License Plate Recognition System

Final Report

by

Arul Manickam (arul@andrew.cmu.edu)
Jeffrey T. Lin (jtlin@andrew.cmu.edu)
Mei Xiang Weng (mweng@andrew.cmu.edu)
Visarut Napaporn (visarut@andrew.cmu.edu)

of

Carnegie Mellon University
Department of Electrical and Computer Engineering
Digital Communications and Signal Processing Systems Design
18-551 Spring 2003 Group 8

Table of Contents

Introduction

Project Description:

We address the issue of automatic vehicle recognition through the scanning of its license plate. This can be very useful for law enforcement. A scenario where the system could be utilized would be an authentication system. This will help combat terrorist activities by detecting unauthorized vehicles attempting to gain access to a restricted zone. Another usage would be in toll booths and various checkpoints, where the system could help identify stolen vehicles. This is useful since one vehicle is stolen every 27 seconds in the U.S and the estimated value of all U.S. vehicles stolen in the year 2000 is $7.8 billion.

Our project is to design the license plate recognition (LPR) system to be installed in places of interest such as toll booths. In order to produce a vehicle's license plate characters as output, we will first have to take an image of the car and its license plate. From the photo, our system will first locate the actual region where the license plate appears. Next it will zone in and concentrate on processing that particular region, where the characters on the plate will be retrieved and recognized.

Past Projects:

There was a previous 18-551 project done by a group in Spring 2000 which also dealt with license plate recognition. At that time, they had a lower quality camera which allowed a maximum photo resolution of only 640x480, so this was the greatest resolution they worked with. The cameras we used had resolutions up to 2048x1536, but we standardized the photos in our data set with a 1600x1200 resolution. The past group also used methods different than ours in determining the location of the license plate and decoding the license plate characters. In the end, our LPR system works much more accurately and precisely than that of the previous group. Detailed comparisons of their algorithms and performance results vs. those of ours will be discussed later on.

System Design

Equipment:

We will be implementing our project on a Texas Instruments TMS320C6701 Evaluation Module (hereafter referred to as an EVM). For more information on this board please refer to: http://focus.ti.com/lit/ds/symlink/tms320c6701.pdf. Since we will be implementing our design on an EVM board, its limitations in terms of speed and memory have forced us to design our system so that we do not exceed its capabilities.

Database:

We captured our images using a digital camera at 1600x1200 resolution. The images were stored as full color JPEGs. The pictures were all taken in broad daylight, from a standard distance and angle. We did not deal with skew as this is not a serious problem in a practical setting as the camera will be in a fixed location. Since the angle and distance are known before hand, the image can be easily de-skewed before being sent into our system.

Initially, we captured 20 images to be used as our training set. However, we originally tried to deal with skew, so many of these images were at angles or rotated. In addition, some of them were taken at night. Since we decided it was unnecessary for our system to deal with skew and we didn't have enough time to take care of nighttime images, we cut down our training set to 8 images that were taken under the appropriate conditions.

Later, we acquired 22 images under the previously mentioned conditions to be used as our testing set. We used these images to measure the performance of our system.

Constraints:

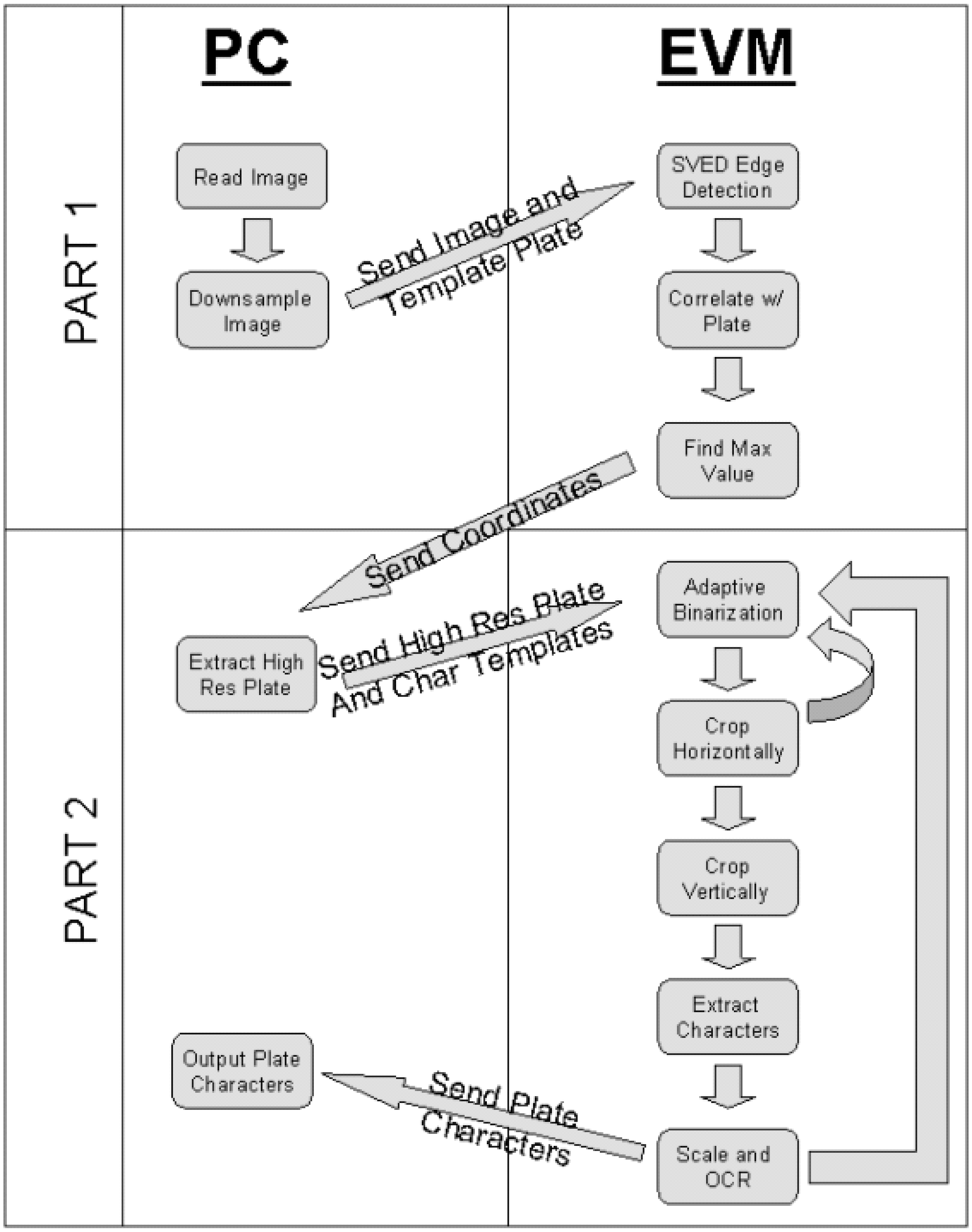
Originally we had hoped our system could be robust in that the license plates characters from most of the 50 states could be recognized, but we then decided to limit its ability to only a particular font style, since the fonts from state to state can vary greatly and would require us to get sample fonts from each state. Thus, we only recognize plates which use the Keystone font, which is the standard for Pennsylvania. We also included one plate from California in our training set and one from Virginia in our testing set since they had similar fonts.

We imposed a timing constraint on our system in that it should perform the license plate recognition under 10 seconds. We felt that this was the maximum practical time for a real life system such as a toll booth or entrance gate to a facility. As discussed in the results section, we not only met this goal but also cut the time in half.

Overview:

Our system is divided into two parts: the first part locates the license plate, and the second part reads the characters from the extracted plate. The PC of the computer where the EVM board is installed serves as the input/output for our system. It initially gets the image and sends a low-resolution copy to the EVM using a synchronous data transfer. The EVM then locates the plate and sends back its coordinates to the PC. The PC then extracts a high-resolution crop of the plate and sends it to the EVM. The EVM attempts to read the characters from the image and sends the PC the characters it thinks comprise the plate. Finally, the PC side outputs the plate characters. This process is outlined in the following schematic:

Flow Chart


PART 1
PART 2
PC
EVM
Read Image
Downsample Image
Send Image and Template Plate
SVED Edge Detection
Correlate w/ Plate
Find Max Value
Send Coordinates
Extract High Res Plate
Send High Res Plate And Char Templates
Adaptive Binarization
Crop Horizontally
Crop Vertically
Extract Characters
Scale and OCR
Send Plate Characters
Output Plate Characters

LPR System Part 1: Locating the License Plate

Reading Image:

The first step in Part 1 is to read the image on the PC side. This is done using the JPEG decoder [4]. The images were originally in color. We converted them to grayscale for our system.

