CSU44053 Computer Vision
Blue Sign Location & Recognition

Séamus Woods
15317173

05/11/2019

Contents

0.1 Introduction (Spec)

Using the provided code in *MyApplication.cpp* (included within TIPS) as a base you are asked to develop a program in C++ to locate and recognise 81 information signs from Dublin train stations in the 33 provided images (in the Blue Signs/Testing. folder provided with TIPS). The provided code will load all images for you, gives you a framework to develop your code to locate and recognise each sign, and provides you with code that will automatically evaluate the performance of your system. It will show each image with all recognised signs and all mistaken signs (for example the stairs sign highlighted in red below was not located by my version of the solution).

0.2 Overview of Solution

There was two main parts to my solution; Locating the signs & classifying the signs. It's probably easiest to give a brief overview of how I went about these two solutions before going in depth. Before doing anything, I resized the image so that it was four times smaller. This allowed the program to run faster and speed up development.

0.2.1 Locating Signs

The first thing I found that was actually useful was to sharpen the image. Some images were a little bit blurry and made it hard to get a decisive edge one some of them, so this helped a lot. The next thing I needed to do was get a nice edge detection, which meant converting the image to gray scale first. I found that adaptive threshold worked really well for my edge detection, since a lot of the images had some lighting problems. After getting a nice binary image, I found all of the contours in that image, and did some filtering on the area of those contours, amount of vertices the contours had and the ratio between width and height to filter all of the noise out. This method allowed me to find a lot of the signs with the exception of a couple.

0.2.2 Classifying Signs

To classify my images I ended up using template matching, which actually worked quite well for the most part. The first thing I had to do was crop my

signs out of the images and transform them so I had a nice front on view of the signs. This was absolutely necessary since some of the images were taken at awkward angles and would've made template matching very ineffective. I found using the TM_CCORR_NORMED method worked the best for me. I set a threshold, found through trial and error, to filter images that got a low score from template matching.

0.2.3 Detailed Explanation of Methods

For a more detailed explanation of the methods I mentioned above.

Sharpening Image: I used the Laplacian operator to do this, which is an operator to detect edges. The Laplacian operator is defined by $Laplace(f) = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta y^2}$. By using the Laplacian operator on an image, and then taking away a factor of the resulting image from the original image, I was able to get a nice sharpened version of the image.

Gray Scale: Gray scale was needed since most computer vision algorithms require a gray scale image as input. Converting an image to gray scale consists of simply averaging all channels of an image and using the average as the pixel value for gray scale.

Adaptive Threshold: Adaptive threshold is very cool, it allows you to get nice binary images regardless of the lighting in the image. The algorithm consists of dividing the image into sub-images, computing thresholds for all of the sub-images and then interpolating thresholds for every point using bi-linear interpolation.

Contours: When I talk about contours, I'm talking about connected components. To find all of the connected components in an image you need to go through each row of the image, labeling each non-zero pixel. If the previous pixels are background we assign a new label, otherwise we pick any label from the previous pixels. If any of the previous pixels have a different label the we note that these labels are equivalent. The result will be an image of connected components.

Transforming: I used a perspective transform for this, since we're looking at the signs from a perspective. This is basically the conversion of a 3d world into a 2d image. It's done by selecting points of the part of

an image you want to change the perspective of, for example the four corner of our signs. Then we create an equal amount of points of where we want the image to be transformed to, so we give it the four corners of a square, because we know the signs should be square. Using this we can create a matrix, and applying that matrix to our image will transform our sign to be face on.

Template Matching: Template matching works by have a template of an object you want to find, and an image to find a match for that template in. For every possible position of the object in the image, it evaluates a match criterion. It then searches for local maxima of the match criterion above some threshold. Because I didn't have to search the entire image for the template, the template would have been like a sliding window across my located signs, although I would be interested to see how it would have worked if I tried to locate the signs using template matching. There is different matching criteria, and I found that cross correlation worked best for me. Cross correlation is defined by $\sum f(i+m, j+n) * t(m, n)$. To find local maxima, we dilate and look for unchanged values, then we threshold.

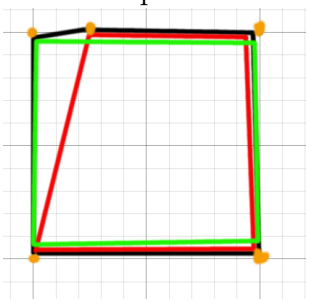
0.3 Issues & Problems with my Solution

I ran into many issues & problems during this assignment. I tried a lot of methods that ended up getting thrown out and even with the methods that I ended up keeping, it was a long process of tweaking to get them working the way they are now.

