Part I - Shape Detection

Viet-Anh Tran*
*Department of Computer Science, University of Bristol
zk20550@bristol.ac.uk

***Abstract*—Given 16 images, we are tasked with implementing an image processing algorithm that detects no-entry signs. We have achieved an overall F-1 score of approx. 0.85 and an overall TPR score of 0.86, by making use of the techniques we have learned in our Image Processing and Computer Vision unit.**

I. Viola Jones and Cascade Classifier

In the 2001 paper "Rapid Object Detection using a Boosted Cascade of Simple Features", Paul Viola and Michael Jones propose Haar feature-based Cascade Classifiers as an effective object detection method. It is an approach that uses machine learning, where a cascade function is trained from a lot of positive and negative images. We used this method as the initial step for our no entry detection system.

We took an unintuitive approach to designing it - we decided to have our Cascade Classifier be "less accurate" by training it for only 2 stages. Instead, we supplied it with more data - we have accumulated 7000 negative images, and created a positive training data set of 14,000.

By using this approach, we are able to get more true positives, at the cost of receiving a lot more false positives. The rest of the algorithm works by going through each detected box and filtering out all the invalid ones - a high-risk, high-reward detection system.

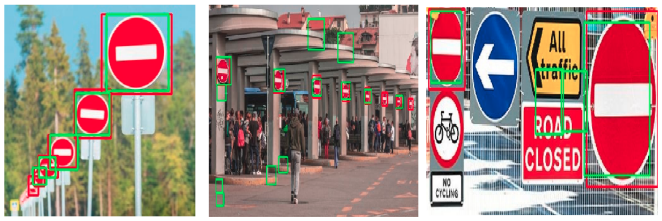

Fig. 1. Merits of using a riskier classifier

In order to make it even easier for Viola-Jones to detect our desired no-entry signs, we decided to apply a mask - all the pixels that are red (or close to red) maintain their original BGR values, and the rest will turn white. More information about masking will be done at a later section.

As shown in Fig. 2, we observe a sharp decrease in the False Positive Rate. This is because our classifier gets better at distinguishing the characteristics that make up our initial No Entry image. It continues this process until obtains the accuracy we set it to achieve (in our case, a maxFalseAlarmRate of 0.01). If it were the case that this accuracy was not feasible, it would have aborted the procedure.

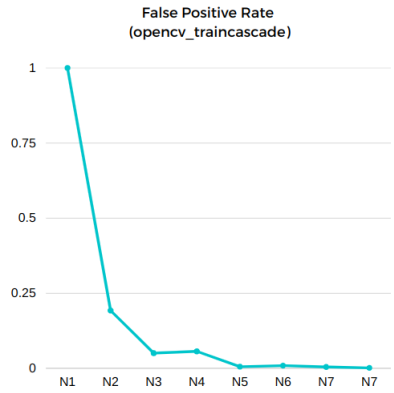

Fig. 2. FPR drops as the Classifier is Training. TPR remains at 1.

Fig. 3. Pre-processing result before Viola-Jones

II. Viola-Jones and Hough Pipeline

Hough is a feature extraction technique that is used in various image processing tasks, with the purpose of identifying instances of objects within a certain class of shapes with the help of a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained if and only if they pass user-selected thresholds.

Our implementation contains the Hough Circle and Hough Line detection algorithms. As these computations are really expensive and time-consuming, we decided against running these on the entire image, but on sub-parts of it. A disadvantage of this is that we are not detecting signs which haven't been caught by Viola-Jones in the first stage, but the vastly diminished processing time and the fact that our Cascade Classifier detects more true positives than usual more than make up for this (during our tests, we have noticed only one sign was not caught by our Cascade Classifier vs. Running Hough Circle on the entire image).

We do this by only going through the detected boxes we have previously identified using Viola-Jones. Given a parameter *delta*, we can get more pixels that surround the box - this is to ensure that we get the entire sign.

For each of the boxes, we then apply Hough Circle. If our Viola-Jones was able to detect a no-entry sign, and the Hough Circle confirms it then we have ourselves a green box.

