Airplane Runway Tracking

Micheal Dunne
Applied Computing Student
Waterford Institute of Technology
Waterford, Ireland
michealdunne14@gmail.com

***Abstract*— This document outlines the design and implementation of runway tracking for planes using MATLAB. This is an important tracking to help the pilot get a visualization of the runway as they are coming in to land. Vision recognition is used to find the runway and identify the number of the runway.**
Keywords – MATLAB, Hough Transformation, Edge detection, PolyShape, detection, OCR

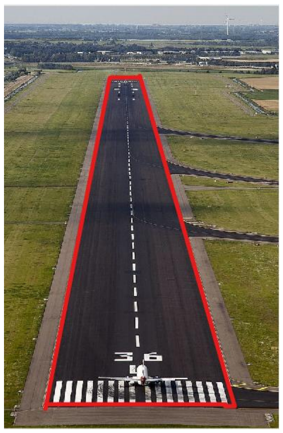
I. Introduction

In airplanes it can be difficult to locate the runway in certain situations. With some ADAS systems that could be used to assist the pilots as they are flying the planes. One of these features could be Lane Keep Assist. This ADAS feature could be used to help the pilot locate the runway by looking at a screen and to be able to clearly see where they need to land the plane. The main goal of this project is to be able to locate the runway as the plane is descending. The goals I hope to set out for this project are that I will be able to identify the runway clearly. I hope to be able to identify the number of the runway. Finally, I would like to be able to test does how well the system works in different weather scenarios.

II. Image Pre-Processing

A. Region of Interest

The area that I am interested in is the centre of the screen and I should not show any of the area around the runway. From the diagrams below we can see from the original image that lines could be detected in the horizon of the image. I will need to filter this out. The area that I am mainly looking at is the runway area. The filter should not take in to account anywhere outside that region. The second area that I am interested is the number that lies at the start of the runway.


Region of Interest

Original Image

Approach 1

III. Image Filtering

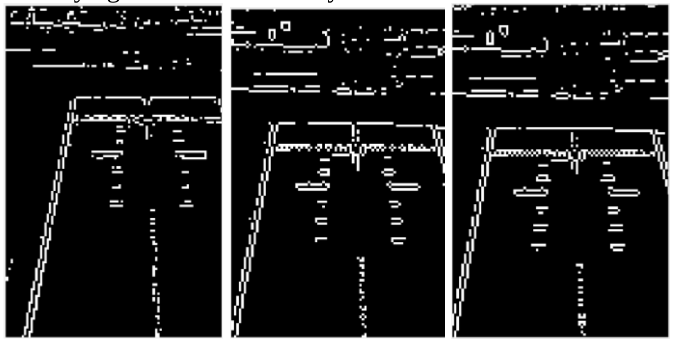
A. Grey-Scaling

The first step for the filtering is to make the image grayscale. The reason why this is needed is because for edge detection the image needs to be two dimensional. The first dimension is rows and the second dimension are columns [1]. If color is added this would make the image three-dimensional and when doing filtering it does not work with an image in color for this reason.

Grey Scaled Image

IV. Edge Detection

The next thing was finding the edges on the image. What this does is it extracts edges from the image that are likely to become an object edge point [2]. The are many different approaches to edge detection such as Sobel, Roberts and Prewitt. Sobel is the most commonly used edge detectors. The Prewitt operator is like Sobel operator and it is used for detecting vertical and horizontal edges in images [2]. From the examples below and doing some testing on which one preformed best I found that using Roberts was the best. The reason why I believe Roberts was the best for my tests was because the top of the runway shows the line not being completely straight in Sobel and Prewitt. Roberts shows that the line is completely straight, and it gave me the cleanest result out of the three. When I applied the result to the identifying lines it worked very well.


Roberts *Sobel* *Prewitt*

V. Hough transformation

The Hough Transformation detects lines in an image. The way that it works is it lets each point on the image vote. What this voting does it helps to figure out prominent lines in the image [3]. In the image below it shows how the it identifies the points in the image. The image only needed to take in to account only certain values. The lines that I was only interested in was the near vertical lines and the near horizontal line. I then needed to find the peaks of the Hough transformation result. The last step of using Hough transformation was to filter out lines shorter than a certain length. The result of Hough transformation can be seen below in Fig.1

