PALESTINE POLYTECHNIC UNIVERSITY
College of IT and Computer Engineering
Department of Computer Engineering

Graduation Project

Project Name

Microcontroller-Based Driver Warning System

Project Team:

Oday Bkerat **Yousef Ashhab**

Supervisors

Eng. Mazen Zalloum **Dr. Hashem Tamimi**

Hebron-Palestine

May-2015

Graduation Project

Project Name

Microcontroller-Based Driver Warning System

Project Team

Oday Bkerat **Yousef Ashhab**

Supervisors

Eng. Mazen Zalloum **Dr. Hashem Tamimi**

According to the system of the College of IT and Computer System Engineering, and to the recommendation of the Project Supervisor, this project is presented to Computer System Engineering Department as a part of requirements of B.Sc. degree in Computer System Engineering.

Hebron-Palestine

May-2015

Signatures:

Project Supervisors signatures

…………………………...

………………………….

Testing Group signature

………...…..……….   ……………..………..…….

Department Headmaster signature

………………………………….

Acknowledgment

It is great pleasure for us to acknowledge the assistance and contributions of our teacher, department staff, college staff, and university staff whom helped us to be here.

We would like to thank our supervisors, Eng. Mazen Zalloum and Dr. Hashem Tamimi, for their assistance, ideas, and feedbacks during all stages of this project. Without their guidance and support, this work could not have been completed on time.

We would like to express our special thanks to Dr. Yaqoub Ashhab for his helpful comments and to Mr. Ameer Al-Qasrawi who helped us to buy the microcontroller from Munich.

Abstract

The problem of pedestrian-vehicle crashes is a major cause of deaths and injuries in road accidents worldwide. The major factor of such crashes is driver. However, there is no effective solution for this problem. In this project, we propose a microcontroller-based pedestrian detection prototype system for this problem. Our approach is based on image processing and machine learning methodologies to detect and classify images with or without pedestrians. The prototype contains a camera that can be installed on the car's dashboard, a microcontroller for computational processes, and a speaker for alarming the driver. The first phase of this project is building a binary classifier, which can predict if the analyzed images contain a pedestrian or not. The Histogram of Gradient (HOG) was used as feature descriptor to encode the processed images. An initial Support Vector Machine (SVM) classifier was built and trained on 250 positive and 250 negative non-redundant images. A 5 fold cross validation was showed a very high accuracy of the created classifier. In the second phase, we tested our trained model on real-time images. Our system achieved a moderate accuracy and a processing speed of 7 frame per second. The accuracy of the system is dependent on the number of training images. In conclusion, we have shown that microcontroller can be used to successfully implement an SVM pedestrian classification system that can be further optimized to a prototype product. To our knowledge, this is the first attempt to implement a pedestrian detection system using a Raspberry-Pi microcontroller.

Table of Contents

Chapter One: Introduction..4
1.1 Overview and Motivation..4
1.2 Causes of Traffic Accidents..7
1.3 Project Rationale...8
1.4 Goal and objectives...9
1.5 Project Importance..9
1.6 Short description of the system...10
1.7 Report Structure..10
Chapter Two: System Analysis..11
2.1 General Description...11
2.2 List of Requirements..11
2.3 Expected Results..12
2.4 Constrains...12
Chapter Three: Background..13
3.1 Theoretical Background...13
3.2 Hardware Parts Background..17
Chapter Four: System Design..20
4.1 Hardware Design...20
4.2 Software Design..22
4.2.1 Training Software Design...22
4.2.2 Testing Software Design...24
Chapter Five : System Implementation..26
5.1 Dataset Used...26
5.2 Hardware Implementation..27
5.3 Software Implementation...28
Chapter Six: Validation and Testing..30
6.1 Validation..30
6.2 Testing..31