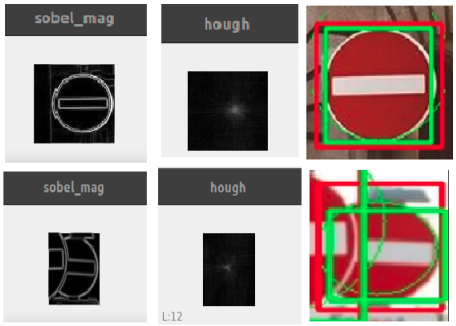

Fig. 4. Hough Circle inside Viola-Jones Boxes

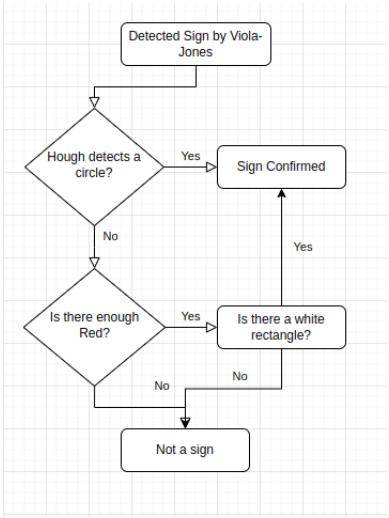

Fig. 5. Detection System Pipelines

If it fails the Hough Circle check, we are going to perform two more checks that both need to be passed - the existence of a white rectangle in the image and how much ”red” is in there.

III. Colour Masking and Closing Morphology

We have chosen to detect how much ”red” there is in a subsection of the image because its one of the distinctive features of our sign.

To detect how much of it is in the image, we first convert it to HSV. The reason we do this is because it will be easier to detect which pixels are red-ish and which are not, based on the *hue* value.

A problem we have encountered was that red-ish hue values wrap around. We overcame this by creating 2 masks - one for hue values between 0 and 3, and one for values between 175 and 180. The Shade and Value channels have no value-restriction. We OR'd these to get our desired mask.

After applying this mask, we decided to apply a Morphological Transform in order to slightly improve the shapes we got. Since Morphological Transforms generally work with binary images, we decided to pre-process our masked image slightly before applying this operation.

The morphological transform we used was Closing. It is useful in closing small holes inside the foreground objects, or small black points on the object. We do this in order to get a more accurate number of red-ish pixels.

Finally, we count how many non-zero pixels we have and return a True or False, depending on whether it passes a threshold check or not.

Fig. 6. Counting number of red pixels

IV. White Rectangle Detection using Contouring and Opening Morphology

White Rectangle is a desirable feature as all no-entry signs have it. Determining if an image contains one was a bit tricky, but it boiled down to - how can we detect a rectangle and how can we know if it is white? The answer was contouring in conjunction with the HLS colour space.

HLS is a particularly interesting colour space, as it contains a *light* channel. This makes it easier to detect white-ish pixels, as opposed to using HSV, by simply looking for pixels with a high value in that respective channel. We construct a mask for this purpose and apply it to our mini image.

In order to get better rectangle detection, we have also applied Canny Edge Detection and Adaptive Thresholding.

Canny Edge detection was chosen as we wanted to highlight the edges as much as possible when we perform contouring.

Adaptive Thresholding was used to make our image binary, and to combat issues such as different lightning conditions in different regions of the image. Unlike normal thresholding, which relies on a single global value, this algorithm detects a suitable threshold for each pixel based on its surrounding area. This will be especially useful to us, as the images are taken from real-world places, where illumination plays a major role in how the resulting pictures are.

We have also made us of a Morphological Transform - Opening. This transform is useful in our purpose, as it removes noise which might have otherwise been accidentally detected as a rectangle.

Contours enable us to detect shapes in images. In order to know if there might be a rectangle in there, we bound one and check its size. We also made sure there is only one single rectangle.

If width is greater than double the height, and the $sizeof(rectangle)/sizeof(image)$ falls between 2 thresholds, we can return True, else False.