Although our database consists of images at 1600x1200 resolution, we down-sampled the image to 800x600, since a high resolution is not required for the correlation of the license plate. Down-sampling our image helps speed up the filters and transforms implemented on the EVM.

In addition to sending the downsampled image to the EVM, the PC also sends an edge-detected image of a default license plate. This is used when the EVM tries to locate where the center of the plate is.

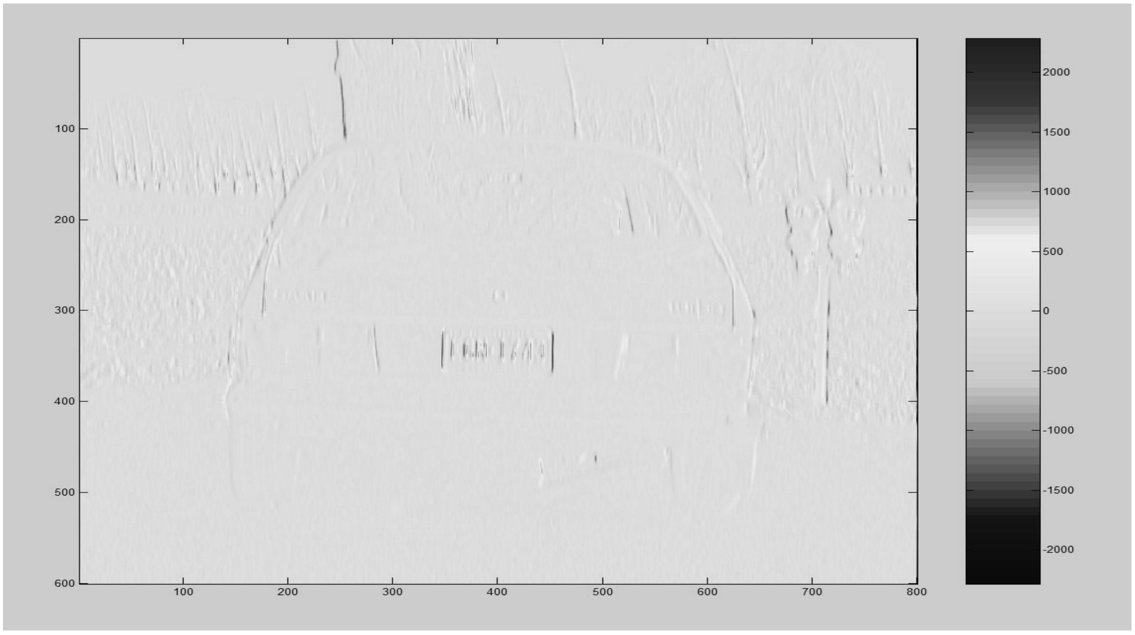
SVED Edge Detection:

To locate the possible regions where the license plate is, the EVM uses a special filter to scan through the entire image that would enhance areas containing short vertical edges, since the license plate is very likely to contain such edges, and the rest of the image is not. The filter we used is called the Short Vertical Edge Filter or Short Vertical Edge Detector (SVED) [2]. It consists of a 9x3 matrix where the values in the left column are -1's, middle column: 0's, and right column: 1's as shown on the next page:

Short Vertical Edge Filter:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

This filter differs from a traditional edge detector, such as a Sobel Filter, in that it picks up short vertical edges only and not horizontal edges. After correlating this filter over the image, we take the absolute value of the result to make sure both rising and falling edges are captured. At this point, the area around the license plate will be enhanced if seen on a gradient map, such as that shown below:



