EYE IN THE SKY

COMP 4102- Final Project Report

18 April 2020

Robin Wohlfarth	100 847 725
Derek Marks	100 974 560

Table of contents

1.1 Abstract ………………………………... pg 2

1.2 Introduction …………………………..…. pg 3

1.3 Background …………….……………….. pg 6

1.4 Approach …………………………………….. pg 10

1.5 List of work …………………………………… pg 10

1.6 Github page …………………………………... pg 10

1.7 References ……………………………………. pg 10

(1.1) Abstract

This project titled Eye in the Sky is an application which takes vertical perspective aerial imagery (Macodrum Library, 2020) as input. It then detects, outlines, and classifies airfield threshold markings. Using standardized regulations on airfield construction (Federal Aviation Administration, 2019), the application can then estimate the airfield width. This was a challenging problem as airfield scale can vary with imagery resolution and flight altitude. Once the airfield threshold has been identified, there are multiple types of markings painted on the pavement which create noise in detecting the appropriate threshold markings. Lastly, the condition of these markings varies, as they can be fully or partially faded preventing effective thresholding.

(1.2) Introduction

Eye in the sky first detects the location of the runway threshold markings using template matching. Once the end of airfield markings have been detected, a localized image is created by cropping around the detection and markings are interpreted by the bounding box classifier.

This system is a good candidate for being placed into a vision application because it automatically identifies features from aerial photography. This process requires many of the image processing techniques learned throughout the term. Gaussian smoothing, thresholding and edge detection are all applied in the process of reading the airfield markings.

Developing this system presented a number of challenges. Many of the airfield markings were faded, making it difficult to find an adequate threshold. Some of the markings were too close to other markings to distinguish them as separate entities. After thresholding, markings had to be classified in order to interpret them.

(1.3) Background

A commercial runway is often a recognizable point on aerial or satellite imagery, due to the nature of its use. Markings on the runway are not different, as they are intentionally designed to be visually distinct for approaches in poor weather conditions. These markings convey information important to pilots when landing like the width of a runway or compass direction. The idea would be if we can parse the information on the ground we can make it human readable and extrapolate more info on the runway.

Tutorials on OpenCV shape identification and corner detection allow for easy identification of shapes and points when thresholding is applied. By identifying patterns of shapes in a specific orientation we can apply this to the beginning of runways to extract information on the runway like width and direction. Specifics on runway marking are published by the FAA and available to the public.

These define rules runway marking must follow and provide us with straightforward layouts of symbols and patterns that will help us process the data and eliminate false positives. Techniques like thresholding and Image Denoising can be implemented to reduce any possible noise, cleaning up any obscured markings and creating stronger results when matching patterns.

(1.4) Approach

The first stage of this application used template matching and scaling. The image is scaled to varying sizes while each template is compared against it using cv2.matchTemplate . Each run through the correlation to the template is measured and compared to the previous highest value.The result is a return of the closest possible match to the template as well as the coordinates .

With the airfield threshold markings location identified, the next step is to prep the image for the bounding box classifier. The markings are cropped from the original image into a localized image using the coordinates created in the template matching process . The cropped image is converted to grayscale, and gaussian smoothing is applied using the cv2 GaussianBlur() function to remove high frequency noise. Next the image is scaled up by ten percent to better enable the program to differentiate contours. Lastly, the image is binarized to black and white using the cv2 threshold() function.

The output of this process can be seen in figure 1 below. The left pair displays near optimal thresholding, with all markings clearly seen and minimal interference from non critical markings. The right image pair shows heavily faded markings, which do not cleanly threshold, presenting a challenge for classification.

