Traffic Sign Recognition
Senior Project Final Report

Jacob Carlson and Sean St. Onge
Advisor: Dr. Thomas L. Stewart

Bradley University
May 12th, 2008

Abstract - Image processing has a wide range of real-world applications from fruit harvesting to autonomous vehicles and beyond. The aim of this project is to create a MATLAB program that will identify a stop sign in various backgrounds and lighting conditions from static digital images. This processing could then output information to a theoretical autonomous vehicle, heads-up display, or other driver assistance device in the future. The software first uses color processing techniques to isolate relevant color data (red intensity) from the image. A variety of MATLAB Image Processing Toolbox commands are used to threshold, filter, detect edges, and further process the image. Morphological processing algorithms are applied in order to remove non-pertinent data and isolate the stop sign. Shape detection is used to determine if a stop sign is present in the current image (if any). Finally, any relevant sign is highlighted and output to the screen.

TABLE OF CONTENTS

I. INTRODUCTION

The objective of the Traffic Sign Recognition project was to identify a traffic sign from a digital photograph. The sign may be viewed from various angles and in many diverse background situations. The sign will then be highlighted after identification. All image processing will be done in MATLAB. Initially, the system will identify a stop sign. Additional signs could be added (yellow signs, etc.) as a continuing project.

II. OVERALL SYSTEM BLOCK DIAGRAM

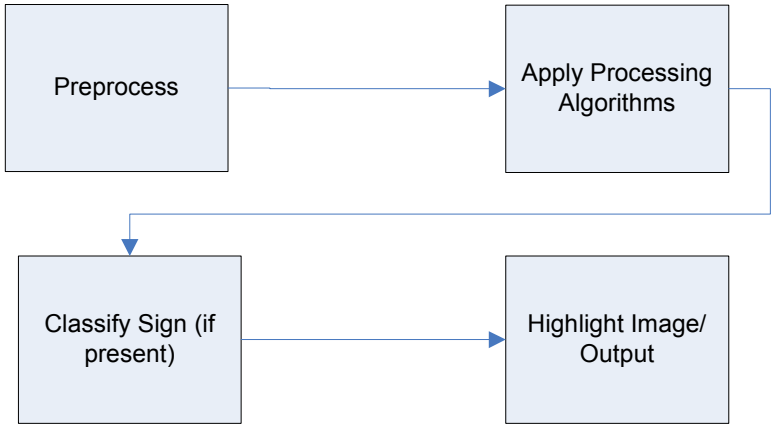

Fig. 1. Overall system block diagram

Input to the system will be an image loaded from the computer's hard drive. Preprocessing including contrast, brightness, clarity will then be performed. This preprocessing will be done outside of MATLAB code initially, with the ability to implement in the future. The actual image processing including color detection and edge detection will be applied next. The software will then attempt to determine if a sign is present. If present, the sign will be classified and highlighted. Figure 1 shows the overall system block diagram. To allow for future expansion, an action would need to be recommended to a hypothetical vehicle based on the nature of the sign observed.

III. FUNCTIONAL DESCRIPTION

A. Preprocessing Block

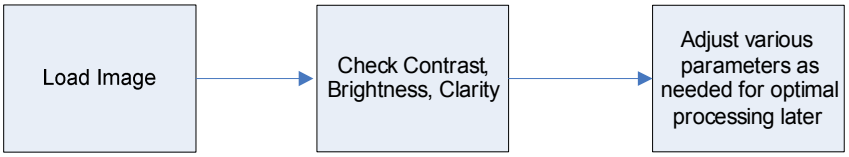

Fig. 2. Preprocessing block

Preprocessing will load the image as well as check contrast, brightness, and clarity. Figure 2 shows the flow of the preprocessing block. If these parameters are off from our desired values for these, adjustments will be performed. This will allow the design team to be able to ensure that the image is

suitable for processing. If the software cannot obtain the contrast or brightness needed, it may not be able to identify if there is a sign in the image.

