:max(G)
    C(ii,1) = mean( IJh( G == ii,1 ) );
    C(ii,2) = mean( IJh( G == ii,2 ) );
end
[~,perm]=sort( cart2pol(C(:,2),C(:,1)),'descend' );
corners=nan(15,7);
corners(:,1:2)=C(perm,:)+centroid;
corners(:,3:7)=nan;
```

6.2.2 Point Filtration and Sort:

```
%%Detecting Border points
k1=nan(10,1);
k1(:)=find((corners(:,1)<2.5)|((corners(:,1)>58.5))|(corners(:,2)<2.5)|((corners(
:,2)>78.5)));
len_k1=length(intersect(k1,k1));


%%This function is used when all 4 border points are detected.
int=0;int1=0;idx=0;idx1=0;idxv=0;idx1v=0;pro=0;


%%Redundant border points are removed.
for j=(1:length(k1))
    if k1(j)==1
        if ismember(corners1(end,7),k1)&&ismember(2,k1)
            corners1(k1(j),1:7)=nan;
        end
    elseif k1(j)==corners1(end,7)
        if ismember(k1(j)-1,k1)&&ismember(1,k1)
            corners1(k1(j),1:7)=nan;
        end
    else
        if ismember(k1(j)-1,k1)&&ismember(k1(j)+1,k1)

            corners1(k1(j),1:7)=nan;
        end
    end
end


%%Points are transformed to global coordinates.
corners1(:,2)=80-corners1(:,2)+x_cur;
```

```
corners1(:,1)=60-corners1(:,1)+y_cur;


%%Points which drone has already passed are removed.
%%This is done by removing points closest corner point from current location
%% until next border points.
int= intersect(corners1(:,7),k1);
int1=intersect(corners1(:,7),corners1(:,7));


idx = knnsearch([y_prev,x_prev],corners1(int,1:2),'k',2);  % get two close points


idx(1) = [];


idx1 = knnsearch([60,80],corners1(int1,1:2),'k',2) ;  % get two close points


idx1(1) = [] ;




rem=corners1(int(idx),7);


if corners1(int(idx),7)==corners1(int1(idx1),7)
    rem=corners1(int(idx1),7);

else
    pro=proj([x_prev y_prev],[corners1(int1(idx1),2)
corners1(int1(idx1),1)],[x_cur y_cur]);
    if ~pro

        rem=[corners1(int1(idx1),7);corners1(int(idx),7)];

    end
end
corners1(ismember(corners1(:,7),unique(rem)),1:7)=nan;


if int(idx)>intersect(corners1(:,7),k1)
    sort(corners1(:,7),'ascend');
else
    sort(corners1(:,7),'descend');
end
fline1=corners1;
Error_tracking_flag=1;
```

```
function pro=proj(p1,p2,p3)
%% This functions returns whether he input point lies inside the given points or
not
    dx = p1(1) - p2(1);
    dy = p1(2) - p2(2);
    pro=0;
    innerproduct=(p3(1) - p1(1))*dx + (p3(2) - p1(2))*dy;
    if (innerproduct<=0)&&(abs(innerproduct)<=(dx*dx+dy*dy))
        pro=1;
    end
end
```

6.2.3 End Marker Detection:

```
if length(B)>1&&(len_k1>0)&&(len_k1<=2)
        landing_flag=1;

    for k = 1:length(B)
        boundary = B{k};
        r1=randi([1 length(boundary)],1);
        r2=randi([1 length(boundary)],1);
        me=mean(boundary);

        if (floor(dis(me(1,1),me(1,2),boundary(r1,1),boundary(r1,2)))-
floor(dis(me(1,1),me(1,2),boundary(r2,1),boundary(r2,2))))<(5/4.29)
 %%Circle is detected if the mean of the blob is equidistant from random
 %%points of the blob.
            circle_points=me;%%Circle mid-points

        end
    end
end
```