Sign-Control

Road sign detection and recognition

Fall 2007 – Group 8

Pratik Barasia | ppb@andrew.cmu.edu
Mudit Aggarwal | maggarwa@andrew.cmu.edu
Kaushal Patel | kaushalp@andrew.cmu.edu
Hussain Tambawala | htambawa@andrew.cmu.edu

Table of Contents

ABSTRACT

Object Recognition techniques have made significant progress in the past few years. Some of this research has been focused towards the development of artificial intelligence systems. Other research is designed to provide efficient solutions for real-life problems we face. Within this context, we discuss in this paper the complete methodology for the detection and classification of road signs. The progress in this field has allowed us to foresee completely dependent systems which could aide the driver in recognizing most common street signs encountered on the roads, and to provide forewarning if the situation calls for it. In our system we implement a reliable detection/recognition model which revolves around accurate detection techniques using color thresh holding and a feature based classification algorithm designed for the specific testing database. Our implementation has been built on the C67 Texas Instruments DSK. We see that this system provides sufficiently high accuracy without using more complex structures such as neural networks.

INTRODUCTION:

The purpose of road signs is to provide the driver with important information about the road ahead. The problem arises when the driver is negligent or blatantly disobedient of traffic laws. In the former case we can ensure a safe and comfortable driving experience by developing an accurate road sign detection and recognition system which can forewarn the driver of approaching signs. Such a system would lower the burden on the driver and prevent unwanted hazardous situations. For example such a system could warn the driver if he were about to enter a *Do Not Enter* zone, or could help avoid the driver from unwanted speeding. Further applications of this system could have it integrated with the cruise control system to release the driver of stressful driving on the freeway. Today's luxury automobiles come with a variety of sensors which assist the driver. One of the most innovative ones is the parking sensor which enables the car to park itself. We hope that this natural integration of signal processing technology in automobiles continues to evolve. We foresee that automated sign recognition systems will one day be carefully integrated with the Global Positioning interface that already exists in cars and will provide the driver with visual information about the surrounding. These are just the few beginning steps as we continue to develop automated systems in everyday life.

THE PROBLEM:

Our objective is to develop a system on the Texas Instruments C67 which will detect and recognize common road signs. Due to the limited time that we are working with and the inherent variability with real time signal processing of video images we have decided to scale back the project to fit into a semester long time frame. The signs that we have attempted to classify are shown in figure 1.

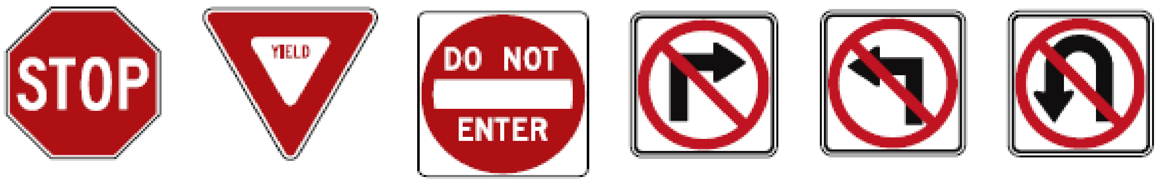


Figure 1

We shall be analyzing digital images of traffic signs, and shall be processing them instead of the frames of a video stream. All our testing images have been taken from the areas that constitute and surround Pittsburgh, PA. However our project can be applied to signs of other states due to their generic nature. We have encountered several difficulties that we hope has not slowed us down:

Handling of Occluded Signs:

Several signs are sometimes partially occluded, which makes it hard for us to recognize. To overcome this problem we have implemented a scale invariant feature as part of our classification.

Bad lighting conditions:

Several problems arise while using the RGB color space in conditions of bad lighting. We can over come this problem by converting our RGB space image to HSV (Hue Saturation Value). These aides in separating the ‘red’ color in good and bad lighting conditions.

Shifted signs:

Some road signs are partially rotated which makes it hard for us to compute its features in the recognition stage. We remove this problem by allowing for aspect ratio errors and resizing features within our system.

Excessive Noise:

Too much background noise can hurt the detection phase. We have adopted an adaptive filter based on the brightness of the picture which helps in removing noise. There is also significant difficulty in identifying signs in the night. Although the image may appear dark, the presence of street lamps shifts the spectrum of the image into the red region resulting in large amounts of noise.

PREVIOUS WORK:

We are basing our work on the basis of various research papers written on the subject. The previous Digital Signal Processing project that addressed this problem was done by Group No 7 in 2002- (Michael Kaye Anusha Krishnakumar Loris Stehle) [1]. We have worked upon improving their detection and recognition techniques which can be broken down as follows:

Detection algorithm:

- Extract Red Areas
- Apply morphology
- Label selected areas and retrieve coordinates

Recognition algorithm:

- Perform RGB differencing
- Perform maximum edge detection
- Perform Mahanalobis distancing to find minimum distance and classify signs

Significant Differences:

- The previous group worked with a sign of much higher resolution. They scaled that picture down and performed their calculations on a 200 x 200 sized image. We have decided to use 640 x 480 resolution image. By virtue of the observer we have decided that road signs can only appear in the upper right %75 of the entire image. Based on this our processing is done on a 360 x 480 sized image.
- We have also decided to perform HSV conversion for our image as it helps us in distinguishing colors better than in the RGB space, which was employed by the previous group.
- Another significant difference between our projects is the labeling methodology for all valid blobs considered. Our algorithm is robust and much more efficient than the previous.
- Color segmentation and noise reduction are very big parts of our projects as the foundation of the recognition stage is in the detection stage.

- Our recognition system is not based on Mahanalobis distances. The idea of our feature extraction and classification algorithm mainly comes from the paper written by Maldonado-Bascon, S. et al [1]. The rest was perfected based on our personal specification and choice of street signs. This gave us a higher accuracy at a very low cost.

Overall, we believe we have made a significant improvement over the previous system in terms of speed and accuracy. We also have a larger number of signs that we wish to classify. The previous group restricted their classification to: Stop, Do not Enter and Yield signs.

Several papers talk about using existing written systems such as Neural Networks and SVM which use test data to train and computes the optimum function that would classify the input correctly depending on its training. However, since we had only six classes, using such a network seemed a lot more time-consuming and not feasible in the short time frame. Moreover, most of our inaccuracies occurred due to issues in detection whereas our recognition algorithm seemed to be very robust.

FLOW GRAPH:

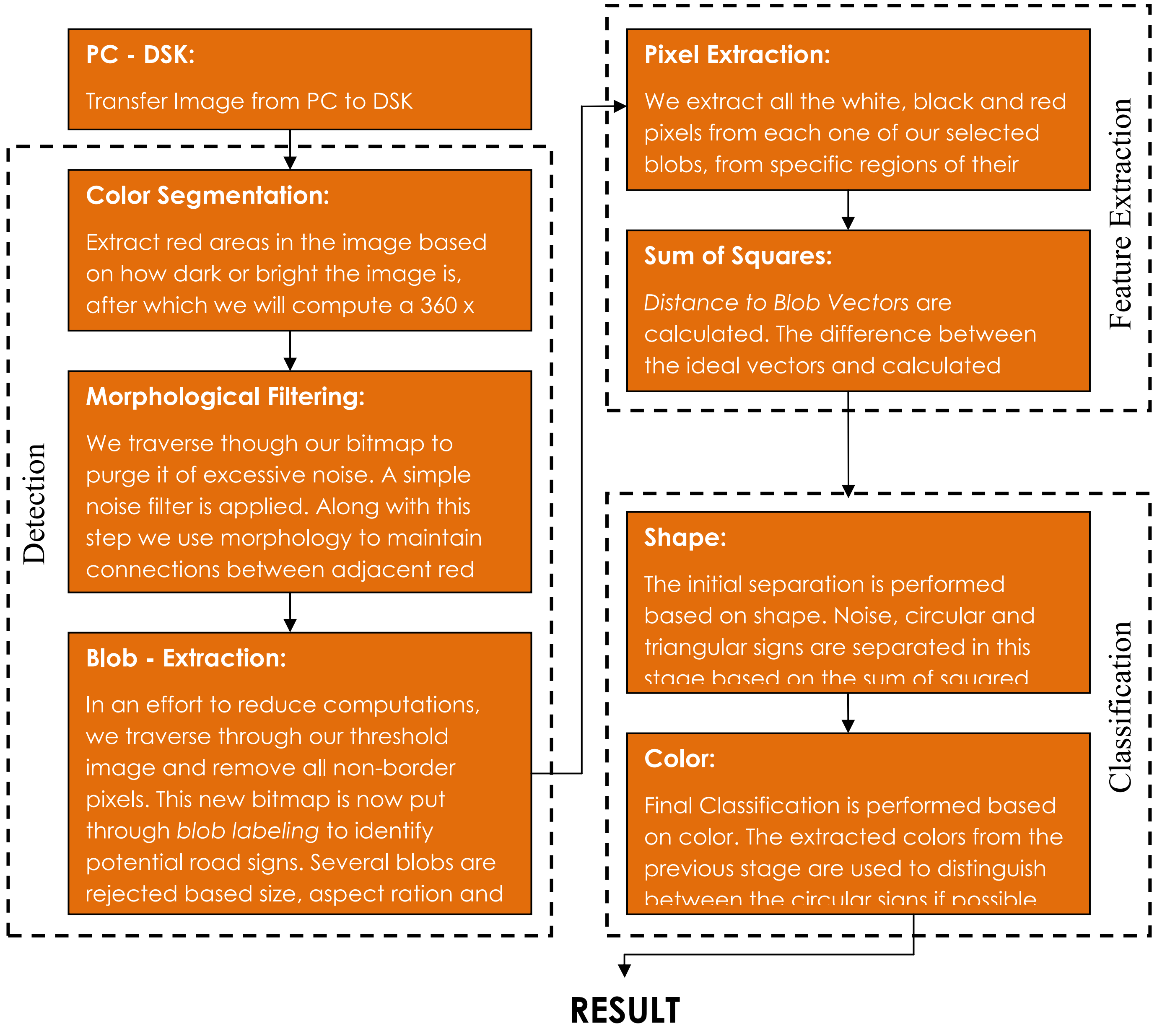

PC - DSK:
Transfer Image from PC to DSK
Detection
Color Segmentation:
Extract red areas in the image based on how dark or bright the image is, after which we will compute a 360 x
Morphological Filtering:
We traverse though our bitmap to purge it of excessive noise. A simple noise filter is applied. Along with this step we use morphology to maintain connections between adjacent red
Blob - Extraction:
In an effort to reduce computations, we traverse through our threshold image and remove all non-border pixels. This new bitmap is now put through blob labeling to identify potential road signs. Several blobs are rejected based size, aspect ration and
Feature Extraction
Pixel Extraction:
We extract all the white, black and red pixels from each one of our selected blobs, from specific regions of their
Sum of Squares:
Distance to Blob Vectors are calculated. The difference between the ideal vectors and calculated
Classification
Shape:
The initial separation is performed based on shape. Noise, circular and triangular signs are separated in this stage based on the sum of squared
Color:
Final Classification is performed based on color. The extracted colors from the previous stage are used to distinguish between the circular signs if possible
RESULT

Figure 2

DETAILED ALGORITHM:

In this section, we will describe more in detail the process and algorithm that we adopted in this project. Although numerous computations are being done on the images throughout, our project can be broken down into three major steps: Detection, Feature Extraction and Hierarchical Classification/Recognition. All the steps were tested in Matlab and successfully implemented on the DSK.

Detection

This is the first major subsection of our projection. The input to this section is the entire image under consideration and the output are all blobs that are likely to be road signs. Surprisingly and inconsistent with most research and work done so far, this section was the most challenging and time-consuming in the entire process. This can be attributed to the fact that we intended to work on images with varying lighting conditions and that this project was limited to six signs which made our recognition a simpler process. There were three sub-steps under detection all of which helped get the most accurate and concise set of blobs.

Color Segmentation:

When trying to identify signs in images with varying lighting conditions, working on the RGB domain is a very bad idea. This is a absolute-color based domain and so the R values of the signs would be different in bright daylight as opposed to twilight or night. For this reason, another domain space was implemented which would provide relative color and give more consistent values for the red signs, the HSV (Hue, Saturation and Value.)

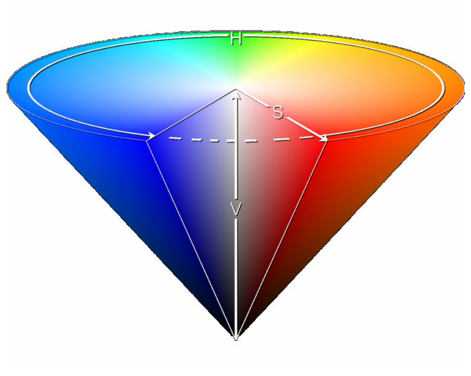


Figure 3

This domain is best described by the cone as in figure 3**.** Hue tells us what color it is and is measured in degrees. 0 represents red and moves counter-clockwise in the cone (30 is yellow, 60 is green and so on.)

Saturation measures the intensity of the color and is the weighted distance from the centre of the cone to the edge. A bright red pixel would have saturation of 255 whereas a pale red pixel would be lowly saturated and white would be close to 0. Lastly, ‘value’ (NOTE: When referring to Value in HSV, it will be indicated in single quotes to avoid confusion) shows how dark the color is and it is represented by numbers from 0 to 255 where 0 would represent a jet black pixel and 255 represents the brightest a pixel can get. In our project we have used normalized values for all three indicators i.e. 0-1.

Since all signs under consideration are red in color, our initial thresholds for hue revolved around 0. To account for a little variation in color, the hue value could be greater than 0.9 and less than 0.1. Saturation was always expected to be above 0.35 as below that the pixels starts becoming pale and white. ‘Value’ had a minimum limit of 0.25, below which the pixels would start looking black. Using these thresholds, we converted the HSV image to a bitmap. All pixels that fell in these thresholds were marked ‘1’ or made white or else a ‘0’ or black.

Even though this domain system gave better results than RGB in case of varying lighting conditions, it was far from the solution. For instance, when the above mentioned thresholds were used, they worked for most of the daylight images. However in twilight and bad lit images the amount of noise made this system inaccurate. This happened because at twilight time most of the image turned reddish, especially the leaves. Moreover, the sign itself was darker and did not pass the threshold mentioned.

For this reason, Dynamic Pixel Aggregation was adopted, which means providing thresholds based on characteristic of the image itself. Initially, the idea was to find the average ‘value’ of the image and based on how dark the image was, the thresholds would be set. However, this did not help while distinguishing some images. When a comparison of all image taken from the streets of Pittsburgh at different times of the day was made, the ‘value’ values did not provide even a remotely accurate representation of the darkness of the image. In some cases, twilight images seemed to have a much higher average ‘value’ than a day image. Thus, this could not be used to dynamically assign thresholds.

The second approach seemed to work better and gave much better results in terms of identifying the sign with white pixels and having less white noise around. In this approach, the image was first passed through the thresholds mentioned above and calculated the number of white pixels that resulted. This number was then used as an indication of how bright or dark the image is. If this number was very high, the thresholds were tightened such that we get lesser white pixels in the bitmap and vice-versa. The assumption we made before trying this method was that the red pixels in the sign would be ‘redder’ and more prominent than other pixels on the image. So when the number of white pixels is very high, we believed that when tightening the thresholds, the noise would disappear rather than the sign itself.

Fortunately, this sum of white pixels was a good indication and helped decide what thresholds the image should be passed through. We made five threshold categories and the sum of white pixels decided which of these categories an image fell into. Working this out was purely trial and error and depended on the database of images. In some papers, researchers have mentioned thresholds that worked for them but they worked terribly for us. Ideally, the more the categories, the better and finer the distinction will be between images of varying lighting conditions. Since our database consisted of only 108 images, five categories worked just fine. The result of this color segmentation process was a bitmap image in which road signs and other red areas were colored white, otherwise black.

Morphological filtering:

Now that a bitmap image which has all red areas indicated as a ‘1’ is available, the image needed to be prepared for blob extraction. This is one of the aspects where we have spent a lot of our research time. A lot of papers talk about using morphological filters and how important it is in image processing. However, none give details as to what kind of morphological filters are good in road sign recognition. When staring at the bitmaps of few of the images, it is very hard to design a filter that would distinguish noise close to the road signs and disconnected of the sign itself. When dilation was attempted, noise pixels that were one pixel away from the sign got connected as well as two big chunks of a sign also one pixel away.

After trying a few filters, a couple of them seemed to help this cause well. This process was divided into a two-step process: Maintaining Connectivity and Weighted-average.

In 'Stop' and 'Do Not Enter' signs, the vertical connections are lost in many cases and if the connection is very thin, it disappears when applying erosion. For this reason, first connections are confirmed and maintained while the rest of the pixels undergo the weighted-average filter. As you can see in figure 4**,** these are the cases under which the pixel under consideration remains white (turns white in the case of first kernel) and the weighted-average filter is not applied on it.