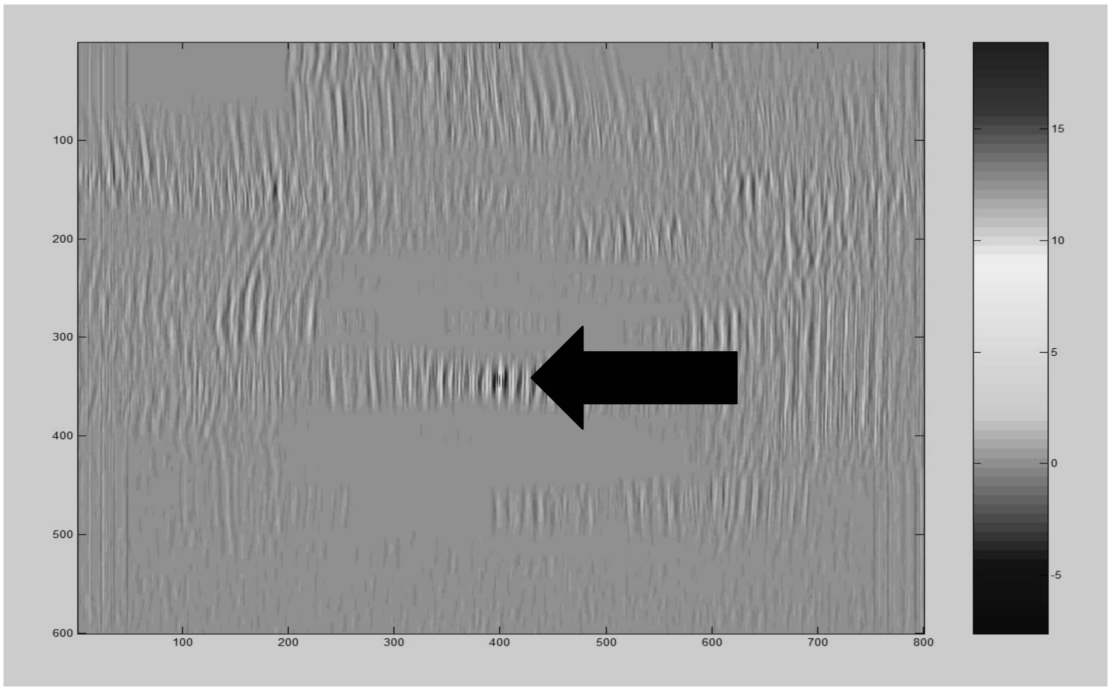
Output of SVED Edge Detector

Correlation with Plate:

From this we do a 2-D correlation with the template plate on the short vertical edge filtered image. Initially we did the correlating at 800x600 resolution, but that took a long time to perform. We then realized that even this downsampled resolution was not needed for the correlation, and we further reduced the image to 200x150 before performing the correlation. In addition, we shift by 4 in the X and Y directions since we do not need the exact center of the plate, but can be off by a bit since we extract more of the image than we need and crop it as described in Part 2.

We did not have much of an issue with this correlation technique since our images of license plates were pretty uniform in distance and camera angle as noted in our constraints, so there was little scale variance. Occasionally we had some false positives since there may be complex areas in the image that resemble the license plate characteristics, such as a rectangle in the background with many vertical lines. Sometimes tree branches in the image can cause this problem also. In addition, text elsewhere on the car such as on bumper stickers may lead to the same problem where our system may believe this is the region where the license plate exists.

Generally, however, the correlation works very well to pinpoint where the center of the license plate may exist. As can be seen in the image on the next page, the middle of the license plate region gave the greatest color intensity. This point will be sent from the EVM back to the PC as an (X,Y) coordinate to retrieve a high resolution section of this region to enter the second stage of our LPR system.



Output of Correlation with Template Plate

Comparison with Previous Group:

Our design with the use of correlation after performing SVED on the down-sampled image quickly limits the possible candidates as a license plate. The past group also performed correlation with a template plate to find the license plate region but they used Sobel edge detection as opposed to short vertical edge filtering. This gave them a lot of false positives. They had used an FFT on the x-projection of the image to eliminate the candidates that were not likely to contain characters, which did not work well. With our image enhancement using SVED we have achieved much higher accuracy, letting us avoid the need to use the somewhat flaky FFT method. They had reported only 73% accuracy in successfully locating the license plate using their methods. Our design gave us 91% success in locating the plate.

LPR System Part 2: Character Recognition

Extraction of High Resolution Image:

After the coordinates of the most likely license plate region is sent from the EVM to the PC, the PC will calculate the corresponding coordinates in the high resolution image. It will then send back to the EVM the cropped segment of the high resolution image at a standard size of 311 pixels wide by 81 pixels high. This size was appropriate for the distances we were using in our license plate (around 10 ft).

