

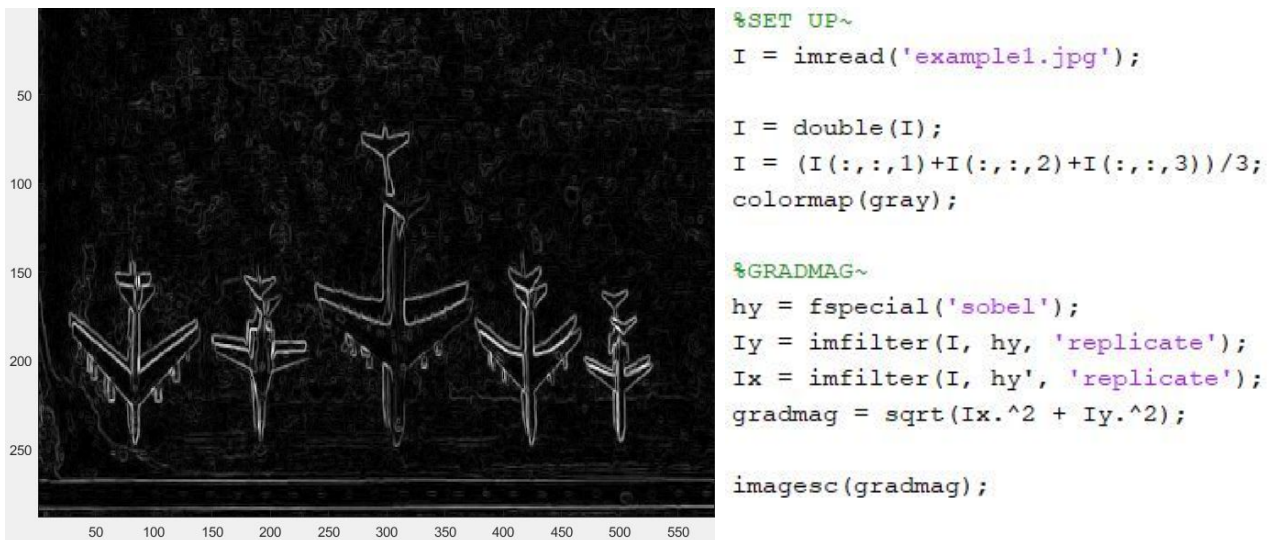
CMPEN 454, Project 1, Fall 2018

Qi Chang
Eryk Maciej Heyssler
Brian Mata
Xi Li

Introduction

This project was intended to provide us a hands on experience with the fundamentals of computer vision. Rather than having a committed step by step outline, this project challenged us to think creatively, realizing the fact that the problem at hand has many different solutions. It forced us to use our imagination to a great extent; approaches that we initially thought would be simple, turned into complex systems.

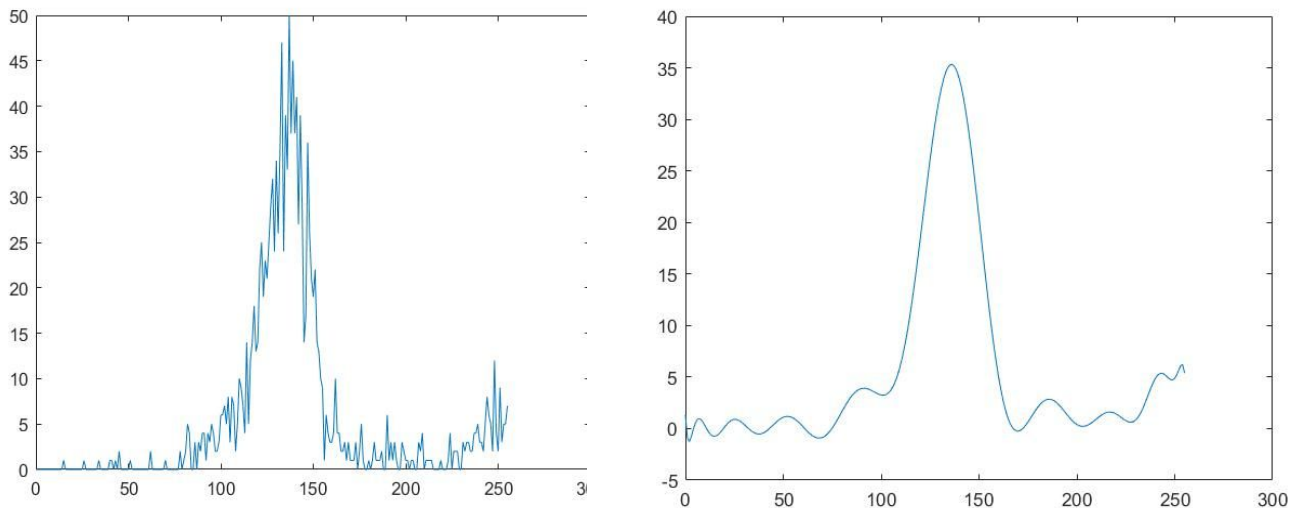
Our first task was to take the image apart by essentially isolating the airplanes from the background. We started by first computing the gradient of example1.jpg. This was done due to the fact that if we were able to successfully find the edges of the planes, we could then proceed to properly segment them. Our initial result was the following:



We noticed that the resulting image was in fact detecting the edges of the planes, but it came with some edges that we did not need. This is because if you take the gradient of a noisy image, the consequence is an even noisier image. Equivalently, the shades of the plane and the non-uniform color in the background will affect the edge information. In addition, the image from which this gradient magnitude was calculated (example1.jpg) is a very small fraction of the larger test image (dma.jpg). This is to say that the larger test image was bound to have areas with substantial noise and unexpected edges. What this meant is that we had to implement a more advanced function in our attempt to segment the image.