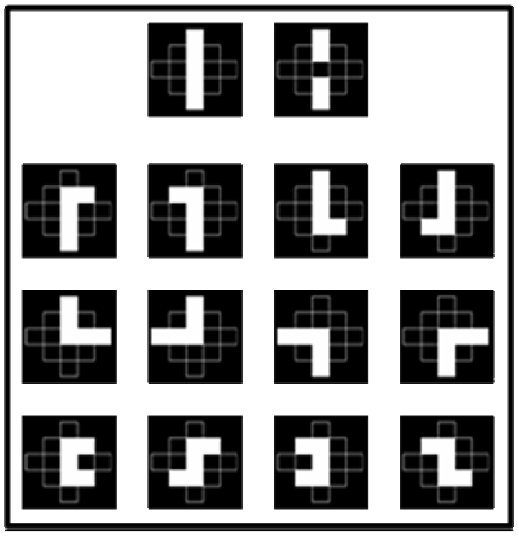
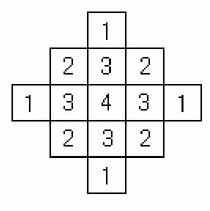


Figure 4

You will also notice from the figures that we have maintained only vertical connections. As I mentioned above, this is implemented to not erode away the thin connections in 'Stop' and 'Do Not Enter' signs. All horizontal connections are passed on to the weighted-average kernel directly since there is a very small chance of this connection belonging to the sign itself. However, the only disadvantage to this is that if there is any noise that is either connected vertically or just one pixel away in a vertical manner, this would be joined to the main blob. This does not cause as much of damage as it helps. The recognition stage is likely to identify the road sign if noise is little but if a valid sign is rejected because it was disconnected by a pixel, it would have no chance of being identified.

After ensuring vertical connections, an erosion filter is applied to reduce noise in the image. This was done using a weighted diamond kernel as shown in figure 5. Whether a pixel should be eroded or kept was decided by itself and its neighbors. If the sum of weights of white pixels was greater than or equal to 18, the pixel was changed to/kept white otherwise it was changed to ‘0’.

The primary reason for adopting this diamond kernel and weights was to correctly identify the neighborhood of a pixel. Giving the pixel itself the same weight as one that is one pixel away did not seem the right way to filter. When using weighted-average, we are using a more accurate representation of proximity of white pixels. The four outermost pixels being white is very different from the immediate neighbors being white and a method that distinguished these two needed to be implemented.



		1		
	2	3	2	
1	3	4	3	1
	2	3	2	
		1		

Figure 5

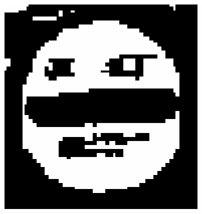


Figure 6

If this averaging filter was applied to pixels that were saved by connections, thin connections would disappear. Figure 6 shows a Do Not Enter bitmap before morphological filter is applied to it. The thin connection in the left is completely cleared even when the smallest erosion is applied to it. However, when we confirm connections, these pixels are left alone. This process does result in a little more noise but this is not very significant as compared to ensuring connectivity in signs. Overall, the image is mildly eroded and increasing the erosion by even a little fraction more was hurting a few images.

Blob extraction:

Now that we have a bitmap image with reduced noise using morphological filtering, the blobs will be identified as separate entities, all with the possibility of being a street sign. When using Matlab, one can simply call the function ‘bwlabeln’ after which all blobs can be treated as separate entities. However, when coding this algorithm in C, this is not very efficient in terms of memory and runtime. For this reason, the edges of all white chunks are retrieved. The algorithm that is later implemented requires accessing every white pixel in a chunk. If we do not work only on the edges, it would become computationally very expensive by trying to access a lot more pixels in a chunk than needed.

To get the edges from the bitmap image, we implement a similar algorithm as the one used in the function 'edge' in Matlab. The result of this is a bitmap where all chunks of white pixel have disappeared and only the borders remain. We then remove all subset edges to reduce complexity and false blobs later. For instance, when getting edges of a Stop sign, there will be an octagonal shape from the edge of the entire sign and also edges of each of the letter of the sign. This is redundant and does not need to be sent for further processing. This step would backfire if the signs were caught within a larger chunk of white but this is extremely rare. Our next step is blob labeling which a computationally expensive algorithm and is significantly helped by using morphological filters (which reduces the number of such chunks) and by finding edges.

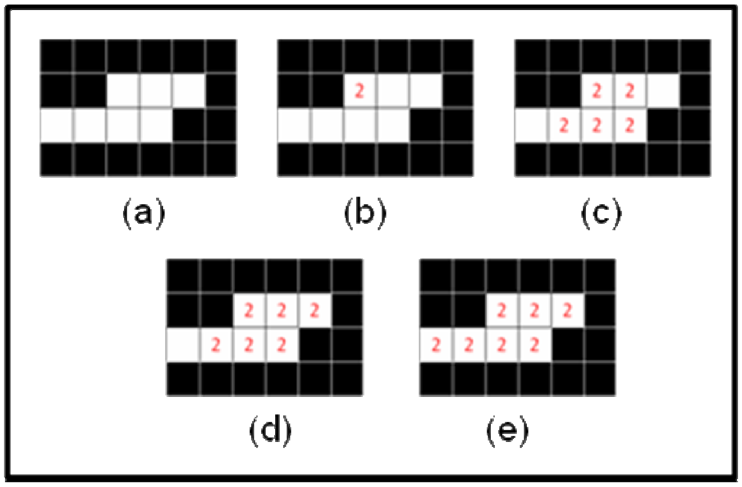


Figure 7

The blob labeling can be explained by looking at figure 7. Figure 7(a) is an example of a chunk of white pixels on the top of the image. Since the black pixels are 0s and white pixels are 1s, we will start labeling blobs with 2s. We browse every pixel in the image row-wise to search for a white pixel. When we find a white pixel, we label it a number not already used. Since this is the first chunk we are labeling, it is labeled 2 as can be seen in figure 7(b). Along with labeling the pixel itself, any 8-connected white pixels are also labeled the same number as shown in 7(c). Then we move on to the next pixel which is labeled 2 and **not** 1.

This is because we have started identifying one blob and we want to finish with one before we move on to the next chunk. Thus when we move to the next pixel labeled 2 (which is the very next pixel), we turn all its neighbors to 2 as well as shown in 7(d). We continue searching the row for another 2 which is again the neighboring pixel. This time when we try to turn its neighbors into 2, we find that there are no unlabeled white pixels. Then we move to the next row and continue searching for a 2 and continue the same process. Since the second pixel is labeled 2, we label the first pixel as well. We move to the next row only when no changes have been made. When we reach a row where there is no pixel labeled 2 at all, this means that we are done labeling this blob and must start searching this image again for an unlabeled pixel and continue with the process by labeling it 3 and so on. This process could have been done iteratively or recursively. Even though it would have been easier to code recursion, using iterations made the algorithm more efficient.

Now that we have labeled all white pixels, we can treat as separate blobs. For instance, to separate the blob labeled 2, we simple find the minimum and maximum row and column values which is all the information that we need to identify the blob. Similarly, these four values are retrieved for all labels and considered to be potential blobs. These blobs include every white pixel that was in the bitmap image and so they are too many to be sending to the recognition stage. We know the minimum size that we wanted to recognize by looking at a few images from our database and since the shapes of all street signs are regular and must fit in a square, we could eliminate blobs based on size and aspect ratio. This reduced the number of blobs by a huge percent since most blobs were little noise blobs all over the image. Also, this excluded those big blobs whose length over breadth ratio was very high or low. The following were the three conditions:

1) The length and width must be a minimum of 40 pixels each
2) The total number of pixels in the blob must be at least 2500 pixels and at most 30000 pixels
3) The aspect ratio (i.e. length/width) must be between 0.67 and 1.5

On average, the number of blobs reduced from a whooping 100 to 2-3 blobs which are referred to as 'valid' blobs.