Conclusion...32
References...33
Appendix...34

List of Tables

Table 1: Major causes of deaths..4

Table 2: The most common reasons of traffic accidents………..8

Table 3: Fivefold cross validation results..30

List of Figures

Figure 1: Distribution of traffic accidents in governorates of West-Bank…....…………...................5
Figure 2: Comparison of death rates for different road user categories..6
Figure 3: Major causes of road traffic accidents...7
Figure 4: steps of HOG extraction..13
Figure 5: HOG feature visualization...14
Figure 6: Histogram of Oriented Gradient explanation...14
Figure 7: Example of before classification and after classification using SVM...............................15
Figure 8: Optimal hyper-plane..16
Figure 9: Logitech C310 Camera..17
Figure 10: RaspberryPi...18
Figure 11: Audio speaker...19
Figure 12: System diagram..20
Figure 13: Connecting camera with microcontroller..20
Figure 14: Audio Jack port on RaspberryPi board...21
Figure 15: USB car adapter...21
Figure 16: Training algorithm flowchart...22
Figure 17: Testing algorithm flowchart..24
Figure 18: INRIAPerson dataset...26
Figure 19: Hardware implementation..27
Figure 20: Training algorithm implementation...28
Figure 21: Detecting algorithm implementation...29

Chapter One: Introduction

1.1 Overview and Motivation

While less than 0.5 million people are killed annually due to wars, military conflicts and homicide, traffic accidents stand behind the loss of 1.24 million lives worldwide (**Table 1**). Traffic accidents have devastating consequences on all levels of society, including the direct accidents' victims and their families. According to World Health Organization (WHO), the majority of traffic accidents' victims are from two age groups: males of 20-40 years whom are mainly involved in car crashes and children less than 15 years whom are the victims of vehicle-pedestrians crashes [1]. There is a serious concern about the economic impact of traffic accidents. In developing countries, the bill of traffic accidents can reach to a value equivalent to 3% of their Gross National Product (GNP) [2].

Table1: Major causes of deaths

Cause	World*	United States*
Traffic Accidents	1,240,000	33,561
Homicide	437,000	16,259
Military conflicts	55,000	127

** 2010 estimates for world, 2012 for the United States [3]*

It is interesting to note that while underdeveloped and developing countries have less than 50% of the global number of vehicles, more than 80% of the world's road deaths occur in these countries [1]. The highest chance of dying in a road traffic accident is for peoples living in Africa where it can reach up to 24.1 deaths/ 100000 inhabitants per year. Unfortunately, the Eastern Mediterranean countries, including Palestine, come in the second worst position with 21.3 deaths / 100000 inhabitants per year. It has been estimated that road traffic accidents costs the economy of Arab countries around 2.5% of their GNP [1].

The figures of road traffic accidents in Palestine are not much better than other Arab countries. According to the annual report of the Ministry of Transportation-Higher Council of Traffic, a total of 7827 traffic accidents were reported in the West-Bank, Palestine during 2013 [4]. **Figure 1** shows the distribution of these accidents on the different governorates. When comparing Palestine with neighboring countries in terms of road accidents statistics, the most noticeable difference is the high rate of deaths among pedestrians. For example, in Jordan and Lebanon 33% of road accidents' deaths are pedestrians whereas the rate in Palestine is 53% (**Figure 2**).