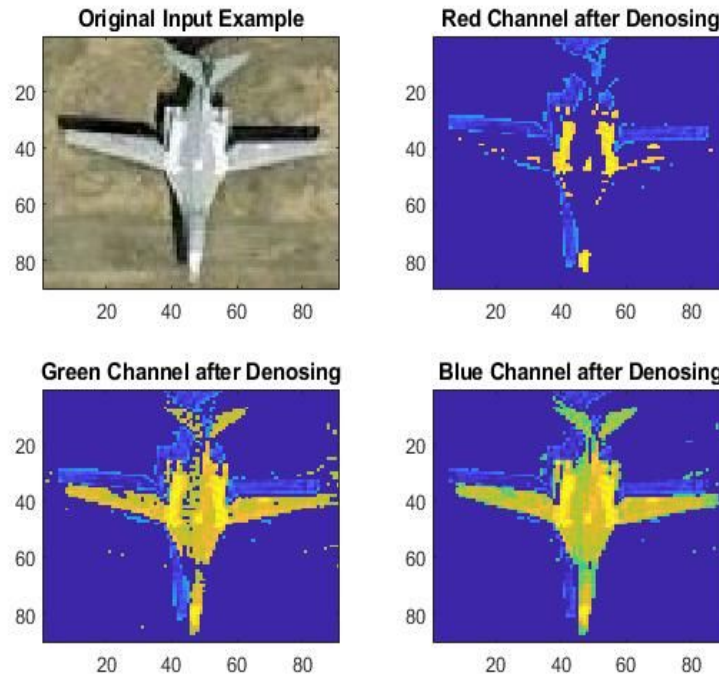
Method

If the test image was properly segmented, we could then use normalized cross correlation in order to match airplane templates within the test image. Thus, we devised a segmentation function that would calculate both intensity information and edge information of a function. This function, called “FeatureExtraction”, is fundamentally based on the principle of histograms and thresholds. Within this function, the input image is denoised by deleting the most common information from a histogram (using a function we wrote called “deleteMain”) which was produced by the matlab function `imhist(image)`. We can effectively smooth the image by finding peaks and nearest valley of the histogram and then delete unwanted information (i.e. the background) of the main mountain from the image. The original histogram and the simulated histogram are pictured, respectively:



This leads to the combination of taking some minimum RGB values as well as using the image `imhist` technique described earlier in order to extract intensity information from the image.

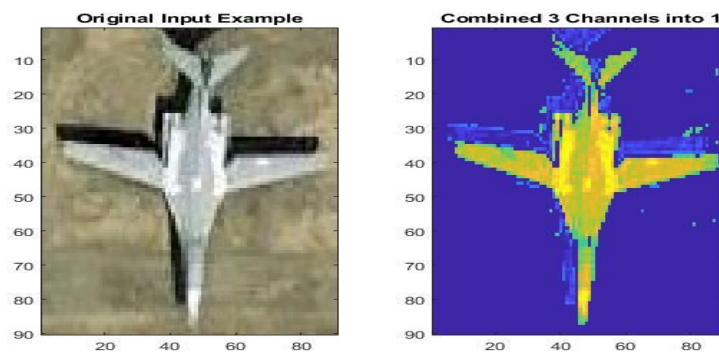
As we can see from the picture below, the plane information is mainly kept on the blue channel. The simple idea is to add them up to reflect the information, and weight high when there is little information in one channel to keep as much of the valuable information as possible.