0.3.1 Main Complications

While I had many issues & problems with my final solution, there were a couple complication that took up the majority of my time. The first was trying to get a really nice binary image. At the beginning I was using Canny edge detection, which looked nice, but the rest of my program wasn't running well with it. I couldn't get rid of a lot of the noise in the image, because setting too low of a threshold meant losing nice edges on the signs. Being honest, this was before I started sharpening the image, so it would be interesting to come back to this. The next thing I tried was just simple threshold.

This actually worked quite a bit better, but still I was having trouble with some edges, because defining a hard threshold was meaning that the darker images with lighting issues were getting little to no edges. Finally I settled on adaptive threshold which was the best of both worlds. It worked nearly flawlessly for all of the images.
The second problem, which took up a huge amount of my time, was dealing with the contours. While I was able to locate the majority of signs using contours, they weren't all easy to filter out. I tried to filter out any contours that didn't have four vertices and any contours with a length by width ratio of less than 0.9 and greater than 1.1. It turned out that a lot of the signs were at such an angle that their length to width ratio was larger than I expected, so for this I just had to increase the threshold. The most annoying part was that some contours of the signs actually had five vertices, which really complicated things later on like transforming etc. I'm guessing that the reason for this was sharpening the image and using smaller image sizes. My approach to fix this was to smooth the contours. I made a function that would iterate through all vertices of the contour, find all of the four sided shapes in that contour, and return the contours with the maximum area. I thought that this would return my sign and for the most part it did with some exceptions.



At an attempt to explain what I'm talking about with the contours a bit better, this is an example of a contour I would get for the signs with five vertices (orange). My idea is the four sided shape with the most area would be what I was looking for (green) but sometimes, because in some instances these two shapes would actually have the same area, the red shape is what would be returned, which is why some of my distortions are very weird. In an attempt to fix this, I kept both contours, did template matching on both of them, and only kept the one with the best score from this, this worked most of the time but not always.

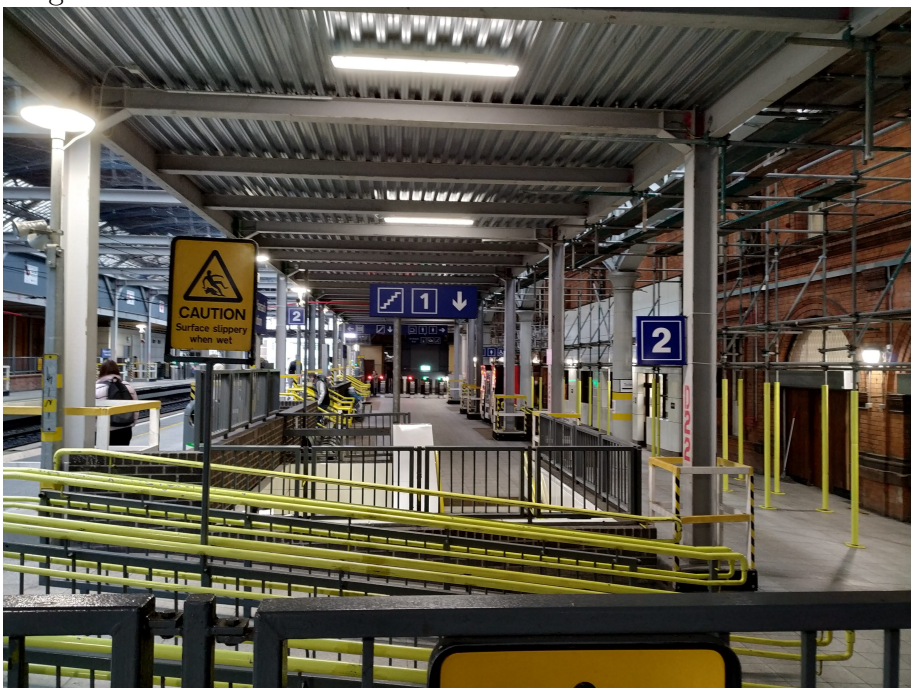
0.3.2 Smaller Issues

The fact that I was using two images introduced some small problems. Looking back, I really wish I didn't resize the image to begin with, because I think I would have gotten a better score and not have to put so much of my time into this assignment. It got to a point where I was getting a decent score with the resized image, and when I tried to use the original image I would get a lot lower of a score, so it would have been a lot of extra time to change everything I had done so far. That being said, because I was using two different images, I was recognising some signs twice. I made a function to remove duplicate signs which just checked if a point of one of the located signs was within a radius of another located sign, and if it was, I would remove the one that got a lower score from template matching.
Another problem I came across was that I couldn't determine the order of vertices in my contours, (e.g.) which index is the top left vertex at? This made it so I didn't know the orientation of my sign and gave me problems with template matching. To overcome this I did template matching with 4 different orientations of my located sign (at 90 degree intervals) and used the orientation which got the highest score. This pretty much fixed the problem.

