CMPEN 454 Project 1, Fall 2018

Ian Hutchinson (ich5023)

William Yoshida (wky5017)

Soleiman Allam (sba5191)

Overview

The project's main goal was to give students experience doing simple object detection in Matlab. Object detection involves many topics discussed in class including smoothing, edge detection, patch matching, scale space and more. Thus it is a good topic to have a project based on because it summarizes most of our knowledge and is also a fundamental task in the computer vision field. The specific object detection problem we were asked to solve was to find and count the number of planes in a satellite image of two different airports. In this report we will delve into our implementation procedure, observations/explanations, experimental visualizations, and finish with an evaluation/discussion of our results.

Implementation Procedure

The first step in the procedure was to decide on what templates to use. After using a singular template of an overall plane+ground picture, we started developing our testing hypothesis. An overall picture of a plane was accurate but only for similar planes with similar orientation and similar size. Our thought process shifted to a smaller sample that targets critical areas of the plane such as wingspan corners or rear plane tail. Our detection results were increased however a new imaging error occurred. The delivered output included a significant amount of false positives. Thus, we gradually shifted back to larger plane images while testing alterations of the template to accommodate different shapes, sizes, and orientation of the planes. However, achieving these accommodations did not seem possible with a singular image. Thus, we took several fitting templates that included different rotations as well as significant edge and corners that would aid in further detection. After we decided on our templates we imported both the image we were searching and the various templates into Matlab. Next we converted both the image and every template to grayscale. Following that we applied a Gaussian smoothing filter to the image and all the templates in order to reduce the noise in the image and templates. Reducing the noise is important because following that we compute the gradient magnitude of the image and templates. If there is no smoothing done before that computing the gradient magnitude will amplify the noise and lead to inaccurate object detection.

Upon computing the gradient magnitudes we performed Normalized Cross Correlation (NCC) between the gradient magnitude image and the various gradient magnitude templates. This gave us a score ranging between -1 to 1 for how similar each pixel is in the image to each template patch. We then create a binary image from the NCC scores, using emptyarray() to create an image first with all pixel values set to 0. For a given pixel if the NCC score was above 0.2 it is marked as a pixel of interest and assigned a value of 1.

For the pixels we performed Non-maximal suppression on we essentially compared their NCC scores to the pixels in their local neighborhood. The function bwmorph() with parameters (result,'shrink',Inf) examines all the pixels in a local neighborhood and shrinks a found connected-component to a single pixel. Likely, the remaining central pixel is also the one that previously held a larger NCC score within its component. We then cycle through the array once again to find all remaining pixels of interest and mark them with the function insertShape() (meaning a plane is at that location). In addition to a mark a number is placed on the current pixel using insertText(). This number is the current plane count (which is initially set to 0) which is tallied via the search algorithm. The final number in the plane count represents the total number of planes found in the image. This final value is also displayed in the center of the image. Unfortunately, due to the exact operations of the bwmorph() function, at some instances, the function was not able to eliminate all other neighboring pixels. This may happen because the bwmorph() function attempts to eliminate all pixels in a connected component, leaving the central pixel. Sometimes patch matches may occur but leave disconnected components, causing two or more central pixels for a single object detected.

**To run a demo of the code, simply run the demo.m file. Otherwise, in order to run the code with a user-specified input image, run the project1.m file. This will prompt the user for a filename.