B. Image Processing and Recognition

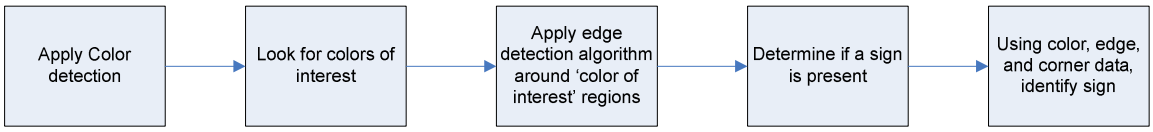

Fig. 3. Image Processing and Recognition (Apply Algorithm block)

This block will process the actual image, and is where the majority of this project is contained. First, the system will detect colors and then look for colors of interest. Red will be considered the first color of interest. Red is the most important type of sign (stop, do not enter, wrong way). Yellow is next most important (yield, crosswalk, curve). The image processing and recognition block is represented in Figure 3. The system will then define the region in which these colors are concentrated and outline the shape of the sign. If no sign is present, there will not be any sign to identify; nothing (or an error) will be output. With the data gathered from the image, the system will determine if a sign is present and will proceed identify it if possible.

C. Highlight/Output

This subsystem will take the data and create the final output. Output subsystem is shown in Figure 4 which shows the flow of the block. Without a sign identified, nothing will be output. If a sign has been identified the system will outline that sign and depending on the type of sign it is, generate a decision variable. This variable will be reserved for future use and could be implemented for an autonomous vehicle control at a later time.

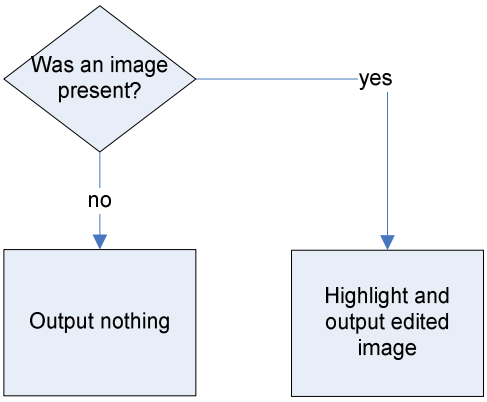

Fig. 4. Highlight/Output block

IV. FUNCTIONAL REQUIREMENTS

Preprocessing shall check contrast, brightness, and clarity. This block shall make sure the image is ready to have image processing done to it. After passing through this preprocessing block, the image shall be ready to have processing algorithms applied to it.

The application of processing algorithms shall take the preprocessed image and find colors of interest and look for shapes relating to the sign or signs we are searching for. This block shall find regions of interest on the image and these shall be further processed to obtain the type of sign. This is done in the following block.

The classify sign block shall take the regions of interest passed from the algorithms block. These regions shall be analyzed and used to compare to 'templates' of known signs. This allows for the system to identify exactly what sign is contained in the image that was processed.

The highlight image subsystem shall create some sort of distinguishing box or highlight around the actual sign.

V. SOFTWARE FLOW CHART

A software flowchart is shown in Figure 5. This is a high level look at what our program accomplishes. Each step is explained in the following software discussion (narrative).