These valid blobs are, however, not yet ready for the next step. The size and location of every pixel in the blob is important. Some images might have larger street signs compared to others while some may have the sign rotated along the vertical axis which makes the sign rectangular. We want to standardize all signs in terms of their size for the next step such that the features extracted from two identical signs, taken at different locations and distances, result in similar features. For this reason, we normalize/resize all signs to 50x50 pixels. These dimensions were chosen based on the minimum size limit i.e. 40 pixels. To resize a blob that has fewer rows (or columns) than 50, we need to add rows (or columns) to normalize it. Adding to many rows (or columns) did not seem like a good idea because of the method we were using (as described below.) Down-sampling was not a problem because the rows (or columns) that remained acted as good representatives of the entire image. Moreover, 50x50 pixels size was a perfect size for us to extract features; neither was it computationally expensive nor was it too small to distinguish from the rest.

To normalize the blobs, we adopted the simple algorithm of averaging and down-sampling. For instance, if we had a blob of 45x55 pixels, we need to add 5 rows and remove 5 columns. The way we select where to add rows is as follows:

1) First we find the number of rows that need to be added (50 – 45 = 5 rows)
2) Adding 1 to this number gives the number of divisions we need to make in the blob (5 + 1 = 6)
3) Find the length of each subdivision. (45/6 = 7.5) This indicates that a row needs to be added after every 7.5 rows (Lets call this number n).
4) Since we cannot divide a pixel any further, we adopted a different method which is as accurate as it can get.

5) We now start from the first row and keep moving until the row number is greater than or equal to n. So when we reach 8, we know that this is the end of the first subdivision and a row needs to be added after this.
6) We then multiply n by 2 and keep browsing till the row number is greater than the new n. The next row number is 15 that would be greater than or equal to 15 and so another row will be added after the 15th row.
7) This process continues till the end during which the required number of rows are added.
8) The value of each added pixel is calculated by checking the average of the 6 pixels around it. If the average is greater than 0.5, the pixel is made white and vice-versa.

The same process occurs for columns as well and when down-sampling. The only difference being that in down-sampling, we do not need to take the averages. The row (or column) is entirely removed. The edge cases are handled appropriately. Some have two or three neighboring pixels and so the averages are taken accordingly. All valid blobs are now resized and ready for feature extraction.

Feature Extraction

In this second major step, we try to find out aspects about the concerned street signs that would distinguish them from the others. This can be called the 'backbone' of our recognition process. Selecting features did not only involve finding significant differences in street signs, but comparing some of the bitmap blobs that we were working on. We had to keep in mind noise, shifted and rotated signs, occlusions, etc. The robustness of our selected features determined the robustness of our system since these were the criteria that we were sorting the street signs on.

SOS of DtBs:

One prominent feature of signs was their shape. Distinguishing shape would categorize triangles, circles and help us eliminate noise. The Yield sign was the only triangle, while the rest were all circles. The Stop sign, though being octagonal, is considered as a circle because it is very difficult to distinguish the two in the method that we are using. Moreover, with little noise, there would be no way to tell one from the other. If the blob did not fall into these two shape categories, it was disregarded as noise. This feature is called the SOS (Sum Of Squares) of DtBs (Distance to Borders). Below are the steps as to how these values are calculated.

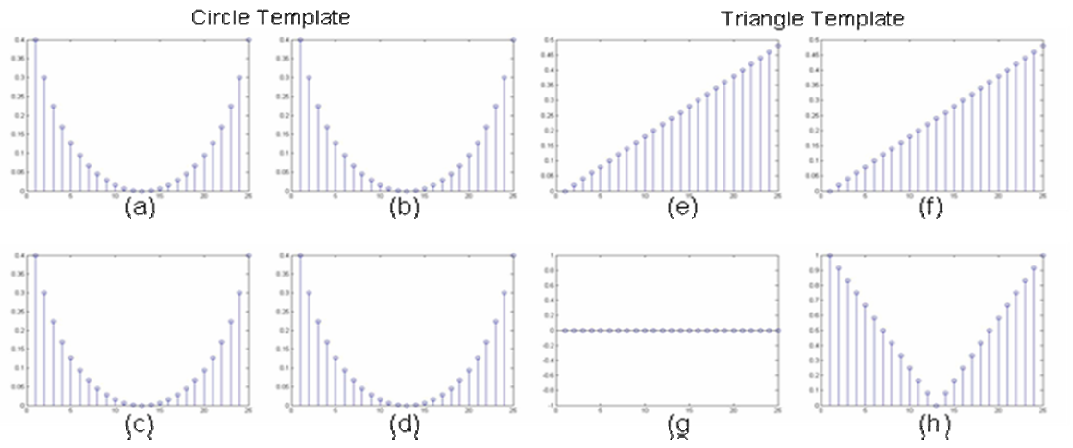
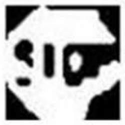


Figure 8

1) Find the distances from the border of the blobs to the first white pixel for each edge i.e. left, right, top and bottom. (as shown in figures 9 and 10**)**
2) These vector graphs give a good indication of the shape of the possible sign from each corner.
3) We compare these vector graphs to ideal triangle and circle templates (figure 8). The differences in these graphs are called the 'error' vectors and give an indication of how different a blob is from a template. The graphs in the figure are arranged in the order: left, right, top, bottom.
4) However, these include vectors that would be completely off because of noise and occlusions. For this reason, we remove the outliers (represented by the bold lines in figures 9 and 10).
5) Any error vector that is outside the range of Mean ± 1.5*Standard Deviation of the error vectors is considered as an outlier.

6) The sum of squares of the remaining vectors is taken for each edge and each shape. Thus we now have 8 sums of squares which act as quantitative measures of the shape (the numbers written on the graphs.)



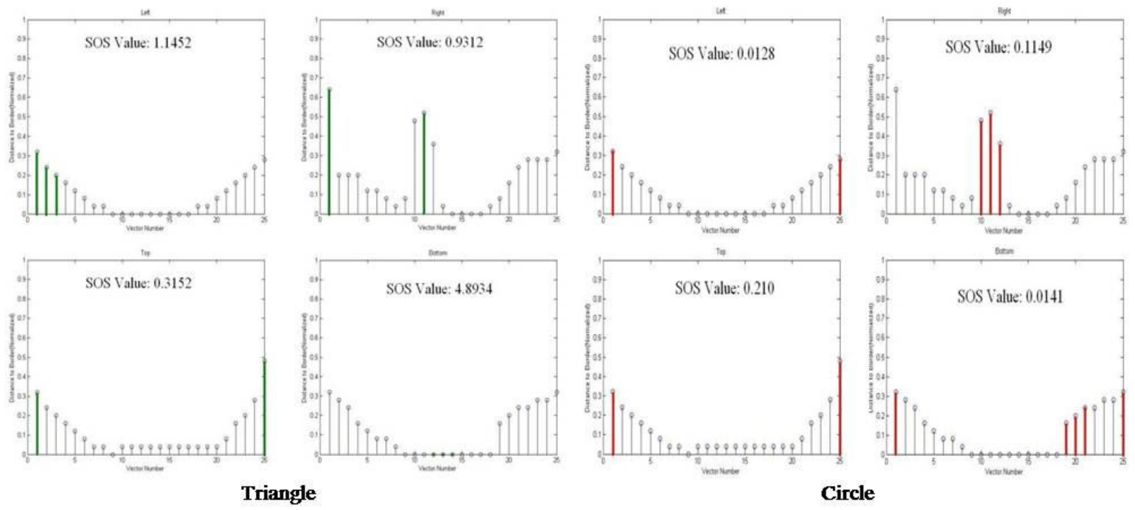


Figure 9

No. of color pixels:

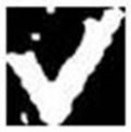
Once we know that a sign is showing the attributes of a circle, we need to find addition features to be able to distinguish exactly what sign it is. The best way to distinguish these signs is by understanding the color combinations of the blob. If we can exactly locate what color of pixels exist in a certain region of a blob, we can categorize it to be a specific sign. For instance, Do Not Enter sign has a white rectangle in the middle whereas the circle signs have some red in that region. In this section, we will only describe the features that we have decided to extract and in recognition as will describe their exact usage.

Red Pixels:

This is the total number of red pixels in the 50x50 pixel blobs. Since we are already working on bitmap blobs that have been computed based on redness, we use these bitmap blobs to find the total number of white pixels. This is an indication as to how red an image is.

White Midsums:

This is a count of the number of white pixels (of color image) in the rows between 23 and 27 and between the columns 8 and 42. To get this, we set new thresholds to retrieve only white pixels. The result is a bitmap in which white pixels actually represent white in a color image and black represents everything else. The midsum is calculated from this bitmap.



