

# Real-time Traffic Sign Detection

Hassan Shojania  
shojania@ieee.org

## Abstract

*An implementation and limited extension of a traffic sign recognition method based on the work by [1] is presented here. This implementation can be extended to target real-time detection. Yield sign, stop sign and red-bordered circular signs are considered. First, image is color segmented based on a thresholding technique. Then, corner features are detected using convolution masks (based on work in [2] called optimal corner detector in comparison to Canny's work). Geometric constraints used for shape recognition along verification methods for each sign.*

**Keywords:** *Traffic sign recognition, corner detection*

## 1 Introduction

Traffic sign recognition (TSR) can be considered part of the bigger problem of autonomous vehicles. An autonomous vehicle system relies on vision-based recognition of surrounding area in order to make driving decisions. This vision-based recognition system functions as the feedback provider for control of steering wheel, accelerator, brake, ..., and needs to:

- Recognize road and lane to allow control system follow the course of own vehicle.
- Detect obstacles on the road till control system avoid them.
- Detect the passing vehicles (e.g. by side or back cameras) to notify the control system about probable hazards.
- Detect and interpret the traffic signs to provide feedback for safe driving.

Traffic signs provide important information for drivers about road condition and hazards. Their discriminating shape and colors make them easily recognizable by humans. Same factors can help development of a vision-based TSR system. Beside the application of TSR in autonomous vehicles, it can also serve as an assistant driver (e.g. when combined with speedometer output) to notify the driver about approaching a traffic sign (e.g. even before driver sees it) or his risky behavior (like driving above the speed limit).

There have been several big projects in the autonomous vehicle area. PROMETHEUS (Program for European

Traffic with Highest Efficiency and Unprecedented Safety) was an initiative started in 1986 in Europe to stimulate research in development of a transport system to exploit full potential of information and telecommunication technologies to improve traffic efficiency and safety [3]. *Safe driving* was one of the three identified main work areas and meant to employ autonomous vehicle control for safer driving with less mental load on the driver. A TSR system was developed in Daimler-Benz as part of the *collision avoidance* project. It employed a *detection process* to scan the image for possible traffic sign candidates and a *tracking process* to identify the sign and track it in the following images. Three sub-units (called specialists) were used: *color segmentation specialist* to find a sign candidate based on color of regions in the image, *shape recognition specialist* to classify the candidate based on its contour and *pictogram recognition specialist* to identify the pictogram inside the traffic sign by comparing it against a library of possible pictograms.

UC Berkeley's PATH is another example. It is a collaboration between the California Department of Transportation (Caltrans) and the University of California with the mission of applying advanced technology to increase highway capacity and safety, and to reduce traffic congestion, air pollution, and energy consumption [4]. It consists of several projects. For example its *Stereo Drive* project explored the feasibility of use of stereo vision (for providing range information) in conjunction with a scanning laser radar sensor to detect obstacles and also maintain a fixed distance from a lead vehicle using feedback provided by range sensors.

In particular area of TSR, there have been many research in academia and also industry (e.g. as part of PROMETHEUS) especially in the mid-90's. Different approaches have been used in different stage of the problem from color segmentation, control theory, feature extraction, learning-based neural networks, morphological based recognition, ... [5] cascades three modules: an ART2 neural network module for color segmentation, a log-polar-exponential grid and Fourier transformation module to extract invariant traffic sign signatures (its first 16 Fourier coefficients), and a back-propagation neural network module to classify such signatures. Instead, [6] uses a split












	European	North American
Warning	 	 
Regulatory & obligation	 	 
Informative	 	

Figure 1: Some of European and North American signs differences.

and merge concept for a color segmentation technique insensitive to noise and brightness variation. [7] discusses pictograph identification of different class of signs (which previously detected and classified): a structural approach for arrow signs, a number detector for speed limit signs, a general nearest neighbor classifier and neural network for prohibition signs.

In an ideal condition (off-line indoor detection of signs with direct front view) traffic sign recognition is not very hard in principle as signs have discriminating color and 2-D shape. Some of the main issues involved are:

- Variety of signs with all different colors, shape and pictographic symbols.
- The complex and uncontrolled road environment (lighting, shadow, occlusion, orientation, distance, ...) which all can be categorized as *noise* (see Section 4 of [5] for description of variety of noise for a TSR system.
- And real-time recognition.

In this paper, we first overview the method in Section 2 and explain it in details in Section 3. We discuss the implementation in Section 4 and provide the test results in Section 5. Section 6 briefly overviews how this implementation can be extended. And finally Section 7 presents our conclusions.

## 2 Overview of the method

Like other classical object recognition problems, our problem can be divided to *localization* and *identification*; first an area with probable sign is segmented, then passed for identification. Not every approach makes such a clear distinction between localization and identification. While [3] does these two steps quit separately, localization in our case also identifies the sign partially (depending on type of the sign).

Our method is based on the work by De La Escalera et al. in [1]. They applied their method to European warning signs (equilateral triangles with one vertex upward) and circular prohibition signs (see Figure 1 for comparison of European and North American signs). Here are the main stages of their method:

**Color segmentation** Based on the traffic signs to be detected, input image is passed through a color mask. The output is a thresholded binary image carrying white pixels for the points where the desired color range exist and black pixels otherwise.

**Corner detection** A set of corner detectors (based on convolution masks) are applied to the binary image from the previous step. To reduce number of false corners, corners within a neighborhood will be combined into their center of mass.

**Shape recognition** Using geometry information of signs of interest, corner combinations creating possible signs are looked up. Every possible sign area is normalized to a 30\*30 pixels image.

**Sign classification** A classification method based on neural network used for classifying the signs at this stage.

Figure 2 describes the above stages. [1] restricts the signs of interest to warning signs (triangles with upward vertex), prohibition signs (circles) with white or blue background and red border, and rectangular informative signs. Yield sign (inverted triangle) and stop signs were excluded. The reason for choice of signs was not provided in the paper. As they experimented with two sets of training sets, one for circular signs and the other for triangular signs, we suspect they were looking for a variety of signs all with the same shape (triangular or circular) but with different pictographic content.

Nine ideal signs (segmented sign without noise) were selected for training the network for each group. Beside the ideal signs, different variation of them created (5 rotated signs within  $\pm 6^\circ$ , three Gaussian noise levels, 4 different color threshold levels, three pixel displacement to the left and right) ending with 1620 training patterns. From three different network size studied, the one with  $30 \times 30/15/5/10$  responded the best with 82 – 98% success rate for detecting signs from another set of *test images*. The achieved recognition speed was 220ms for detection phase for a  $256 \times 256$  image on a special image processing card. The neural network was taking 1.2s on a PC which the authors intended to decrease with implementing the algorithm on a DSP instead (hoping to achieve 30 – 40ms).

In our project, we implemented the above method but with the following restrictions and modifications:

- No pictographic sign classification implemented as implementing the neural network too was making the project beyond its intended scope.

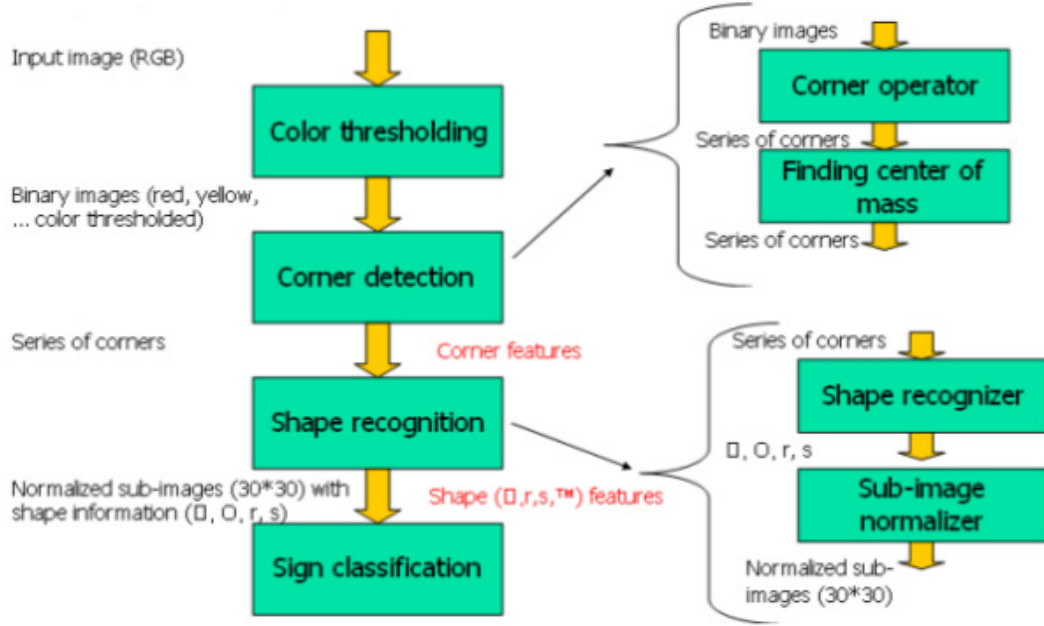


Figure 2: Flow of processing in the method.