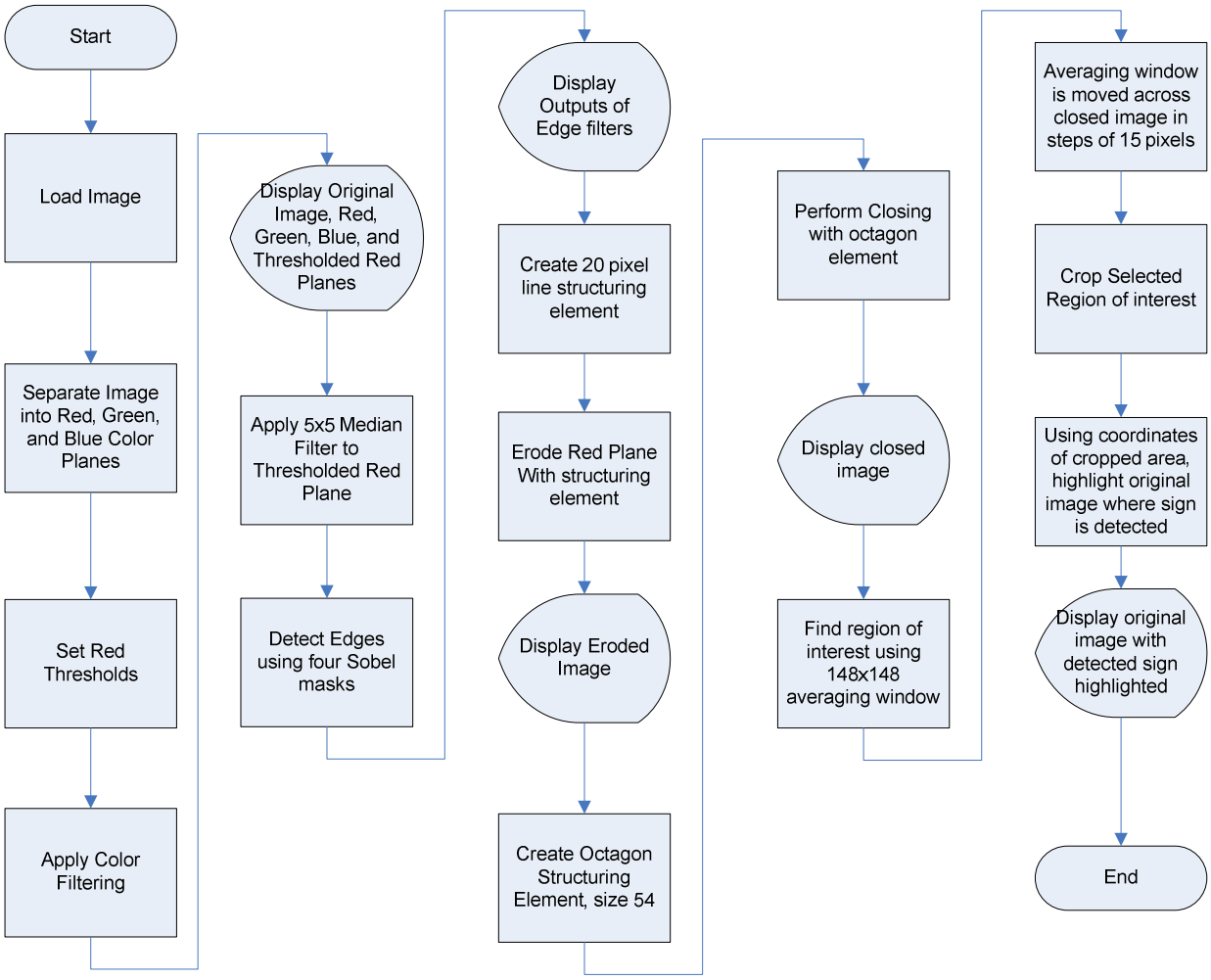

Fig. 5. Software Flow Chart

VI. SOFTWARE DISCUSSION

Our software first loads the image from the computer's local hard drive. This image has already been preprocessed manually. This makes the image able to be processed by our software. Next, the image is broken into its red, green, and blue color plains. Of these three plains, the red plain is of most interest to us for the initial task of locating a stop sign. Green and blue color plains contain little to no

data of interest. Now we want to threshold the red plain to further isolate the stop sign. Our threshold is set to 50 based on observation of a number of stop sign images. Any lower than that and a lot of noise is present after filtering. After the threshold image is calculated we display the original image, the three color plains, and the threshold red plain.

After threshold operation there will still be some noise. A median filter kernel of five pixels by 5 pixels is slid across the image. At each pixel the median of the pixels overlapped by the kernel (25 pixels total) is calculated. That value replaces the center pixel and the next pixel is computed and so on until the entire image is filtered.