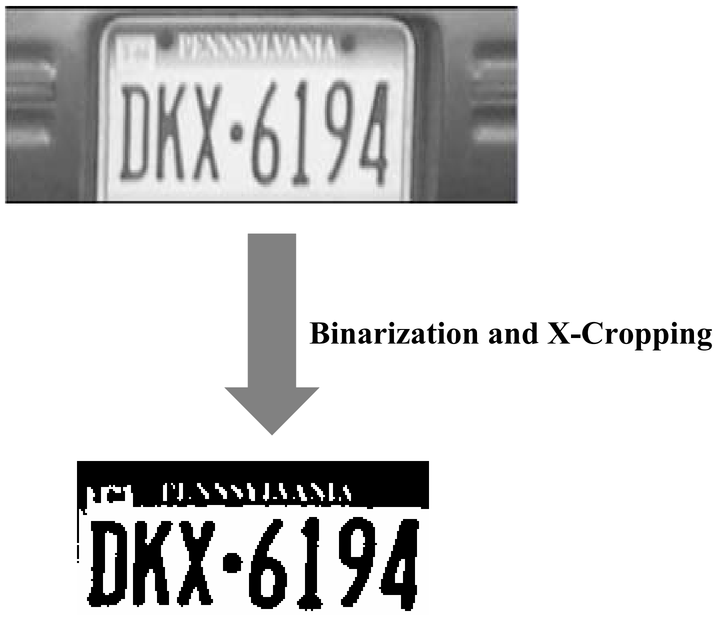
Adaptive Binarization:

We utilize a technique we call Adaptive Binarization to enhance letter contrast without introducing too much noise. We start out with a high binarization threshold, which is the number which determines what grayscale values are mapped to white and what values are mapped to black when we binarize. Thus initially, most values are mapped to black. We then perform the cropping of the plate, and as described below, if we are unable to do this successfully, we return to the adaptive binarization and lower the threshold. We perform this process until we can successfully crop the image (defined as being able to crop at least half the image width). Then we attempt to perform the Optical Character Recognition (OCR). If the number of characters is less than a minimum threshold (in our case this threshold is set to 7), then the adaptive binarization is performed again and the whole process is repeated until we read the minimum number of letters or reach a binarization level of 0.

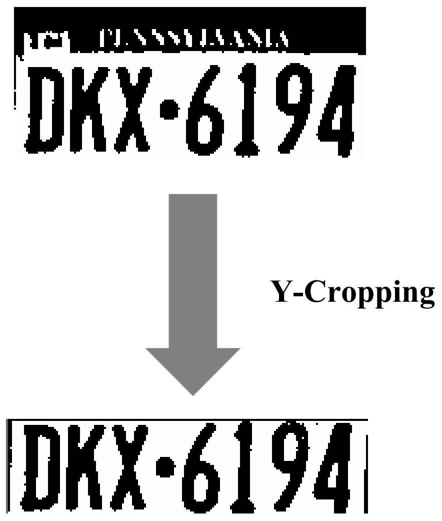
Cropping of Plate:

As described above, the next step after adaptive binarization is to perform a cropping of the image. The goal of this cropping is to isolate the letters in the plate from any background graphics which may be present. We proceed to do this in two stages. Stage1 (X-Cropping) crops the image horizontally and stage 2 (Y-Cropping) crops the image vertically.

To begin the X-Cropping, we start in the center of the image. We proceed to the left and right edges of the plate until we reach a column which is at least a certain threshold of black pixels (in our case we used the value of 90% based on our test set). If no column meets these requirements when proceeding to the left or right, we choose the corresponding edge of the image. Generally this procedure will remove the outside border of the license plate since the plate itself will be very light colored compared with the body of the car. Typical results of this process are as below:



The next step is to perform the Y-Cropping. For this stage, we do a Y-Projection of the image [1]. This is just a summation of the values in each row. We compute the mean value over all rows for the Y-Projection. Starting in the center of the image, we proceed to the top and bottom of the image until we reach a row that is below or above a certain threshold of the mean (in our case we used 20% based on our test set). We utilize this method since the y-projection of the rows containing the letters should be rather uniform, and immediately above and below these rows, there should be a sharp difference in the Y-projection since there needs to be white space above and below the letters. The procedure results in an output as shown below:



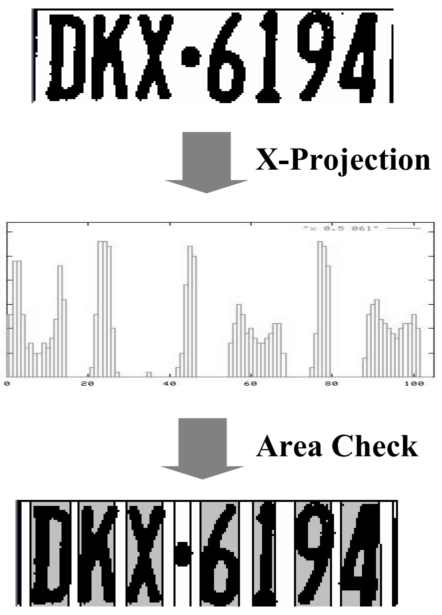
After these two croppings, we should get a rectangle encompassing the license plate characters where the height is approximately the height of the characters and the width about the width of all the letters. We arrived at this cropping procedure after experimentation with many different techniques and this one seemed to work the best. In general, it seems to perform successfully.