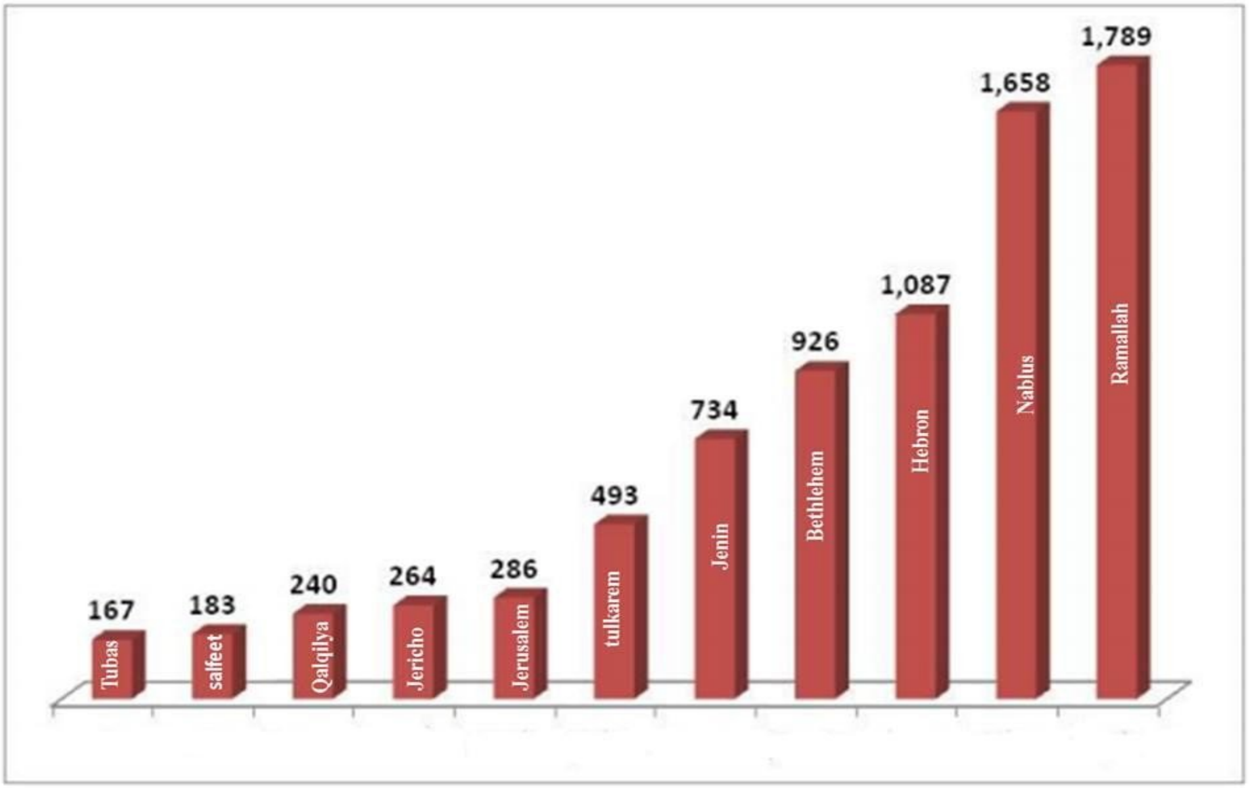
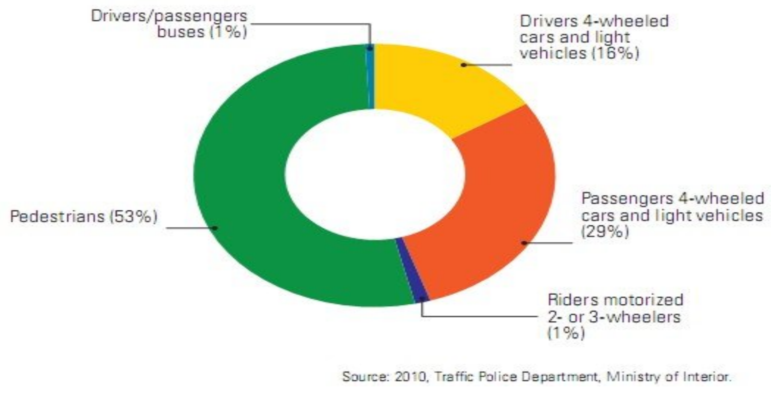
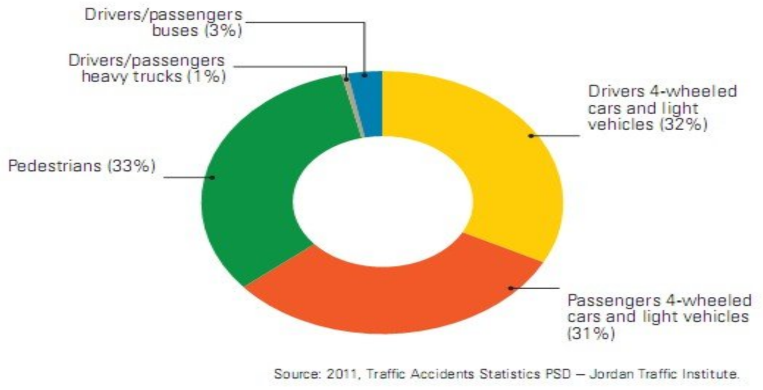