Next we want to calculate the edges of the binary image that results after the filtering and threshold operations. This is essentially a differentiation of the image, finding edges where the image goes from black-to-white or white-to-black. This creates a sharp 'peak' and in the edge detected image only transitions are seen. This is accomplished using Sobel filter masks (shown in Figure 6). Four edge detections are completed horizontal, vertical, and both diagonal directions. The horizontal and vertical directions locate most edges, but for future consideration diagonal edges were also detected as certain signs and angles may require diagonal edge detection. These are displayed alone as well as summed and displayed for consideration.

Horizontal			Vertical		
1	2	1	-1	0	1
0	0	0	-2	0	2
-1	-2	-1	-1	0	1
Diagonal			Diagonal		
-2	-1	0	2	1	0
-1	0	1	1	0	-1
0	1	2	0	-1	-2

Fig. 6. Sobel Edge Detection Masks

After edge detection, as with any differentiation, noise is amplified. Because of this, any small pixels or group of pixels is outlined. To remedy this we construct a structuring element. This element is a twenty pixel long 'line'. This line is used to erode the image. Erosion moves this structuring element across the image. If the element overlaps 20 'white' pixels, the pixels stay. Any time the line does not overlap entirely erosion 'erases' the pixels. This gets rid of excess noise as well as the lettering in the sign which may interfere with further processing and identification. This is also displayed for the user.

Next, another structuring element is created. For this structuring element an octagon of size 54 is needed. This is allows us to close the image. Closing consists of morphological dilation followed by erosion; both processes use the same structuring element (octagon). Dilation takes the structuring element and replaces the pixels with values with the maximum of its neighbors. This is used to fill in the gap left by the erosion. It also helps fill in any problems around the edges of the sign. This result is displayed also.

At this point the image is fairly well filtered and cleaned up. This allows us to implement 'blog' recognition. Blog recognition basically looks for an area in the image with high intensity (white value). To accomplish this we slide a 148 pixel x 148 pixel window around the image and compute the average value contained within that window. This average is then compared to the highest average. If it is greater the coordinates are stored as the area of maximum value. After the averages across the entire image the coordinates of the maximum are used to crop the area containing the 'blob' out of the picture.

If a blog was recognized, the software then uses the coordinates to highlight the original image. This basically adds a small value to all of the pixels contained around the coordinates to brighten the region against the original background. An example is shown in Figure 7.

Fig. 7. Highlighted Stop Sign

VII. ANALYSIS OF RESULTS, CONCLUSIONS, AND FUTURE WORK

The goal of this project was to prove that a sign can be extracted from an image using MATLAB code. MATLAB is too inefficient for this to work in real time. A full size image (3072x2304 pixels) can take several minutes to process. Most 800x600 pixel images run in less than a minute on most systems. This speed issue can be fixed by using C or C++ code to implement the design laid out in this project. Doing some quick color processing

The code does a good job of isolating most red signs in most situations. Heavy snow or ice covering the sign inhibits the processing. Graffiti, bullet holes, trees, or other obstructions can also make the sign difficult to distinguish. The blob recognition, though it cannot draw correlation to an actual stop sign shape, is able to find signs that still have a good portion of red showing. Another improvement would be to actually compare the resulting 'blob' to a standard octagon.

We tested several ways of doing this during the last couple weeks of the project, but could not find a satisfactory method in that short amount of time. One method attempted was to find different characteristics of the blob and compare them to an ideal octagon. This method ran into issues because MATLAB sees octagons as being more 'circular' than circles (do not enter signs). This method would work, however, for distinguishing between squares or rectangles and circles or octagons. A better method would be to use the center of the blob to plot distances to the edges of the shape. This would allow easy correlation to an ideal octagon pattern and should be robust enough to handle different viewing angles.

We also authored some code attempting to isolate yellow signs. This was essentially done experimentally to validate our approach being viable for other applications other than red sign identification. Yellow signs are more difficult to isolate and the processing algorithm we designed would need quite a bit of optimization to be utilized.

REFERENCES

"Traffic Sign Detection in Static Images using Matlab", Garcia, M.A.; Sotelo, M.A.; Gorostiza, E.M. Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA apos;03. IEEE Conference Volume 2, Issue , 16-19 Sept. 2003 Page(s): 212 - 215 vol.2