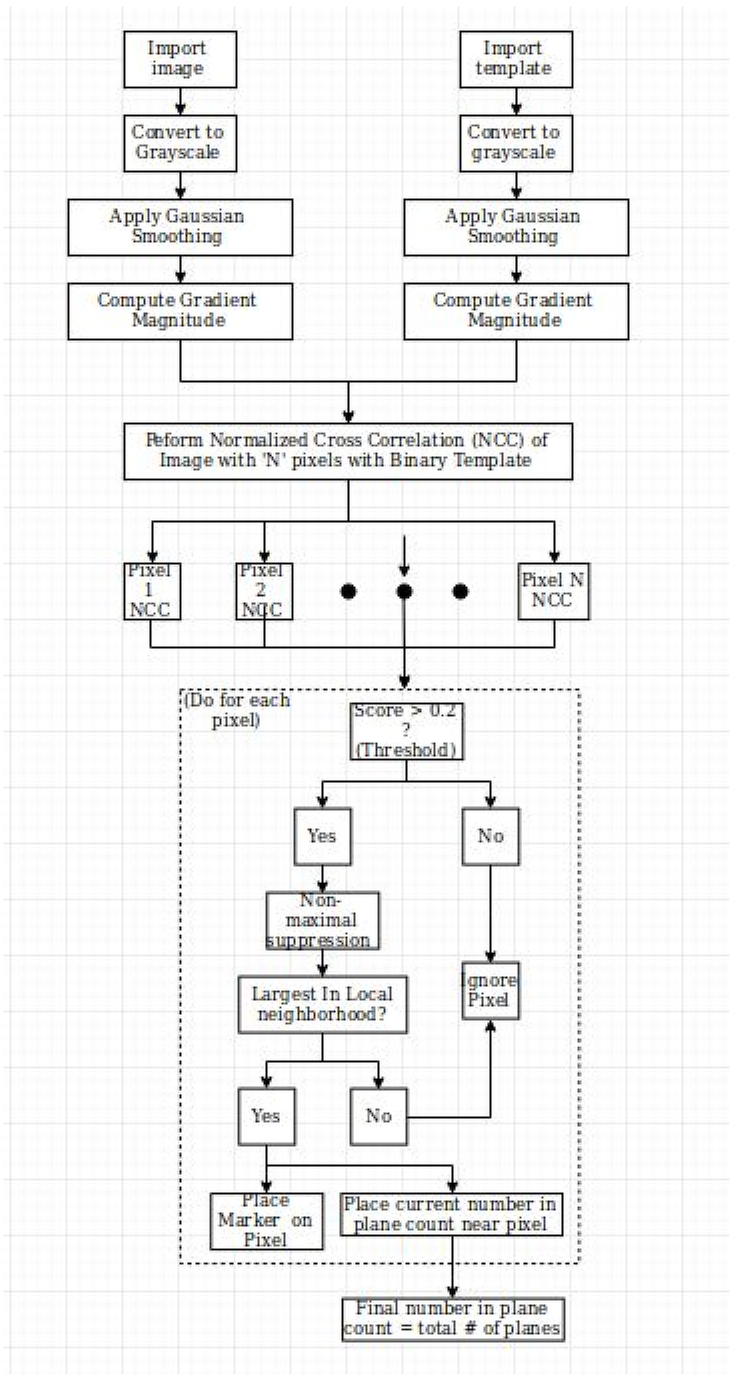
Import image
Import template
Convert to Grayscale
Convert to grayscale
Apply Gaussian Smoothing
Apply Gaussian Smoothing
Compute Gradient Magnitude
Compute Gradient Magnitude
Peform Normalized Cross Correlation (NCC) of Image with 'N' pixels with Binary Template
Pixel 1 NCC
Pixel 2 NCC
Pixel N NCC
(Do for each pixel)
Score > 0.2 ? (Threshold)
Yes
No
Non-maximal suppression
Largest In Local neighborhood?
Ignore Pixel
Yes
No
Place Marker on Pixel
Place current number in plane count near pixel
Final number in plane count = total # of planes

Figure 0: Flow Chart

Observations and Explanations

Our observations towards this project stems from the technical knowledge challenges that we had to overcome. The first step was to find the airplanes in the image through matlab. This required our prerequisite knowledge from class. Therefore, our starting point included two factors; the given image, and the free choice of a comparative template. We manually cropped part of the full image down to display the template using an image editor such as paint or windows photo gallery. This template needed to be cross referenced across the entirety of the original image for similarities. Using no filters or gradients yield the expected results of only finding the extracted template. We started by using grey scales and simple gaussian blurs across in either the X or Y axis. This increased the number of template matches however it was not very effective. Therefore, we decided to incorporate gradients with either convolutions or correlations that would aid us in catering to more airplanes. We came across normxcorr2() which was successful at finding several airplanes with valid detections. As mentioned previously, our thought process led us to understand that further importance must be stressed on choosing the correct template. The shift was from the plane+ground picture to edges and corners then finally back to multiple overall plane templates. The learning curve to picking the most suited templates was reliant on trial and error but we noticed that including several unique templates was the most effective approach.