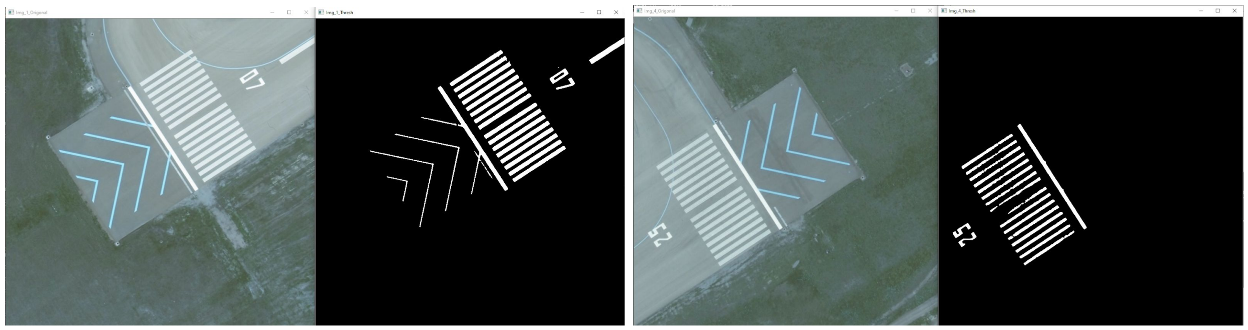


Figure 1: Screenshots of two images prepared for contour detection.

With a thresholded image prepped, the next step is to find the image contours using the cv2 method findContours(). This returns an array of n contour groups, each represented by a list of (x,y) vertices. The results of this operation can be seen below in figure 2. Left image pair has chevrons markings that are clearly detected, however the right side displays problematic detection of threshold markings due to fading, and does not pick up chevrons at all.

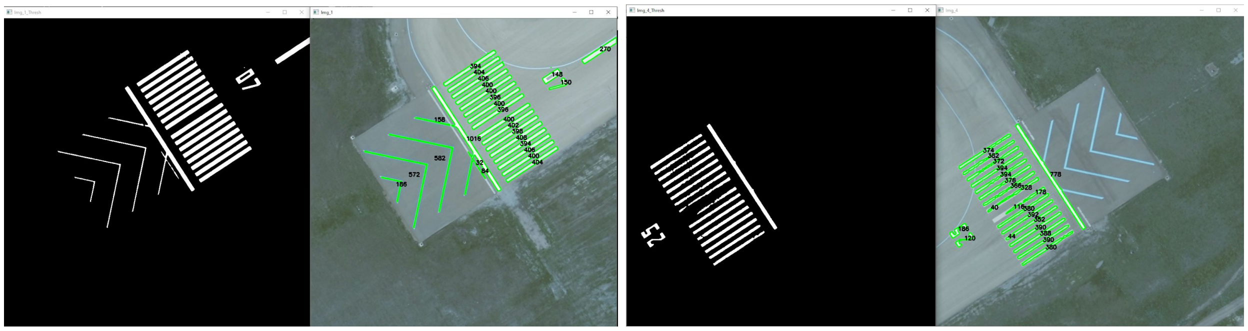


Figure 2: Examples of contour detection on thresholded images.