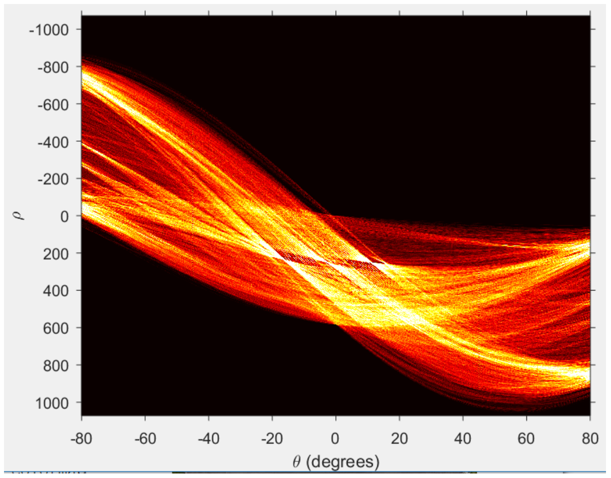


Figure 1

VI. Writing Lines

For writing the lines I looped through each result from the Hough Lines. I then got the endpoints of my result and added them to the image. The result can be seen in Fig.2. When this was applied to the image it worked very well.

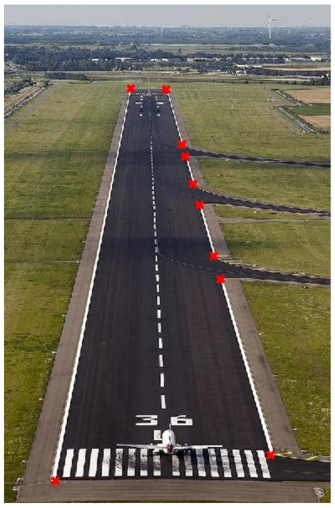


Figure 2

The next step was to plot lines between the two points. Each one of the points in fig.2 is connected to one endpoint. This result can be seen in Fig.3

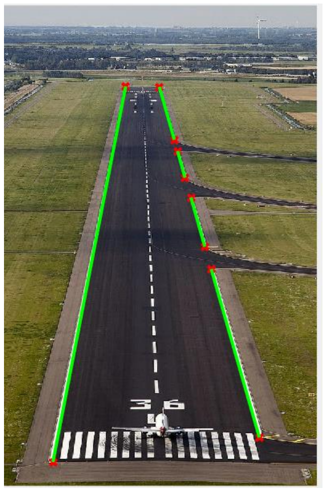


Figure 3

Doing this was nice and it worked well but a problem arose was that each point only had one end point. This meant that none of the points could be connected horizontally to each other which was a problem. I tried some solutions such as using K Nearest Neighbor. Which would get a point on the graph and find the nearest point. The way that this would work is by using dsearchn which would find the nearest point to the current point. I ran in to a problem where it would search and find the nearest point, but I could not connect the green point and the red point together. I could connect the red points together and the green points, but this was not the result I was looking for. An example of dsearchin working is shown in fig.4.

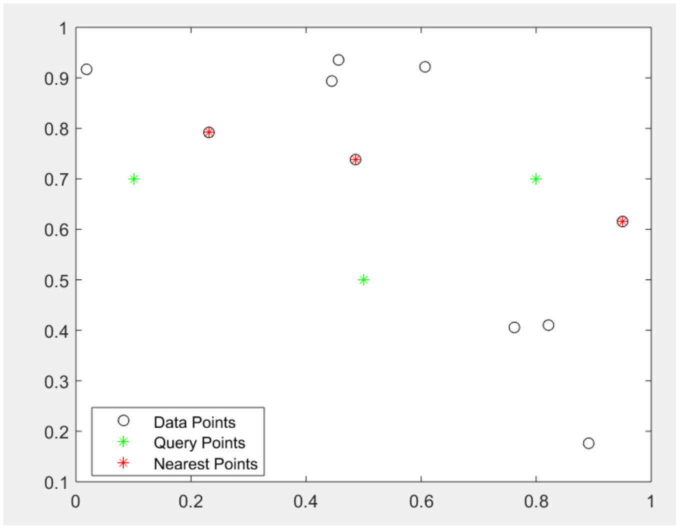


Figure 4

The next step was not to connect point to point but instead draw lines straight threw the gaps that were appearing. I got a good result out of this as seen in fig.5. The way that the Hough works is that it only deals with one angle. I dealt with the vertical lines first. I then done the same process with the horizontal lines. I got a good result and it can be seen in Fig.6.