In the next step of assigning the individual locator circles and index numbers, a new problem was observed. Our method of locating the planes included a cluster of points/pixels that exceed a threshold and therefore create a bigger locating mark for the overall plane. This resulted in a problematic barrier to plotting individual marking locater such as a red circles around the plane. The red circles would be clustered together and drawn around each individual pixel. The same problem would be interfering with the numbering system based on detected indexes. Addressing this issue we needed the non-maximal suppression provided by the function bwmorph() which cycles through and finds each center point for our clusters. However, in order to be able to use bwmorph we had to convert our image into a binary image. This is why we mentioned earlier the extra steps of creating an emptyarray() with all zeros that references the NCC scores for when our chosen threshold is crossed. Pixels that exceeded the threshold would be assigned a value of 1.

Experimental Visualizations

This section contains a step by step visual representation of intermediate and final results. Essentially it will walk through our algorithms creation and its evolution to its final state.

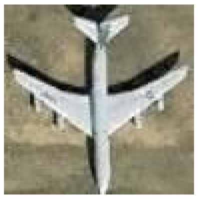

Figure 1a: Template 1

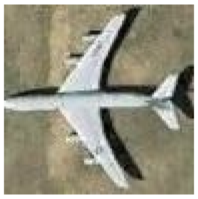

Figure 1b: Template 2

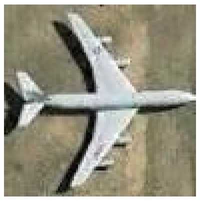

Figure 1c: Template 3

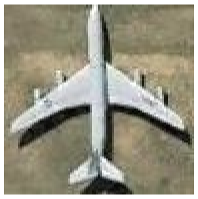

Figure 1d: Template 4

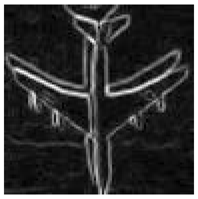
Figure 2: Template 1 Gradient Magnitude

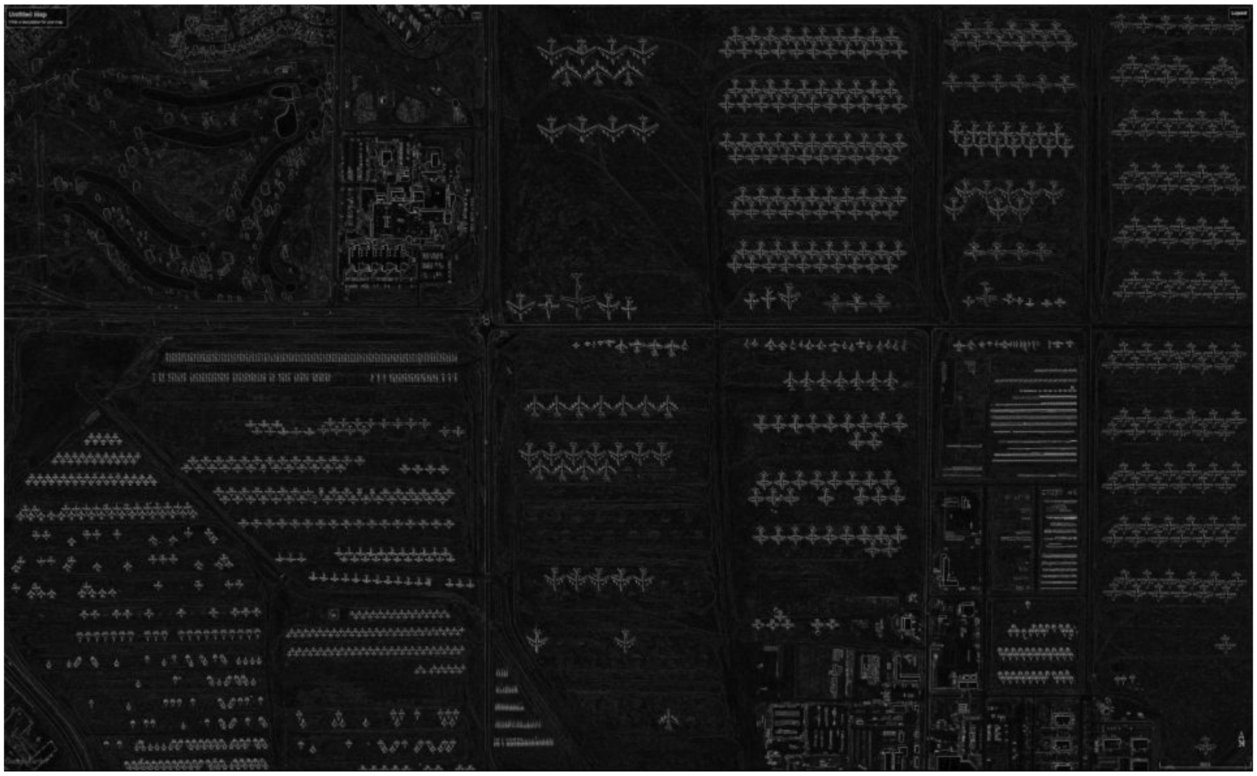
Figure 3: Gradient Magnitude for dma.jpg