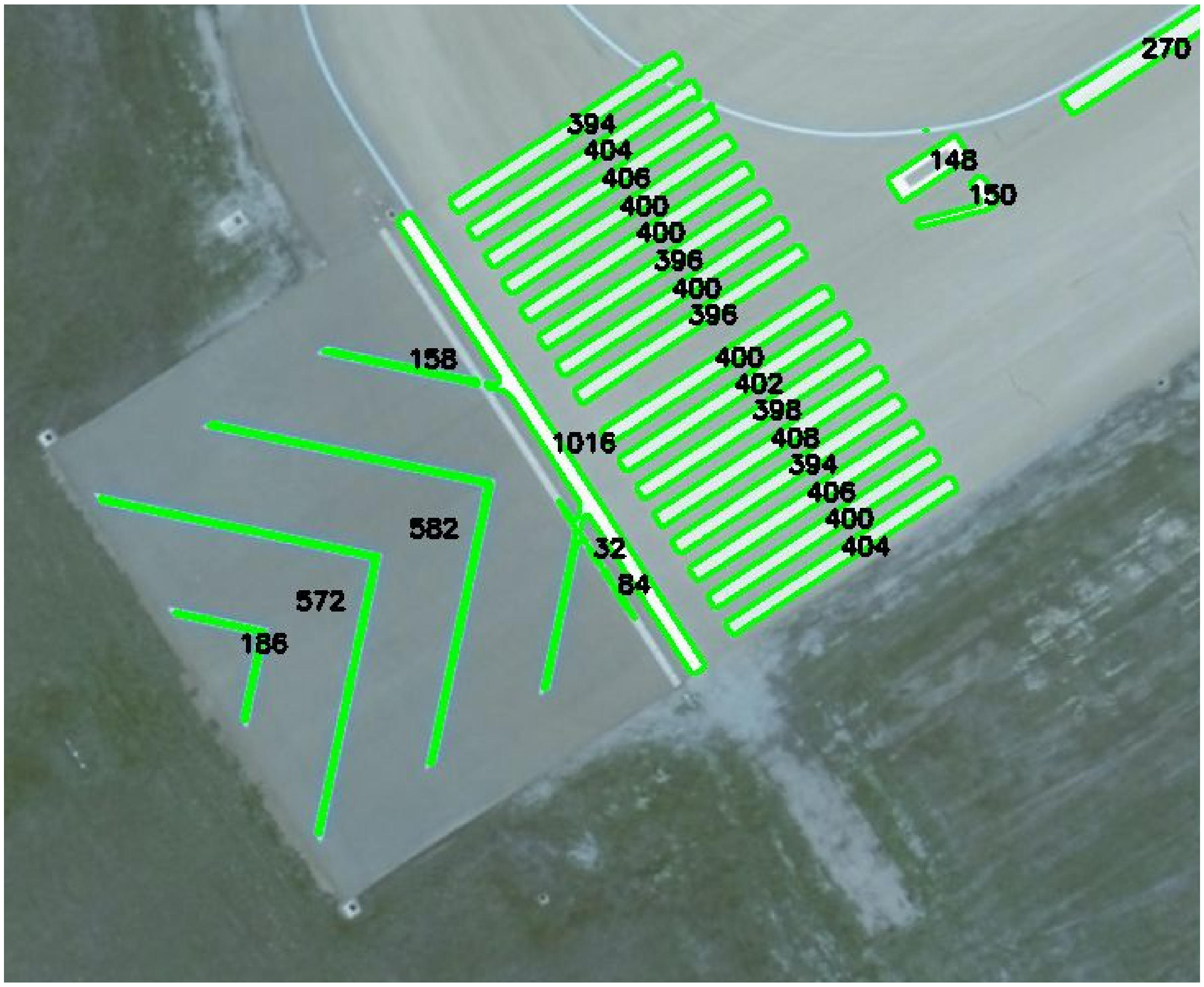


Figure 3: A close up of the cv2 contour detection outcome. Numbers written in black are the total count of points in the respective contour set.

With contours drawn on markings, the next step is to filter markings that are not relevant to determining runway width. By applying a simple range filter, any contour with an extremely small (less than 300) or extremely large (greater than 600) set of vertices is discounted from the detection process. The results of this can be seen in figure 4. Relevant markings are drawn in green, and discarded contour sets are drawn in red.