- To limit the variety of signs, only yield sign (with similar detection as upward triangles of [1] and stop sign (as an extension) were targeted in the beginning. Later, support for circular red-bordered signs added too.
- Similar to the original method, only limited degree of sign rotation is acceptable (increased to  $\pm 10^\circ$ ) in shape recognition.
- Similar to the original method, image signs taken from narrow angles are not supported. Only minor skew is allowed. Basically minor sign rotation and minor change of view-angle (from the sides) are supported (along sign translation and scaling).
- Similar to the original method, occlusion is not supported. All corners of a sign must be visible.
- The original method didn't need to worry about verification of detected shapes as the neural network in the classification stage would ignore unrecognized images (treat them as outliers). But here we need to have verification mechanism to be able to distinguish a real sign from just a few points which happen to satisfy our border points condition. More on this provided in Section 3.3.
- Though [1] targets a real-time TSR system, we limit ourselves to still image to simplify our implementation.

Each one of the steps is explained in details in the following section.

### 3 Description of the method

As Figure 2 shows, the main stages of our implementation are color segmentation, corner detection and shape recognition. We describe each step in details here.

#### 3.1 Color segmentation

Color information in contrast to shape information is invariant to scale, rotation and view, and possesses high discriminative power. Traffic signs have distinctive colors, so using color segmentation methods are very popular in TSR systems.

As the signs of interest in our implementation all have red color in their borders, we can decrease the complexity of our input image (which is in RGB format) to a binary image by applying a color thresholding algorithm for shades of red color. Then, detection of corners can be done in this binary space instead of the original image leading to less computation time.

Color thresholding in RGB color space can be achieved like Equ. 1 where  $g(x, y)$  is the output of thresholding,  $k_1$  and  $k_2$  are the output binary values and  $f_r(x, y)$ ,  $f_g(x, y)$  and  $f_b(x, y)$  are the red, green and blue components of input image respectively (see [1]).

$$g(x, y) = \begin{cases} k_1 & \text{if } \begin{cases} R_{min} \leq f_r(x, y) \leq R_{max} \\ G_{min} \leq f_g(x, y) \leq G_{max} \\ B_{min} \leq f_b(x, y) \leq B_{max} \end{cases} \\ k_2 & \text{otherwise} \end{cases} \quad (1)$$

$R_{min}$  and  $R_{max}$  define the range of red component for pixels to be considered *red* in traffic signs. Note that traf-

fic signs don't have pure red. So green and blue components are non-zero though smaller than the red component. As the result, similar range for green and blue components need to be defined too.

But there is a big problem with working in RGB color space as it is very sensitive to lighting changes (e.g due to sun angle, weather, clouds, ...). So with only minor change of lighting, a previously defined color range in Equ. 1 will not be valid anymore. A common method to attack this is to first convert the image to HSI color space (stands for hue, saturation and intensity; also called HSV color space). HSI space decouples the intensity component from color-carrying information (hue is color purity, saturation is color dilution by white light). For more information about HSI color space see section 6.2.3 of [8] or RGB to HSI tutorial in [9]. But converting to HSI requires evaluation of some non-linear and trigonometric equations which are expensive to evaluate for every single pixel. Other variation is proposed by [10] to use ratio of pixel's red component to its overall intensity to define the range of redness. Another simplification will use a range for red component and other ranges for ratio of blue and green components to red component for defining the threshold criteria as shown below.

$$g(x, y) = \begin{cases} k_1 & \text{if } \begin{cases} R'_{min} \leq f_r(x, y) \leq R'_{max} \\ G'_{min} \leq \frac{f_g(x, y)}{f_r(x, y)} \leq G'_{max} \\ B'_{min} \leq \frac{f_b(x, y)}{f_r(x, y)} \leq B'_{max} \end{cases} \\ k_2 & \text{otherwise} \end{cases} \quad (2)$$

See the figures in Appendix A for some samples of color thresholded images.

### 3.2 Corner detection

Now we have a binary image resulted from thresholding the input image for a proper shade of red color using Equ. 2 and defining proper ranges we determined by investigating some samples images. We need to find corners in this binary image (i.e. corresponding to red corners in the original image) to be used for shape recognition (a later stage).

Using mask convolution for detection of different patterns is a well-studied area beside its common applications in filtering and picture enhancements. Canny's edge detector convolves the image with masks representing first derivative of Gaussian as the first step in its detection process. Similar masks can be derived for other shapes. For example [11] generates angle-dependent convolution masks for detecting circular and radial edges in detection of circular traffic signs. Their two detectors called Detector of Circular Edges (DCE) and Detector of Radial Edges (DRE) are created by a set of convolution masks derived by aligning a base function (i.e. an edge detection mask like Sobel) with tangents to the circle (for DCE) or radi-

als (for DRE). Of course, their method has the deficiency of relying on center of circular sign be known as a priori. Hough transform is a generalized method that can be used for detecting many shapes. But its main deficiencies are high processing time and large memory requirements.

Our shape recognition uses corners as primary features to find shape information from geometrical relations of corners. Common corner detectors use edge information or eigenvectors of gradient in a pixel neighborhood. They involve many steps. [1] employed a corner detector called "Optimal corner detector" which was developed by [2]. This corner detector has two advantages. First, since it works by convolving the image against some masks, it works faster than previously mentioned detectors (needs some simplifications as did by [1]). This makes it very suitable for our case of real-time detection where speed is a very important factor. Second, this method can classify *type of detected corners* (by angle and direction) which helps to reduce complexity of shape recognition phase (more on this in section 3.3).

#### 3.2.1 Optimal corner detector

Optimal corner detector models the local gray level around a corner point and attempts to find an optimal function (represented by the mask) which when convolved with the image yields a maximum at the corner point. Noise is modeled by additive Gaussian noise. The corner has fixed angle and orientation. The problem is then formulated as an optimization problem and variational calculus is used for solving it.

This approach is very similar to Canny's approach in his famous edge detector ([12]) and that is why they named it as *optimal* corner detector. Similar to Canny's qualitative objectives (good detection, good localization and single response to an edge), [2] sets the following qualitative goals:

- The method should not be sensitive to noise.
- It should not delocalize the corner.
- Detected corner should be an edge point too.
- The corner point should have at least two neighbors with different gradient than the corner itself.

Consider a corner at the origin oriented as Figure 3 with  $m$  as slope of the upper bounding edge and  $-m$  as the slope of lower bounding edge. The function describing the gray levels around the corner is

$$I(x, y) = \begin{cases} A & \text{if } x > 0 \text{ and } -mx < y < mx \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

The additive Gaussian noise  $n(x, y)$  changes the input image to

$$F(x, y) = I(x, y) + n(x, y). \quad (4)$$

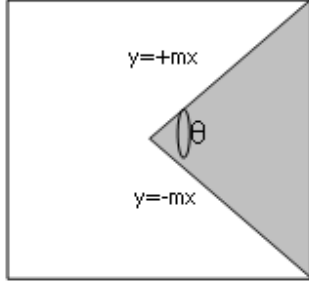


Figure 3: Corner model.

If the corner operator is named  $g(x, y)$ , then output image will be the convolution of  $F(x, y)$  with the mask as

$$O(x, y) = F(x, y) * g(x, y). \quad (5)$$

The first two qualitative objectives can be converted into quantitative functions by defining  $SNR(\Xi)$  for first criteria and  $E[x_0^2 + y_0^2]$  for  $Delocalization(\Lambda)$  where  $(x_0, y_0)$  is the detected location of original corner in  $(0, 0)$ .

$$\Xi = \frac{A \int_0^\infty \int_{-mx}^{mx} g(x, y) dy dx}{n_0 \sqrt{\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g^2(x, y) dy dx}}. \quad (6)$$

$$\Lambda = \frac{n_o^2 \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g_x^2(x, y) dy dx}{(A \int_0^\infty \int_{-mx}^{mx} g_{xx}(x, y) dy dx)^2} + \frac{n_o^2 \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g_y^2(x, y) dy dx}{(A \int_0^\infty \int_{-mx}^{mx} g_{yy}(x, y) dy dx)^2}. \quad (7)$$

$n_o^2$  is the variance of Gaussian noise  $n(x, y)$ , and  $g_x, g_y, g_{xx}$  and  $g_{yy}$  are the partial derivatives of the desired mask  $g(x, y)$ . Ideally a zero delocalization and high  $SNR$  is expected, so

$$\Sigma = SNR/Delocalization \quad (8)$$

is tried to be maximized as the performance measures of corner detector. This is very similar to Canny's definition of  $SNR$  and  $Localization$  (instead of  $Delocalization$ ) where he maximizes  $SNR \times Localization$ . Another difference is that input and derivation are in 2D space now compared to 1D for Canny. Similar approach using variational calculus used to solve this optimization problem.

