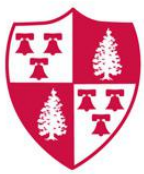

MONTCLAIR STATE UNIVERSITY

Road Sign Recognition System on Raspberry Pi

MS Project Report

Department of the Computer Science
Montclair State University
Montclair, NJ 07043

Enis Bilgin

Fall 2015

Advisor: Dr. Stefan Robila

Table of Contents

List of Figures:

Acknowledgements

First of all, I would like to take this opportunity to express my profound and deep regards to Dr. Stefan Robila for his guidance, and constant encouragement throughout the course of the project. The help and guidance given by him time to time shall carry me long way in the journey of my professional life which I am about to embark. One simply could not wish for a better advisor.

Beside my advisor, I would like to thank to; Dr. Constantine Coutras, Dr. Herman Dolezal for their insightful advices, and encouragement as well as Dr. Leberknight and Antoniou for their support as Graduate Project Committee members.

I would also express my gratitude to School of Business Technology Department (SBUS-Tech), who have instructed and supported me by any means throughout my formal education. I thank David Santos, the former SBUS Instructional Services Technology Assistant, who always made sure I am having access the latest technology equipment within the department, and also provided me direction as more of a mentor and friend than a manager.

Since to thank everyone can be interpreted as thanking no one, I especially want to single out a select group of three: Serkan Yavuz, Varsha Nimbagal, Nikita Panchariya. I am glad to count all of them as good friends. Serkan influenced my student life in a positive way. It was his idea to work on coding projects as team, including an android application, a modern furniture start-up website etc.

And behind everything on this journey, I have benefitted immensely from the support of my family. My parents exposed me to all kind of opportunities as I grew up and they were always encouraging when having a life altering decisions. While I could write their impact on my life for several more pages, I can just easily summarize in a single sentence: A kid could not ask for better parents.

Any future success that I may enjoy will be built on the educational foundation laid by all I mentioned above. I am eager to start paying everyone back, I will never forget the large debt I owe to those people and the society.

Enis Bilgin

Abstract

Digital image processing, i.e. the use of computer systems to process pictures, has applications in many fields, including of medicine, space exploration, geology and oceanography and continues to increase in its applicability.

The main objective of this project is to demonstrate the ability of image processing algorithms on a small computing platform. Specifically I created a road sign recognition system based on an embedded system, that reads and recognizes speed signs. The project report describes the characteristics of speed signs, requirements and difficulties behind implementing real-time base system with embedded system, how to deal with numbers using image processing techniques based on shape and dimension analysis. The system also shows the techniques used for classification and recognition.

Color analysis plays a specifically important role in many other different applications for road sign detection, this report points to many problems regarding stability of color detection due to daylight conditions, so absence of color model can led a better solution.

Neural networks are also widely used techniques in road sign detection and recognition. Additionally some other techniques such as template matching and classical classifiers were also employed, however in this project light-weight techniques were mainly used due to limitation of real-time based application and Raspberry Pi capabilities. Raspberry Pi is the main target for the implementation, as it provides an interface between sensors, database, and image processing results, while also performing functions to manipulate peripheral units (usb dongle, keyboard etc.).

Chapter 1 Introduction

The field of traffic sign recognition is not very old, with the first paper on the topic published in Japan in 1984when the aim was to try computer vision methods for the detection of objects. Since then however, the field has continued to expand at an increasing rate[23].

Traffic sign recognition is used to maintain traffic signs, warn the distracted driver, and prevent his/her actions that can led an accident. A real-time automatic speed sign detection and recognition can help the driver, significantly increasing his/her safety. Traffic sign recognition also gets an immense interest lately by large scale companies such as Google, Apple and Volkswagen etc. driven by the market needs for intelligent applications such as autonomous driving, driver assistance systems (ADAS), mobile mapping, Mobileye, Apple, etc. and datasets such as Belgian, German mobile mapping[3].

Methods of recognizing and detecting traffic signs continue to be published as the number of systems and tools to interpret images becomes more available on multiple platforms. In this project we used a mini embedded computer Raspberry Pi, that is capable of doing everything you would expect a desktop computer to do, from word processing, image processing to playing games. The idea behind using Raspberry Pi is mainly because its cheap price and requires minimal maintenance. The system has originally been developed by Raspberry Pi Foundation in an effort to give young people an easy solution to learn coding and computer programming. Raspberry Pi Foundation was founded in 2009, by a game developer David Braben, and supported by a tech firm, Broadcomm, and the University of Cambridge Computer Labs[2,6].

In order to provide fast processed results, this project aimed to demonstrate use the of simple shape recognition algorithms and open source optical character recognition (Tesseract OCR) on Raspberry Pi[1,2].

Tesseract OCR is an open source optical character recognition module for various operating systems. And its development is supported by Google since 2006. Tesseract OCR is one of the top character recognition engines in terms of accuracy. Tesseract can detect letters in various forms of images, and it uses the open source C library Leptonica library. In this project we will be able to pass images to Tesseract OCR and read them. To improve accuracy we had to do pre-processing on images before pass them Tesseract OCR engine[18,19,20]. The system can be scaled down to improve the conditions of highly automated driving systems.

The remaining of this report is organized as follows:

Chapter 1 Introduces the motivation and background of the project

Chapter 2 Explains the methodology and its requirements for the implementation

Chapter 3 Demonstrates how to setup the system

Chapter 4 Explains the concept of capturing and detecting speed signs

Chapter 5 Explains how to recognize speed signs

Chapter 6 States test results

Chapter 7 Provides conclusions and future work

Chapter 2 Methodology

2.1 System Overview

Speed sign recognition systems have a great potential to save time, money, and lives, and to improve our environment. Speed sign detection and recognition systems have a considerable promise future for commercial success. These systems can also be closely integrated with other major technologies, such as internet, mobile data services, artificial intelligence, automated cars. Speed signs define a new meaning of visual language interpreted by drivers and law enforcers such as traffic police, campus security etc. The systems determine the current road speed regulations on the road record them with GPS coordinates to claim legal recourse on speed limit violation providing information such as time and location.

Road and speed signs specifically have been designed to be recognized from natural and man-made background. They are required to meet certain criteria according to recent federal law changes. For example, stop and warning signs must be red with white lettering, all road signs must be larger than 18 inch width and 24 inch height with white and background black lettering. They are designed in 2D shapes such as triangles, rectangles, octagons and circles etc. The tint of the paint that covers sign should correspond to specific wavelength to make it in the visible spectrum. The problem is speed signs can be in any condition like distorted, damaged, clouded[7,14].

Figure 1. A Typical Speed Sign on a Road

Although most projects in road sign detection focus on color based algorithms, in this project shape detection techniques are used, because speed signs correspond a gray-scale background (close to white) and it is hard to extract the information from background. It is proven by many research groups that shape-based road sign detection can be efficient enough for preprocessing data. Systems that rely on color detection should be checked and adapted to various weather and light conditions, that can result in color and light intensity variation. Thus shape-based detection is chosen as a good alternative to color-based detection[4,5].