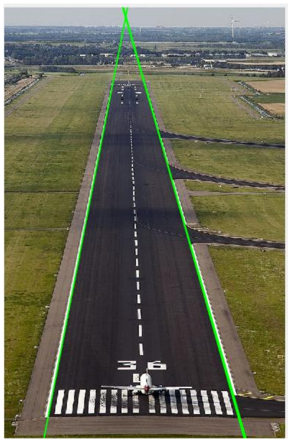


Figure 5

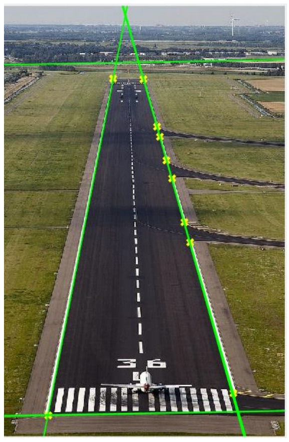


Figure 6

VII. Conclusion to Approach 1

In conclusion to this approach I found that it worked well and that you could clearly see where the runway is. When I applied this to a video it worked very well. The problem with this approach is that it does not completely find the runway. There are lines going beyond the length of the runway which is not good. I hope to resolve this problem in Approach 2.

Approach 2

VIII. Introduction

In approach 2 I used some of the same components that where in approach 1. These include the Hough transformation, Gray Scaling, edge detection. This approach will focus more on trying to resolve some of the issues that arose while doing approach 1.

IX. Image Filtering

For image filtering I used grayscale, imbinarize and imtophat. What imbinarize does is it binarizes a 2-D grayscale image by threshold [4]. When I done this, I began to get better results with other images as this would filter out the background and all points to be more focused on the runway rather than the area around the runway. The result can be seen below when imbinarize is applied in fig.6. The imtophat preforms top-hat filtering [5]. The result from the imtophat can be seen on fig.7. This imtophat is then applied to the edge detection.

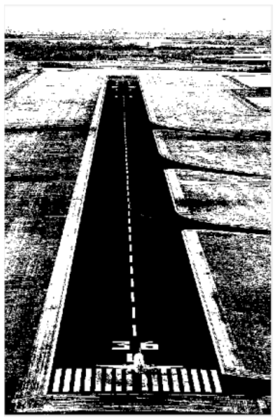


Figure 6

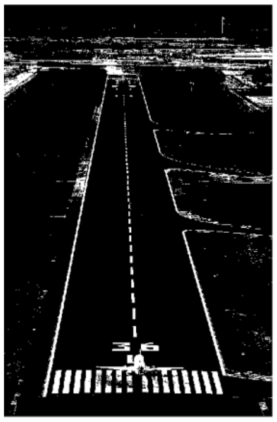


Figure 7

X. Polyshape

When every point was gotten the next step was to add a polyshape to the image. What was required for this approach to work was to identify the correct coordinates for the polyshape to connect too [6]. The way that I approached this was by first adding each point got from the Hough transformation lines to an array. The next step was to find the correct points that I was looking for. This was done by searching threw each value in the array. I first multiplied the coordinates of the X and Y coordinates together. This would gives me one value. I then found the three smallest values in the array. These three would get a corner of the runway. I then got the largest value out of all the coordinates. This would give me the final point of the runway. I then set these coordinates as the four points of the runway. The results that I got where good. The result can be found on fig.8.

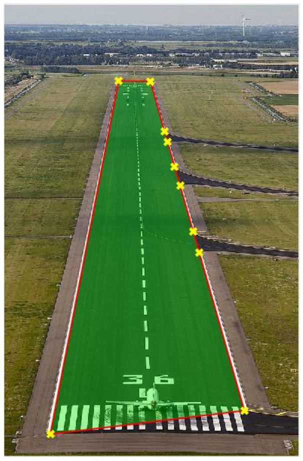


Figure 8

Once this image was working, I then applied it to a couple more images to see what result I would get. These results can be seen below:

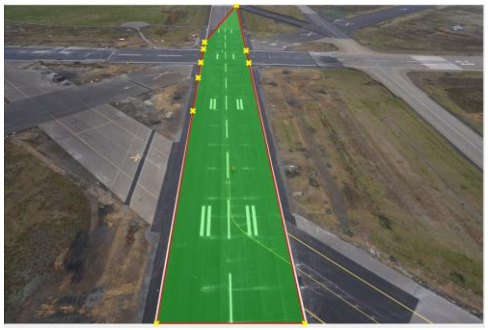


Figure 9

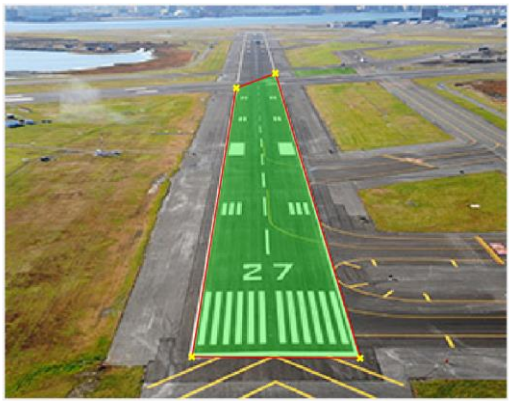


Figure 10

When I applied it to other images, I was happy with my results the only issue was that it may not cover the entire runway as the distance to the end of the runway was too far away.