0.4 Sample Result Images

0.4.1 Run through of solution

Original:



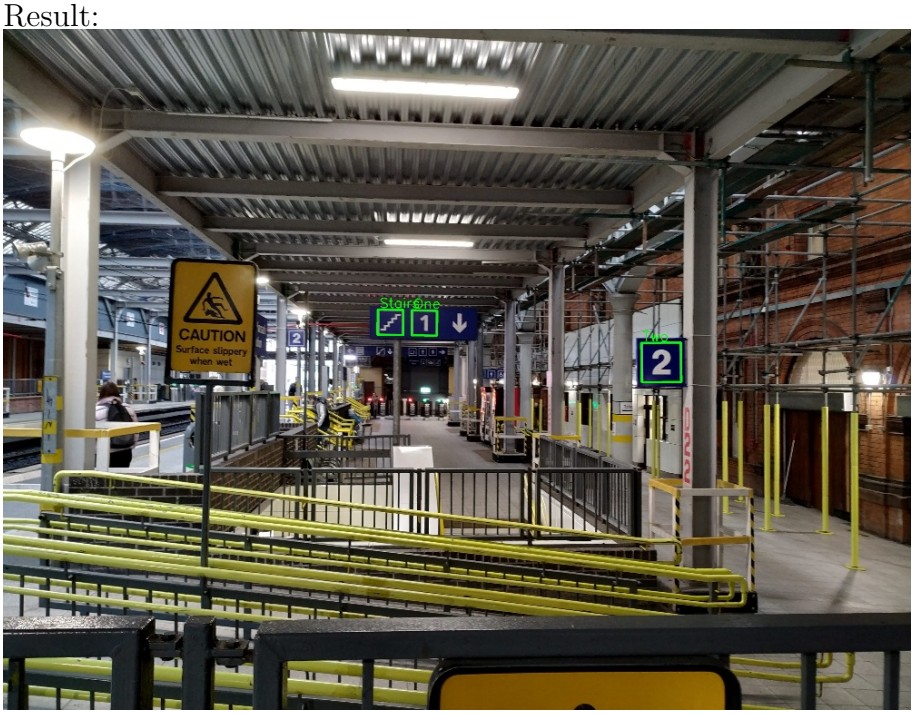
Sharpened:

Adaptive Threshold:

Contours on image/4:

Contours on image/3:

Result:



Hopefully it's clear here the reason I tried to use both images and not just a larger one. The larger image would help recognise signs that were far away, but for some reason couldn't recognise signs that were closer, and vice versa for the smaller image. This was the case for most images, but by using both I was able to find nearly all of the signs.

0.4.2 Example of mismatch & duplicates

Duplicate:

Mismatch:

Duplicate & Mismatch:

I think the following images show where I lost most of my marks. With template matching I had trouble correctly classifying the 'male' sign. I guess this is because the shape of it looks a lot like the 'information' sign or the 'one' sign. I also lost marks for duplicate signs which I tried to get rid of with some success, maybe if I spent more time on this part I could have gotten rid of them completely. I also lost marks for signs that I just didn't recognise at all (red signs), but this was ultimately a result of trying to filter out the mismatches and duplicates, and not because I couldn't locate them.

0.4.3 All Recognised Objects

You can see here where some of my methods let me down. There are some signs that are very distorted, and the reason for this is because my method of trying to smooth the contours failed and actually returned the wrong co-ordinates. You can also see how I sometimes incorrectly classified the 'male' sign with information in a couple of instances, because the male upside down

was a better match with information than male was with male the right way up. Same thing with 'bike' being recognised with 'one' when it's at that orientation.