The use of shape information to detect speed signs also has certain difficulties. Similar objects to speed signs may exist in the same scene like walls, cars etc. Speed signs might be damaged, distorted by other objects or they may appear rotated vertically or horizontally. As the distance between a camera and the sign changes by the time, the scaling factor to recognize signs is important as well. When the sign is very small (it is too far from camera), it may be unrecognizable.

Another important concern is the speed of data processing. Since the data needs to be collected on a regular fashion. The pipeline of the system can be exhausted by image-capture flow. Working with shapes also required an edge detection and a matching algorithm as well[4].

2.2 Development of the System

To design a good recognition system, the system needs to have a good discriminative power and a low computational cost. The system should be robust to the changes in the geometry of sign (such as vertical or horizontal orientation) and to image noise in general. Next the recognition should be started quickly in order to keep the balanced flow in the pipeline of Raspberry Pi allowing for processing of data in real time. Finally, the optical character recognition engine must be able to interpret a pre-processed image into a text file.

Thus, to reach the required computational power, the following list of hardware and software requirements were used.

Hardware requirements:

- A low-cost embedded system hardware (Raspberry Pi 2 Model B, DC duino)
- A camera module for a Raspberry Pi 2 Model B
- A convenient global positioning system (GPS) module for the embedded system(Adafruit GPS breakout)
- Peripheral units (Keyboard, Mouse, HDMI input screen, Network Cable etc.)

Software requirements:

- A free Linux distributed Debian optimized operating system for Raspberry Pi 2 Model B
- Required Linux libraries (libavformat-dev, libgtk2.0-dev, pkg-config, libswscale-dev)
- A file transfer protocol (netatalk)
- A high level object oriented and structured programming language (Python 2.7) [20]
- An open source computer vision library for real time applications' efficiency (OpenCV 3.0) [8]
- An open source optical character recognition engine and a recognition library (Tesseract OCR, Leptonica respectively) [19]
- A path, an environmental variable within Raspbian OS, that tells the shell where image processing algorithms' executables located

A flow chart of speed sign recognition and detection is shown in Figure 2. The system can be implemented by shape recognition and K-Nearest detection. The identification of the speed signs is achieved by two main stages: detection and recognition. In the detection phase the image is pre-processed, enhanced, and segmented according to sign properties such as color, shape, and dimension. The output of segmented image contains potential regions which can be recognized as possible speed signs. The effectiveness and speed are the important factors throughout the whole process, because capturing images from video port of Raspberry Pi and processing images as they come into to the pipe should be synchronized[8,17].

In the recognition stage, each of the images is tested with K-Nearest algorithm according to their dimensions, which is an important factor to detect the speed signs, since we want to process images only once as they come, it also emphasizes the differences among the other rectangle shapes. The shape (rectangle) of the signs plays a central role in this stage.

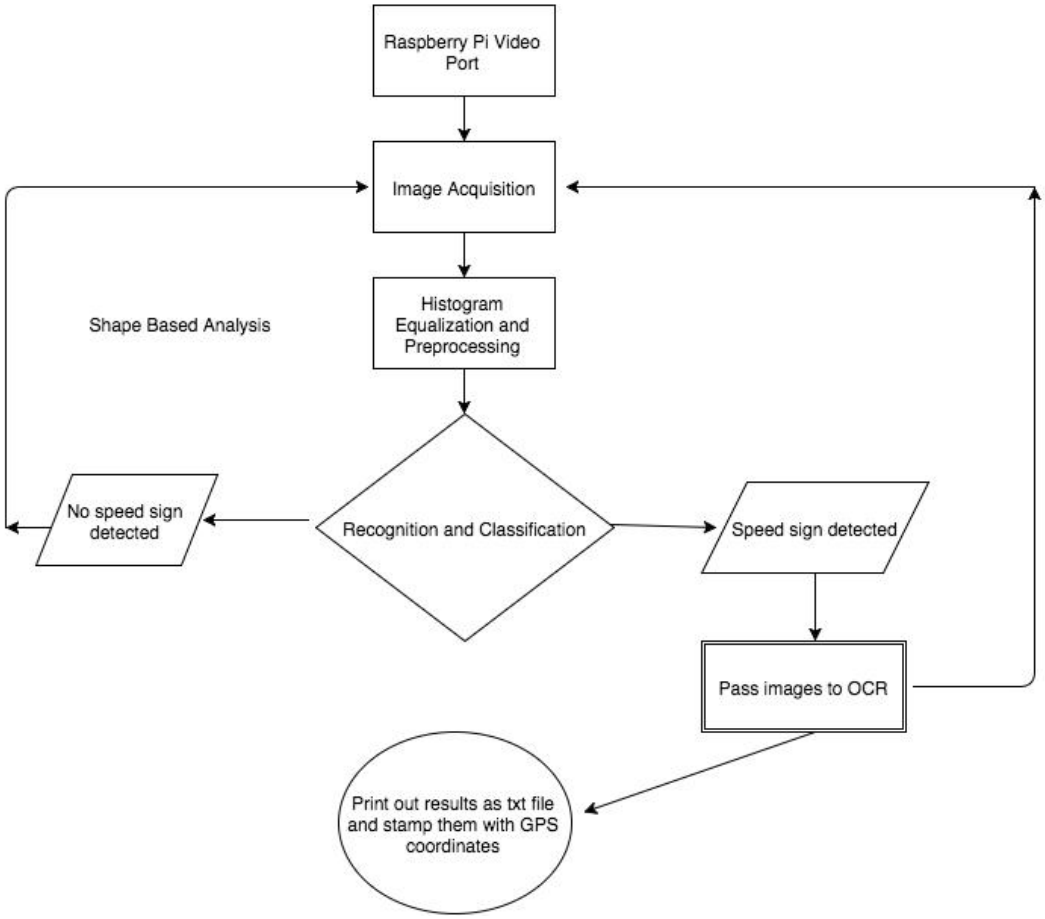

Figure 2. The speed sign detection and recognition flowchart

When a speed sign detected, it is passed into the optical character recognition engine to convert them and write out as a text file[20].

The GPS module is used to add to the results with time and location. To retrieve GPS data and write them into text file another open source platform Arduino-Uno used. This platform consists of both a physical programmable circuit board (micro-controller) and its integrated development environment (IDE). Arduino was used for this project because, unlike most complex programmable circuit boards, the Arduino does not need a separate piece of hardware to program instead, the only thing it needs a USB cable to load new code onto the board[16].

Chapter 3 Setting Up the System

3.1 Installing Raspbian on Raspberry Pi

A version of Snappy Ubuntu Core, is available as Ubuntu MATE for the Raspberry Pi. It can be downloaded from Ubuntu Mate official website[21]. We can also install all compatible versions of Linux on Raspberry Pi including Red Hat, Mandrake, Gentoo and Debian. But since in this project GPIO pins on Raspberry Pi were extensively used for camera module and USB WiFi-dongle, Raspbian OS is installed.