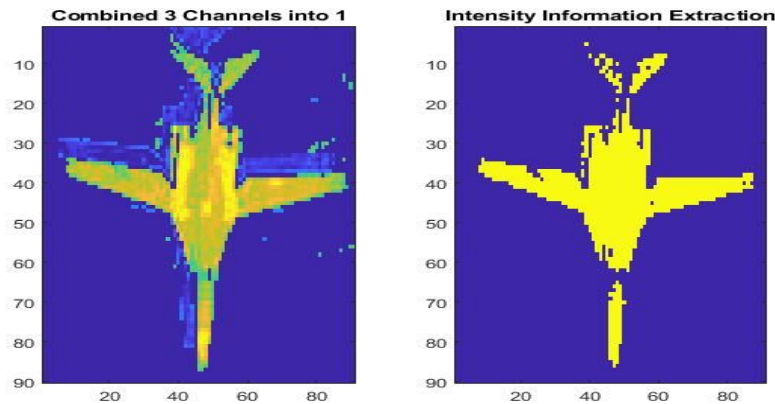
Afterwards, we apply a calculation to the coefficients of each channel and combine the 3 channels into 1 channel.

```
R_num = length(find(Example_delete(:,:,1)~=0));
G_num = length(find(Example_delete(:,:,2)~=0));
B_num = length(find(Example_delete(:,:,3)~=0));
Total_num = R_num + G_num + B_num;
R_coe = (Total_num - R_num)/(2*Total_num);
G_coe = (Total_num - G_num)/(2*Total_num);
B_coe = (Total_num - B_num)/(2*Total_num);
Image = R_coe*Example_delete(:,:,1) + G_coe*Example_delete(:,:,2) + B_coe*Example_delete(:,:,3);
```

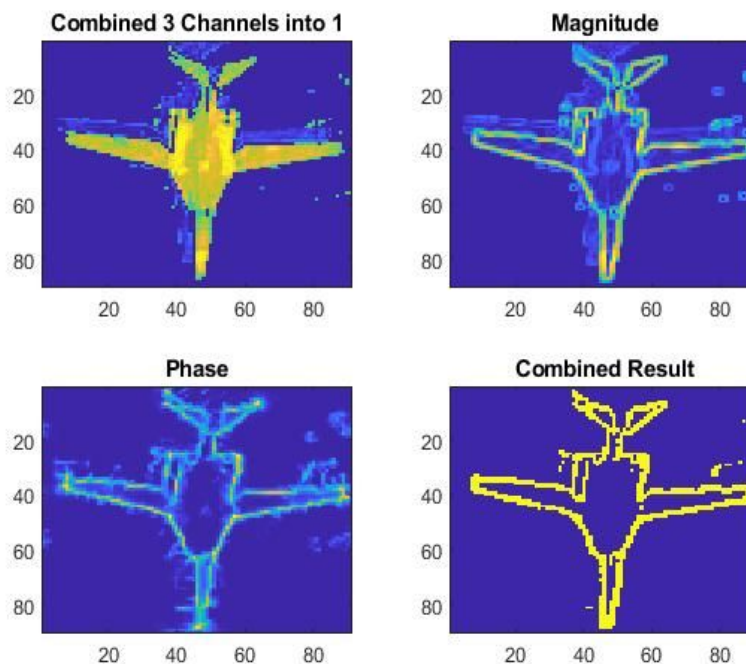
The result is shown in the following image.



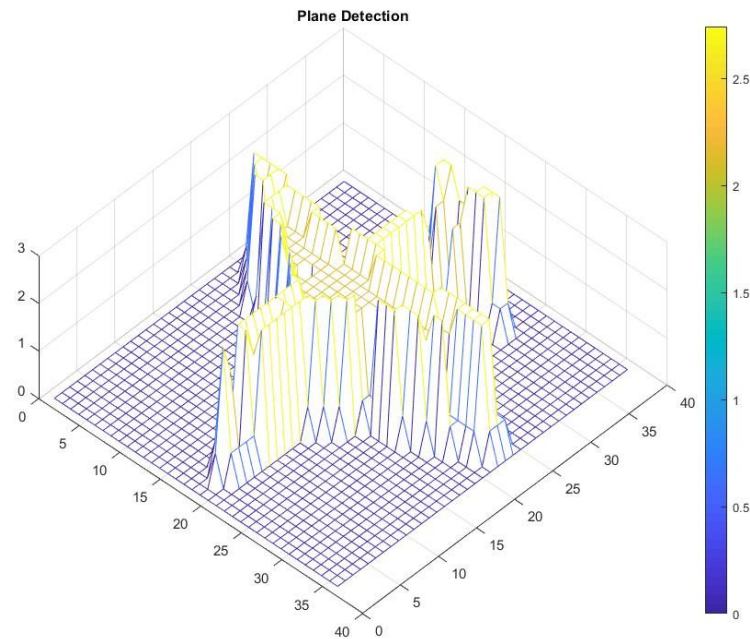
In order to make the intensity information only reflect the existing information with 0 or 1, we make a threshold to select the top 60% non-zero points and label them with 1, others with 0.