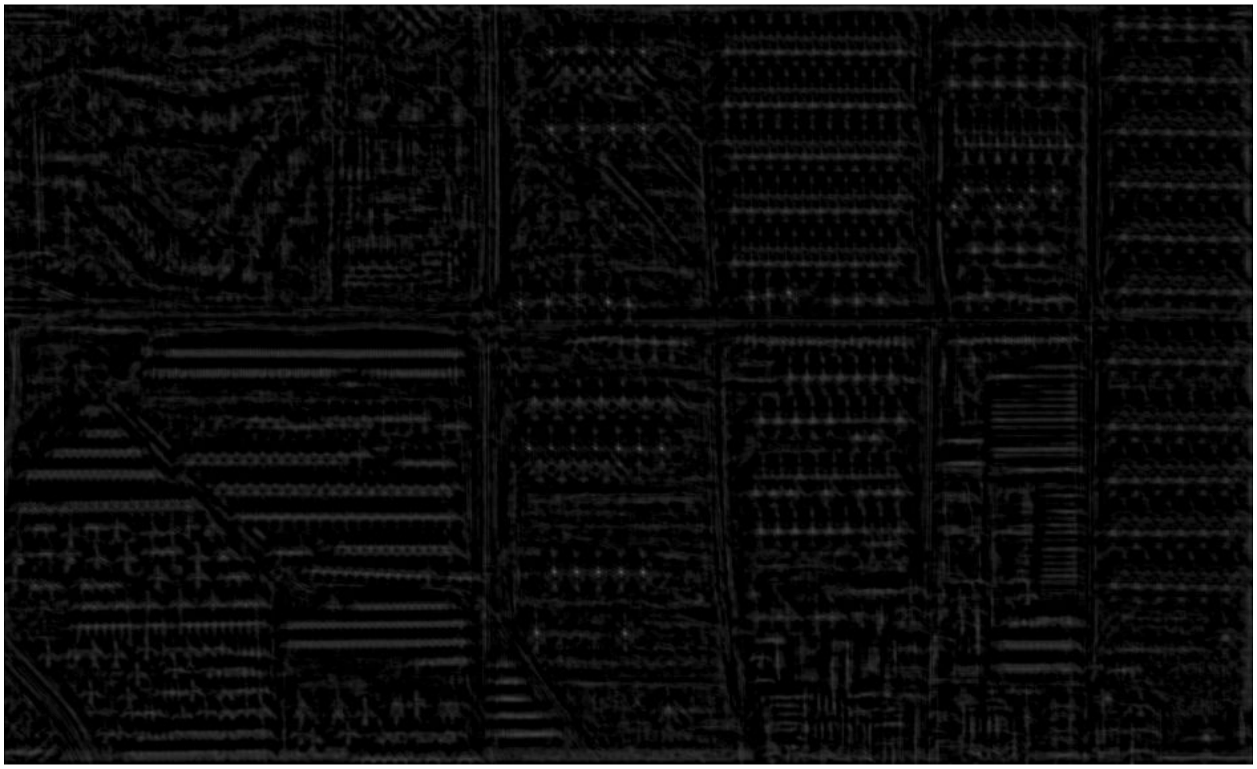
Figure 4: NCC Score for dma.jpg

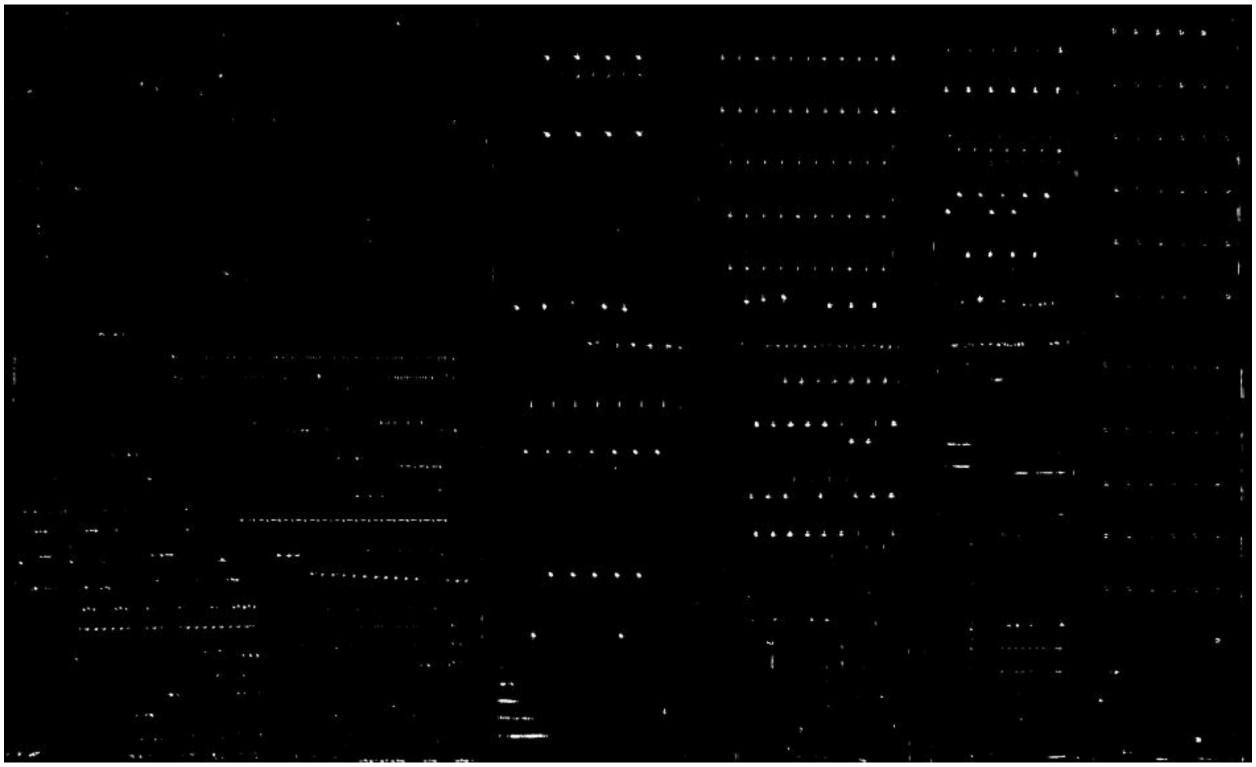
Figure 5: Binary image of NCC scores for dma.jpg

Figure 6:Binary image after non-maximum suppression for dma.jpg (*Note small light pixels of interest/matches in the image)