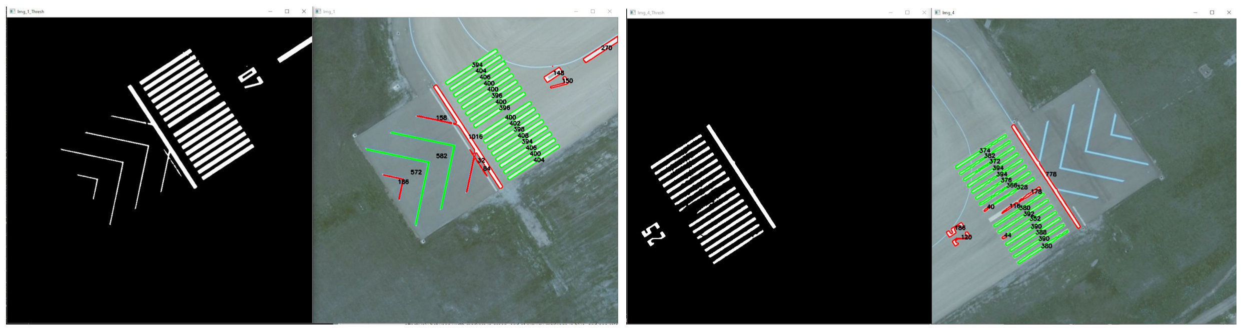
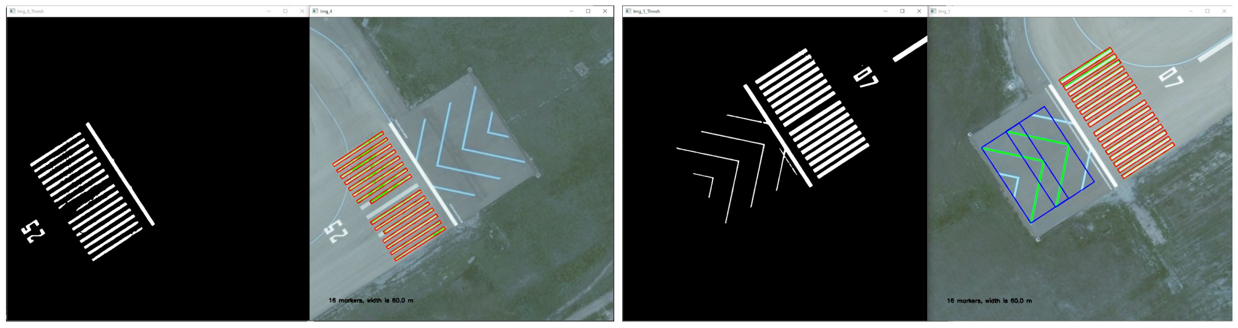


Figure 4: An example of the vertex count filter. Discounted contours are drawn in red, valid ones are green. Images produced using rangeClassifier() function in funct.py. The right pair shows one threshold marker was not detected due to fading.

This step is still not sufficient to determine runway width. It can not discriminate effectively between threshold and chevron markings. A final method, bndBoxClassifer() is needed to classify these objects separately from one another. First bndBoxClassifer() draws a minimal area bounding box around each of the remaining contour groups by calling the cv2 method minAreaRect(). Second, it calculates the average area of these contour bounding boxes. Lastly, it compares the bounding box of every contour group. Any bounding box with an area significantly greater than the mean area is classified as a chevron. The remaining boxes are classified as threshold markings. The output of this method is seen below.



16 markers, width is 60.0 m

16 markers, width is 60.0 m

Figure 5: bndBoxClassifier() output. Successfully discriminates between end of runway chevron markings (drawn in green, bounded in blue boxes), and threshold markings (bounded in red boxes).

(1.5) Results

The following images are the results of the bndBoxClassifer() function on airfield imagery. The number of identified threshold markers and determined width of the airfield is printed in black at the bottom left corner of each image. Threshold markers are identified in red, and chevrons with blue bounding boxes.