[2] ends up with the following solution for Equ. 8:

$$g(x, y) = \begin{cases} c_1 \sin \frac{m\pi x}{W} [-(e^{zW} + e^{-zW}) + e^{zy} + e^{-zy}] & \text{for cone portion} \\ c_2 \sin \frac{n_1\pi x}{W} \sin \frac{n_2\pi y}{W} & \text{for non-cone portion} \end{cases} \quad (9)$$

where  $W$  is half of the mask size,  $c_1, c_2, m, n_1, n_2$  and  $z$  are all constants. Then they argue how proper values should be chosen for these constants, for example they pick  $m = -1$  to end up with positive values for cone-portion of the mask as the second term (exponentials) is always negative since  $x$  and  $y$  range between 0 and  $W$ . They couldn't find any rational for selection of value  $z$  though noted that its higher value equals higher noise fighting capability of the mask.

-3	-5	-5	-3	0	-3	-5	-5	-3
-5	-9	-9	-5	0	-5	-9	-9	-5
-5	-9	-9	-5	0	-5	-9	8	5
-3	-5	-5	-3	0	6	9	9	6
0	0	0	0	0	6	10	10	6
-3	-5	-5	-3	0	6	9	9	6
-5	-9	-9	-5	0	-5	-9	8	5
-5	-9	-9	-5	0	-5	-9	-9	-5
-3	-5	-5	-3	0	-3	-5	-5	-3

Figure 4: Optimal 60°mask.

The optimal mask of size 9 for  $\theta = 60^\circ$ ,  $m = -1$ ,  $n_1 = 1$ ,  $n_2 = -1$  and  $z = 0.2$  is shown in Figure 4. The effect of sinus wave is evident in the mask.

Any mask generated as above is dependent on the corner angle and orientation. So a restriction of this method is its dependency on the slope of the lines making the corner. This means the requirement of a large number of masks to cover all possible slopes of lines for all corners present within an image. To overcome this deficiency, they came up with a small class of corners which can approximate most of the possible corners in real images. They suggested to divide corners into groups of corners based on the quadrants they occupy and assign one mask to each group. For example one mask for all corners in quadrant I, another for quadrant II, another for corners that occupy both quadrant I and II and so forth. They end up with 12 masks. Though these masks will not have the same high response as a mask tailored for a particular angle and orientation of a corner, they can still give a good result.

Another advantage of these 12 masks is that the response of convolution to each can be built by smaller masks. For example consider the 9 by 9 mask for corners

-6	-11	-11	-6	0	4	7	7	4
-11	-18	-18	-11	0	8	13	13	8
-11	-18	-18	-11	0	10	17	17	10
-3	-5	-5	-3	0	12	19	19	12
0	0	0	0	0	12	20	20	12
-6	-11	-11	-6	0	-6	-11	-11	-6
-11	-18	-18	-11	0	-11	-18	-18	-11
-11	-18	-18	-11	0	-11	-18	-18	-11
-6	-11	-11	-6	0	-6	-11	-11	-6

Figure 5: 90°corner detector. Responds well to all corners in quarter 1.

of first quarter shown in Figure 5. This mask can be built with 4 smaller sub-masks shown in Figure 6. And then using separability, convolution response to the full mask can be written as sum of convolution response to the smaller sub-masks as

$$I(x, y) * g(x, y) = I_{C1}(x + d, y + d - 1) + I_{CX}(x + d, y) + I_{NC}(x - d + 1, y + d - 1) + I_{NC}(x + d, y - d) + I_{NC}(x - d + 1, y - d). \quad (10)$$

where  $d = (W + 1)/2$ . This will decrease the computational expense significantly, for example for the 12 generic masks (assume mask size of 9 by 9) it was required to do  $9 * 9 * 12$  multiplication for each pixel in the image. Now instead of convolving each  $9 * 9$  mask, smaller sub-masks are convolved with each pixel once and all the 12 generic masks can be built with proper combination of response to these sub-masks. They also go further and show that each sub-mask can be decomposed to 1-D masks.

As we described before, [2] defined four qualitative targets for the optimal corner operator. The first two were quantified by creating corner masks using equations 9. After convolving the image with proper masks and applying a preset global threshold, a set of candidate corners are found. For the third criteria, they used Canny edge detector to find pixel edges for comparing the candidate corners against them. For the forth criteria they used gradient of neighboring pixels to discard pure edge points. But [1] relaxed these conditions and didn't use the third and forth criteria as they're very costly and wouldn't be advantageous to other corner detectors cost-wise.

4	7	7	4	0	12	19	19	12	0
8	13	13	8	0	10	17	17	10	0
10	17	17	10	0	8	13	13	8	0
12	19	19	12	0	4	7	7	4	0
0	0	0	0	0	0	0	0	0	0

-6	-11	-11	-6	0	12	20	20	12	0
-11	-18	-18	-11	0					
-11	-18	-18	-11	0					
-3	-5	-5	-3	0					
0	0	0	0	0					

12	20	20	12	0
----	----	----	----	---

(a)
(b)

(c)
(d)

Figure 6: Basic sub-masks: (a) C1; (b) C2; (c) NC; (d) CX.

### 3.2.2 Corners of interest for our signs

Using optimal corner detector, we can find a particular type of corner (known angle and orientation) by applying a corner mask. For each traffic sign we need to find different type of corners in the image. Here we go through each sign and type of corners we need. In total we need to detect five types of corners (later we have to expand it to eight). As described in the previous section, optimal corner detector has the interesting property of *corner classification*. So after convolving the color thresholded image (binary image) with a mask, we end up with the list of all detected corners of that type.

Let's go through each sign and corner types needed for detecting that sign:

**Yield sign** The yield sign has three corners each with 60°angle. For detecting the bottom corner, we apply Y1 mask shown in Fig. 8. The masks needed for upper corners can be approximated with 90°masks instead of 60°masks. As Fig. 7 shows there is only little difference between the two masks (see [1]). Since we need to use 90°corner masks for other shapes anyway, this will decrease the number of required masks. These 90°corner masks are called C2 and C3 because of naming convention used in the next steps.

**Stop sign** For stop signs, we first detect  $p1$  and  $p8$  points using mask Y1 (see Fig. 9). This mask identify these corners quite easily though it was intended for 60°corners originally. Then line segments  $L1, L2, L3$



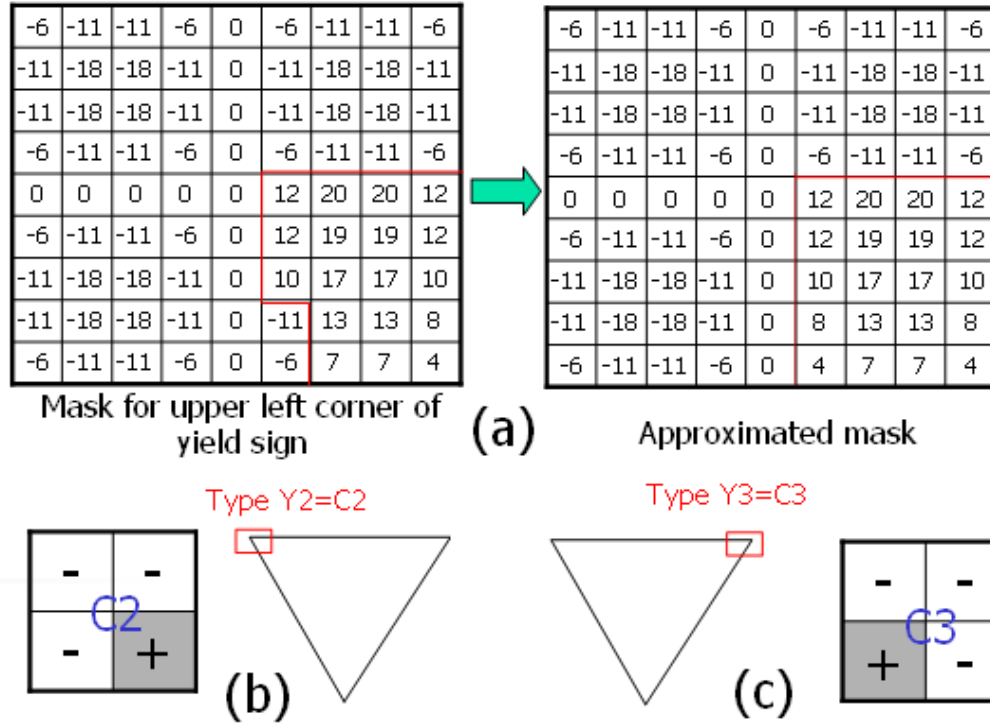


Figure 7: Yield sign upper corners: (a) approximating 60°corner with 90°; (b) mask for upper-left corner; (c) mask for upper-right corner.