XI. Conclusion to Approach 2

Approach 2 works well as it gets a perfect box around the runway. The only problem that if that the four points don't go to each corner of the runway it does not cover the entire runway as seen in fig.10. By doing this approach I resolved the issue that came about in approach 1.

XII. Runway Number

In different runways there might be multiple runways. The pilot will want to be able to identify the runway that they need to land on. The way that I identified the text by filtering the imtophat image with imerode which would then erode the image [7]. This would clean up the image seen in fig.11. After this was done then imreconstruct was applied to the image. What this does is it reconstructs the image using both the image from imerode and imtophat [8]. The result of this can be seen in fig.12. This is then applied to an imbinarize image again to filter out more of the image. The result of this can be seen in fig.13. The result searches for numbers using OCR. OCR recognizes text that is used in the image [9]. The result of this is then filtered down to only show the number values that it gets back.
The last step is to draw boxes around the result of number. This can be seen in fig.14.

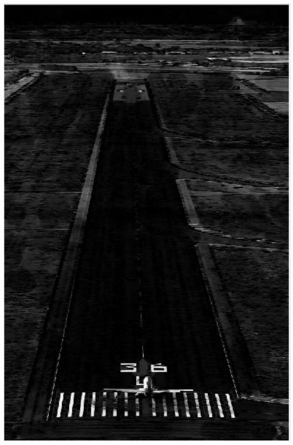


Figure 11

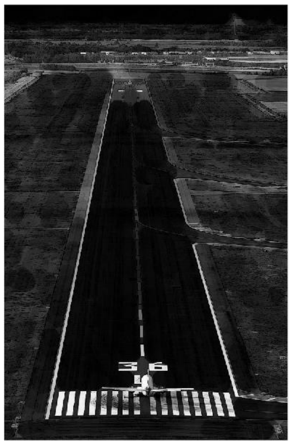


Figure 12

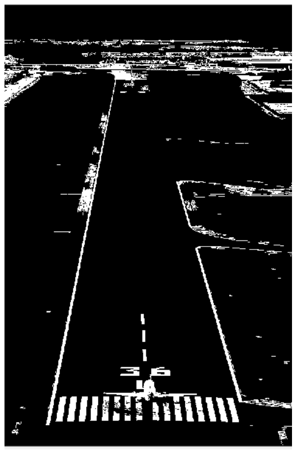


Figure 13

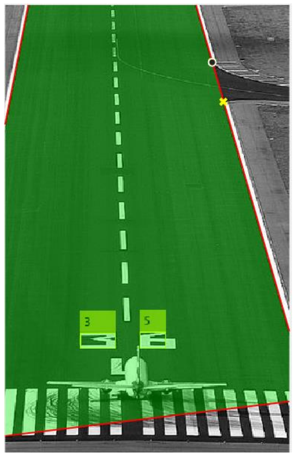
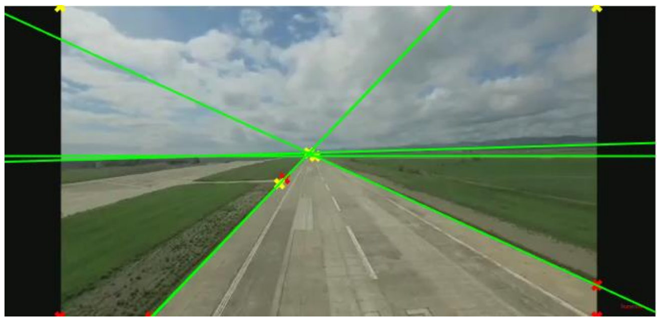


Figure 14

The result that I got was good as it identified that the number locations where correct. [10]

XIII. Video

I applied each approach to a video. The result I got from approach one was good and it worked well, and example of the video can be seen below. When I tried to apply the video to approach 2 it did not work as expected.