Segmentation:

The next step consists of extracting each of the characters in our cropped image. We begin by doing an X-projection [1]. The idea here is to use the spaces in between the characters to separate them. The X-projection values below the characters will be very different from the values below the spaces.

Starting at the left end of the plate, we proceed until we reach an X-projection above a certain threshold (in our case 3 black pixels based on our test set). We continue from this point till we reach a point where the X-projection falls below this threshold. We then extract the box whose horizontal segment begins at the first point and ends at the second point. The height is the height of the cropped image.

Taking this box, we calculate the number of black pixels it contains, and if this is below a certain threshold, we eliminate it (in our case, we used 200 black pixels as our threshold). This is useful for removing noise or items such as a hyphen, which is present in the center of the Pennsylvania license plate. This process is shown below for a typical image. The accepted boxes are shown in gray.



Next, using a bilinear scaling function [3], we scale the image proportionately to the height of templates we have for the letters of the alphabets and the digits 0-9. The width is scaled by the same amount as the height, but may not exactly match the template's width. In our project, the template letters are 18 pixels wide and 42 pixels high. We then attempt to perform Optical Character Recognition on these letters as described below. After the OCR, we proceed to get another letter by starting at the second point and repeating the above process. This continues until we reach the right end of the image.

Character Recognition:

For each box extracted above that met the threshold for the minimum number of black pixels, we attempt to recognize the letter. To do this, we perform a correlation between this box against all of the 26 template letters and 10 template numbers. Since the width between the box and the template could be different, we used the smaller of the two as the width of the correlation. The images were aligned at their centers.

We did not perform a direct correlation but rather looked at the difference between a pixel in the box and its corresponding pixel in the template. Taking the sum of these differences over the whole correlation range gave a likelihood number for each template. The template with the lowest number was chosen as the most likely candidate to match the box. After getting all the characters, the EVM would return them as a string to the PC, which would output them.

Comparison with Previous Group:

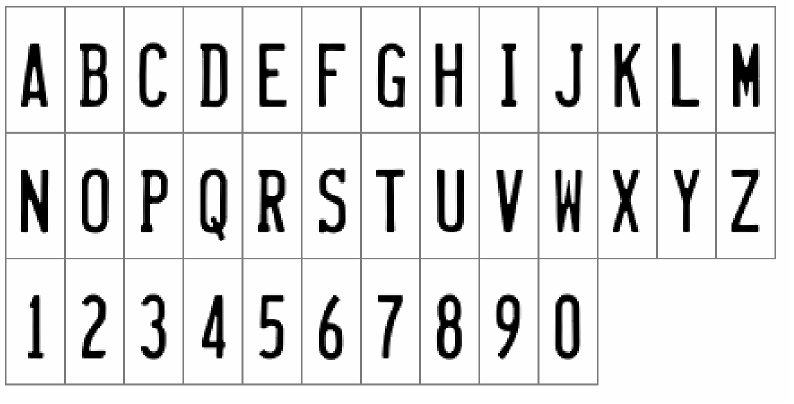
The previous group also used correlation with templates to perform OCR. They used different algorithms to extract the characters, however, and this led to some key performance differences.

Our first major difference was that we used adaptive binarization to increase image contrast. The previous group used histogram equalization to achieve the same goal. However, their system is only useful when the image's values are in a small range. If the image already has good contrast, their system will actually make the final character recognition worse, since polarization of the histogram is desired.

The second key difference is that we scaled our extracted characters to the size of the templates. Their project was extremely sensitive to distance. While our system is also sensitive to distance, this is not as much an issue as long as we can still locate the plate since we scale the letters.

Finally, their character segmentation algorithms did not always work. For example, sometimes the characters H and L would be separated, and sometimes there were extra spaces around the extracted characters. Our higher resolution along with our algorithms helped us achieve 96% overall character recognition rate. The previous group achieved this rate for numbers, but only got 70% letter recognition rate.