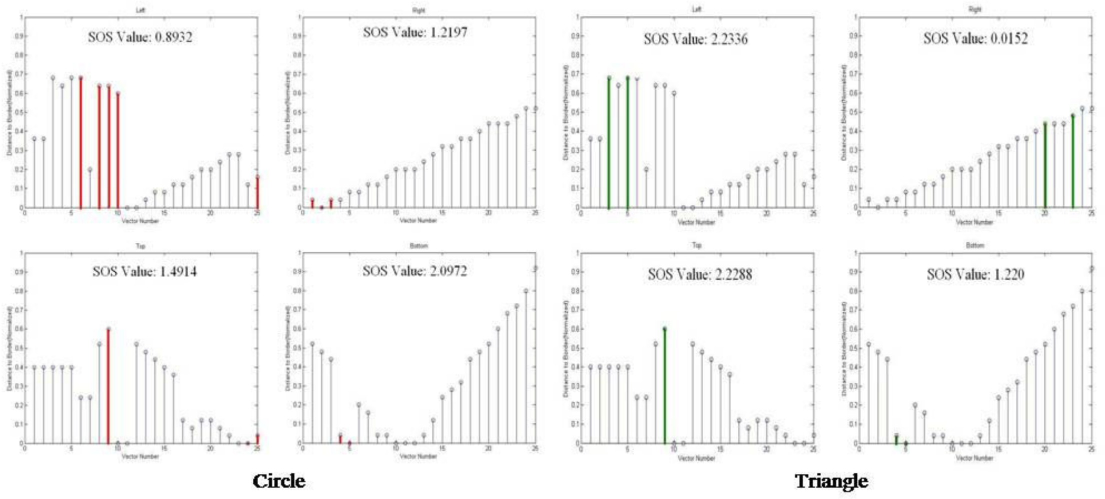


Figure 10

Black pixels:

The no turn signs have everything in common except the black arrow in the middle. We needed to find a way such that we can differentiate between the left, right and u-turns. We again use dynamic pixel aggregation to retrieve the black areas of a blob with the least noise and maximum density in the right areas. Moreover, since the arrow was in the middle of the blob, we decided to forcefully remove noise in areas that we are sure would not contain the arrows. We used the template shown in figure 11 for this purpose. If a pixel is white in the template, that particular pixel in the blob is turned black. As you can see in the figure, all red areas in a no turn sign and everything outside is turned black. This leaves only the pixels within the circle. Based on location of the arrows, we count the number of white pixels in the bitmap of different locations as shown in figure 12. The regions are referred to as Top, Top-Right, Bottom-Left and Bottom-Right.

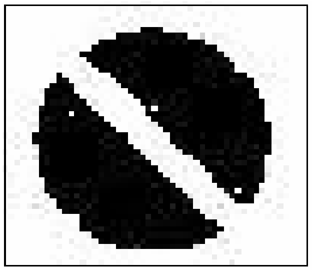


Figure 11

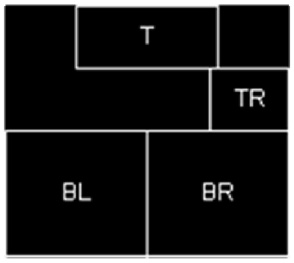


Figure 12

Hierarchical Classification/Recognition

This is the final step of our project in which we determine the status of each blob i.e. if its noise or actually a sign and if it is a sign, what sign is it. This step uses all the features calculated in the previous step to find the identity of the blobs. The entire process can be summarized as a tree structure as shown in figure 13.

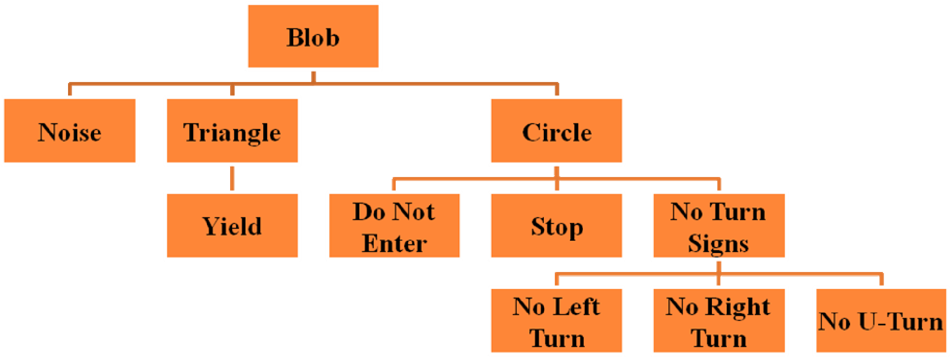


Figure 11

Firstly, we use the SOS of DtBs to determine the shape of the blob. As mentioned in the previous section, we get 8 numbers which must be used to determine what the shape looks like. Since our system was being designed keeping in mind the possibility of noise and occlusions, it was not a good idea to enforce all four sides to demonstrate the properties of a particular shape. We believed that allowing for two sides to be affected by noise and occlusions is allowable. However, if three sides are affected, it would be hard to distinguish between a three side affected sign or noise that by chance demonstrated the properties of a circle or a triangle. Thus, if DtBs of any two sides of the blob is lesser than the threshold provided, we consider it to be that shape. However, we do not take care of the condition if which two sides corresponding to a circle and the other two to a triangle and neither have we encountered any such image.

If we find that it is a triangle, we conclude the sign to be a Yield sign or if the vectors do not match both, we disregard it as noise. If the sign is classified as a circle, there is further classification that needs to be done to determine whether it is Stop, Do Not Enter or no turn signs (No Right Turn, No Left Turn, No U-Turn.) We now use our color pixel features to help in this further classification.

If the number of red pixels in the sign is low and the midsum number is moderately high, it is considered to be a no turn sign. When the midsum number is low, it is identified as Do Not Enter and if the number of red pixels is very high and midsum is also high, it is a Stop sign. This does not give very accurate results when the Do Not Enter sign is significantly shifted because the midsum value would not represent what it should. We realize that these are very few features to distinguish three signs. However, this is the best we could do in the given time-frame and they actually gave us very good results. Ideally we would decide on a couple of more features to confirm our identification but most of our time was dedicated to detection, the part which required the most effort.

Lastly, when we know that a blob is a no turn sign, we must use the black arrow to determine exactly which no turn it is. We take the number of white pixels in the bitmap in each of the four regions as mentioned in the previous section. Figure 14 shows exactly how the three no turn signs differ in these regions.

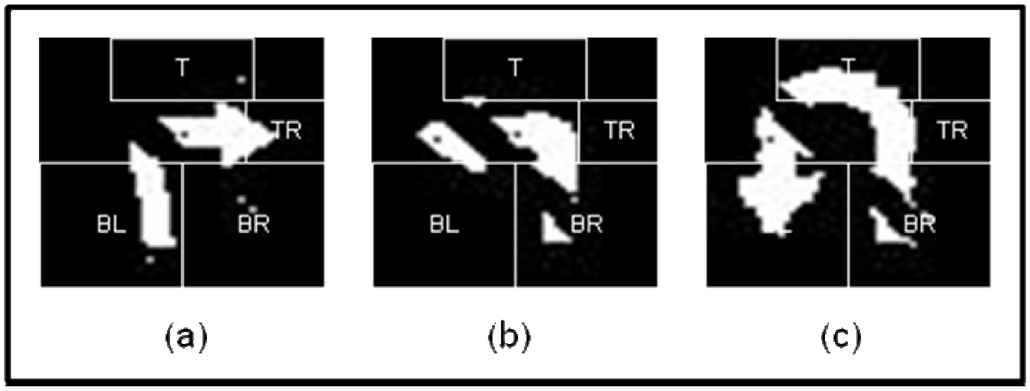


Figure 12

RESULTS:

Test Data

Our recognition system processed 640x480 pixel RGB images and attempted to recognize any of the six traffic signs mentioned earlier in them. The images we tested were taken from a Nokia N95 cell phone camera to mimic the field of vision of a driver of a car. Below are the results of our testing.

	Total	Positives	False Positives	Accuracy
Stop	28	20	1	71.4%
Do Not Enter	28	23	0	82.1%
Yield	10	8	1	80.0%
No Left	19	14	1	73.7%
No Right	16	11	0	68.8%
No U-Turn	7	6	0	85.7%
	108	82	3	75.9%

Figure 13

Stop and Do Not Enter signs were tested extensively with very satisfactory results. No Left Turn signs were not far behind in accuracy. It was the No Right Turn where we would have liked a better recognition rate. We see a discrepancy in the number of signs tested of each type. This is because No U-Turn and Yield signs are found particularly on highways and it is rather difficult to stop and click a shot of such signs. This explains the low number of those signs tested. Overall, the recognition rate was pretty impressive.