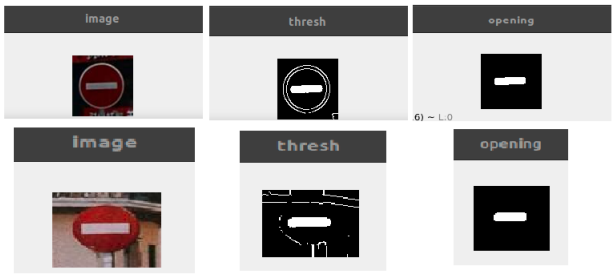

Fig. 7. Rectangle detection

V. Analysis

As expected, we are not going to get a TPR score of 1, as the Cascade Classifier predicted. This is because the Classifier is trained on a really simple image, whereas the images we apply our algorithm on are really complex, as they contain illumination issues, occlusion issues etc.

For our analysis, we decided to compare our first and final implementations and see how many true positives we lose because of our filtering. We do this by comparing the their TPR values on each image. We notice that there is no sharp decrease in the score. We can therefore conclude that our filtering works effectively without affecting the true positives too much.

We are also looking at the final F-1 score of each image, together with the TPR values and obtaining the deltas. We can notice that the overall TPR score is higher than the overall F-1.

However, our final implementation scores a much higher F-1 score. This is because the F-1 score is based on more factors, such as False Positives and False Negatives. Compared to the initial version, this one performs better because it is able to reduce the number of False Positives. The number of False Negatives, however, stays the same.

TPR Initial vs. Final			
Image	TPR Init	TPR Final	Δ
NoEntry0	1.0	1.0	0.0
NoEntry1	1.0	1.0	0.0
NoEntry2	1.0	1.0	0.0
NoEntry3	1.0	1.0	0.0
NoEntry4	1.0	1.0	0.0
NoEntry5	0.6	0.2	-0.4
NoEntry6	0.5	0.5	0.0
NoEntry7	1.0	1.0	0.0
NoEntry8	0.833	0.666	-0.167
NoEntry9	1.0	1.0	0.0
NoEntry10	1.0	1.0	0.0
NoEntry11	1.0	1.0	0.0
NoEntry12	0.375	0.375	0.0
NoEntry13	1.0	1.0	0.0
NoEntry14	1.0	1.0	0.0
NoEntry15	1.0	1.0	0.0
Overall	0.894	0.858	-0.036

F-1 Initial vs. Final			
Image	F-1 Init	F-1 Final	Δ
NoEntry0	0.8	1.0	+0.2
NoEntry1	1.0	1.0	0.0
NoEntry2	0.153	1.0	+0.847
NoEntry3	0.666	0.8	+0.134
NoEntry4	0.571	0.8	+0.229
NoEntry5	0.5	0.333	-0.167
NoEntry6	0.363	0.666	+0.303
NoEntry7	0.4	1.0	+0.6
NoEntry8	0.909	0.8	-0.109
NoEntry9	0.5	1.0	+0.5
NoEntry10	1.0	1.0	0.0
NoEntry11	0.125	1.0	+0.875
NoEntry12	0.285	0.5	+0.215
NoEntry13	0.666	0.666	0.0
NoEntry14	0.666	1.0	+0.333
NoEntry15	1.0	1.0	0.0
Overall	0.598	0.847	+0.249

VI. What was Implemented but Not Used and Why

Hough Line was implemented by following the algorithm presented in the lectures. Initially, our plan was to detect lines across the image, compute line intersections and check which boxes have about 4 intersections (signifying the presence of a rectangle).

However, while our implementation was successful in identifying all types of lines (vertical, horizontal, diagonal), it outputted too many lines for the same ”real-world” line. This would overestimate the number of line intersections in a region of the image, rendering us to look for alternatives for rectangle detection.

K-Means Colour Clustering was initially used to help with Hough Circle detection, as a pre-processing step. It does this by performing Colour Quantization, which is the process of reducing the number of colours in an image. By how much is entirely dependent on what value we set K to be. Since this is entirely context-dependent, and the contexts differ dramatically from image to image, after further analysis we decided that this feature was not worth using because of its randomness.

While not used, the previous features can be found in our code.

VII. Further Improvements

If processing time was not an issue, running Hough on the entire image would help find the small number of signs that were missed by Viola-Jones.

Additionally, improving our filter by, for example, implementing Hough Ellipse Detection would certainly decrease the number of false positives.

VIII. Conclusion

We would like to thank the lecturers for an interesting and intellectually-challenging piece of coursework.