The templates that we used are of the font Keystone, which is common to Pennsylvania license plates. Originally we had used a font similar to Arial's type, but that gave poor results. The decision to use Keystone drastically improved the correlation results on the segmented boxes vs. the template characters. The templates that we tried are shown below:



Keystone Font – the template characters for our final design

ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

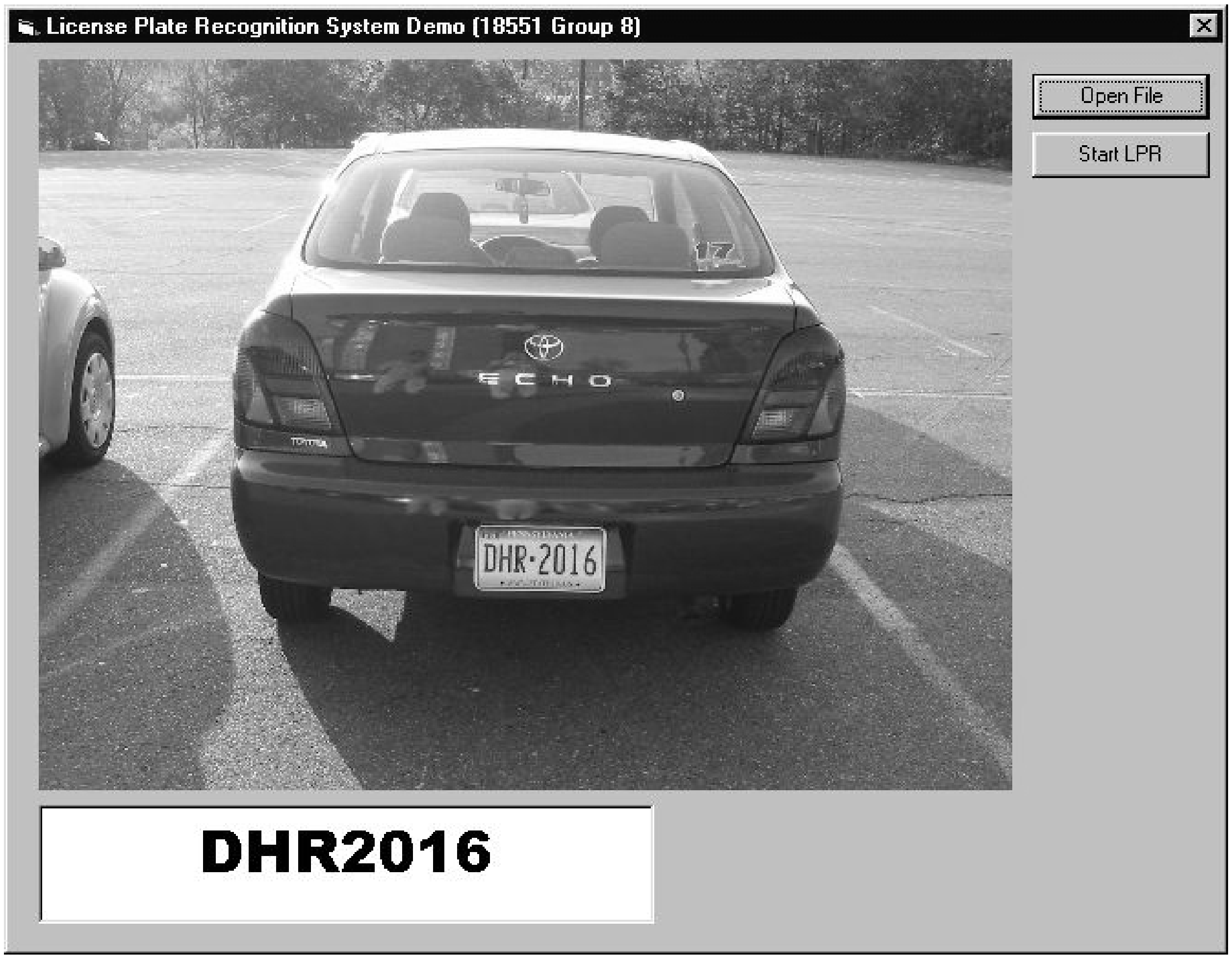
Arial-like Font – the template for our earlier design, not too accurate

The system is very font sensitive. To recognize plates from other states, more template fonts should be added to the system.

Graphical User Interface

To make the system more user-friendly, we decided to write a graphical user interface that will allow easy selection of images and streamline process without the need for Code Composer.

We develop the graphical interface in Visual Basic. It consists of a simple window with a panel for image display, a textbox for result output, and two buttons for opening images and starting the recognition process. Below is a screen capture of the graphical user interface:



Since the majority of our PC-side code is written in Visual C++, it's not feasible to port them all to VB. Instead, we compile the C++ portion into a dynamic link library (DLL) file.