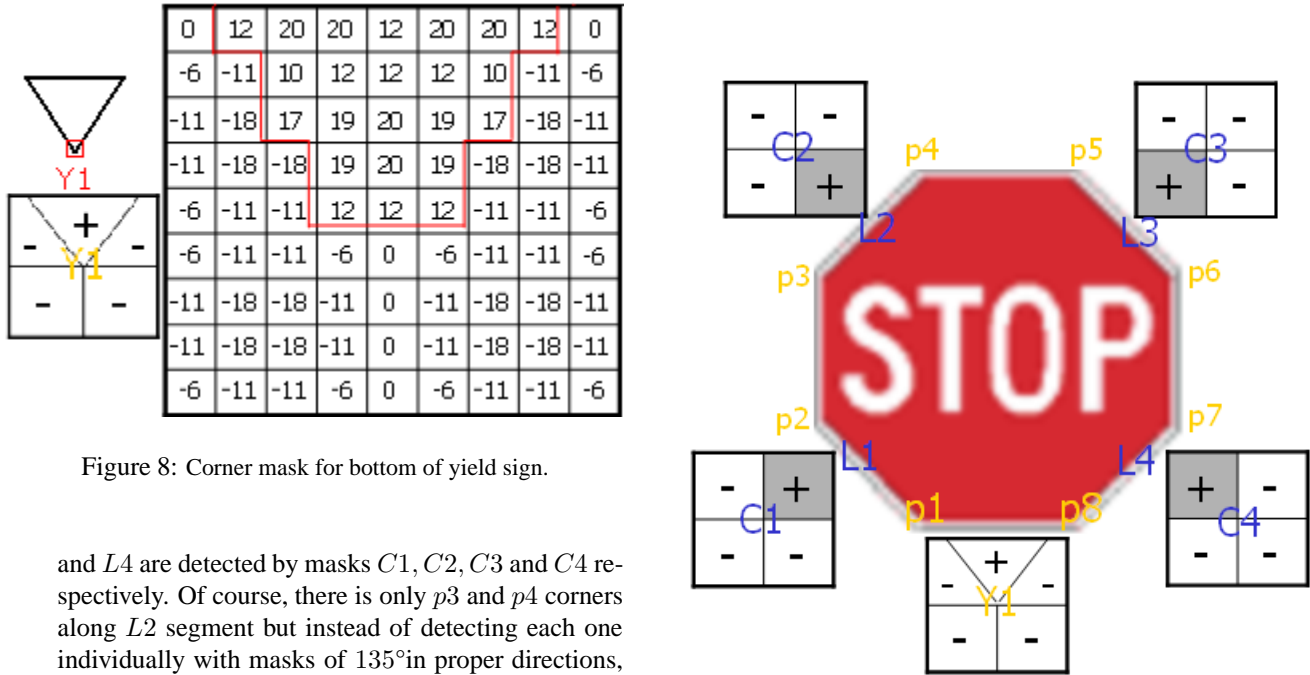


Figure 9: Corner masks for stop sign.

and  $L4$  are detected by masks  $C1$ ,  $C2$ ,  $C3$  and  $C4$  respectively. Of course, there is only  $p3$  and  $p4$  corners along  $L2$  segment but instead of detecting each one individually with masks of 135° in proper directions, we learned that 90° masks identify the line segments quite well, so we can use response to these 90° masks (which we need anyway for circular signs) rather than introducing new masks.

**Circular sign** For circular signs, we need to detect four

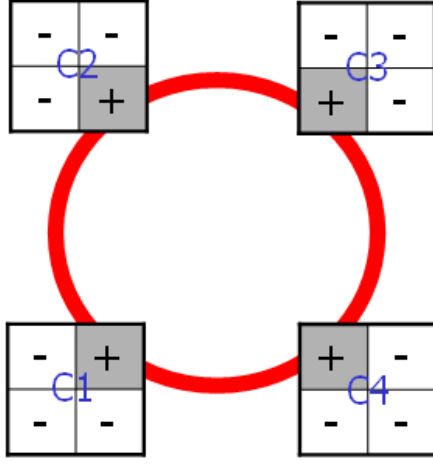


Figure 10: Masks for circular sign detection.

points on corners of the circle. Each 90° mask pretty well detects portion of the circle in upper-left, upper-right, lower-left and lower-right arcs. Figure 10 shows this correspondence.

Now we have five list of corners corresponding to  $Y1, C1, C2, C3$  and  $C4$  corner masks. We call each one as a *corner set*.

### 3.2.3 Center of mass calculation

Because of all sorts of error (input image not having a perfect corner, optimal corner operator not being perfect, errors in finding the perfect threshold values, approximating a corner mask with another mask close to corner angle, noise, ...) we never get a *single* response for a corner and multiple close corners will be detected instead. To compensate for this, we pass detected corners through a center of mass calculator to end up with less corners but more closely concentrated towards the real corners. [1] doesn't mention how they are implementing their algorithm. Here is what we are using:

- For each corner point, all points within the neighborhood around the corner (size of 7\*7) are considered and using the convolution response at each point as weight, average point within the neighborhood is calculated. This point represents the center of mass in the neighborhood of that corner.
- Since neighborhoods of close corners overlap, the center points in two overlapping neighborhoods are compared and the one with higher average of convolution response is picked.

Corner Set	# of raw corners	# of corners (after center of mass)
Y1	806	218
C1	303	96
C2	279	88
C3	298	97
C4	243	82

Table 1: Number of corners of a different class for sample image Figure 22

Table 1 shows the number of detected corners and corresponding corners after applying center of mass calculation for sample image shown in Fig. 22 of Appendix A.

### 3.3 Shape recognition

Our shape recognition method is based on finding the proper geometry between detected corners of proper class for each shape. This method is like an Interpretation Tree where geometrical constraints limit the number of searches and eliminate many sub-trees. Also, because our corners are *classified* (like  $Y1, C1, C2, \dots$ ), the number of corners in each class is reduced and since we pick corners by a particular order from each class (depending on the shape type), the number of detected features combinations to be tried is heavily reduced. Here corners are our low-level features and their (distance, angle) function as binary constraints for feature match.

Now we go through the recognition process for each shape.

#### 3.3.1 Yield sign recognition

Here are the steps (see Figure 11) which are similar to [1] but with added verification methods:

- Pick a point like  $p1$  from  $Y1$  corner set (60° set).
- Find points like  $p2$  from  $C2$  corner set and  $p3$  from  $C3$  corner set such that:
  - $(-60 - \theta_t) \leq \angle p3p1 \leq (-60 + \theta_t)$  and  $(-120 - \theta_t) \leq \angle p2p1 \leq (-120 + \theta_t)$  where  $\theta_t$  is configurable parameter. Note that angles are shown as negative because vertical coordinate ( $Y$ ) is really downward in image coordinate.
  - $l_{min} \leq \|p3p1\| \leq l_{max}$  and  $l_{min} \leq \|p2p1\| \leq l_{max}$  where  $l_{min}$  and  $l_{max}$  are configurable parameters.
  - Note that only corners within the limited area satisfying above conditions are picked.
- Verify that  $p2p1$  and  $p3p1$  have roughly the same size.  $p2p3$  is not compared with the other two as some yield signs in our test images were isosceles rather than



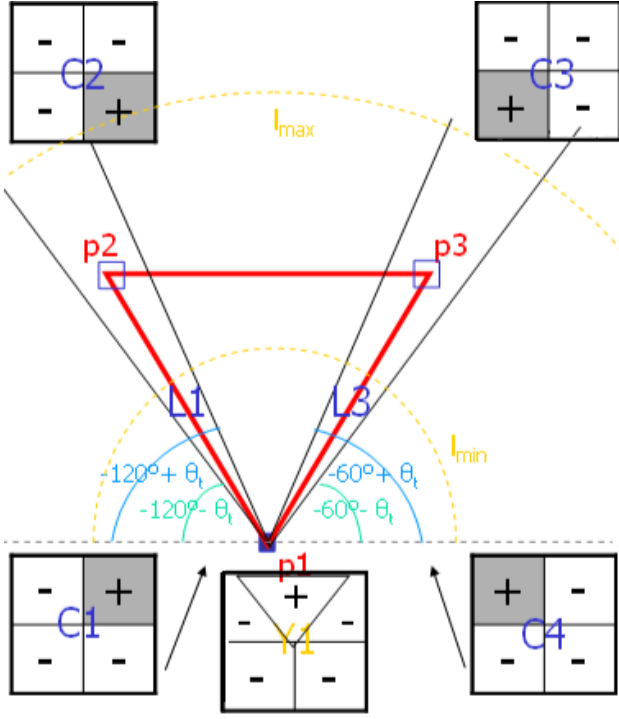


Figure 11: Recognizing yield sign.

equilateral. Also, picture of a yield sign taken from an angle not parallel to its face looks more like an isosceles.

- Now we have the three possible points that can make a triangle. But an extra verification step is required as these might be just isolated points not on border of a valid red triangle. This can be done by sampling few points in L1 and L3 segments and checking if there is any corner around. This is the reason that we use C1 and C4 corner sets for yield sign too as they detect some points along L1 and L3 segments as corners.
- Later we had to add another verification method as some triangles were detected on stop signs or in a case like triangle detected on back of the gray car in Figure 23. To do this, we calculated the ratio of red pixels in an area at the center of the triangle. This ratio must be very small as there is no red pixels in center of an ideal yield sign. Small ratio was allowed to tolerate error in calculating center point.