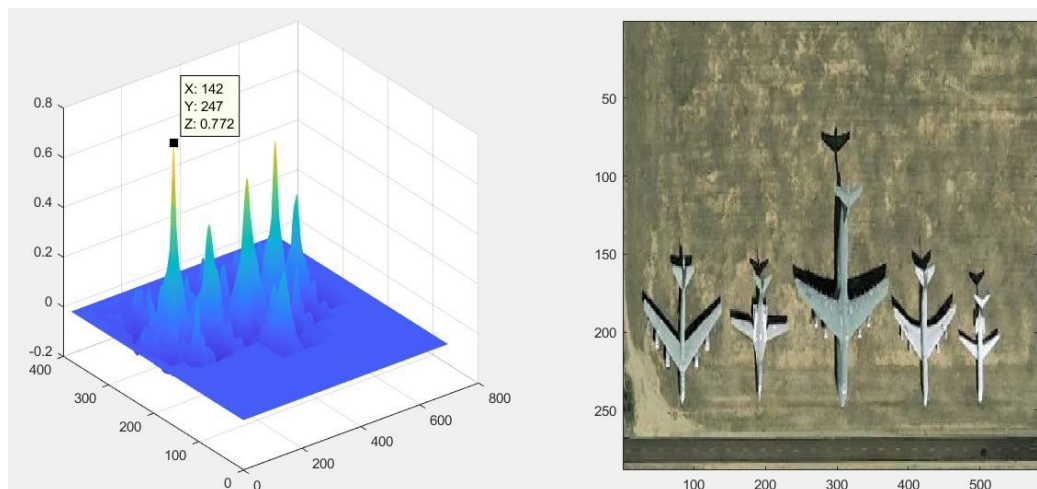
The same combined 3 channels image is used to find edges again; the gradient magnitude and frequency phase information are used to determine the edge information. Also the concept of thresholds (top 30% non-zero points) is used so that we may further decrease unwanted results and present existing information.



After that, the intensity and edge information are combined to use when calculating norm-crosslation in the next steps. We would like to think that the edge existing information is a little bit important than the intensity existing information.



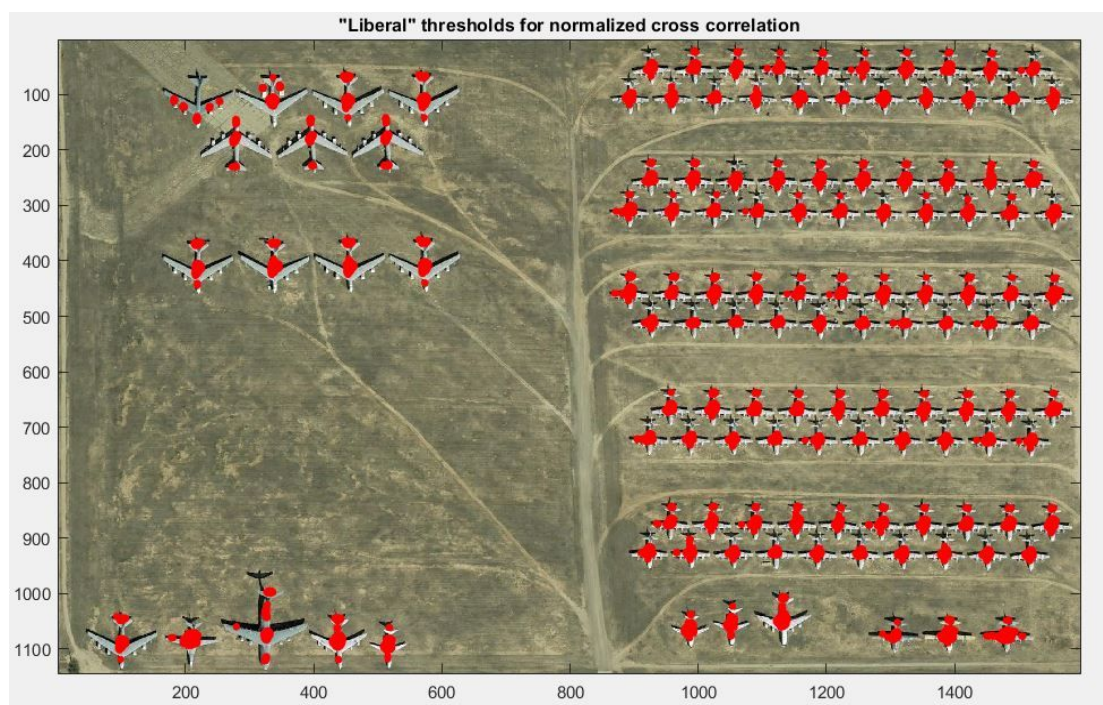
Using this information, we then constructed templates of each type of airplane in order to match them within the images using normalized cross correlation. Matlab has a built in normalized cross correlation function called `normxcorr2`, which simplified this process. Keeping in mind that the end goal of this project is to detect airplanes from a bird's eye view, we first needed to see how well normalized cross correlation responds to the template matching for each template. We explored this by plotting the normalized cross correlation with one template (one type of airplane) on the small test image, to see if the singular airplane template could match with different types of airplanes. Plotting this as a surface yields this result:



In this case, the template was the intensity information obtained from the leftmost plane. The correlation obtained from this information exactly matched our hypothesis; the highest peak in fact matched the correct airplane. In the surface diagram, the X and Y coordinates represented the location of the correlation, while the Z coordinate represented the magnitude itself. Upon observation, we noticed that the correlation actually had peaks in the exact spot of every airplane respectively, differing in magnitude, meaning it “picked up” every single airplane in the image. The next highest peaks coincided with the middle airplane, and the fourth airplane from the left. This made sense as these two were very much alike to the template. Unsurprisingly, the peak with the lowest correlation corresponded to the airplane that was the most dissimilar to the template, most likely due to the shape of the wings, as well as the slightly smaller scale.