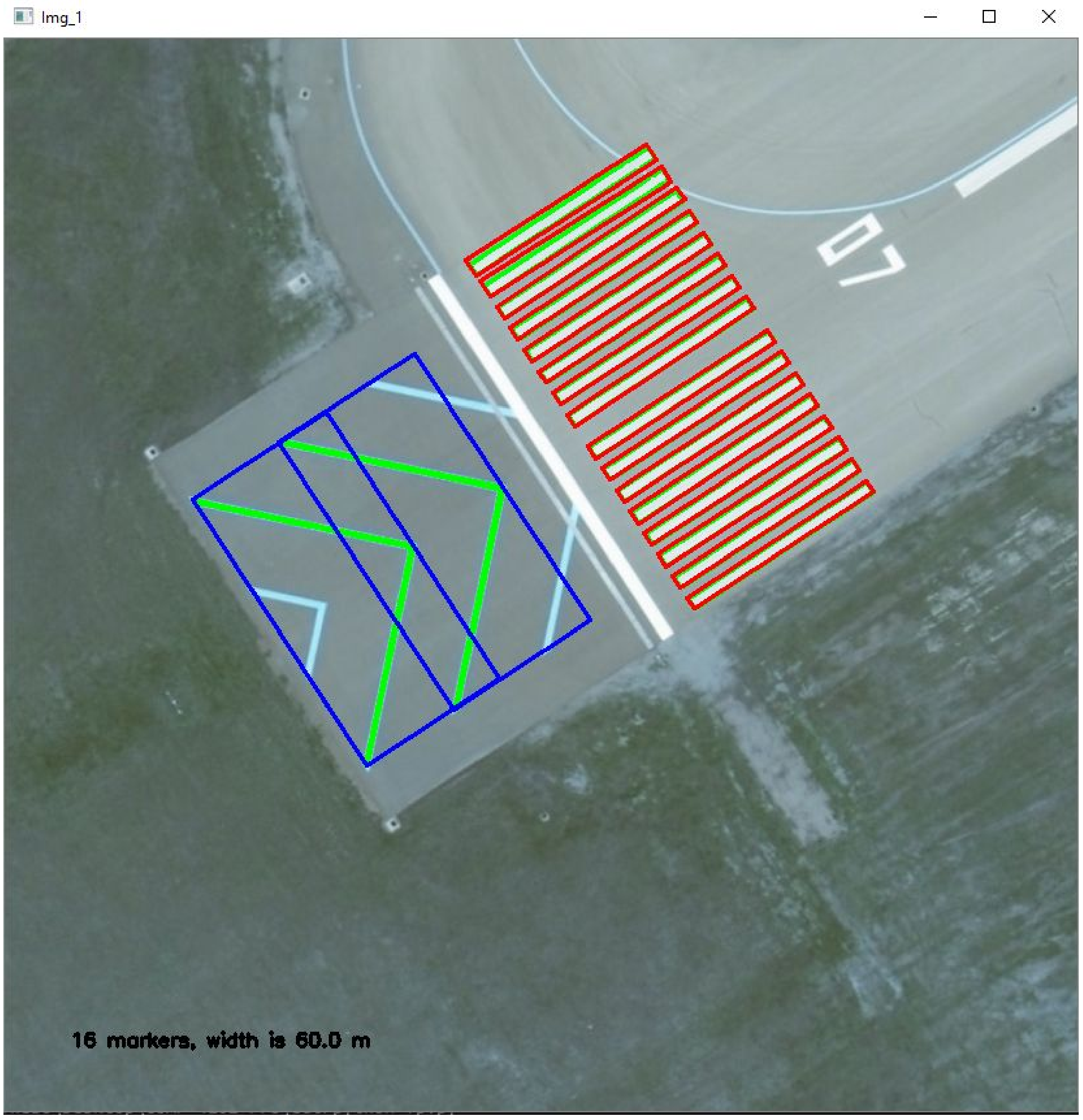


Figure 6: Image one has perfect classification, and successfully identifies every threshold marking and end of runway chevron.