### 3.3.2 Stop sign recognition

The steps involved here is more complicated than the yield sign (see Figure 12) and is based on our own extension:

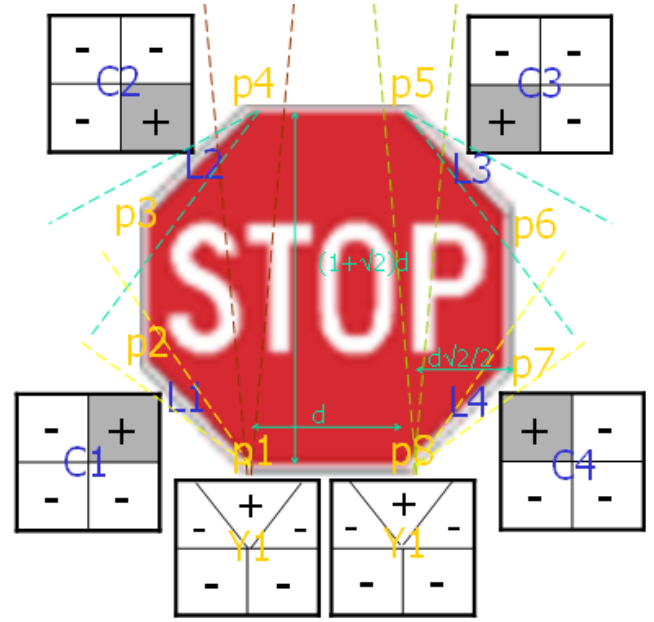


Figure 12: Recognizing stop sign.

- Pick pair of corners like  $p1$  and  $p8$  from Y1 corner set and make sure they are roughly on a horizontal line (another configurable parameter). Calculate the distance between the two corners (assume it to be  $d$ ).
- Find a corner like  $p4$  from C2 corner set above  $p1$  (requires a range of angle) and also in a range around  $(1 + \sqrt{2})d$  from  $p1$ .  $(1 + \sqrt{2})d$  is derived because of octagon shape and because each corner angle is  $135^\circ$ . A minimum and maximum for the angle and distance ranges are considered (another set of configurable parameters). Note that for pictures taken from sides of the sign (not parallel to its face), all horizontal segments are compressed relative to the vertical segments, so we had to widen the minimum and maximum range for  $p1p4$  search area to be able to detect these side pictures. Note that C2 mask detects many corners along L2 segment as it is a  $90^\circ$  detector and is not meant for such a corner. As the result, there will be multiple candidate for  $p4$ .
- Find a corner like  $p5$  from C3 corner set similar to  $p4$  case described above but comparing it with  $p8$ .
- Find a corner like  $p3$  from C2 that satisfies constraints from  $p4$ .
- Find a corner like  $p6$  from C3 that satisfies constraints from  $p5$ .

- Find a corner like  $p_2$  from  $C_1$  that satisfies constraints from  $p_1$ .
- Find a corner like  $p_7$  from  $C_4$  that satisfies constraints from  $p_8$ .
- Verify verticality of  $p_3p_2$  and  $p_6p_7$  segments within some angle range.
- We didn't used to have any further verification as probability of having some outlier points satisfying all above conditions was not likely. But after starting to experiment with circular signs, some circles were detected as stop signs. Also some isolated outlier points were detected as stop sign too. An easy method to distinguish the stop sign from a circular sign is to calculate the area of red pixels within candidate sign's bounding box. For stop sign this is much higher than a circular sign (around %70 ideally). Of course, a range of acceptable ratio had to be defined as the ratio varies for not parallel pictures. This area calculation was done in domain of binary image where it was easy to distinguish red and non-red pixels.

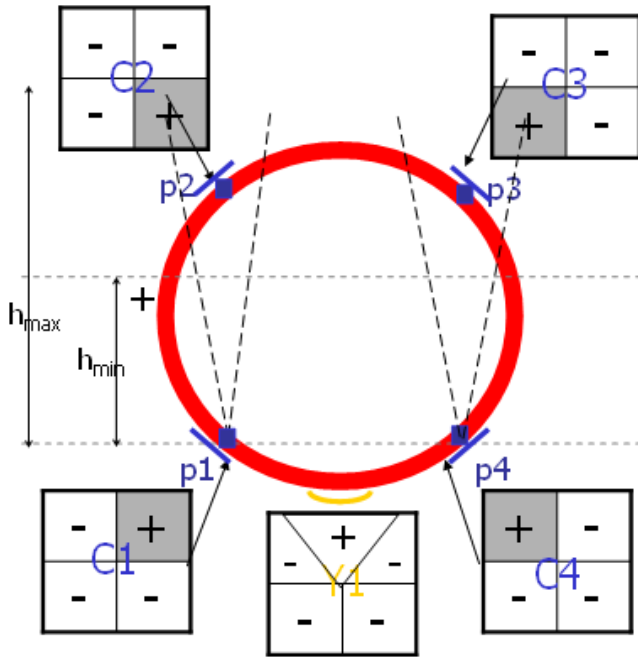


Figure 13: Circular sign recognition.

### 3.3.3 Circular sign recognition

Here are the steps (see Figure 13):

- Pick a corner like  $p_1$  from  $C_1$  corner set and a corner like  $p_4$  from  $C_4$  corner set that are roughly on a horizontal line (configurable parameter). Calculate the distance between the two points.
- Using  $p_1p_4$  calculated in previous step, setup a search range in vertical direction (configurable parameter) to find  $p_2$  from  $C_2$  corner set above  $p_1$ . In calculating the search range, we need to widen it a bit as horizontal distances are compressed in pictures taken from sides of the sign (similar to stop sign case discussed before).
- Do similar thing to find  $p_3$  from  $C_3$  above  $p_4$ .
- Now for verification, we use  $Y_1$  corner set to detect a few points in the bottom of circle. So the likely bottom area of the circle is calculated from the four known points and then  $Y_1$  corner set is searched for corners in that area. There should be a few points at least. Of course this verification method is simple and inefficient. We are working on an improved method to calculate the circle equation from the detected points (actually can be an ellipsis considering the effect of projection). Then sample proper corner sets for corners in that area and based on a vote, accept or reject the circle. This method is not completed yet and we see frequent outliers circle in our results.
- Note that very similar method can be used for rectangle detection as [1] use it for rectangle detection. Here we don't care about rectangular signs. But there shouldn't be problem of confusing circles and rectangles as rectangular signs doesn't have a red border and they need to be color thresholded for a different color.

### 3.3.4 Comparing with Geometric Hashing

Another alternative recognition method could have been Geometric Hashing. Traffic signs are 2D objects but their image is not a 2D transformation of the model because of the projection effect. If we approximate the perspective effect with a parallel projection for simplification (considering ratio of a sign dimensions is usually small to the long viewing distance), the projection effect can be done by an affine transform mapping the model plane to the image plane. With three points such a plane can be specified and there exists a unique affine transformation mapping any three non-collinear points in the model plane to another triplet in the image plane. For more discussion of this see [13] and [14] written by Lamdan and Wolfson who developed the original Geometrical Hashing method.

The big advantages of geometrical hashing to our method will be its support of occlusion, any degree of rotation and projection effect in the input image. In our method

object model is hard-coded into the recognition code based on some geometrical constraints while in Geometric Hashing the model is kept in the database. This limits the flexibility of our method and makes it more difficult to extend it to other shapes.

## 4 Implementation

Our code is using OpenCV but only for loading the input image files, display and save of the intermediate and output images. After loading the input image (image files are 24 bpp bitmap files), color components are separated to different images of 8 bpp. Then color thresholding is applied to them and a single 8pp image is generated with pixel values of 0 for non-red pixels and 255 for red pixels.

A mask of 60°corner (covering 60°to 120°in image coordinate space; note that  $Y$  direction is downward) and another mask for a 90°(covering 0°to -90°) are hard-coded as arrays. All of our required masks ( $Y1$ ,  $C1$ ,  $C2$ ,  $C3$  and  $C4$ ) are built from these two masks by simple mirroring relative to axes of symmetry (depending on each mask). For each corner mask, the color thresholded image is convolved with the mask resulting in raw corner set which itself will be converted to a smaller corner set after applying center of mass calculation. Now we have a corner set for each corner mask. Note that use of sub-masks as suggested by [2] (building a full mask with smaller masks to save time in convolution calculation) is not implemented yet.

For each sign type, a detection function is called starting with yield sign, stop sign and finally circular sign. Each function receives its required corner sets along the thresholded image (used for verification code for stop sign by calculating the area of red pixels and similarly for verification of yield sign) and an output image. The output image is initially the clone of input image (24 bpp). Each detection function updates it by marking the detected signs on it.

The reason for ordering the detection functions as yield, stop and then circular is related to robustness of each detection method. Each detection function removes the corners belonging to a detected sign or falling inside the sign (which are outliers) from the related corner sets to reduce the detection time (by shrinking the corner sets). So calling the detection method in order of their robustness decreases the chance of detecting outliers by the next detection. Bounding boxes of all detected signs are kept all the time. After detecting a new sign, its bounding box is compared against all previously detected signs and if there is an overlap, the newly detected sign is considered as an outlier. This is functioning as a verification method currently (used after a new sign is detected). Of course doing this test right after selecting each pair of corners (as a line segment candidate) will remove them early from the loop of corners and will help the detection time.