Utilizing this knowledge of peaks, and how different templates react to different airplanes, we were able to determine appropriate values for which to plot these peaks onto the test image. In other words, we could establish a minimum for the normalized cross correlation of each airplane, so that the function detects only those airplanes which we wish to detect.

Our expectation was that very “liberal” thresholds would at certain points detect multiple airplanes on one airplane, as well as *minor* variations within airplanes of the same type (e.g. minute differences in shade, color, scale, rotation), while “conservative” thresholds would detect less airplanes due to those very minute differences, but would ultimately be more accurate in their detection. The following image is an example of what we consider to be “liberal thresholds”.



As expected, the liberal thresholds did in fact identify the airplanes, multiple times per airplane. What was actually quite surprising from this result was how normalized cross correlation was identifying these planes. We expected the dots on the image, which were supposed to be indicative of peaks with respect to normalized cross correlation, to collect roughly around the centers of the plane. However, in many cases, the function would detect the planes at either the nose or the tail of an airplane.

This gave us some insight about how normalized cross correlation works with multiple templates imposed upon one airplane, especially if a template is considerably smaller than an airplane that we are trying to match. In other words, the reason as to why there were points on the tails and noses of certain airplanes is precisely because they were matched with templates of small airplanes. This issue would require further manipulation of the thresholds to optimize the program.

In order to optimize the values for normalized cross correlation, we had to run tests on the largest planes first. This is because the largest planes were the most susceptible to having other planes found within them. For example, the tail of the biggest plane could have been registered as a small plane, in addition to being detected as itself.

First, we sorted the planes according to size and identified the biggest plane first. Here, we use the size of the template to represent the size of plane, since each input template is square and unique in size. To illustrate, the big plane is in the middle of the big template picture and the boundary is very near to the body of plane.



Then we delete the intensity existing information from the target picture in order to avoid the misidentification caused by the small plane.

Second, at each plane identification processing, we set a threshold according to the self-correlation information of each input template. Since we cannot extract the same existing information even they are the same plane in the target picture. We set a range from 0.6 to 1.1 of the maximum value happened in the self-correlation.

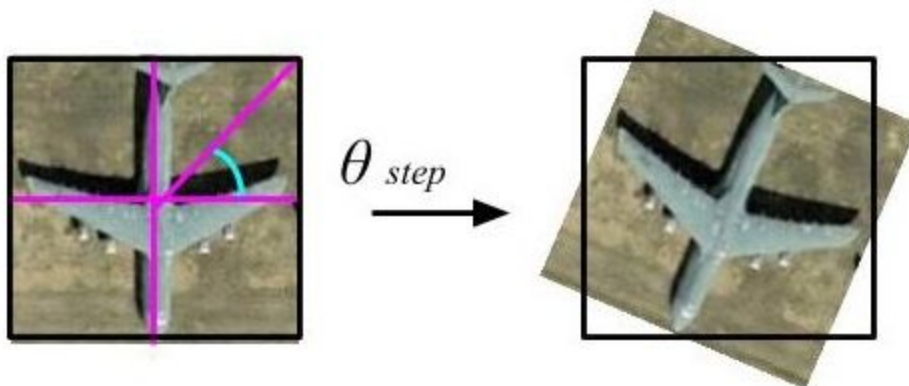
Third, we make selection in the area of one plane identification and decide the maximum value point as the position of the plane and delete other points and locations. The following is the example of the result.

Plane Detection

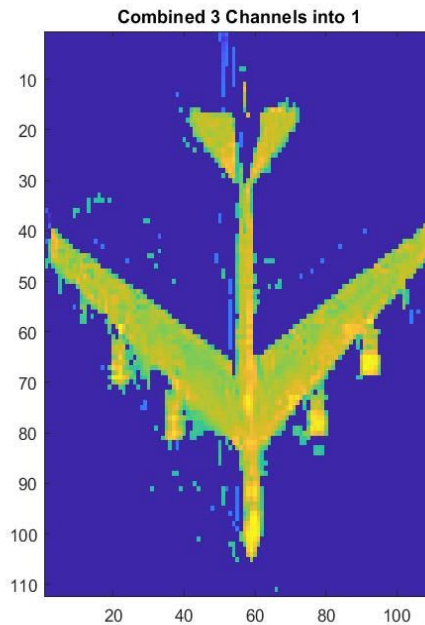
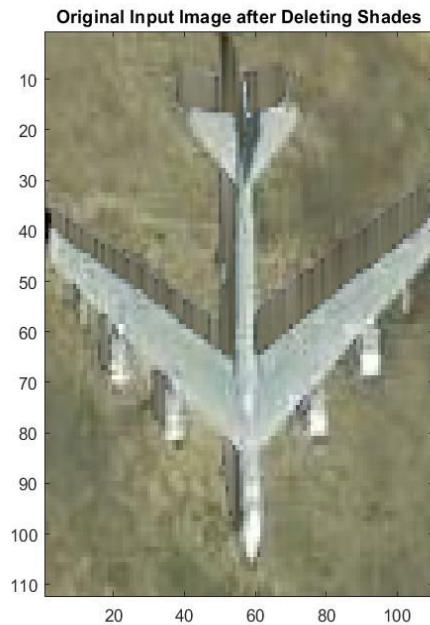