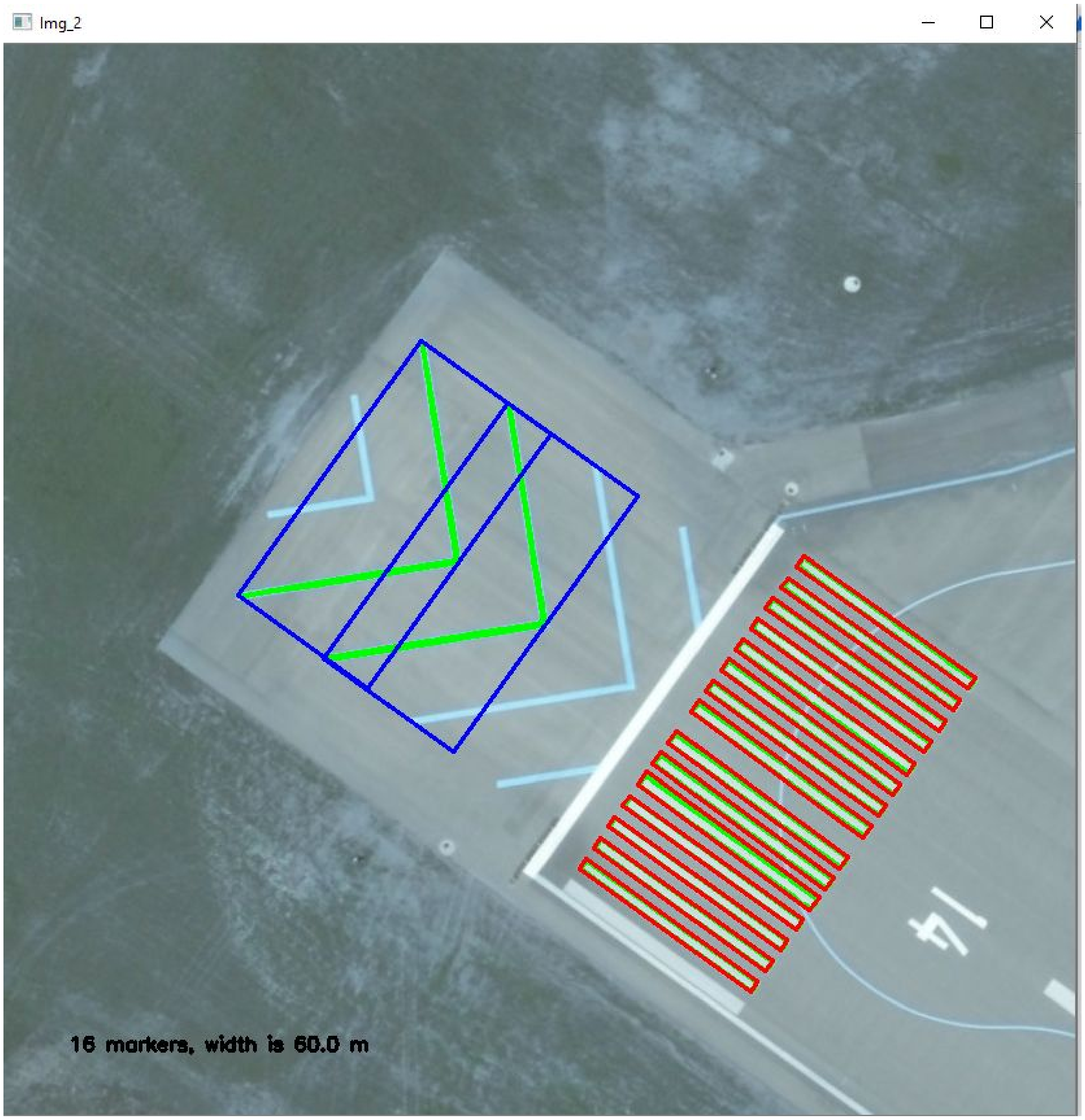


Figure 7: Image two has near perfect classification. The final threshold marker is not detected due to its proximity to the boundary marker on the airfield. The correct result is still obtained because per federal guidelines there are only ever an even number of threshold markers, so the system rounds up.