The input image filename need to be hard-coded into the program. All intermediary images (including thresholded image, each corner set, output image, ...) are shown and written to files when program executes.

## 5 Results and observations

First, we started our early tests with some synthetic sign images found over the web and developed our algorithms over them. Figures 14 and 15 in Appendix A show the output image, along its thresholded image. Figures 16 and 17 show its  $Y1$  and  $C1$  corner sets. As each sign has its ideal shape, there is no problem detecting them.  $Y1$  corner set was originally meant for detecting bottom corner of the triangle but as picture shows, it can detect bottom corners of the stop sign and bottom portion of circular signs.  $C1$  corner set can detect the left segment of the yield sign (used for verification of the sign), lower-left segment of the stop sign and a lower-left arc of circular signs.

In the next step, we experienced with some real images found over the web which didn't have the perfect shape of previous test. Figure 18 shows a sample of them. With this test case, our bugs and deficiencies started to show up. After changing our configuration parameters most of them went away.  $S1$  and  $S2$  have little degree of rotation and detected pretty easily.  $S7$  and  $S3$  needed quit lot of effort to change search area parameters.  $S6$  could not be detected originally because of its small size. We figured out that our center of mass calculation is causing the problem. Center of mass calculation is done over a neighborhood around each corner and it can cause few pixel shift of the major corner depending on other corners. After reducing the neighborhood size to  $7 * 7$  and some corrections in detection code, it was finally detected.  $S5$  can not be detected because of very narrow angle of the picture.

Surprising  $S4$  was not detected though its shape is relatively ideal (excluding the fact that the "KEEP RIGHT" sign causes multiple responses for bottom corner). It turned out that three proper corner points are found successfully but this is our verification algorithm that fails because  $C1$  mask is not capable of detecting  $L1$  segment properly (remember that some points along  $L1$  are sampled to make sure there is a real triangle there not just isolated corners). After further investigation we figured out that relatively large slope of  $L1$  causes that many red pixels (white in the thresholded binary image) falls into the negative portion of the mask reducing the final value of convolution for pixels along  $L1$  and basically failing to detect any corner there (see Figure 19). Note that  $C1$  was never intended for triangle detection and we just learned about its usefulness by experiment. A mask like  $C5$  is expected to respond better though not implemented yet.

Figure 20 shows another example.  $T1$  is not detected because of the projection effect but a circle is detected in-

side letter 'P'. Figure 21 shows an early result before implementing triangle verification where a red leaf in T4 was causing so many outliers.

And finally, we took some pictures around Queen's campus. Figure 22 is the most complex one having many red objects. As our circle verification is not complete yet, those outliers are detected as circles. Figure 23 shows an earlier result with many outliers.

Figures 24 to 32 show more examples. With the new samples we figured out that our stop sign detection doesn't work sometimes even with small rotation of signs. It turned out that  $C1-C4$  masks might not be able to detect line segments  $L1$  to  $L4$  (refer to Figure 12) properly so some of the points  $p2$  to  $p7$  won't be detected causing failure. We were already aware of good response of  $Y1$  mask for finding  $p1$  and  $p8$  corners. So by using similar  $60^\circ$  corner but from left ( $Y2$ ), top ( $Y3$ ) and right ( $Y4$ ) directions the performance improved. Of course this has the drawback of three added new masks. Also some problem with thresholding of some shades of red was noticed (see Figure 24 as red-border of "No-right-turn" sign is not detected). This was solved after changing the threshold range.

Figures 26 and 28 show the effect of thresholding parameters. Figure 30 shows some red shades because of light reflection which are not easily visible in the original image. In Figure 31, though the sign is partially occluded, it is still detected because the four corner points were detected and our circle verification is *not* complete yet. Our new circle verification will sample the perimeter of the circle for red pixels. By relaxing the number of required samples within each circle pie, it should be still detectable (refer back to Section 3.3.3). And finally Figure 33 shows some outliers detected as stop and circular signs.

Here are the summary of the issues we experienced:

**Center of mass calculation:** Center of mass calculation can cause shift of real corners. When the effect of such shift on multiple corners added up, it could break the detection especially for small signs. Such a case happened for  $S6$  sign in Fig. 18.

**Need for new corner masks:** Verification of yield sign by sampling  $L1$  and  $L2$  line segments using  $C1$  and  $C2$  corner set might not work depending on segment slope. Need to use a new corner mask for better result (Fig. 19). Also we saw similar issues with stop sign detection and had to use new corner masks  $Y2$ ,  $Y3$  and  $Y4$  to improve corner detection.

**Importance of color thresholding:** Color thresholding parameters severely affect the number of detected corners, and change probability of outlier signs and also recognition time. We still see the need for working more on this. Some images work better with

one settings while the rest work better with another settings. These two settings should be converged.

**More tolerance for vertical parameters:** Projection effect could cause smaller size for line segments in horizontal direction than vertical direction (because we consider side views only and in practice looking at a sign in upward or downward direction doesn't happen on views from a passing car). As we start by detecting the horizontal base segment for the stop sign and adjust our search area in vertical direction based on that segment size, we need to have a larger vertical search than what the mathematical relation of the ideal model dictates.

**Importance of verification:** To distinguish outliers, robust verification methods are essential. [1] didn't need to worry about outliers much as their neural network stage (for pictographic recognition of sign content) was able to catch them. But we didn't have such a block, so had to develop verification methods which still need improvements.

**Difficult to debug:** As complexity of the input image increases, the number of detected corners increase too. This makes the debugging difficult as need to traverse many corners (e.g. wait for the 105th corner) to debug a failing case.

**Configuring the parameters:** There are many parameters to adjust for each shape. After changing each parameter of a sign detection algorithm to fix a failure in a particular image, there is the need to test the whole test image set again to make sure that fix is not breaking the previously working cases. Also, as each detection removes some corners from the corner sets, changing a parameter can change the order of sign detection or produce new outliers.

**Code size:** The code grows very fast as each shape model is hard-coded into the code. Good programming practices like object-oriented programming should be used to define common shape properties, objects like line segments, ... to make debugging easier. Our code has reached 2000 lines currently.

**Code optimization:** There are many searches involved in each shape detection algorithm. Better search algorithms (like sorting corners of each corner set based on the area) will decrease recognition time. Many loops are involved too. Keeping some data structure to optimize the corner combinations will reduce loop iterations (for example testing each corner against the bounding box of previously detected signs to prevent redundant iterations).

## 6 Future work

There are many areas that this work can be improved. All issues discussed in 5 can be improved. Among them color thresholding and sign verification are the most important ones. The more efficient way of calculating mask convolution (by using smaller basis-masks discussed in Section 3.2.1 based on [2] can be easily implemented. After doing so there will be a better argument for using new corner masks for better recognition (like the case of Y2, Y3 and Y4 discussed briefly in Section 5). But we shouldn't forget the bigger picture as this implementation in its current form has a limited application as we completely ignored the pictographic recognition stage. The method chosen for that stage could affect requirements of previous stage like the robustness of sign verification (e.g. [1] didn't need to worry much about sign verification because of their neural-network based pictographic recognition stage). Also, implementing Geometrical Hashing for shape recognition stage using our classified corners could be an interesting extension of this work.

After all, more advancements in car navigation systems based on Global Positioning System (GPS) will allow addition of each single traffic sign's *location* in a form of library in future (for a limited area of course, based on system's storage capacity). So the driver can be informed about a sign through the *library of sign locations* rather than recognition. This system will not have all recognition issues of a real-time vision-based TSR system (like high processing overhead or noise issues like occlusion, night or foggy condition, ...). But navigation systems will keep their off-line nature for the next few years (requiring library update for new changes of road system). Even with a frequent or even online update of the navigation library, a reliable TSR system can provide better safety because it reports what *really* exist in the road not just library information. As future vehicles will be equipped with cameras for collision avoidance and road detection, having a TSR system even as a companion to a GPS-based navigation system is still advantageous. Anyway, we have to wait and see the future commercial application of TSR systems.

## 7 Conclusions

In this report, we presented an implementation of a traffic sign recognition system based on the work done in [1]. This method can target real-time sign recognition (not in its current implementation, of course) by trying to increase processing speed. A simple color thresholding method (similar to functioning in *HSI* space) employed to segment colors of interest (only red in our case). The *optimal corner detector* from [2] (used in corner detection stage) efficiently produces *classified* corners (based on angle and orientation) that can be further improved by calculating convolution masks using smaller basis masks. Since

detected corners are classified, many sub-trees of the interpretation tree are not traversed in shape recognition stage. Angle and distance between two corners function as the binary constraints for feature matching.

We limited the signs of our interest to yield sign and red-bordered circular signs based on [1] work and extended it to stop signs. We also had to add sign verification methods as we didn't have the pictographic recognition stage of [1]. The implementation works pretty well for a variety of test cases from synthetic signs to real pictures. However, we didn't do an empirical study of test images based on their difficulty level (distance, view angle, possibility of outliers from surrounding objects, ...) as more test images were necessary and higher priority was given to improving the implementation.

And as the final words we should mention that a great deal was learned in this project about all sort of issues that could arise in a real computer vision project and gave us a better perspective of the subject area.