Raspbian is a Debian based Linux distributed de-facto standard operating system, which comes with pre-installed peripheral units libraries (GPIO, Camera Module). It is jointly maintained by the Raspberry Pi Foundation and community. It also has raspi-config, a menu based tool, that makes managing Raspberry Pi configurations much more easier than other operating systems, such as setting up SSH, enabling Raspberry Pi camera module etc[15].

Because of all reasons we listed above, Raspbian has been chosen for this project, with it's stability, and range of example projects within the community.

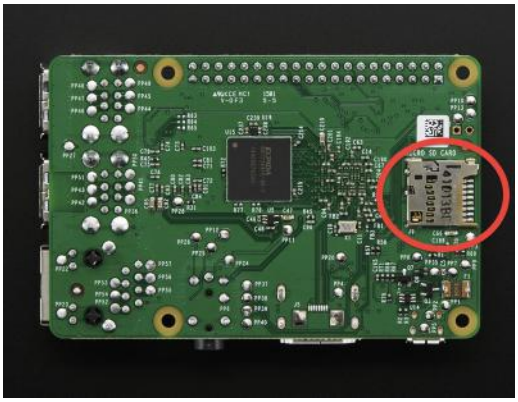

Figure 3. Raspberry Pi SD Card Slot

To install Raspbian to a Raspberry Pi we need to write the Raspbian image on SD card which mounted onto Raspberry Pi as shown in Figure 3. After we identify the SD card on our computer we can write a Raspbian image to SD card from the Linux command line. Then the Raspbian OS should be updated and upgraded in order to use latest camera functions available. The following commands were used:

```
sudo dd bs=1m if=image.img of=/dev/rdisk<disk# from diskutil>
sudo apt-get update
sudo apt-upgrade
```

Figure 4. Installation of the Camera Module

Finally, the camera module can be installed by basically inserting cable into the Raspberry Pi's camera module slot which is between Ethernet and HDMI ports(see Figure 4). After we connect Raspberry Pi to HDMI display and reboot, we can run the command from bash-script of the Raspberry Pi:

```
sudo raspi-config
```

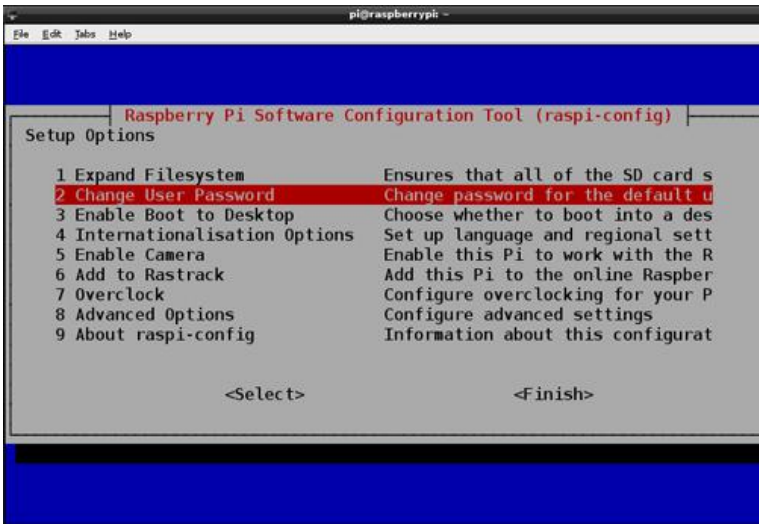

Figure 5. Raspberry Pi Configuration Screen

Next, from prompted screen (GUI for Raspberry Pi) we can enable camera options as seen in Figure 5. After navigating to the enable camera option the Raspberry Pi needs to be rebooted with the new

configuration, which enabled camera port. At this moment Raspberry Pi is ready to execute the commands for camera functions from Raspbian bash-script.

3.2 Raspberry Pi Wi-Fi Setting

In order to assure that the Raspberry Pi is connected to Wi-Fi properly, the setup needs to be done manually by changing content of /etc/network/interfaces file via command line. To ensure that we changed the wlan0 to the following.

```
allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

After the necessary changes for wlan0, the network name should be added to content of /etc/wpa_supplicant/wpa_supplicant.conf file as following.