While optimizing the program to fit a certain level of normalized cross correlation worked well for airplanes that were mostly the same as the template, we also had to account for the factors that would possibly slip under our method of detection. Scale was not an issue, as each airplane had a fixed size due to the perspective of the images we were testing against. They were aerial shots that lied on (nearly) perfect horizontal planes, so there wasn't any way for the size of the same types of airplanes to vary. Rotation, on the other hand, was a completely different matter, in that any slight rotation of an airplane based on this "optimal" range of normalized cross correlation values would be passed by undetected.

Accounting for rotation proved to be difficult, as we were restricted by not having access to the integrated matlab harris corner function. Instead, we took a more naive approach to this question. By rotating each template a number of radians and then creating a new template out of this rotation, we could in theory match every airplane depending on the bounds of the normalized cross correlation as well as the amount of rotation we do at every step of this template-creation process. Of course, the matter of how much to rotate by comes into question; having a small factor of rotation would make sure that every plane would have a peak in the normalized cross correlation at the cost of performance, while having too large of a factor would lead to inaccuracy.

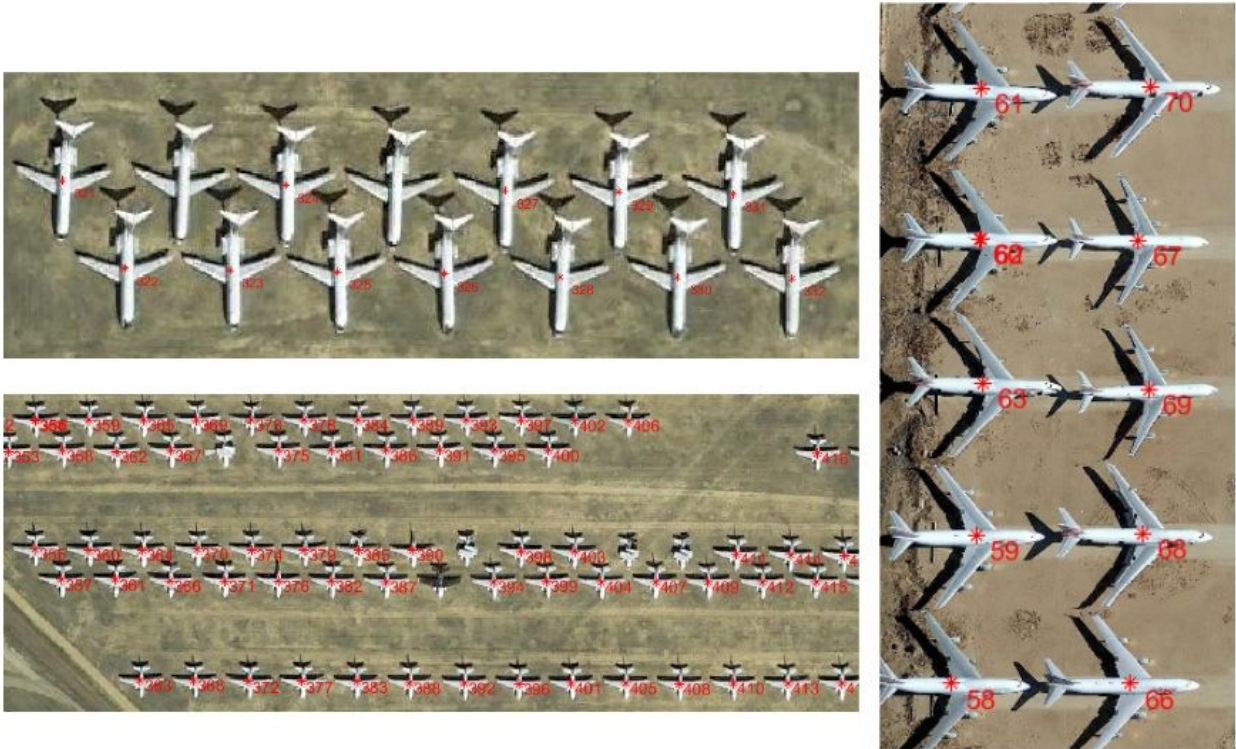


Another issue when dealing with rotation is the problem regarding shadows. The shadows coming off of a plane no longer match with the template due to the airplanes' rotation. This would make it especially difficult to detect larger planes. In order to circumvent this problem, we decided to create a function that would delete any shades in the test image. Our function "DeleteShade" uses the matlab function imhist in order to refine the image such that our extraction process bypasses the shades. The following image illustrates this implementation:



Final results

As expected, simple airplanes that matched our original templates were detected very well by our program.