## References

- [1] A. Escalera, L. Moreno, M. Salichs, J. Armingol, "Road Traffic sign detection and classification," *IEEE transactions on industrial electronics*, vol. 44, no. 6, Dec. 1997.
- [2] K. Rangarajan, M. Shah, and D. Van Brackle, "Optimal corner detector," *Computer Vision, Graphics and Image Processing*, vol. 48, no. 2, pp 230-245, Nov. 1989.
- [3] F. Heinze, L. Schafers, C. Scheidler, W. Obeloer, "Trapper: eliminating performance bottlenecks in a parallel embedded application," *IEEE Concurrency [see also IEEE Parallel & Distributed Technology]*, vol. 5, issue 3, July-Sept. 1997, pp. 28-37.
- [4] "PATH project website," <http://www.path.berkeley.edu/> or "Computer Vision Group," <http://http.cs.berkeley.edu/projects/vision/>.
- [5] N. Kehtarnavaz, A. Ahmad, "Traffic sign recognition in noisy outdoor scenes," in *Proceedings of the Intelligent Vehicles '95 Symposium*, 25-26 Sept. 1995, pp. 460-465.
- [6] D.S. Kang, N.C. Griswold, N. Kehtarnavaz, "An invariant traffic sign recognition system based on sequential color processing and geometrical transformation," in *Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation*, 21-24 April 1994, pp. 88-93.
- [7] L. Priese, R. Lakmann; V. Rehmann, "Ideogram identification in a realtime traffic sign recognition system," in *Proceedings of the Intelligent Vehicles '95 Symposium*, 25-26 Sept. 1995, pp. 310-314.
- [8] R. Gonzalez, R. Woods, *Digital Image Processing*, 2nd edition, Prentice Hall, 2002.
- [9] *Tutorial section of website for "Digital Image Processing" book*, <http://www.imageprocessingbook.com/DIP2E/tutorials/tutorials.htm>

- [10] H. Kamada, M. Yoshida, "A visual control system using image processing and fuzzy theory," in *Vision Based Vehicle Guidance*, I. Masaki, Ed. Berlin, Germany: Springer-Verlag, 1992, pp. 111-128.
- [11] H. Sandoval, T. Hattori, S. Kitagawa, Y. Chigusa, "Angle-dependent edge detection for traffic sign recognition," in *Proceedings of the IEEE Intelligent Vehicles Symposium 2000*, Dearborn (MI), USA, Oct. 3-5, pp. 308-313.
- [12] John Canny, "A Computational Approach to Edge Detection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume PAMI-8, Number 6, November 1986, pages 679-698.
- [13] Y. Lamdan, H.J. Wolfson, "Geometric Hashing: A General And Efficient Model-based Recognition Scheme," in *Proceedings of Second International Conference on Computer Vision*, December 5-8, 1988, pp. 238-249.
- [14] Y. Lamdan, J.T. Schwartz, H.J. Wolfson, "On recognition of 3-D objects from 2-D images," in *Proceedings of IEEE International Conference on Robotics and Automation*, April 24-29, 1988, pp. 1407-1413.

## Appendix A - Test Figures



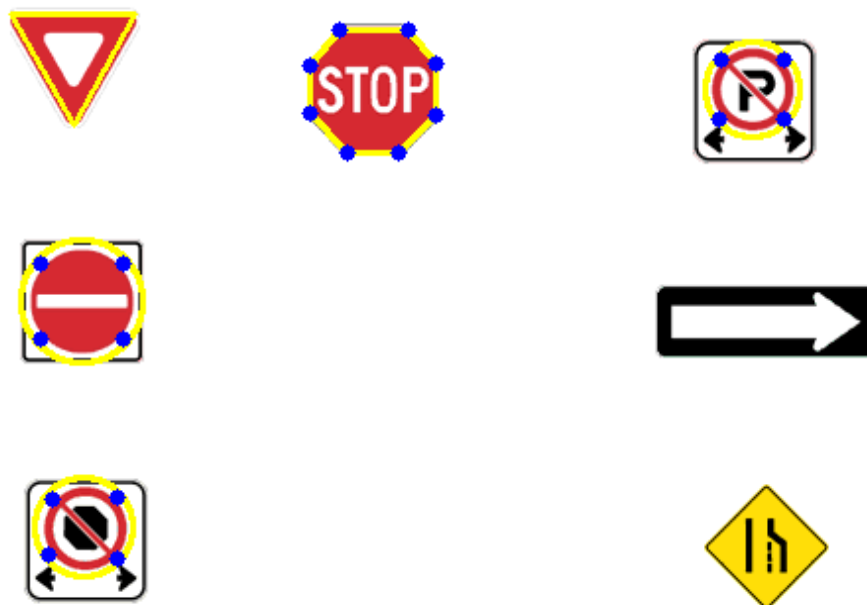


Figure 14: First sample image after detection.

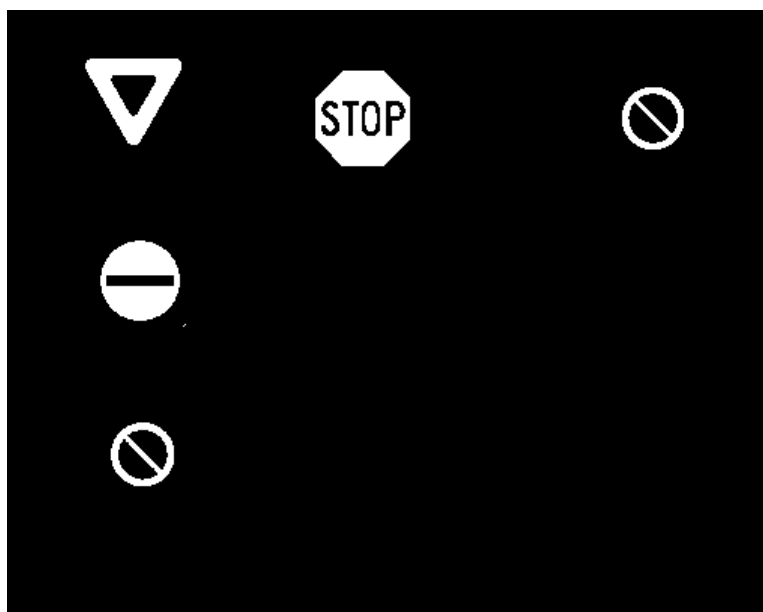


Figure 15: Red color segmentation of original image of Figure 14.

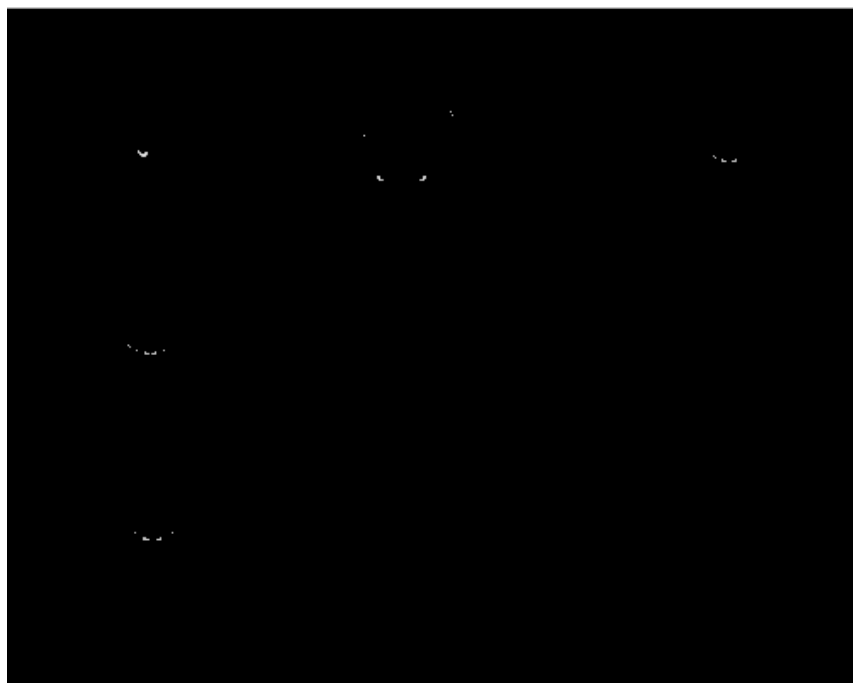


Figure 16: Y1 corner set of Figure 14's original image.