0.5 Performance Metrics

			Recognised as:											
		Coffee	Disabled	Escalator	Exit	Gents	Information	Ladies	Lift	One	Stairs	TicketDesk	Two	False Negative
Ground Truth	Coffee	4	0	0	0	0	0	0	0	0	0	0	0	1
Ground Truth	Disabled	0	3	0	0	0	0	0	0	0	0	0	0	0
Ground Truth	Escalator	0	0	2	0	0	0	0	0	0	0	0	0	0
Ground Truth	Exit	0	0	0	3	0	0	0	0	0	0	0	0	2
Ground Truth	Gents	0	0	0	0	4	2	0	0	0	0	0	0	1
Ground Truth	Information	0	0	0	0	0	7	0	0	0	0	0	0	1
Ground Truth	Ladies	0	0	0	0	0	0	6	0	0	0	0	0	1
Ground Truth	Lift	0	0	0	0	0	0	0	8	0	0	0	0	3
Ground Truth	One	0	0	0	0	0	0	0	0	10	0	0	0	0
Ground Truth	Stairs	0	0	0	0	0	0	0	0	0	6	0	0	3
Ground Truth	TicketDesk	0	0	0	0	0	0	0	0	0	0	5	0	1
Ground Truth	Two	0	0	0	0	0	0	0	0	0	0	0	6	3
Ground Truth	False Positive	0	0	0	0	0	0	1	0	1	1	0	0	0
Precision = 0.927536														
Recall = 0.780488														
F1 = 0.847682														

Precision: 0.93
Recall: 0.78
F1: 0.85

I was happy enough with this score, but I feel I could have done a bit better if I got SVMs to work and if I had used the original full size image from the beginning. I lost some precision from recognising signs such as the bike and the luas sign, and because I was using template matching these signs were still getting a relatively high score, and I found it difficult to threshold them out. I also got quite a few false positive for random square like shapes like the 'Poblacht na hEireann' sign. Template matching was just giving too high of a result to threshold out. As I lowered the threshold, my precision would go up a bit but my recall would go down. I just had to strike a balance between the two to get a good score, but if I got SVMs working and picked good features, I would probably have been able to filter these out and keep my recall score high. Just to show that I understand the metrics.. precision is the ratio between the number of true positives and the sum of true positives and false positives, recall is the ratio between the number of true positives and

the number of relevant elements (sum of true positives and false negatives), and F1 score is $2 * \frac{precision*recall}{precision+recall}$.

0.6 Final Results & Possible Improvements

I did a lot of tweaking to get the highest possible F1 score. The max recall I could get was around 0.85, but my precision would go way down, and in turn my F1 score. I was actually able to locate nearly all of the signs, the real problem was classifying them. So in order to improve my F1 score, I would try to implement a different method of classifying sign, for example support vector machines. I really tried for a long time to implement SVMs, but I wasn't getting good outputs when I tried to predict. I even trained it with the training images, and then tried to predict one of the training images and still I was getting weird outputs. I tried multiple combinations of features and still didn't have any luck. I debugged your example and the output is similar.. for your example you just need to classify whether an o-ring is good or bad, so it's a binary classification, and you just check whether your prediction is less or greater than 0. The output for the prediction would be something like 1.006^9 or -1.006^9, which is similar to what I was getting even though my label vector had multiple classes. I wasn't really sure what was going on here.

Even without implementing SVMs, I could have improved how I was doing the template matching. In my program I'm doing template matching with the raw images, perhaps doing some kind of threshold on the images would have produced better results, but I just didn't get around to it.

Beyond the methods I used, it probably would have made a big difference if I just used to full size image and not two separate images. It should have been better for locating the signs and left less room for error. I also wouldn't be losing score for having duplicate recognition on some signs. Its maybe good to mention that while I used smaller images for locating the signs, I transformed the vertices of the signs to the larger original image for cropping, so I would get the best resolution for template matching.

0.7 Did I try anything else?

Yes, I tried a lot of different methods for locating the signs. If you look at all of the commented out parts of my code this is some of the things I tried (some I just deleted). I spent quite a bit of time trying to located the signs using a Hough transform for line detection. My idea was to find all of the lines in the image, find all of the intersection of those lines and then maybe I could find the signs using that. In the end it didn't work out because there was too many lines being detected because at the time I couldn't get a nice binary image. I looked into Harris corners for a bit, which looked promising but I didn't know how to distinguish between other corners in the image without doing something overly complicated. I tried using dilation and erosion to repair broken lines of signs in my binary images. I was actually surprised that this didn't work, I got much worse results using this. I tried equalizing the images which worked okay for some images, but ultimately it wasn't an improvement overall. I tried to remove noise from the image using Gaussian blur, but this made it so I couldn't get nice edges on the signs. I tried k-means clustering to get a solid colour on the signs, and then back projection on the images, this worked only for very few images, I guess because the training data wasn't large enough and the ground truth images were quite a bit darker. I tried different colour spaces which I thought would be less sensitive to light like LAB, which didn't turn out great, and as mentioned above, I tried to implement SVMs with no luck.