The VB graphical interface would simply need to include the DLL file, and call InitEVM(), LPR(), and CloseEVM() at appropriate times to trigger the process; namely, it calls InitEVM() upon program start-up, CloseEVM() when the program exits, and LPR() along with the image to do the recognition. LPR() would return a string with the output once the recognition is completed.

In addition, the PC-side C++ code is changed to use the auto-load feature of the EVM to remove the system's dependency on Code Composer. To run the system, the user would simply need the VB executable, the DLL that contains the PC-side codes, and the compiled EVM .out file.

Results

EVM Statistics:

Timing Information (in cycles):

Overall(includes overhead):	**280,650,581**
First stage:	**256,662,626**
SVED:	**18,231,738**
Plate Correlation:	**214,537,875**
Find Max:	**513,989**
Second stage:	**15,453,808**
Readplate (OCR):	**11,430,923**

Memory Configuration:

Name	Origin	Length	Used	Attr
ONCHIP_PROG	00000000	00010000	0000e300	R X
SBSRAM_PROG	00400000	00014000	00000000	R X
SBSRAM_DATA	00414000	0002c000	00000000	RW
SDRAM0	02000000	00400000	00002710	RW
SDRAM1	03000000	00400000	00000000	RW
ONCHIP_DATA	80000000	00010000	00000d04	RW

As can be seen above, our total number of cycles on the EVM was 280,650,581. The previous group used 1,283,640,000 cycles. Thus we needed less than a fourth of their EVM time. Our real time running performance was approximately 5 seconds.

The bulk of our processing time was spent in the correlation routine. Initially, as described in Part 1's section, we performed the correlation on a 800x600 image. The program took approximately one minute to run. After reducing the size to 200x150, we improved the time by a factor of 16.

We compiled the code with full optimization and we personally performed some tweaks such as converting frequently used variables to global data, thereby having them stored in the on-chip memory and not having to be allocated for every routine. Due to time constraints, we did not pursue further optimization techniques such as loop-unrolling or memory paging. Loop unrolling would help increase the compiler's ability to parallelize our code. Since most of our memory is accessed off-chip, we incur the memory latency frequently. Thus, memory-paging would allow us to increase performance significantly.

Discussion:

As previously stated, our test set consisted of 22 images. Out of these, the plate was correctly located in 20 of them, giving us a plate location accuracy of 91%. The previous group reported an accuracy of 73%, giving us a marked improvement in this regard. For a detailed discussion of why this is so, please refer to the end of the section dealing with Part 1.

Out of the correctly located plates, we read 134/140 characters correctly, giving us an OCR accuracy of 96%. The previous group achieved this for numbers, but only got 70%

accuracy for letters. For a detailed discussion of the reasons for the differences in accuracy, please refer to the end of the section dealing with Part 2.

Possible Directions for Future Work:

There are a number of possible improvements that could be made to our project, which we did not implement due to timing constraints or lack of expertise. We list some of the ones we think could improve the project significantly:

- Expand the system to deal with different plates.
    - This could be done using more font templates.
    - Another option is to use a neural net to recognize the characters.
- Use more image processing to increase the number of correctly located plates.
    - Standard image processing techniques could be used, for example, to deal with problems such as glare; we only deal with contrast issues.
- Automatically detect plate angle and rotation.
    - This would remove the setup time of figuring out the angle and rotation of the camera.
    - We attempted to do this using a Hough transform, but were unable to do so as many lines other than that of the plate would often be identified as the main lines in the picture.

References

We looked online for algorithms that may be useful to our LPR system design. These are the sites that we found useful:

1) http://trident.mcs.kent.edu/~nochiai/thesis/menu.html - Provided information on X and Y-Projection of license plate characters, important in part 2 of our process where we determine the characters on the plate.

2) Object Detection by Template Matching, 2002 18-798 project by Juhasz, Patel, Wang, and Wang - We got the idea of a Short Vertical Edge Detector from this project.

3) http://www.manning.com/Kabir/Files.html - Provided code for the bilinear scaling function, which we use to scale extracted license plate characters to the size of letter templates used in correlation.

4) CMU's Computer Graphics course (15-462) - Provided code for decoding JPEG images into raw format.

All other algorithms we wrote ourselves as they were relatively straightforward and we could tailor them to our specific needs.