Analysis

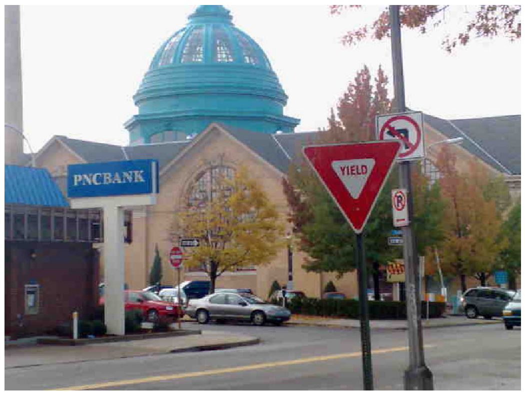


Figure 14

← Original Image

In this image, there are two signs, varied in size and one is obstructed by the other.

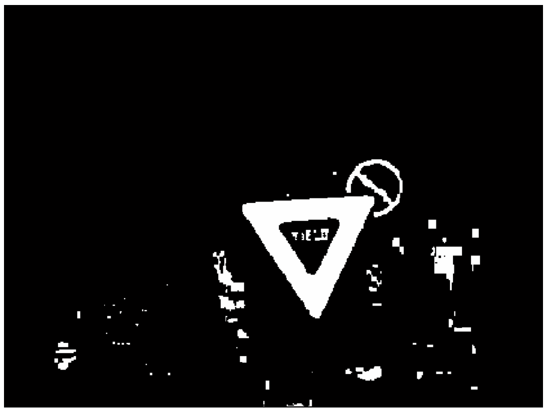


Figure 15

← Morphed Threshold Image

Even though one sign is obstructing the other, they are two distinct blobs. Our threshold is successful in extracting the relevant red areas. Further Morphological filter enhances the blobs.

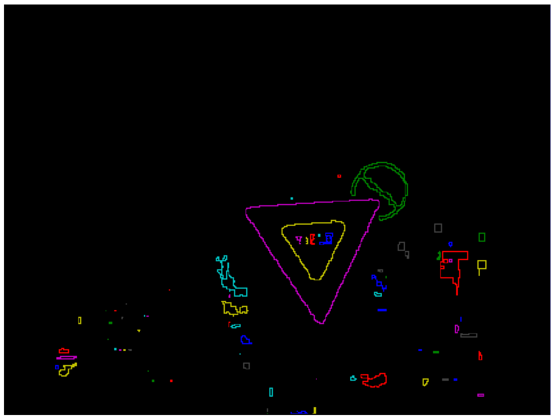
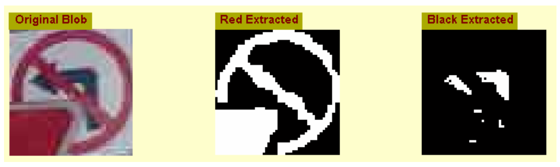
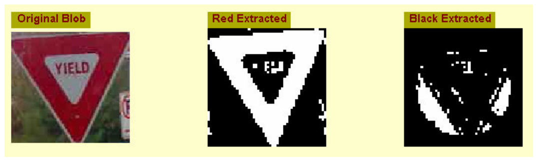


Figure 16

← Blob Extracted Image

As expected, the two blobs are identified and forwarded to the recognition. Our recognition is successful in identifying both the signs as No Left-Turn and Yield sign.





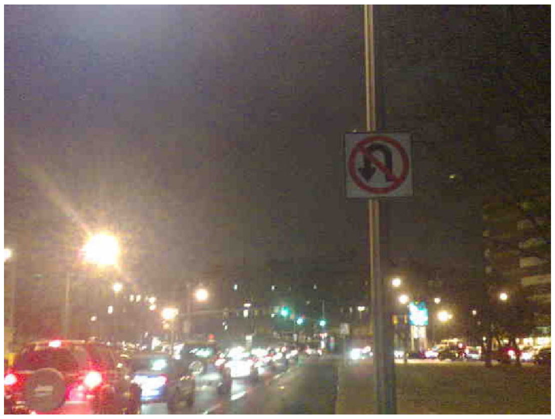


Figure 17

← Original Image

This image is a little darker taken sometime after twilight. We can see that the sign is not that prominent. Also there are some bright street lights and red rear car lights.

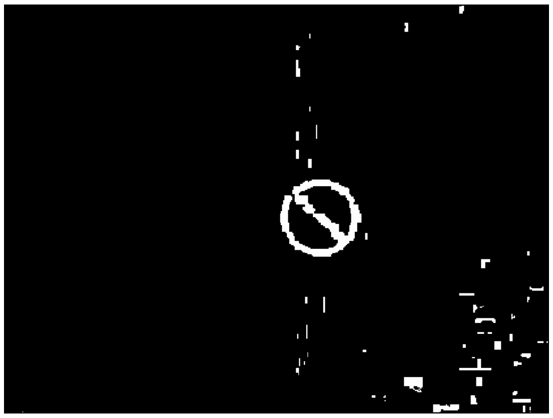


Figure 18

← Morphed Threshold Image

This is where our adaptive threshold is in full effect. Even though the saturation and value of the sign is not high enough for a regular bright sign, our detection relaxes the boundaries due to the dark ambience.

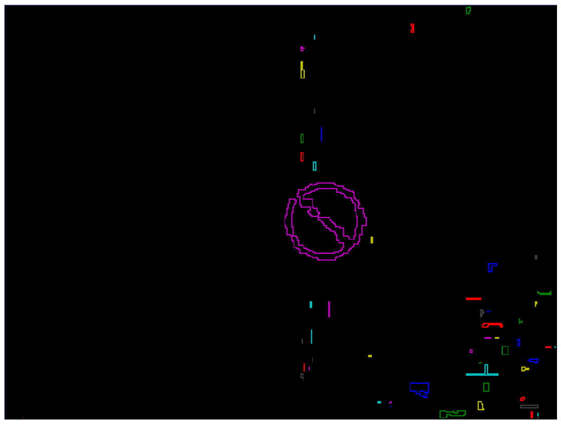
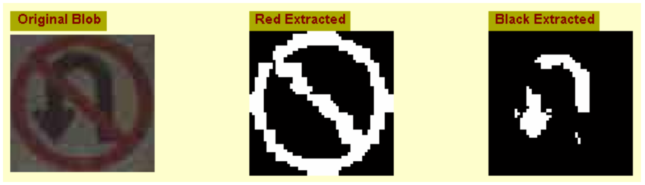


Figure 19

← Blob Extracted Image

Only one blob meets the size and aspect ratio requirements. It is forwarded to our recognition which is successful in classifying it as a No U-Turn Sign.



Noise Blobs:

Our feature extraction lays the foundation of rejecting noise blobs. Our hierarchical classifier uses the features calculated to recognize the shape of the blob. If it doesn't find a triangle or a circle, it marks the blob as noise. As we can see in figure 20 (a) and (b), both the blobs passed our detection phase but were correctly classified as noise by our recognition phase.

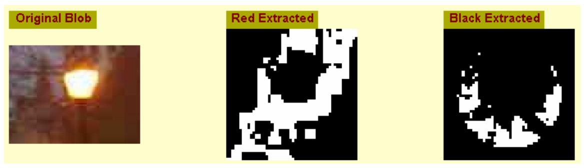


Figure 20(a)

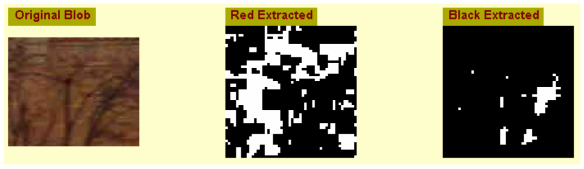


Figure 20(b)

Noise Interference:

In some unwanted cases, noise from the background interferes with the sign. Typically, a red building or something in the shade of red in darker images displays such phenomenon. Below is one of those examples.

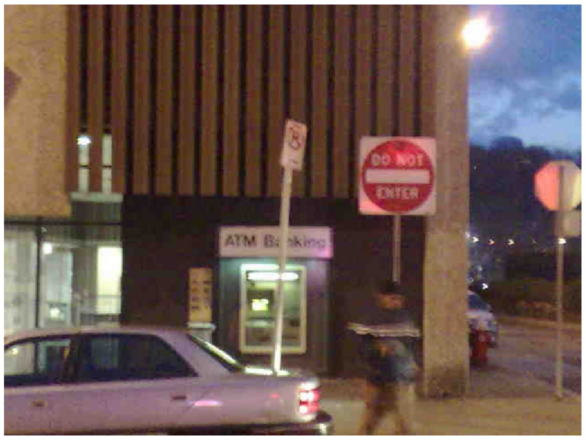


Figure 21

← Original Image

This image displays a problem which our project fails to address. Under bad lighting conditions, with a more relaxed threshold, more and more unwanted areas are tagged.