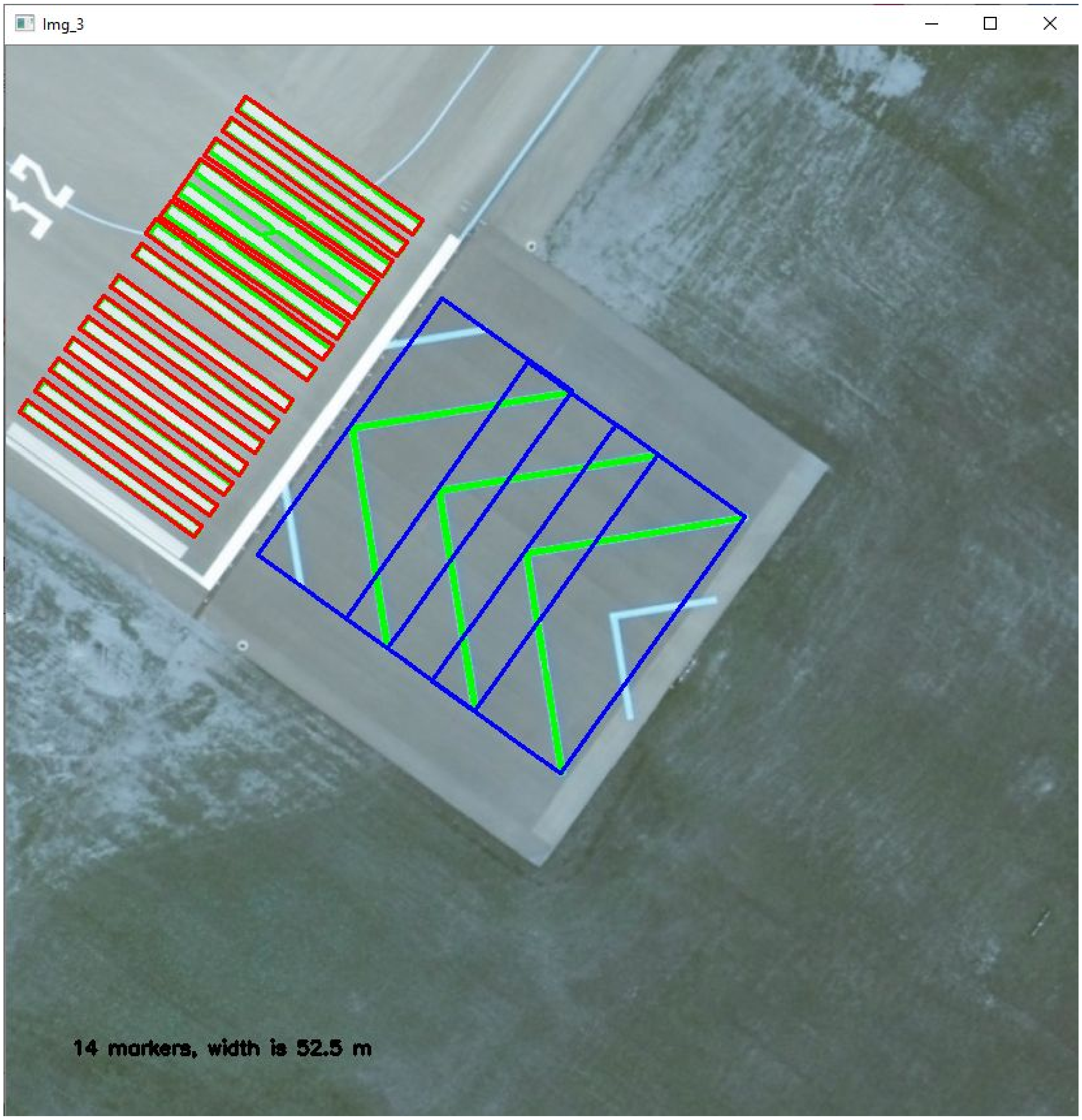


Figure 8: The only runway image of four tested that is incorrectly classified. The leftmost marker is not identified because it is close to the left airfield boundary marker. This error is not caught by simple rounding, as two of the threshold markers (4th and 5th from the right) are also adjoined by a particularly intense runway centerline and are identified as only one marker.