```
network={
        ssid="M3IA1"  <network name>
        psk="BACDD5193" <password for the specified network>
        key_mgmt=WPA2-PSK <password protocol>
}
```

After these necessary steps are done, Raspberry Pi can be used headless (meaning no keyboard, mouse, or additional display) can be accessed via network connections and file transfer protocols[16].

3.3 Installing GPS module

In this project Dcduino (Arduino clone circuit) is used to pull out GPS data from Adafruit GPS module and the connections is shown in Figure 6 [16].

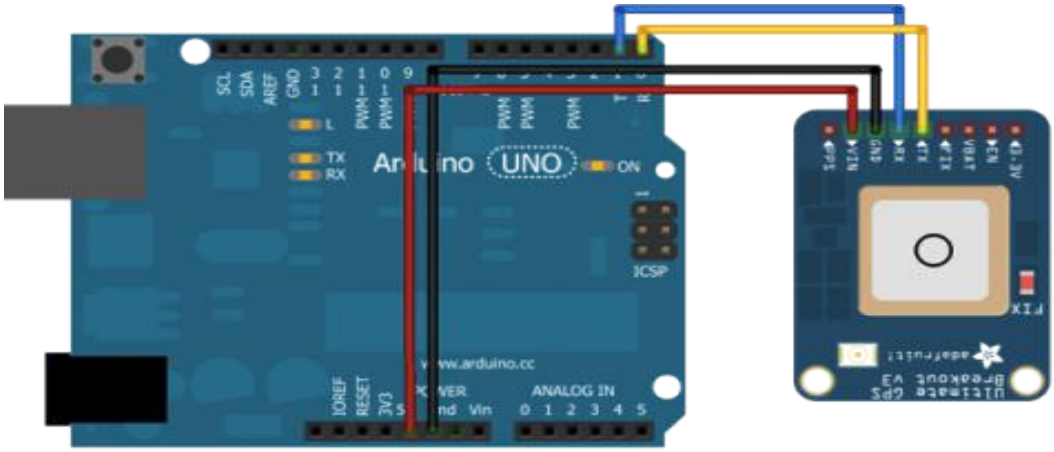

Figure 6. Adafruit GPS module connections

After Dcduino (Arduino-Uno Copy Circuit) connected to development environment (Users Machine) the Adafruit GPS library is installed in order to use GPS functionality. With the serial connection to our machine we can transfer data at rate of 9600 baud [16].

Chapter 4 Capturing and Detecting Speed Signs

In this chapter the algorithms, that are used to detect and extract the signs from the original image, are explained. To achieve better and faster results, OpenCV libraries are extensively used along with the Raspberry Pi camera module.

OpenCV was first created at Intel Labs by Gary Bradsky. In 2005 after OpenCV supported car project won the DARPA challenge, its development was sped up by Gary Bradsky and his team, and it now supports a multitude of algorithms related to Machine Learning and Computer Vision within a variety of programming languages including but not limited to C++, Python, Java etc. In addition OpenCV is available on multiple platforms including Linux, Windows, Android, IOS etc.

Since Python is a slower language than Java and C++, OpenCV uses C++ as main code and Python libraries to wrap up the code, which is running on the background. This has two advantages, first C++ code is getting executed faster than Python and second is we are able to code in easier to learn language(Python). OpenCV Python also makes use of Numpy(highly optimized library) which is specifically designed for numerical operations in MATLAB format. All OpenCV arrays are converted into Numpy to make variables easier to integrate other platforms such as Matlab etc[8].

As OpenCV is an open source development environment, people can contribute to its repositories by requesting pull from OpenCV development team on code management website Github. Furthermore contributors to OpenCV are extending OpenCV libraries by adding new algorithms and methods everyday.

4.1 Capturing Images

A Raspberry Pi is capable of capturing a sequence of images rapidly by utilizing its video-capture port with JPEG encoder. However several issues need to be considered before we use video-port:

- The video-port capture is only capturing when the video is recording, meaning that images may not be in a desired resolution/size all the time. (distorted, rotated, blurred etc.)
- The JPEG encoded captured images do not have exif information (no coordinates, time, not exchangeable).
- The video-port captured images are usually “more fragmented” than the still port capture images, so before we go through pre-processed images we may need to apply more denoising algorithms.
- All capture methods found in OpenCV (capture, capture_continuous, capture sequence) have to be considered according their use and abilities. In this project, the capture_sequence method was chosen as it is the fastest method by far.

Using the capture sequence method our Raspberry Pi camera is able to capture images in rate of 20fps at a 640×480 resolution. As in shown top of the Figure 10, there is a captured image by Raspberry Pi at chosen resolution.

One of the major issues with the Raspberry Pi when capturing images rapidly is bandwidth. The I/O bandwidth of Raspberry Pi is very limited, and the format we are pulling pictures makes the process even less efficient. In addition, if the SD card size is not large enough, the card will not be able to hold all pictures that are being captured by camera port, leading to cache exhaustion.

4.2 Multithreading in Capturing and Processing Images

Because of limited I/O Bandwidth of the Raspberry Pi, structuring the multithreading is an extremely important initial step of the pre-processing algorithm for images. To succeed this first we need to capture an image from video-port then process. Raspberry Pi maintains a queue of images and process them as the captured images come in. Most importantly, the Raspberry Pi image processing algorithm must run faster than frame rate of capturing images, in order to not to stall the encoder[17].

In addition, special care must be taken to ensure proper synchronization. Here the GIL (Global Interpreter Lock) is difficult to use in Python compared to low-level languages' multithreading. Python is an interpreted language thus, the interpreter is not able to execute code aggressively, because interpreter does not see the Python script as whole program. This means other than that the algorithm within the Python, to account for processing speed, we must rely on how fast interpreter works. Additionally developing a multi-threaded application becomes more complex rapidly in both developing and debugging compared to its single-threaded counterpart. In our schools we learn a mental model well suited for sequential programming that just does not match the parallel execution model. In this case the Global Interpreter Lock is really helpful to ensure consistency between the way of our thinking and between threads. Technical details about GIL can be researched by CPython repository[22].

```
class ImageProcessor(threading.Thread):
    def __init__(self): ...

    def run(self):
        # Separate Thread
        global done
        while not self.terminated:
            # Wait for an image to be written to the stream
            if self.event.wait(1):
                try:
                    self.stream.seek(0)
                    stream = self.stream
                    #Doing Some Processing on Images
                    #Then set done to True to exit
                    done=True

                finally:
                    # Reset the Stream and Event
                    self.stream.seek(0)
                    self.stream.truncate()
                    self.event.clear()
                    # Return ourselves to the pool
                    with lock:
                        pool.append(self)

def streams():
    while not done:
        with lock:
            if pool:
                processor = pool.pop()
            else:
                processor = None
        if processor:
            yield processor.stream
            processor.event.set()
        else:
            # When the pool is starved, wait for images to get into the pipeline
            time.sleep(0.1)