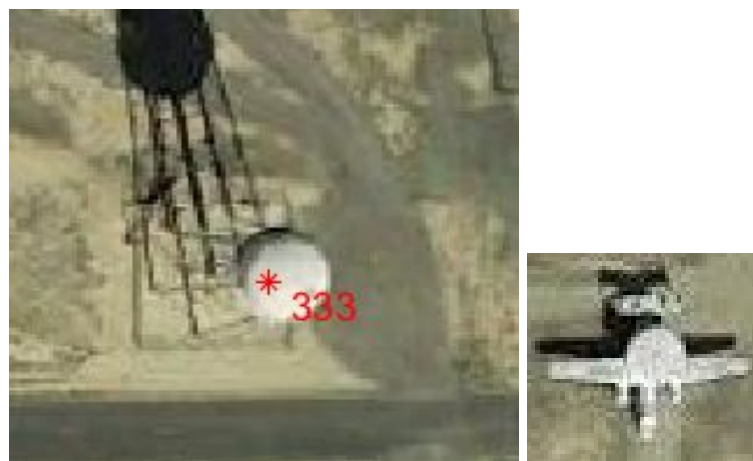
However, we experienced some difficulties when either the shade of the airplane itself was different from our template, or they had additional features that we did not account for. For example, the following image shows a near-perfect match, with the exception of two airplanes. One of the airplanes that was not detected was one that was slightly off-color to our template, while the other one did not have the characteristic thrusters.



The following image exhibits this issue of color more obviously, however it's still important to note that the templates matched for both the rotated and unrotated planes. This color issue was most likely due to the fact that when we deleted unwanted information from the histograms (discussed on page 3), we might have also cut off information that was important with regards to these slightly green-tinted airplanes.



In addition, we found that our program was detecting some false positives. This was most likely due to having one template that was rather ambiguous, meaning that it was prone to detecting objects that were similarly ambiguous. The images below represent a false positive, as well as the template that was most likely the culprit of this detection.



Within our rotated-template-creation process, we rotated each template by 15 degrees. The results are shown below.



Other planes however, proved to be difficult to detect, due to rotation. Our implementation was not robust enough to capture every rotated plane. As mentioned before, our original templates were aligned to be completely vertical (90 degrees), and we created new templates by rotating them by 15 degree steps. There is of course a “buffer zone” of about half a degree, due to the thresholds applied onto normalized cross correlation, so planes that were aligned at barely 15 degree increments were still picked up. The issue however, arose when airplanes were not aligned at roughly 15 degree steps- we would have to adjust the increment to a lower value in order to account for these planes.



Experience

Throughout our journey, we realized that things that were very obvious to the human eye were major structures within the realm of the computer. Our biggest problem area was detecting the rotation of the airplanes. Our rather simplistic approach of rotating filters and directly applying them through normalized cross correlation was rather faulty, as we were limited not only by the actual accuracy of our templates themselves, but also computing power; ideally we would create a template every 1 degree, though managing this idea with 10 templates, each going through an extraction filter, is a major strain on computing power.

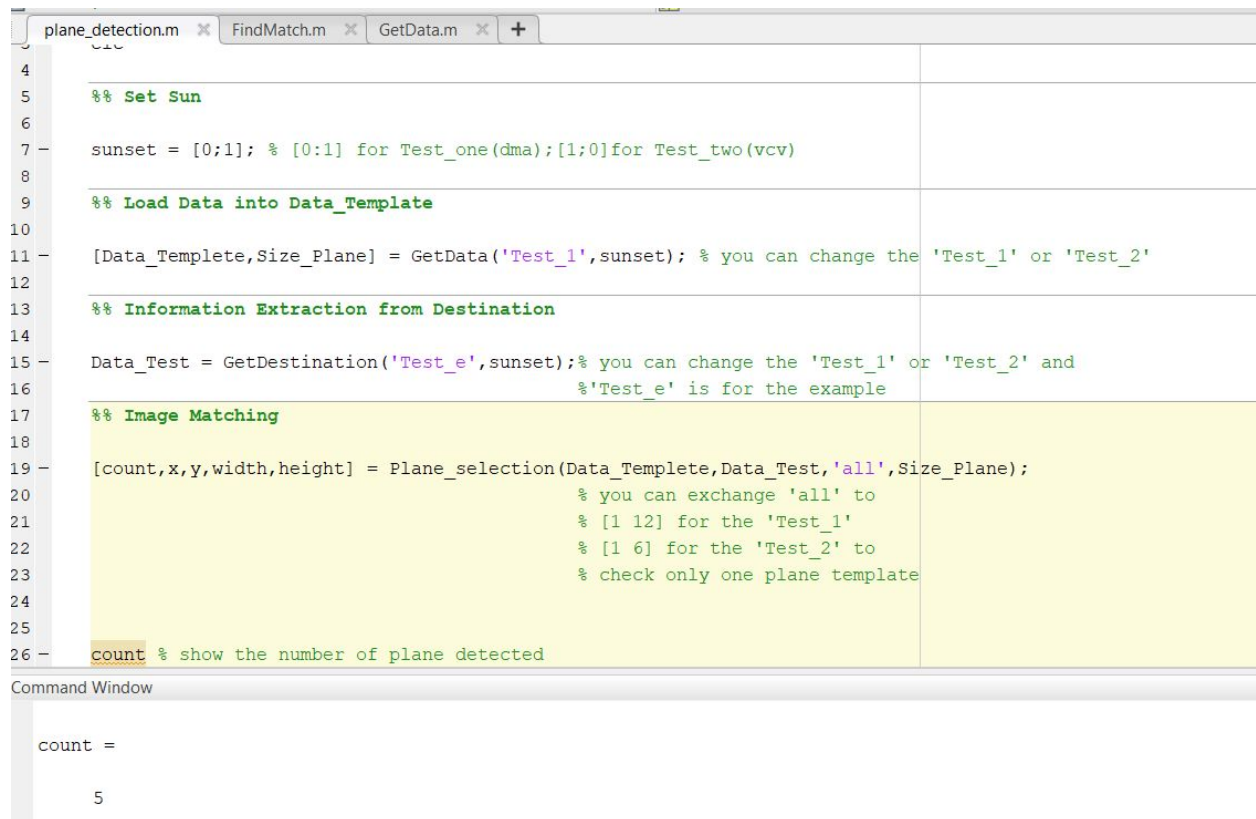
It is very difficult to apply some simple idea into programming such as select the location of the plane, since the position is recorded by x axis first and then y axis. Even the two point are pretty close (such as (1,1) and (4,5)), we can not find them only through the judgement like distance value is less than 10, because we will have the (2,100) , (3,44) after the (1,1).