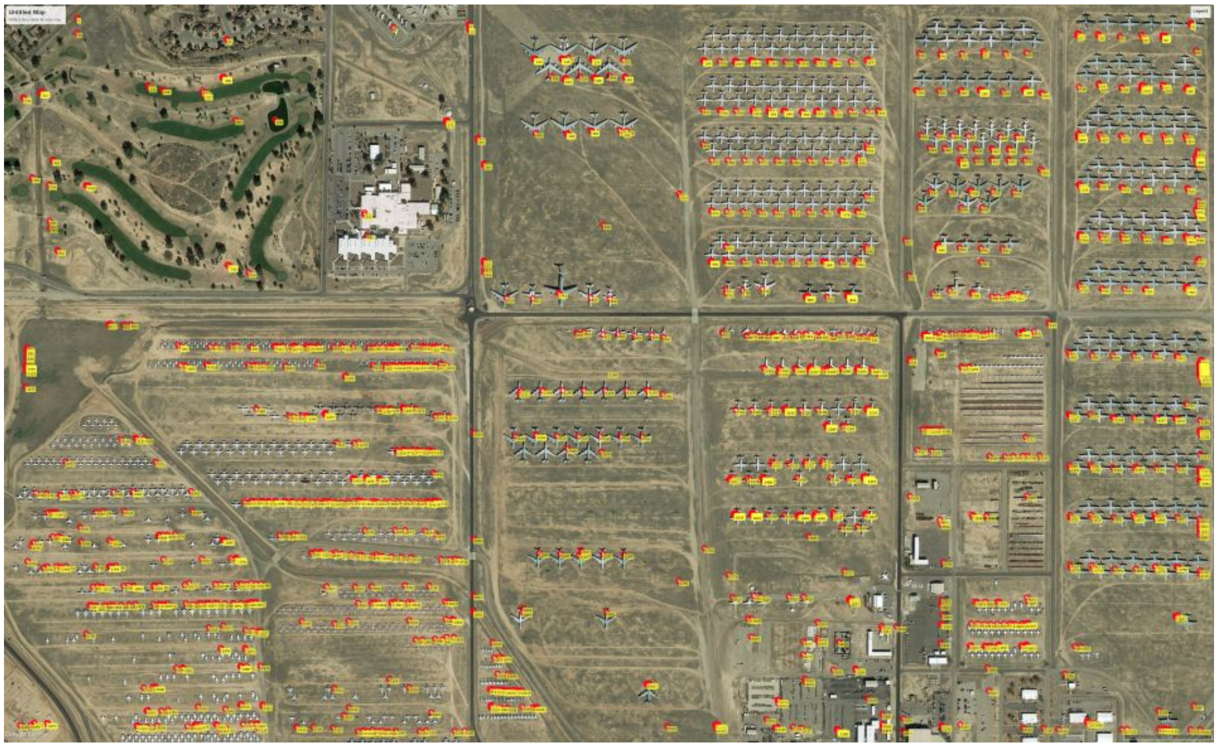
Figure 7: Final Result for dma.jpg

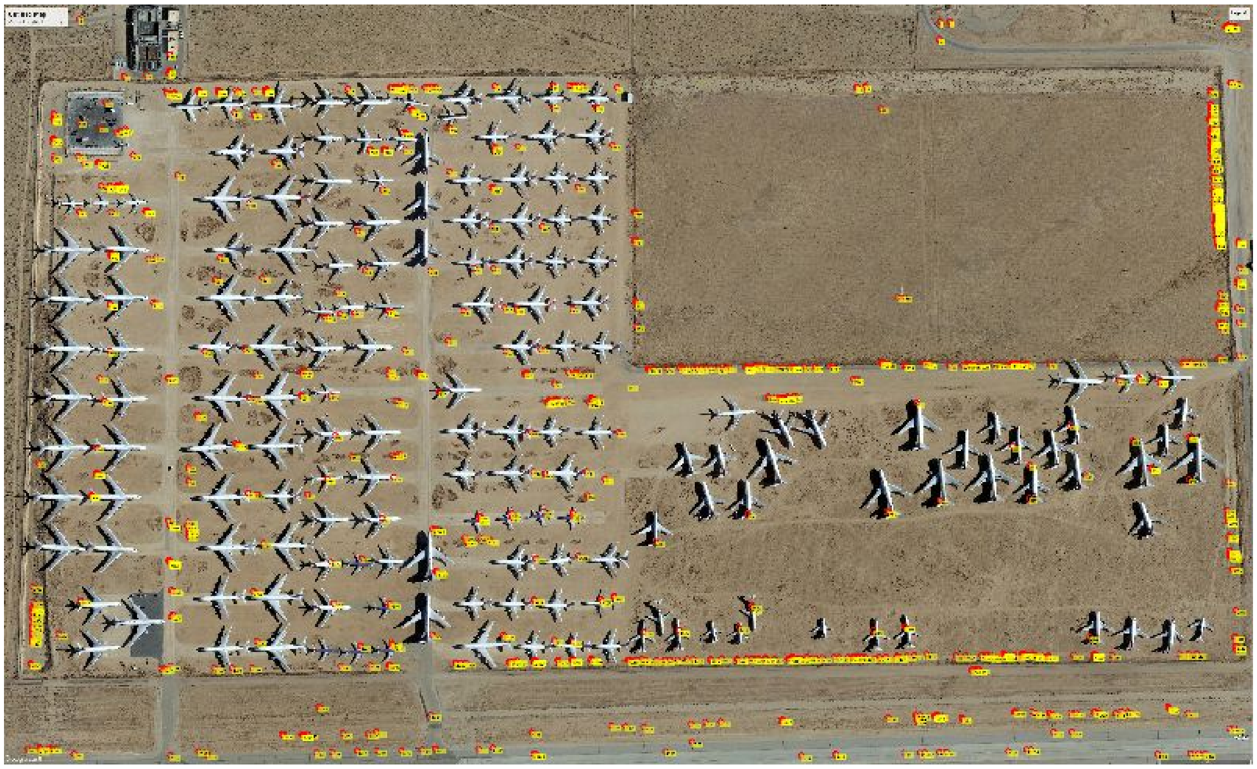

Figure 8: Final Result for vcv.jpg

Evaluation

Our evaluation of the final code results would be considered a success. We were successfully able to extract information from the templates and apply it to the original image. Furthermore, we circled the found results and labeled them accordingly. Considering the true positives, our code clearly matches up with a lot of the planes and is correctly displaying the required alignment. Even though we conceptually succeeded, there are false positives and false negative cases to be found. In comparison to our brain and eye analysis of looking at the image, what the human perception is capable of doing, it would be lacking in the category of accuracy. We were limited by the tools we could use in