# Shut down the processors after each use
    with lock:
        processor = pool.pop()
    processor.terminated = True
    processor.join()
```

Figure 7. Multithreading Method in Python 2.7

The multithreading sample code snippet given in Figure 7 shows how the multiple threads are run. In the algorithm, threading is used for simple optimization. On the other hand multiple threads share the same data space with main thread. Due to the design of the Python interpreter, we need to make sure only one thread is running at a time. In order to accomplish this we need to create a lock, the beginning of the code a threading lock and a pool for processes are created. As we can see from code, the video streaming and processing running on different pools (different cores). After each use of the core for image processing, the pool has to be refreshed for the incoming tasks.

4.3 Detecting Speed Signs

When we look at the pictures of speed signs shown in Figure 8, the most defining feature of a speed sign is rectangular shape with mostly round edges.

Figure 8. A set of speed signs (src: graphicriver.net)

Before finding the rectangles in a captured image, we should retrieve contours, thus the sahpe detection algorithm employed loops through a subset of contours and checks if the contour shape is rectangle.

The shape detection is based on the OpenCV's Python implementation proceeded by filtering and edge detection.

4.3.1 Smoothing and Gaussian Filter

All the images captured from video-port have some amount of noise. To prevent the noise from being mistaken as edges, and produce wrong results, the noise must be reduced to certain level. Thus, the images are smoothed by applying Gaussian Filter. In a two dimensional space an isotropic Gaussian form F(x, y) equals to:

$$F(x, y) = \frac{1}{(2\Pi\sigma^2)} e^{\left(\frac{-x^2+y^2}{2\sigma^2}\right)} \quad (1)$$

The kernel of Gaussian Filter is so similar to mean filter, but the different kernel represents the shape of a Gaussian hump shown in Figure 9.

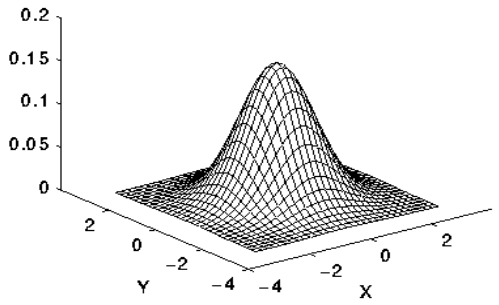

Figure 9. A Gaussian Distribution with mean (0,0) and σ = 0

4.3.2 Canny Edge Detection

The purpose of the canny edge detection is to significantly reduce the amount of data in the image by converting the image into binary format. Even though it is quite an old method, it has become one of the standard edge detection algorithms. In general, any edge detection should meet following criteria.

- **Detection:** Probability of real edge points should be maximized and connected, and the probability of falsely detected edge points should be minimized
- **Localization:** Detected edges should be as close as possible to the real edges in the image.
- **Number of Responses:** It should have simplicity, meaning that detected edges should not be resulted as more than once.

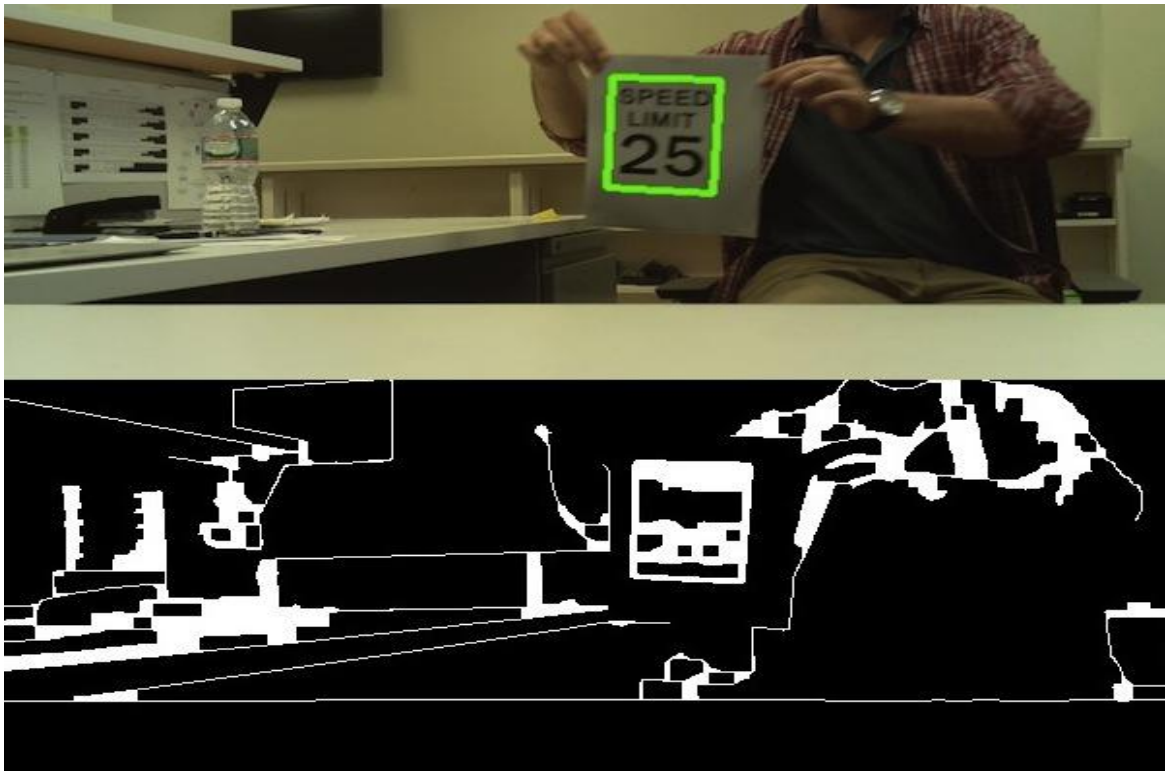

Figure 10. Sample image to demonstrate canny edge detection

As seen from Figure 10. in the bottom picture, all major edges have been detected and in the upper picture a rectangular shape is detected in certain size.
In this project Gaussian smoothing in the Canny edge detector performs two necessary steps: first it controls the amount of detail that appears in pre-processed image, second it suppresses noise in images.

4.3.3 OpenCV Contour Features

In doing image processing and especially shape analysis it is often required to check the objects shape, and depending on it perform further processing of a particular object. In our application it is important to find the rectangles in each of frames as these may potentially correspond to road speed signs. This shape detection must be done at least once in every 40 frames to ensure close to real processing. Once we check all the contours retrieved, we should look for closed loops then that closed loop should meet the following conditions to become a rectangle.

- It has to be convex
- It has to have 4 vertices
- All of its internal angles should be approximately 90 degrees

4.3.4 Bounding Rectangle and Contour Approximation

Contour approximation is a really important step for finding desired rectangles. Suppose we are trying to get a rectangle but because of distortions other issues in the image we do not have a perfect rectangle. Then the contour approximation might be in this case better choice for finding convex hulls. After this step we can approximate the rectangular shape in the captured image.

To compute a complex hull in OpenCV is a computational geometry problem. As formal definition the convex hull of S is the smalles convex polygon that contains all the points of S [24].

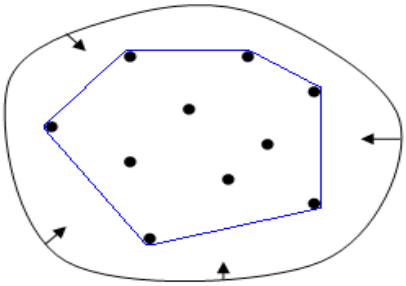

Figure 11. A Convex Hull

Length and area are simple characterizations of a rectangle. The next level of detail should be creating bounding boxes around the shapes.

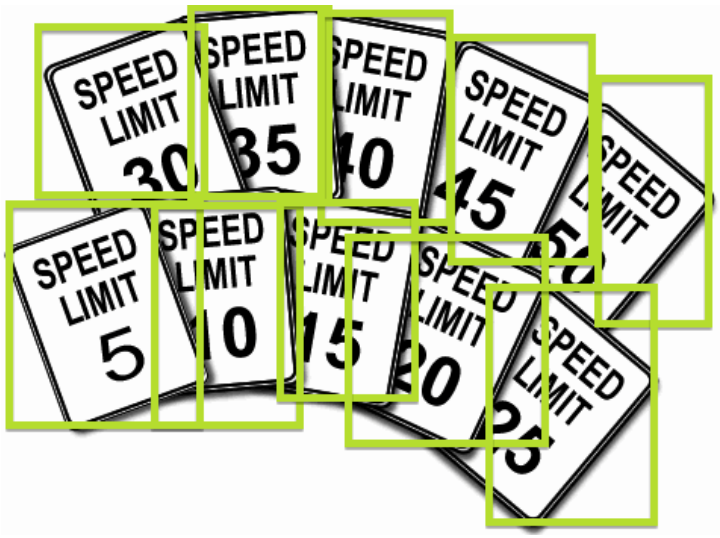

Figure 12. Bounding Rectangles for a Set Speed Signs, Generated by OpenCV cv2.boundingRect() function

After we have individual speed signs, speed signs can be recognized by OCR. Before speed signs passed to OCR we need to orientate the bounding rectangles seen in Figure 12. To do that first we find 4 max and min points in a rectangular shape. Once the speed sign has been orientated the number in the speed signs located in bottom of the sign. We look for this space to extract numerical data to read.

There are many special uses for scaling. In this project scaling factors are used for timing to decide when to process images. If the given sizes in the contour do not match the desirable limits, then the captured images are not passed to the digit recognition step making algorithm faster and more accurate.

Chapter 5 Recognizing Speed Signs

5.1 K-Nearest Neighbor Algorithm

It is mentioned in many of the image processing algorithms setting K to the square root of the number of training patterns and sample can help to have better results. K nearest algorithm is very easy to apply and it works very versatile with recognizing algorithms range from computational geometry to graphs[11].

K Nearest Neighbor algorithm is a non parametric, basic algorithm, that does not make assumptions on the underlying data distribution such as Gaussian mixtures etc. Since K Nearest Neighbor algorithm does not use the training data points, it can be considered as lazy algorithm, that means there is no special training phase or it is on very minimal levels.

Basically the K Nearest Neighbor algorithm assumes that the given data is in a feature space (geometrical metric space) and the points are in 2D geometrical space, where they have a notion of distance.

Each of the training data consists of a set of vectors and classes assigned to each vector. As we can see from the name of the algorithm K stands for number of neighbors that influences the classification. K is usually an odd number.