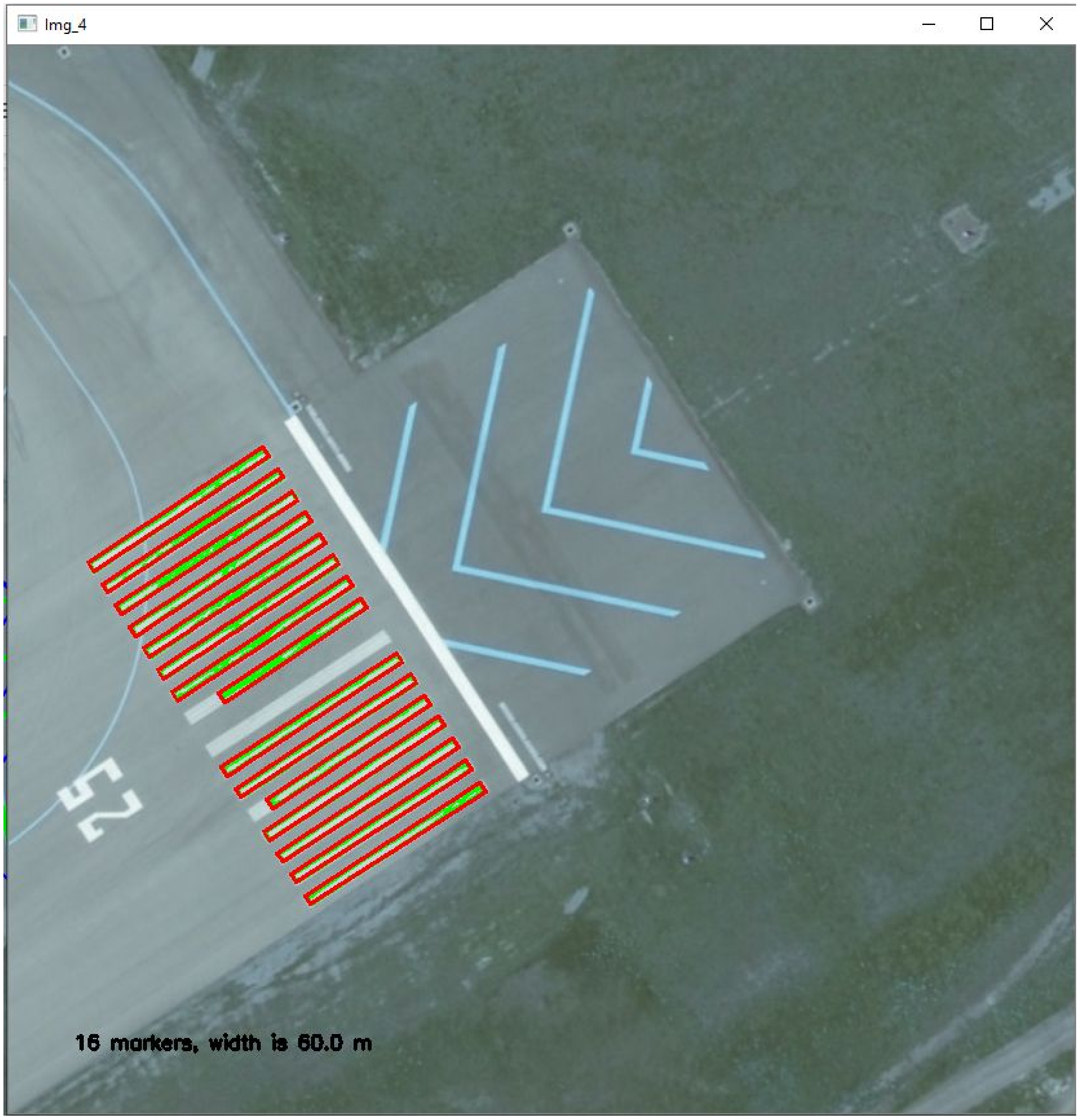


Figure 9: Threshold markers identified on an end of runway with faded markings. The chevrons are not intense enough to be detected, and portions of the threshold markings are not picked up, resulting in short bounding boxes. However, it is still correctly classified as the area of these markings is sufficiently close enough to the mean to be counted.

In conclusion, this project was able identify and classify end of runway markings in a mostly consistent manner. However, error was introduced by the presence of fading and intersecting runway markings, Possible corrections to this error include more effective thresholding techniques, implementation of canny edge detection, and using colour to remove intersecting runway lines.

(1.6) List of work

Equal work was performed by both project members.

(1.7) GitHub Page

https://github.com/derekmarks/COMP-4102-Project

(1.8) References

1. Public Information Map. (n.d.). Retrieved from http://carleton-u.maps.arcgis.com/apps/PublicInformation/index.html?appid=977e55c0c435488f9e5c38b4c8700501
2. Runway And Taxiway Marking. (2017). Louisiana Department of Transportation and Development, 13–1-13–3. Retrieved from Aviation/Airport_System_Plan/Chapt_13 Runway and Taxiway Marking.pdf?fbclid=IwAR1tMCdiK0jk5sLn3a2To9QZ_0HYmhL01Bi5f7aazgmRj0wxgMFEK1yrhdU
3. Federal Aviation Administration. (2019). AC 150/5340-1M - Standards for Airport Markings. Retrieved from https://www.faa.gov/airports/resources/advisory_circulars/index.cfm/go/document.current/documentNumber/150_5340-1