matlab and using our very fundamental knowledge of image processing we missed some airplanes as well as had ghost indexes that should not be there. This was a very challenging yet rewarding project that pushed our coding, logic, and conceptual knowledge to its limit. Our team included two electrical engineering students, Soleiman and Ian, and the third member William is studying computer engineering. We adjusted our roles based on our majors. William's excellent coding knowledge and fluency in Matlab came in handy for the implementation phase. Soleiman and Ian distributed the work in terms of coming up with logic and documenting for the written report.

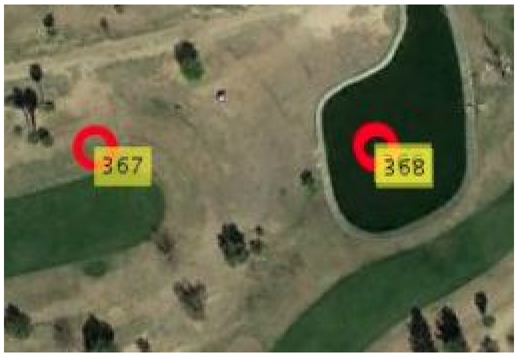

Figure 9: False Positives

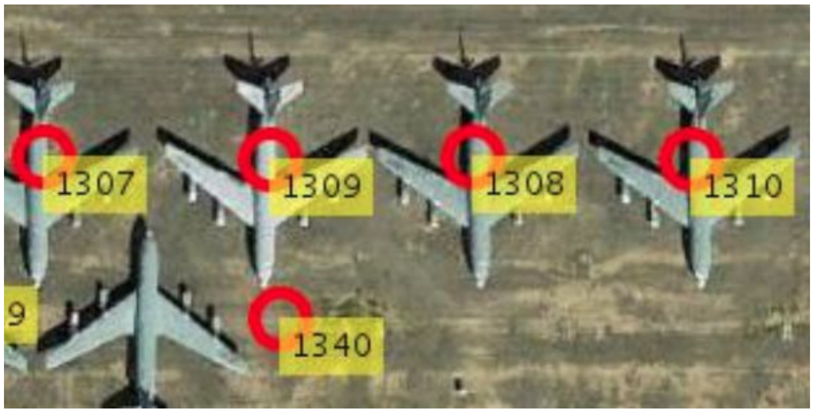

Figure 10: True Positives

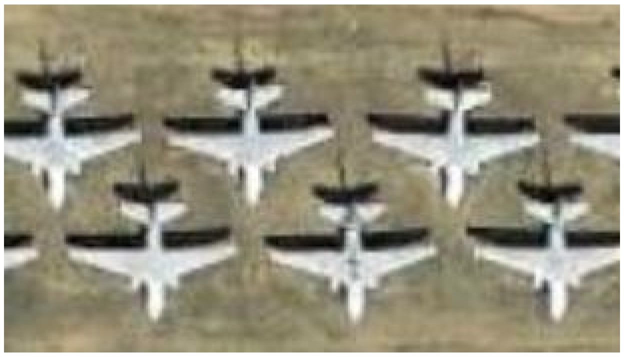

Figure 11: False Negatives