As it is mentioned before, each of the characteristics of a training set as a different dimension in space, and take that value for matching. And after the considerations of similarity in two points to be between that space and object we can define some features such as numbers, letters etc.

The way that the algorithm works, if the points from the training set to object are similar enough, then it will be considered as a new class, closest to K data points. To run the algorithm perfectly during the recognition phase, we need accomplish following steps.

- Assign random weights
- Calculate the classification error
- Adjust the weights according to error
- Repeat till acceptable level of accuracy is reached

$$\forall P \leqslant P \leqslant P * \left(2 - \frac{c}{c-1} * P\right) \tag{2}$$

According to the Bayes error rate P, c is the number of classes and P is the error rate of the nearest neighbor.

The result clearly states that if the number of points are fairly large then the error rate of Nearest Neighbor is less that twice the Bayes Error Rate. Also it is clear that the larger "K" means, the better accuracy we get.

In our project we used a set numbers to define the size limitations, and space features as we can see below Figure 13.

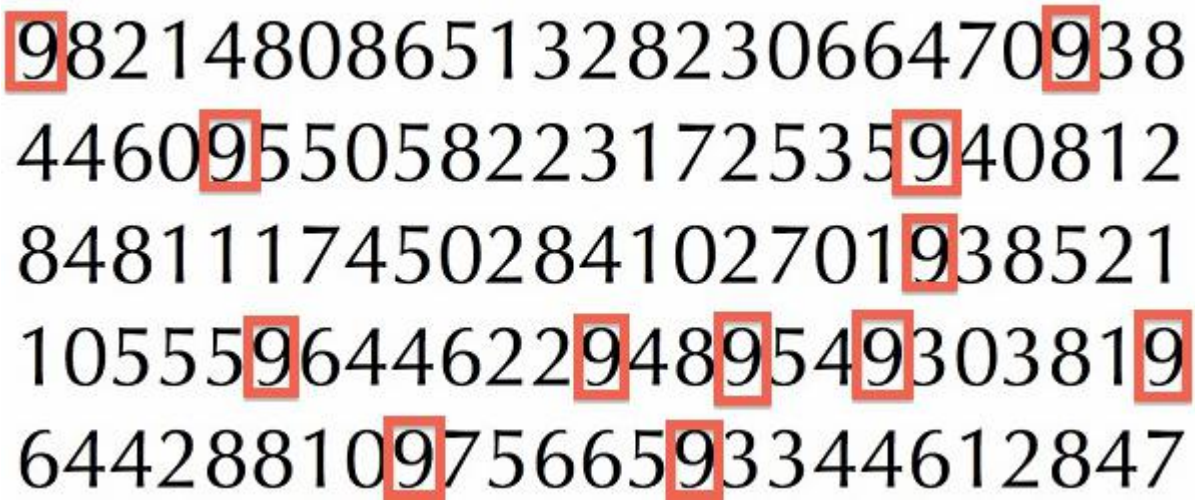

Figure 13. Sample Training Data for K-Nearest Algorithm

In OpenCV the algorithm does the followings:

- ➔ Loads the image
- ➔ Creates the instance of classifier for K-Nearest Neighbor
- ➔ Selects the digits one by one with contour approximation and bounding rectangles
- ➔ Once the corresponding numbers are found (in Figure 13, sample number is 9)
- ➔ Saves the pixel values into an array

5.2 Open Source Optical Character Recognition Engine

In order to read the speed signs accurately, Tesseract Optical Character Recognition(OCR) is chosen for the project. Tesseract OCR is and open source engine started as PhD research project at the HP Labs, Bristol. After a joint project with HP Bristol Labs and HP scanner division, Tesseract OCR was shown to outperform most commercial OCR engines.

The processing within Tesseract OCR needs a traditional step by step pipeline. In the first step the Connected Component Analysis is applied, and the outlines of the components stored. This step is particularly computational intensive, however it brings number of advantages such as being able to read reversed text, recognizing easily on black text on white background.

After this stage the outlines and the regions analyzed as blobs. The text lines are broken into

characters cells with the algorithm nested in the OCR for spacing.

Next, the recognition phase is set two parts. Each word is passed to an adaptive classifier, and the adaptive classifier recognizes the text more accurately. Adaptive classifier has been suggested for use of OCR engines in deciding whether the font is character or non-character.

Like most of the OCR engines Tesseract does not employ a template (static) classifier. The biggest difference between a template classifier and an adaptive classifier is, that adaptive classifiers use baseline x-heigh normalization whereas static classifiers find positions of characters based on size normalization.

The baseline x-height recognition allows for more precise detection and recognition of upper case, lower case characters and digits, but it requires more computational power in return[19,20].

The Tesseract OCR is quite powerful but at the same time it has the following limitations:

- Tesseract is unable to read rotated speed signs and digits (unlike other engines such as the one used by the U.S. Postal Service).
- Tesseract needs to preprocess to improve results. Images need to be scaled appropriately, the contrast should be adjusted and text should be aligned horizontally.
- Tesseract only works on Linux and Windows based machines. To use for another systems like iOS and Android a wrapper should be used for OCR.

To improve quality of results, the images ideally should be given to the OCR module in form of clear black text and white background. By default Tesseract OCR applies Otsu's thresholding method to every image, however since we have our custom pre-processing algorithm, this step was bypassed in order to improve speed. To disable internal thresholding of Tesseract OCR; the tesseract delegate option should be set as “self”(tesseract.delegate = self).

Chapter 6 Testing and Results

6.1 Accessing Raspberry Pi without additional hardware

Raspberry Pi with Raspbian operating system is just as good as a headless computer connected to a TV. Sometimes the system needs also to be used without a display. To accomplish that we need to first setup a file transfer protocol between Raspberry Pi and client's computer. In this project, the client/server file sharing, Apple Filing Protocol (AFP) is used to share files over the local network. First, installed Netatalk by booting up Raspberry Pi with additional screen, and from Raspberry Pi's bash-script typing:

- "sudo apt-get install netatalk"

That will install Netatalk and allows us to access Raspberry Pi after it boots up "headless".

To access Raspberry Pi; from Macbook's Terminal one can type:

- "open afp:// <IP Address of Raspberry Pi>
- Enter username and password for Raspberry Pi

Now the client is able to use Finder (or any other file manager on their computers) to transfer files to or from Pi, using the same techniques that would be used for linux distributed systems. A similar approach can be followed for Windows based systems.

6.2 Testing the Application (1st Experiment)

After we run the final Python script from bash-script manually, the application starts reading the road signs (which includes numbers) within an infinite loop. A total of six different speed signs are used to show to the Raspberry Pi camera module during testing. See figure 14 for examples.

To test the application the following steps should be taken:

- Change file directory to the path where the python script is stored;

  cd <path> (example: cd /usr/enisbilgin/pi/home/Finals)
- Run the application with the Python interpreter installed (version 2.7)

  python <application name> (example: python oneForAll.py)
- To halt the Python script manually, keystroke (Command+C) can be pressed

6.2.1 Comparing Test Results (1st Experiment)

Once the image processing of the sample image set is complete, the results can be compared to the actual speed signs we used for testing showed in Figure 14, that are printed on paper.

Figure 14. Sample stop signs used for testing

Many practical observations were gained while performing data collection and analysis. It was obvious that while the software was robust and capable of detecting speed signs, the corner of the speed signs (shown in figure 14. with red circles) were misinterpreted by Tesseract OCR in some cases as shown below Table 1.

