Name: Zaigham Abbas Randhawa

Precision = 0.942029

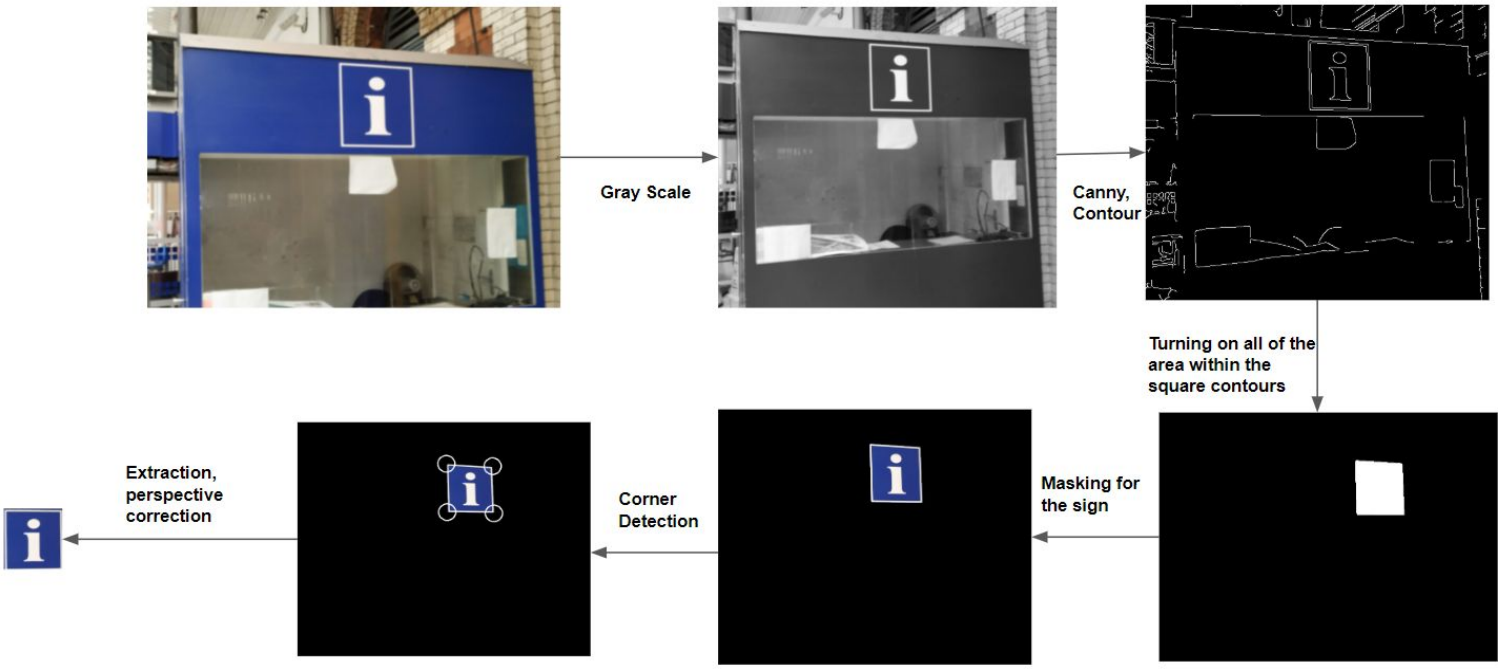
Recall = 0.802469

F1 = 0.866667

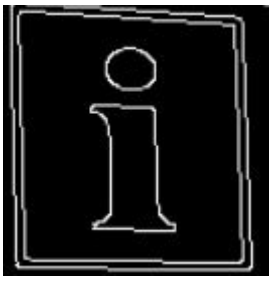
- **Location of the white boxes and issues, problems, and suggestions**

I began with the understanding that the white-signs in the images were squares. Even with a perspective distortion in the image, they formed a perfect parallelogram. Not a lot of objects that you will see in these images had such qualities. Hence, the problem was just a matter of how well I could identify such near-rectangular features within an image.

The best idea was to use a connected component analysis like contour-detection and then filter through them for the most rectangular regions and filter them out. The flowchart below summarises my approach.

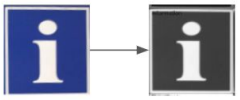


So after finding the contours, I just filter through them. The ones which meet the custom built standards (like the aspect ratio, area ratio of the contour to that of the Bounding rectangle, Area of the contour above the given threshold etc) are the only ones which are used further. Please look at a segment of the contour image below. The white lines represent the drawn contours.



The problem is that at the moment we don't know which contour represents a sign: Even within a sign, there are multiple contours which fulfill the criteria of rectangularity. A single contour doesn't hold enough information to represent the entire information of the sign. Fine tuning the edge-detection/contour detection to only detect the external-square contours was going to take too much time. The solution was to turn-on/fill the area of these contours to get the total area of the sign and then mask it with the original image to get back the complete sign.