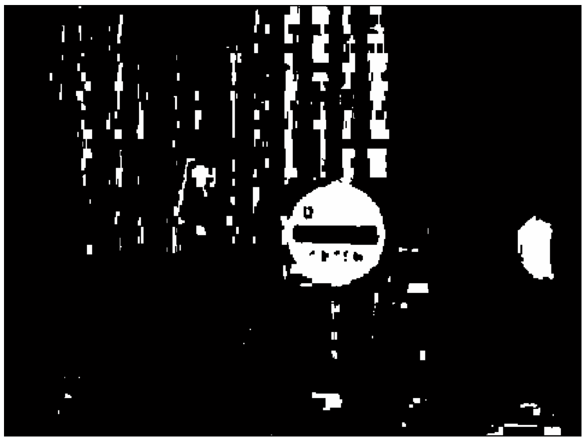
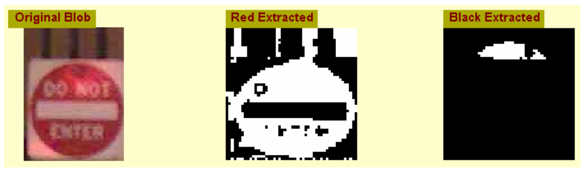


Figure 22

← Morphed Threshold Image

As you can see from this bitmap image, the background noise (very slightly red in shade) is also classified as part of the sign. Now the extended blob is forwarded to the recognizer. Due to such an anomaly, our recognizer classifies it as noise.



False Detection:

In some extremely rare cases, our algorithm falsely classifies a blob as a sign. We can see below in Fig 23, where a bunch of red leaves is recognized as a Yield sign. This is happening because of the mere shape of the arrangements of the red leaves. If we observe carefully, the red extracted bitmap image looks like a broken upside down triangle.

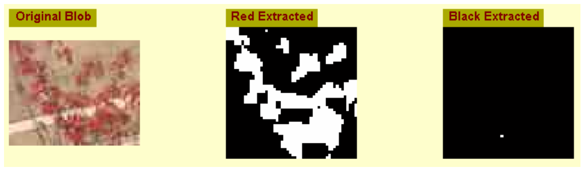


Figure 23

Irregular Illumination:

Irregular illumination on the sign results into abnormal shading. In figure 24 we can see that a strong light from the right side is actually dividing up the sign into two distinct colors. The left half is normal red, while the right half is bright orange in color. Here the left half fall within the selected threshold for the image and hence being identified as seen in the morphed threshold image. While the second half is invisible as it did not satisfy our threshold. Thus our detection phase fails to label the sign as a blob.

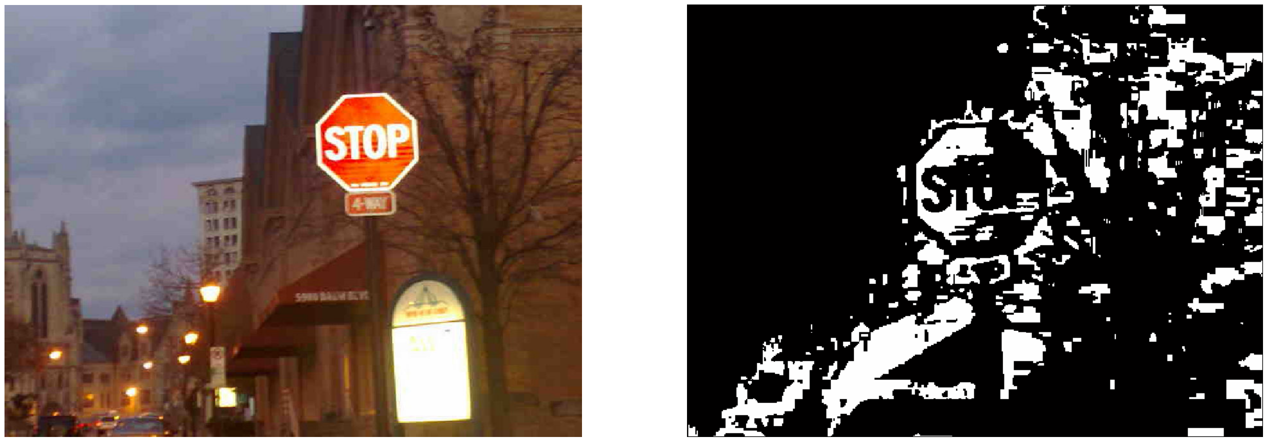


Figure 24

Rotation Invariant:

Our detection and recognition algorithm is not too sensitive to rotation. As long as the overall shape and the features are maintained, rotation has no effect on the result. But if a sign is restated to such an extent so as to change the shape, it would not recognize the blob. Not only the shape but also the features play an important role in rotation especially in circular/octagonal signs. Rotation has no effect on the shape, but might disrupt the features our classifier uses to identify them. Figure 25 is an instance of a rotated Stop sign which is recognized accurately.

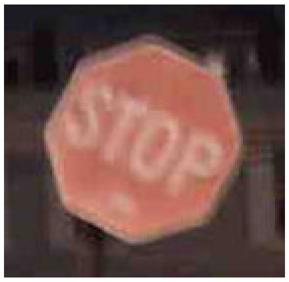


Figure 25

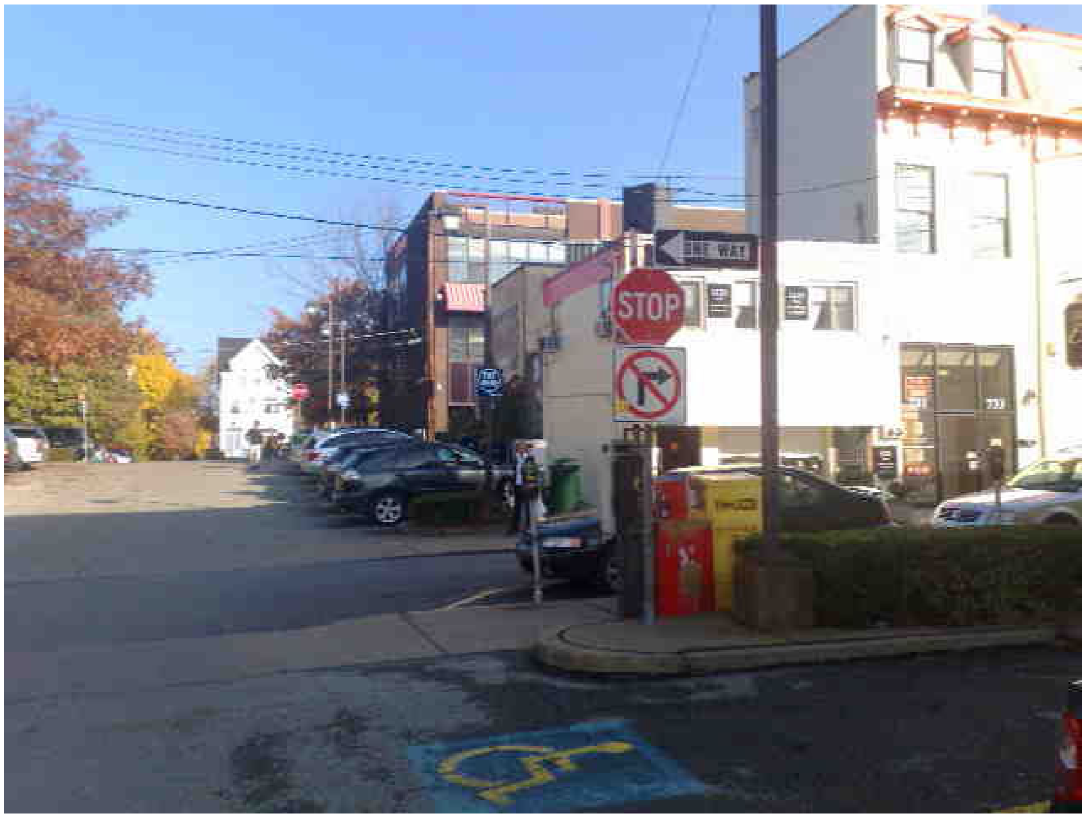


Figure 26 – This Stop is an indication of the smallest sign that our system could recognize. The No Right Turn sign is not recognized.