Number of Signs	Time Stamp	Image	Result
S01	Mon Nov 16 16:57:00 2015	25	25
S02	Mon Nov 16 16:57:02 2015	25	25‘
S03	Mon Nov 16 16:57:04 2015	25	V25’.
S04	Mon Nov 16 16:57:06 2015	25	25
S05	Mon Nov 16 16:57:07 2015	80	80T
S06	Mon Nov 16 16:57:09 2015	80	80L
S07	Mon Nov 16 16:57:10 2015	80	80
S08	Mon Nov 16 16:57:12 2015	80	T80
S09	Mon Nov 16 16:57:13 2015	Changing	
S10	Mon Nov 16 16:57:15 2015	Changing	
S11	Mon Nov 16 16:57:16 2015	25	T25
S12	Mon Nov 16 16:57:18 2015	25	25
S13	Mon Nov 16 16:57:20 2015	25	25
S14	Mon Nov 16 16:57:21 2015	Changing	
S15	Mon Nov 16 16:57:23 2015	60	60L
S16	Mon Nov 16 16:57:24 2015	60	60
S17	Mon Nov 16 16:57:26 2015	60	Not Recognized
S18	Mon Nov 16 16:57:28 2015	60	Not Recognized
S19	Mon Nov 16 16:57:29 2015	No signs	
S20	Mon Nov 16 16:57:31 2015	No signs	

Table 1 Observed Values for Speed Signs (1st experiment)

Also the background information presented in speed signs were challenge for the accuracy of the system. In testing the application, the initial accuracy of the system was %35 (7 correct results out of 20 speed signs).The following issues were identified as being contributing factors:

- Field conditions: Issues related to sign location within the frame; Based on the scenery that surrounded a sign could effect the accuracy of system such as having multiple rectangular shapes on the background along with road signs.
- Physical obstructions: Surrounding objects and other physical obstructions can all contribute to reduce the visibility of speed signs.
- Geometry: Due to fixed distance interval at which images are captured and the fixed field of view for the camera, in certain cases of grade changes prior to the sign may cause the sign to be missed at least one of Input/Output streams.

6.2.2 Testing the Application (2nd Experiment)

The first examples of experimental results are presented in section 6.2.1. In that sequence at least more than six images' should have been recognized correctly (Table 1). Although most speed signs in Input/Output stream of Raspberry Pi are detected at the detection stage, there are some misinterpretations such as V25, T80, 60L etc. Misinterpretation usually occurs when the speed signs are rotated or the Tesseract OCR converts corner of the frames into letters.

After the cropping algorithm applied on pre-processed images, and adding a delay between the frames (delay between processing and reading images from stream) we had better results as shown in Table 2 below. We tested the system with 20 frames in each experiment, but there is a few lost frames, when we are switching speed signs manually.

The accuracy rate of the system (out of 17 frames) improved to:

- 53% → (9/17) speed signs matched with results completely (25-25, 55-55, 60-60 etc.)
- 70% → (12/17) speed signs matched with results approximately (45-43, 55-56 etc.)

Outcomes of the system:

- ✗ The algorithm which has been implemented is quite accurate but very slow, the total average time for both detection and recognition is ~1.5 frames per second (fps) on a 700 MHz Broadcom chip.
- ✗ The detection phase is based on the shape, and runs faster than recognition, that causes an increase of the images available for processing. To prevent errors caused by prior images, the processor should be stalled for a small time period.
- ✗ Because of shape based detection, the biggest issue is obstacles close to the speed signs
- ✗ Python-OpenCV is a wrapper around the original C/C++ code, which makes it simple and fast. But native Python script written functions that do not exist within OpenCV (our own functions in Python), reduces performance drastically.

Number of Signs	Time Stamp	Sign in the Scene	Result
S36	Mon Nov 30 17:40:13 2015	45	43
S37	Mon Nov 30 17:40:15 2015	25	25
S38	Mon Nov 30 17:40:17 2015	25	25
S39	Mon Nov 30 17:40:19 2015	25	
S40	Mon Nov 30 17:40:20 2015	60	60
S41	Mon Nov 30 17:40:22 2015	60	60
S42	Mon Nov 30 17:40:24 2015	Changing	
S43	Mon Nov 30 17:40:26 2015	55	55
S44	Mon Nov 30 17:40:28 2015	Changing	
S45	Mon Nov 30 17:40:30 2015	45	43
S46	Mon Nov 30 17:40:32 2015	60	0
S47	Mon Nov 30 17:40:33 2015	Changing	
S48	Mon Nov 30 17:40:35 2015	Changing	3
S49	Mon Nov 30 17:40:37 2015	45	43
S50	Mon Nov 30 17:40:39 2015	Changing	
S51	Mon Nov 30 17:40:41 2015	25	25
S52	Mon Nov 30 17:40:42 2015	25	
S53	Mon Nov 30 17:40:44 2015	Changing	
S54	Mon Nov 30 17:40:46 2015	25	25
S55	Mon Nov 30 17:40:47 2015	25	25
S56	Mon Nov 30 17:40:49 2015	Changing	
S57	Mon Nov 30 17:40:51 2015	Changing	
S58	Mon Nov 30 17:40:53 2015	55	56
S59	Mon Nov 30 17:40:55 2015	80	80

Table 2 Observed Values for Speed Signs (2st experiment, revised algorithm)

Chapter 7 Conclusions and Future Directions

Recently, many companies and research groups are involved in sign recognition research, and very good results have been achieved. In many applications road sign recognition systems use computer vision and artificial intelligence which is based on training with certain data sets. But even with improved techniques in the image processing field, road sign recognition may suffer from different problems like; blurred signs, fading of colors, disorientation, difference in shape and sign, uncontrollable lightning conditions, and occluded road signs by objects etc.

The task in this project is split into two parts, similar to other applications in the field, as “detection” and “recognition”. For detection part, shape-based algorithms used because color-based segmentation is much less reliable than shape-based segmentation. In similar cases to speed sign detection, they were many different techniques used, such as genetic algorithms, hough transforms, and artificial neural networks based algorithms. Some other researchers used colors and standard color spaces to find relations between them for improving quality, as lightning is a major issue since it varies for outdoor objects.

Despite the fact that color spaces using HUE, saturation like HSI and HSV are very common, there is also other color spaces like YIV, YUV were used. In some other projects, researchers went beyond that with using color spaces to develop databases, look-up tables etc.
The color-based applications vary from simple techniques to complex techniques (fuzzy, neural network), that shows it is not standardized as well. Thus, the color-based approach was not desirable for extraction of speed signs in this project [15,16].

Due to limited computation power of Raspberry Pi, complex techniques were not chosen despite the fact that there is many OpenCV libraries and examples including Eigen faces, SURF-SIFT template matching, and Fuzzy matching. Template matching is a good alternative to detect speed signs. It can be used to classify regions and can be combined with other techniques. In order to keep Raspberry Pi running smoothly, other classifiers like nearest neighbor, and euclidian distance were chosen for this project. Speed sign detection in different conditions like lighting, disorientation were not tested well.