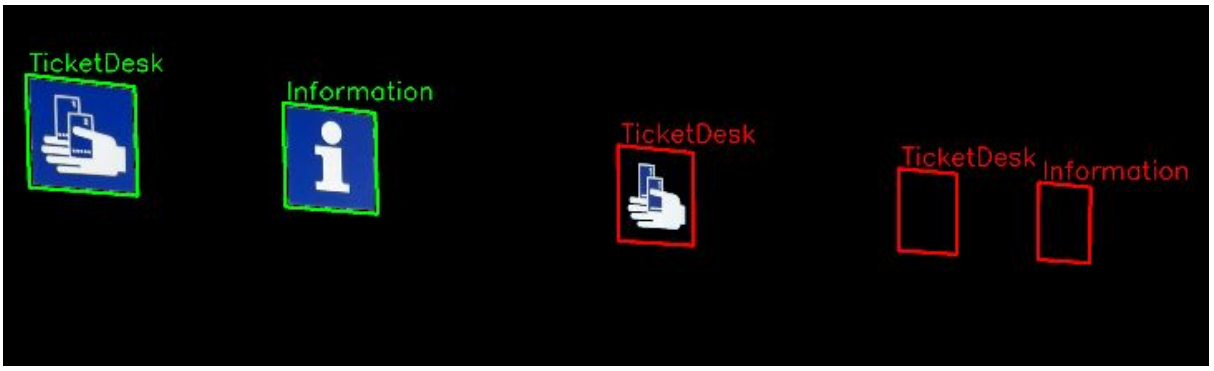
Since we should ideally have only the blue signs inside the image now, we can find the "**external**" contour for each sign. Then for each of these external contours we find the top-left, top-right, bottom-left, bottom-right points, and its height and width. This is followed by applying perspective transformation and data extraction on the image to get each sign which are added onto the ObjectAndLocation-vector. This is followed by converting the image to grayscale for template matching. Ideally we should have the image below.



However, honestly speaking for computer vision, the real world is not the most "ideal" of test grounds. There are just so many factors that you can't control. For example, my system failed to identify these images because of an intensive light reflection on the white boards (I used the masked image to see how much of the edges had I detected). My system just didn't find any significant edges in here. Maybe having had applied some kind of light-reflection correction would have helped.

The system also fails if the sign has a bit of variance from my given standards for the contours. Like in the picture below, the right-two most signs are failed to be detected. It could

be improved by fine-tuning the parameters even more.



This sign detection system will fail if we come across signs which are square but not blue, as shown below. The only solution is to check for an average-sample color within these detected signs to see it they are somewhere near blue or not.

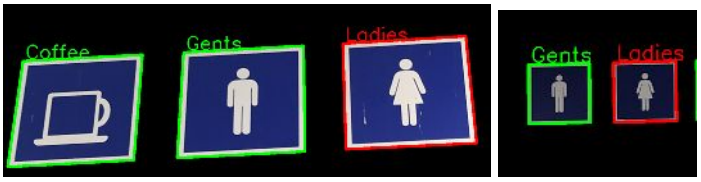
- **Recognition of the signs and the required performance metrics.**

Recognition of the signs was the easiest part (I just compared templates of each type with the given image and choose the best score, a OvsO approach). I aso iterated through all of the available methods and concluded that **cv.TM_SQDIFF_NORMED** was the best. Additionally, I had to devise a data-transformation technique to make it compatible with the given frameworks. Below you can see the output.

		Recognised as:												
		Coffee	Disabled	Escalator	Exit	Gents	Informatic	Ladies	Lift	One	Stairs	TicketDes	Two	False Negative
Ground Truth	Coffee	5	0	0	0	0	0	0	0	0	0	0	0	0
Ground Truth	Disabled	0	2	0	0	0	0	0	0	0	0	0	0	1
Ground Truth	Escalator	0	0	2	0	0	0	0	0	0	0	0	0	0
Ground Truth	Exit	0	0	0	5	0	0	0	0	0	0	0	0	0
Ground Truth	Gents	0	0	0	0	6	0	0	0	0	0	0	0	0
Ground Truth	Informatic	0	0	0	0	0	7	0	0	0	0	0	0	1
Ground Truth	Ladies	0	0	0	0	0	0	0	0	0	0	0	0	7
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	7	0	0	2
Ground Truth	TicketDes	0	0	0	0	0	0	0	0	0	0	4	0	2
Ground Truth	Two	0	0	0	0	0	0	0	0	0	0	0	9	0
Ground Truth	False Posi	0	0	0	0	0	1	0	0	0	3	0	0	0
Precision = 0.942029														
Recall = 0.802469														
F1 = 0.866667														

- **Failed classifications.**

My system generally works well, it has a decent ReCall and precision. Most of the signs are classified quite well. However, the template matching strategy fails badly on some types of signs, e.g. the Ladies sign. It is probably because the female sign resembles the male sign very closely. This issue could possibly be improved by implementing some kind of a decision tree.



Here are some success cases:

- **Discussion of the results including any issues/problems with the metrics and the ground truth (and suggestions for improvements if possible)**

In this Image, the blue sign is way too close to the screen than the image in the background. Since, it is closer to the camera it should be detected as well (and it would actually help an autonomous robot more to be aware of the signs closer to it than father away). However, the framework provided doesn't let us do it because of an imposition of the minimum sign area. It makes for most of the images because the farther away images are smaller in size, however here this imposition changes the ground-truth for this image in a way which might not have been the best idea. Maybe having a 2.5D image (the ones with some kind of a liDar

data in them) might help with this distance problem (i.e. instead of using an area threshold, we could use the actual real life distance as a threshold for significance).

Another huge issue is the fact that the bicycle sign below is meant not to be detected by the system because this sign is not in our training data. However, it does fulfill the criteria of a blue sign and is detected by my system. By making the system in such a way such that it refuses to accept this sign, we are calling the credibility of the system into question? Maybe we could have some kind of a model in the backend which creates a new category of signs in the training data upon stumbling a new sign (i.e. make the model learn unsupervised over time). This will make the system more robust and useable.

Additionally, the system had a “REQUIRED_RATIO_OF_BEST_TO_SECOND_BEST” parameter. Honestly speaking, I feel like it wasn’t needed. I played around with it and discovered that my final accuracy went up by a couple of points if I reduced it.