Approach one

XIV. Different weather scenarios

The weather while flying a plane is not always clear skies. I tested how the system will hold up under different scenarios. The way that I done this was by adding noise to the image. The way I will do this is by using imnoise. What imnoise would do is that it would make the image noisy by adding random dots on to the image. The result of this can be seen in fig.15. What would happen when I done this was that it would not be able to identify the runway. I added a filter to try and combat this. The filter that I used was called wiener2. What this does is that removes the noise from the image. This will be the way of combating the bad weather. When I applied this to result, I got the expected result. The result can be seen in fig.17.

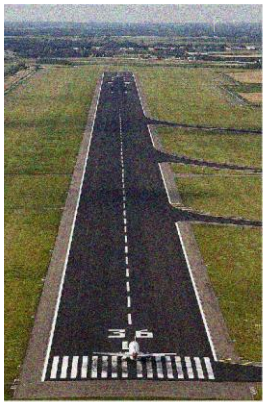


Figure 15

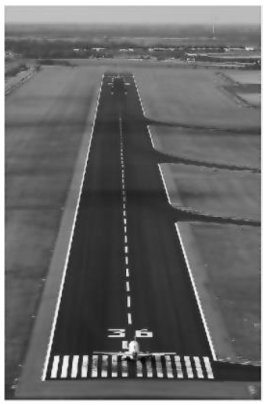


Figure 16

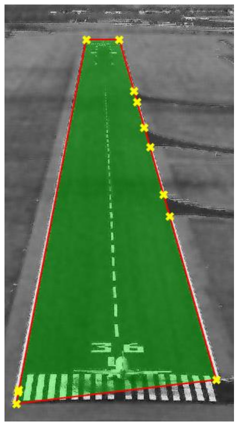


Figure 17

XV. Potential Improvements for the future

To help make this system work better for pilots. It needs to be more accurate and not return any false values. It will need to make sure that it identifies the runway numbers correctly. The system will need to preform faster than it currently does while working with a video.

XVI. Conclusion

Doing this project, I have completed all the tasks I set out to complete. These include being able to identify the runway, testing how well the system worked under different weather scenarios and identifying the number on the runway. There are many improvements that can be made to make this tool useable. But achieving this is outside my goals of this project.

XVII. References

[1] "Why does 'imread' reads grayscale images as two-dimensional - MATLAB Answers - MATLAB Central," [Online]. Available: https://uk.mathworks.com/matlabcentral/answers/47823-why-does-imread-reads-grayscale-images-as-two-dimensional.

[2] "Comparing Edge Detection Methods - Nika Tsankashvili - Medium," [Online]. Available: https://medium.com/@nikatsanka/comparing-edge-detection-methods-638a2919476e.

[3] "The Hough Transform: The Basics - AI Shack," [Online]. Available: http://aishack.in/tutorials/hough-transform-basics/.

[4] "Binarize 2-D grayscale image or 3-D volume by thresholding - MATLAB imbinarize - MathWorks United Kingdom," [Online]. Available: https://uk.mathworks.com/help/images/ref/imbinarize.html.

[5] "Top-hat filtering - MATLAB imtophat - MathWorks United Kingdom," [Online]. Available: https://uk.mathworks.com/help/images/ref/imtophat.html.

[6] "2-D polygons - MATLAB - MathWorks United Kingdom," [Online]. Available: https://uk.mathworks.com/help/matlab/ref/polyshape.html.

[7] "Erode image - MATLAB imerode - MathWorks United Kingdom," [Online]. Available: https://uk.mathworks.com/help/images/ref/imerode.html?searchHighlight=imerode&s_tid=doc_srchtitle.

[8] "Morphological reconstruction - MATLAB imreconstruct - MathWorks United Kingdom," [Online]. Available: https://uk.mathworks.com/help/images/ref/imreconstruct.html?searchHighlight=imreconstruct&s_tid=doc_srchtitle.

[9] "Recognize text using optical character recognition - MATLAB ocr - MathWorks United Kingdom," [Online]. Available: https://uk.mathworks.com/help/vision/ref/ocr.html.

[10] "Recognize Text Using Optical Character Recognition (OCR) - MATLAB & Simulink - MathWorks United Kingdom," [Online]. Available: https://uk.mathworks.com/help/vision/examples/recognize-text-using-optical-character-recognition-ocr.html.

[11] "Extract line segments based on Hough transform - MATLAB houghlines - MathWorks United Kingdom," [Online]. Available: https://uk.mathworks.com/help/images/ref/houghlines.html.