A real front end traffic sign detection and recognition system must offer high performance with precise results, but also it should be real-time. In this paper we approached the problem mostly on shape characteristics of speed signs. Also the most important step in real-time speed sign recognition system is tracking captured images. For recognition, images should be pre-processed before passed to Optical Character Recognition Engine (Tesseract OCR), in that case captured images must be processed faster than capturing images, which is practically really difficult with today's technology. So, while running the application, the camera module and the processing algorithm should be synchronized, in order to keep things running smoothly.

By using a pre-built OCR engine, a detailed study of recognition techniques is outside the scope of this project. Detection, tracking and recognition are interwoven, while recognition reduces false positives and detection narrows the choices and increases accuracy of a system. Keeping the results is another important part, as a result, it needs to be improved as more databases are being developed by the time.

A comparison of the performance within an embedded system of this project will provide the baseline of the improvements. However, the lack of an open source evaluation framework for similar systems (datasets for speed signs, labelled data etc.) makes it hard to perform that comparison. This problem should be solved in future. I think the complexity of traffic sign recognition systems will be shrink in the near future as embedded technology advances. With the advancement of technology more and more embedded systems will available for developers and every car is equipped with a high-resolution camera and GPS receiver.

Several ways of reading speed signs were tested during the last weeks of the project, but could not find a solid method without exhausting the pipeline of the Raspberry Pi, in that short amount of time. Scale Invariant Feature Transform (SIFT) template matching method generates a large number of key-points described by objects location, orientation and scale as well as a series of weighted local gradient data based on histograms. Even though SIFT features are highly distinctive and invariant to objects scaling and rotation, it is a really expensive process within a small embedded system. And when it is applied within the capture-process algorithm, frame rates per-second (fps) for reading images, drops down drastically (less than 1 fps).

This project implemented as real-time video processing, for future work, the use of car's dynamics (direction, trajectory, speed changes etc.) should be considered to improve the system's robustness of the speed sign reading process. Speed sign recognition system would be useful as a driver-assistance system, it also has other possible applications, such as database for locations and limits of signs, building and maintaining maps of signs, automated inspection to provide better maintenance and safer signposting. All of those listed improvements need challenging research works in the near future.

References:

[1] L. Fletcher, N. Apostoloff, L. Petersson, and A. Zelinsky, "Vision in and out of vehicles," IEEE Intelligent Systems, Jun. 2003

[2] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, "Towards fully autonomous driving: Systems and algorithms," in Intelligent Vehicles Symposium (IV). IEEE, 2011

[3] "Mobileye Project".[Online]. Available.http://www.mobileye.com/ Accessed October 4th 2015

[4] Benallal, M., Meunier, J. "Real-time color segmentation of road signs" Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering (CCGEI). 2003

[5] Ritter, W., Stein, F., Janssen, R., "Traffic sign recognition using color information" Math. Comput. Model. 22(4-7), 149–161. 1995

[6] Bahlmann, C., Zhu, Y., Ramesh, V., Pellkofer, M., Koehler, T., "A system for traffic sign detection, tracking and recognition using color, shape, and motion information" Proceedings of the IEEE Intelligent Vehicles Symposium, pp. 255–260. 2005

[7] Paclik, P., Novovicova, J., Pudil, P., Somol, P., "Road signs classification using the laplace kernel classifier. Pattern Recognittion Letter" 21(13-14), pp. 1165–1173. 2000

[8] "OpenCV Documentation". [Online]. Available. http://opencv.org/ Accessed November 10th, 2015

[9] "Road and Traffic Sign Recognition and Detection". [Online]. Available. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.2523&rep=rep1&type=pdf Accessed December 4th, 2015

[10] A. de la Escalera, J. Armingol, and M. Mata, "Traffic sign recognition and analysis for intelligent vehicles," Image and Vision Computer., vol. 21, pp. 247-258, 2003

[11] "K Nearest Neighbors Algorithm". [Online]. Available. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm Accessed November 5th, 2015

[12] Belongie S, Malik J, Puzicha J "Shape matching and object recognition using shape contexts", IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(4), pp.509– 522, 2002

[13] "Scale Invariant Feature Transform". [Online]. Available. http://www.scholarpedia.org/article/Scale_Invariant_Feature_Transform Accessed December 1st, 2015

[14] M. Bicego, A. Lagorio, E. Grosso and M. Tistarelli. "On the use of SIFT Features for face authentication. Proc. Computer Vision and Pattern Recognition" (CVPRW'06) 35-35. 2006

[15] "Raspberry Pi Foundation". [Online]. Available. https://www.raspberrypi.org/ Accessed October 29th, 2015

[16] "Adafruit Industries on GitHub". [Online]. Available. https://github.com/adafruit Accessed November 6th, 2015

[17] "Concurrency and Parallelism". [Online]. Available. http://www.toptal.com/python/beginners-guide-to-concurrency-and-parallelism-in-python Accessed November 20th, 2015

[18] R. Smith, "An Overview of the Tesseract OCR Engine". [Online]. Available. http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/33418.pdf Accessed December 2nd, 2015

[19] "Tesseract OCR API", [Online]. Available. https://code.google.com/p/tesseract-ocr/wiki/APIExample Accessed November 16th, 2015

[20] "Pytesseract python wrapper for Google's Tesseract OCR", [Online]. Available. https://pypi.python.org/pypi/pytesseract/0.1 Accessed November 16th, 2015

[21] "Ubuntu Mate Website".[Online]. Available. https://ubuntu-mate.org/about/ Accessed December 14th 2015

[22] "Cpython Repository on Github".[Online]. Available. https://github.com/python/cpython Accessed December 14th 2015

[23] R.M. Gray, "Vector Quantization" IEEE ASSP Magazine April, 1984

[24] "Convex Hull", [Online]. Available. http://www.dcs.gla.ac.uk/~pat/52233/slides/Hull1x1.pdf Accessed December 15th 2015