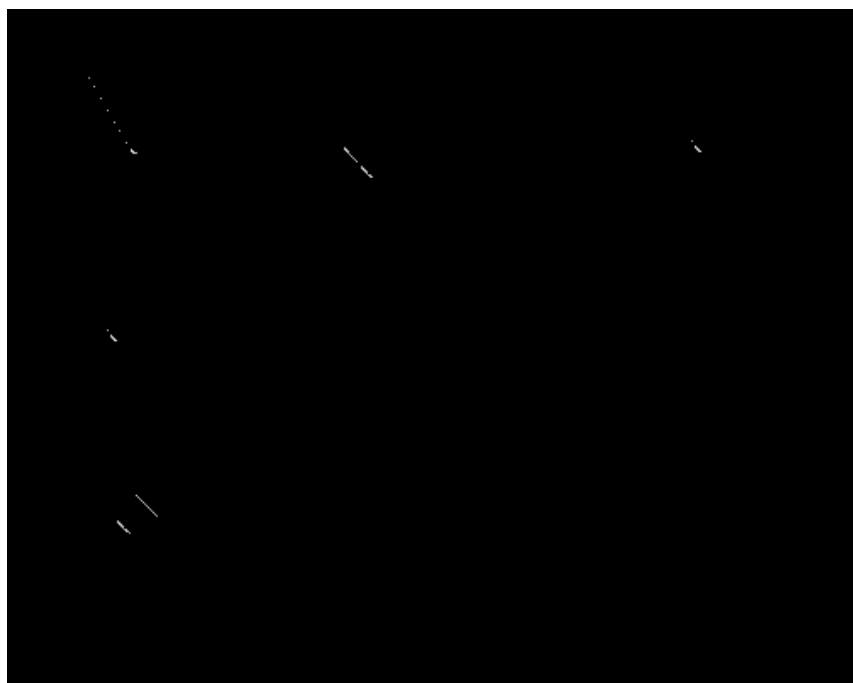


Figure 17: C1 corner set of Figure 14's original image.



Figure 18: Sample image.

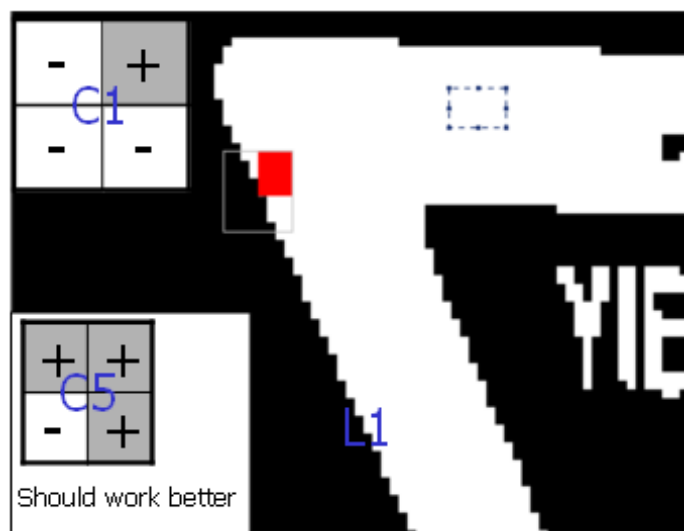


Figure 19: Reason why S4 yield sign in Fig. 18 is not detected.

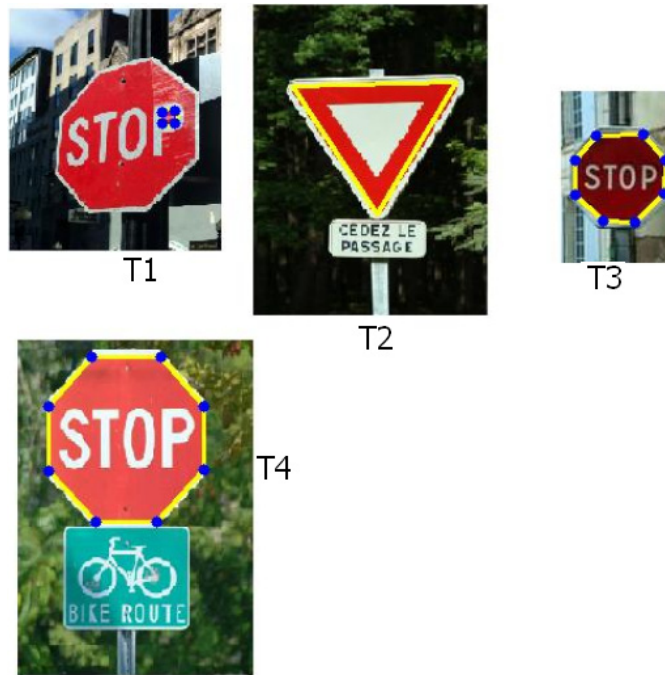


Figure 20: Another sample.

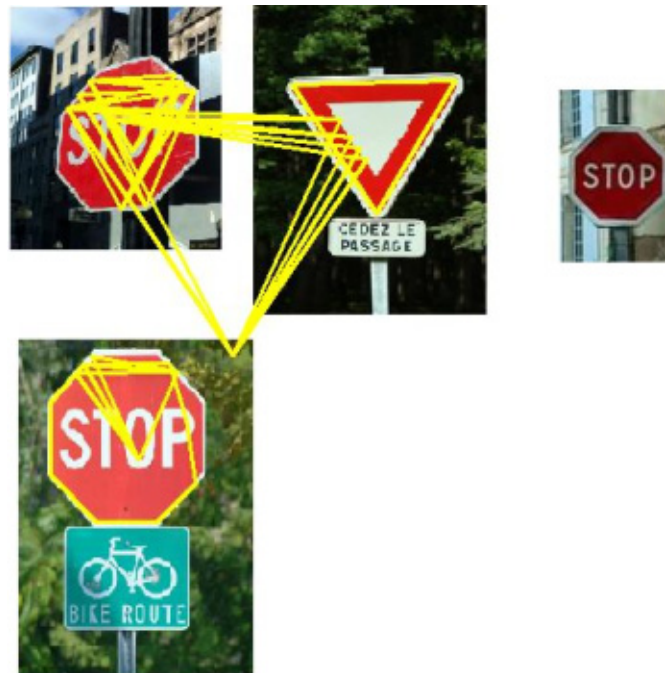


Figure 21: An earlier result for Fig. 20 with many outliers.



Figure 22: A complex sample with many red objects. Circular outlier is because circle sign verification is not completed yet.



Figure 23: An earlier result for Fig. 22 with many outliers.





Figure 24: A sample failing to detect the "no-right-turn" sign because of bad thresholding.



Figure 25: Relatively small sizes of the signs caused a verification failure as the ratio of red pixels was not in the original range set. After expansion of the range verification passed. See Section 3.3.2.





Figure 26: Another sample.



Figure 27: The color segmented image of Figure 26. Still better threshold settings should mask most of the yellow pixels.



Figure 28: An earlier color segmented image of Figure 26 before tightening the threshold range. This resulted in larger number of corners and higher possibility of outliers.

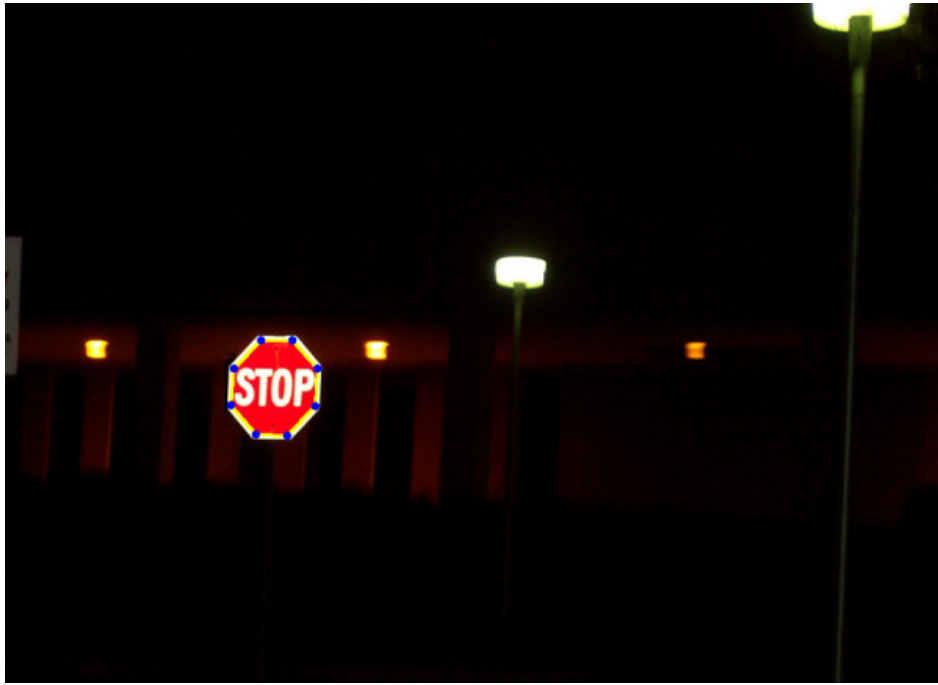


Figure 29: West campus's Jean Royce Hall.



Figure 30: Color segmented image of Figure 29. The reflection of red light on the hall entrance caused detection of many corners and larger detection time.



Figure 31: Another sample. Though the sign is partially occluded, the four basis corners were detected. Implementation of our new circle verification method would have dropped this sign (because it samples the perimeter of the circle) unless the test conditions were relaxed.



Figure 32: Signs on entrance of ECE department.



Figure 33: See the outliers on the ambulance lights. Need to extend verification step to add test of internal parts of circles to make sure there is a non-red portion (or check its area like the stop sign case) to ignore such cases.