What Didn’t Work/Issues

In this project, we set off with a target of recognizing both red and yellow traffic signs. But in our testing phase, yellow signs did not work well for us. The reason behind this was that the spectrum was very narrow for yellow in HSV domain unlike that of red. Another problem with yellow signs was that yellow lies between red and green in the hue domain. Both red and green are dominant colors found extensively in the background. Hence, we dropped the idea of recognizing yellow signs and redirected our focus on red signs.

Another aspect which did not work as well as we would have wished was the noise interference. Even with our adaptive threshold of red color segmentation, some pictures give false impression of brightness. Typically, a picture shot at night time with many cars and street lights in it. Even though the ambience is dark along with the signs, the lights increase the number of bright pixels and thus forcing a threshold corresponding to a real bright image. Also different lighting conditions affect the recognition. Even if a picture is shot in bright daylight, a sign may not have any illumination on it and hence not be recognized. Typically, an image in broad daylight, except the light is behind the sign and not on it. This will result into a dull sign with bright background.

DSK IMPLEMENTATION:

Memory

The on board memory is classified into two parts, namely internal and external. Internal memory is the 256 KB available on the chip itself. External memory is the on board memory sizing to 1.2 MB. As we mentioned earlier, the image size we are processing is 480x360 pixels (25% cropped from the bottom and left portions of 640x480 pixels). Now with that size, it is impossible to keep the original image in RGB domain in the internal memory. Hence, we keep it in the external along with the red extracted image and the bitmap border image. The no turn template (50x50 pixel bitmap image) is also stored in external memory. The internal memory is used by all the local counters and flags used in the algorithm described. The blobs which are extracted are resized (50x50 pixel bitmap image) and saved in the internal memory also, along with the black extracted bitmap image of the same size.

Code size:

- DSK code: 1650 lines (46 KB)
- PC code: 508 line (14 KB)

Upon testing the access times, we find that internal memory access are substantially faster than external memory (1.5 cycles for internal memory compared to 5.5 cycles for external memory). Hence the most computationally complex functions in our project are:

RGB to HSV (w/ thresholding): **10.9 million** cycles (63.5 cycles/pixel)

Morphology: **13.8 million** cycles (80.4 cycles/pixel)

These times are higher than expected. This is because the image RGB to HSV conversion processes is located in the external memory and hence every iteration will need to access each pixel from the external memory. Morphological filter too works on the red extracted bitmap image located in the external memory, hence take so many cycles to process.

Paging

To optimize the speed of our implemented algorithm, we tried implementing paging in the computationally complex functions. No improvement was seen in the RGB to HSV function as each element got accessed once every iteration. Hence paging the image data would not help save accesses. Also, paging would we impossible to implement in the blob extraction function as it is a recursive implementation. Hence, paging the image blocks would just increase the downtime of each recursive call dusting function. Also there is no guaranteed pattern access in the recursive calls for blob extraction. It could be that paging would result into paging a whole block for just one access, which would increase the processing time rather than making it quicker.

However, paging could have been useful in the morphological filtering as kernel of two neighboring pixel will have. As shown in figure 27, 1 and 2 are the pixels which are processed and yellow pixels are the ones which are repeated in the two consecutive kernels. Hence, paging the image while applying the morphological filter helps increase performance time. But it will not be visible in our project as the kernel we use is extremely small. If a larger kernel were to be used, paging would definitely be advantageous..

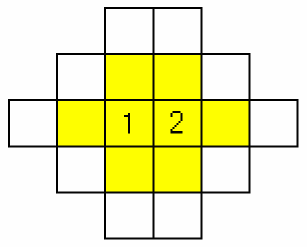


Figure 27

Optimization

Using the GCC optimization level three, we saw a need to further optimize the code for better timing results. The major optimization changes could be categorized in the following four topics:

- Loop unrolling – We unrolled the big loops which iterate over the rows and columns of the image. This allows the processor perform computations in parallel than in series, hence making the loops go faster.
- Local variables for repeated calculations – Instead of computing small calculations over and over again, its better to save that value in a local variable and use it wherever needed. This helps in speed by saving unnecessary calculations already done before and by saving accesses to external memory.
- Minimizing function calls by saving values used in future – Rather than calling the functions every single time to calculate the same values, we save the return values of specific functions and use them in other functions. This helps minimizing functions calls, hence reducing the complexity of the code.
- Writing to external memory only when features are finalized – When something is to be written to external memory, we make sure it is the final value to be written. We avoid writing base values to external memory and then changing it as we go. This again saves accesses, reducing the total time.

POSSIBLE IMPROVEMENTS

- Detection – For future, more ranges in hue domain could be defined in the detection phase to enable the system to detect signs of different color. Traffic sings in the US are mainly red, yellow and blue. All three could be detected if the hue domain was properly divided and adjusted according to the brightness of the image.
- Recognition – More features are to be added if the number of signs to be recognized are increased substantially. Also with more features, it will be extremely hard to define a hierarchical classifier with just human intuition. Some sort of neural network (like SVM) would be required to weight those features for recognition.
- Optimization – Paging will definitely help if a bigger kernel is used for morphology.
- Video processing – This system could be easily extended to a video. By using some image enhancing algorithms with the help of consecutive frames, a decent image can be obtained. From that image, a sign can be recognized. This forms the basis of autonomous driving.

SCHEDULE:

- 17 Oct – Starting initial Matlab testing of detection on web images.
- 29 Oct – Detection modifications and test data collection.
- 31 Oct – Starting DSK coding for detection.
- 12 Nov – Detection complete on DSK, weighing different recognition algorithms.
- **12 Nov – Midterm Oral presentation.**
- 19 Nov – Recognition options, SVM or Hierarchal Classification. More test data collection.
- 22 Nov – Features decided, reject SVM and go ahead with hierarchy. GUI started.
- 29 Nov – Everything working on DSK, last minute optimization on all algorithms, coding and transfers for better results. More data collection.
- **03 Dec – Oral in class, demo at night.**
- **10 Dec – Final project report due.**

WHO DID WHAT?

It is difficult to classify different aspects of our project to particular individuals. Even though we all had our areas of interest and expertise, we worked as a team and contributed wherever we thought our input would be crucial. Below is an overall description of various topics and who was involved:

- Matlab testing – All the group members were involved in this step. At this stage we didn't know much about the algorithms helpful to our project.
- Detection – Kaushal started molding the detection algorithm with some valuable inputs from Pratik. Hussain was responsible for the C implementation of detection algorithm.
- Feature Extraction – Pratik spear headed the key feature extraction methods. Kaushal and Mudit assisted him with some of the issues he faced.
- Hierarchy Classification – Mudit was mainly responsible for making sense of the features in this recognition part. Valuable contributions were given from Pratik and Kaushal.
- DSK coding – DSK C coding was mainly done by Hussain and partly by Kaushal. Debugging was done by Pratik, Kaushal and Hussain jointly.
- GUI – Hussain was the man behind the fancy GUI shown in the demo. It basically combined the PC with DSK, initiated transfers between them and displayed results in a graphical fashion which would make sense to a common man.

REFERENCES:

[1] Maldonado-Bascon, S., Lafuente-Arroyo, S., Gil-Jimenez, P., Gomez-Moreno, H., Lopez-Ferreras, F., (June 2002) "Road-Sign Detection and Recognition Based on Support Vector Machines," in IEEE Transactions on Intelligent Transportation Systems, pp.685-688, Tokyo, Japan.
(HSV implementation, Shape classification, DtB vectors, Hierarchical Classification)

[2] Shaposhnikov, D., Podladchikova, L., Golovan, A., Shevtsova, N., Hong, K., Gao, X., (2002) "*Road Sign Recognition by Single Positioning of Space-Variant Sensor Window",*
(Color Segmentation)

[3] Nguwi, Y., Kouzani, A., (June 2006) "A Study on Automatic Recognition of Road Signs", 2006 IEEE Conference on Cybernetics and Intelligent Systems, pp.1-6, Bangkok, Thailand.
(Color Segmentation, Dynamic pixel aggregation, Shape classification)

[4] Kaye, M., Krishnakumar, A., Stehle, L. (Spring 2002) "What's Your Sign? Improving Driver Safety with Road Sign Recognition", Pittsburgh, USA.
(Previous project done on the same topic)

(All C code written from scratch)