Also, through the project, we mainly using the intensity information and edge information to detect the plane. Although we can get the pretty clear of them, we still can not find some plane with color changes and queued in the rows. If we can have more time to add SIFT into our program, the result would be better. We have try to use the matlab built-in function 'corner' to detect possible corner and compared with the possible plane we selected. However, the work of that idea is much complex and time wasting. At the meantime, we are also try to use some properties like the symmetric to detect plane. But we do not have much experience how to detect that.

Everybody met up the first week to talk about how we should approach the problem- we all brainstormed ideas and came to a general game plan for an outline of the project. This rough outline entailed creating a feature extraction program, applying normalized cross correlation, and template creation function which included rotation. While every group member contributed to every aspect of the project, Qi Chang and Eryk Maciej Heyssler primarily worked on extracting features, shade deletion, and normalized cross correlation, while Xi Li and Brian Mata primarily worked on template creation and presentation.

Manual (how to run the program)

The main function is plane_detection.m and you can simply run this function to check our program. There is only one value to change on this function that must be changed; it is called sunset (value we use to account for shade and rotation). We use the value [0 1] in Test_1 (dma.jpg) and use [1 0] in Test_2 (vcv.jpg). Additionally, we use 'Test_e' for example.jpg. In line 11, you can change 'Test_1' with the 'Test_2' to determine what database you are going to use. You can change line 15 'Test_1' with the 'Test_2' to determine what target picture you are going to search. You can also change 'All' to any number under the database, to check for a single airplane template.



```
plane_detection.m x FindMatch.m x GetData.m x +
4
5 %% Set Sun
6
7 sunset = [0;1]; % [0:1] for Test_one(dma);[1;0]for Test_two(vcv)
8
9 %% Load Data into Data_Template
10
11 [Data_Template,Size_Plane] = GetData('Test_1',sunset); % you can change the 'Test_1' or 'Test_2'
12
13 %% Information Extraction from Destination
14
15 Data_Test = GetDestination('Test_e',sunset);% you can change the 'Test_1' or 'Test_2' and
16 %'Test_e' is for the example
17
18 %% Image Matching
19 [count,x,y,width,height] = Plane_selection(Data_Template,Data_Test,'all',Size_Plane);
20 % you can exchange 'all' to
21 % [1 12] for the 'Test_1'
22 % [1 6] for the 'Test_2' to
23 % check only one plane template
24
25
26 count % show the number of plane detected
```

Command Window

```
count =
5
```

Also, the FindMatch function includes ranges for the normalized cross correlation. The range for Test_1,Test_e should be .6 - 1.1, while the range for Test_2 should be .55-1.25. The reference image is pasted below.

```
plane_detection.m x FindMatch.m x GetData.m x +
1 function [x,y,width,height,D] = FindMatch(template,destination)
2 [height,width]=size(template);
3 [SrcHeight,SrcWidth]=size(destination);
4
5 D=zeros(SrcHeight,SrcWidth);
6 max_value = SelfCorr(template);
7 y_o = [];
8 x_o = [];
9 D = normxcorr2(template,destination);
10 % figure;
11 % imshow(D);
12 for i=1:SrcHeight
13     for j= 1:SrcWidth
14         if D(i,j)>=0.65*max_value&&D(i,j)<=1.1*max_value
15             y_o = [y_o i-height];
16             x_o = [x_o j-width];
17         end
18     end
19 end
```