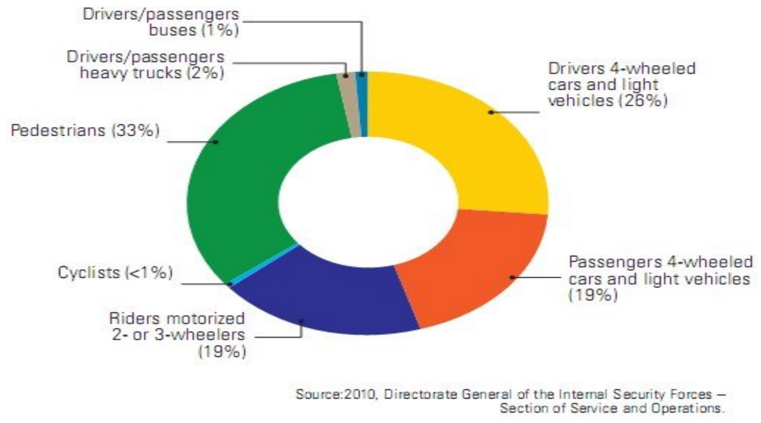
Figure 1: Distribution of traffic accidents in governorates of West-Bank [4].



Palestine



Jordan



Lebanon

Figure 2: Comparison of death rates for different road user categories: The comparison is mad for Palestine, Jordan and Lebanon [1].

1.2 Causes of Traffic Accidents

Generally speaking, there are 3 major causes for road traffic accidents and they are: human factors, road design factors, and vehicle factors. Most studies have shown that around 55-60% of vehicle accidents are due to driver related factors as a primary cause [5]. On the other hand, the road design factors are responsible for 25-30% and vehicle factors count for 10-20% (**Figure 3**).

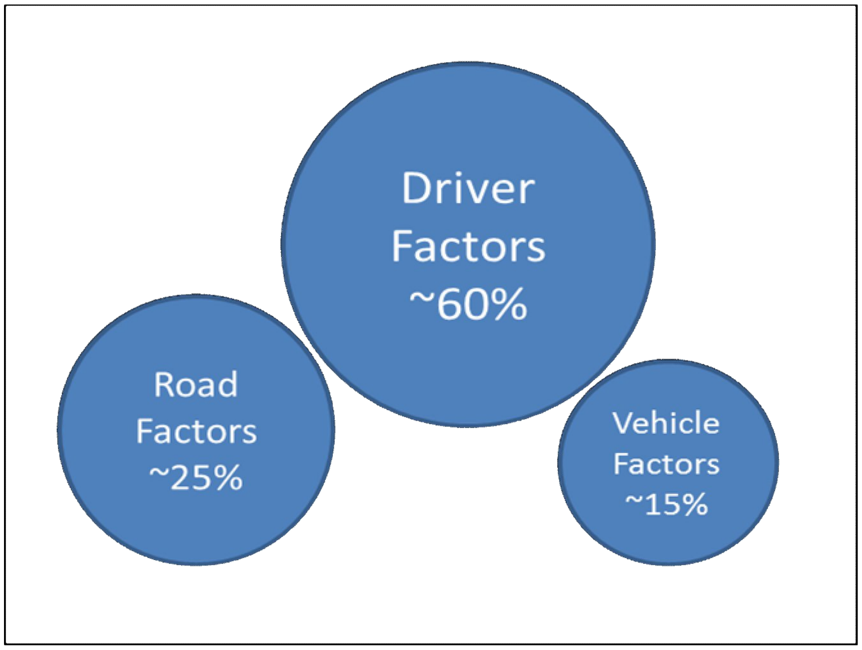


Figure 3: Major causes of road traffic accidents. The percentages represent estimations of how much deaths were due to the indicated factors. It is important to recall that certain accidents might be due to a combination of two or more factors together.

It is important to note that while road design and vehicle safety criteria are relatively amenable to improvement and optimization, driver associated factors are difficult to control (**Table 2**). This means that a lot of effort and creative solutions are needed in order to minimize the impact of driver associated factors.

Table 2: The most common reasons of traffic accidents: The examples shown in the middle column is for the most frequently reported cases that are associated to the three main factors. The right column shows how these problems were tackled by traffic authorities.

Type of factor	Example	Solutions
	♦ Distraction (using mobile, applying make-up, eating…etc)	Except the law to control alcohol consumption, the
	♦ Sleep deprivation	difficult to control.
	♦ Passenger physical shock	♦ Seatbelt and airbags
Vehicle	♦ Bad tires	♦ Periodical vehicle testing
	♦ Unmarked crossroad junction	♦ Traffic lights
Road	♦ Over-speed in urban areas	♦ Pedestrians crosswalks, bridges, or road bumps

1.3 Project Rationale

There is an increasing worldwide concern about the catastrophic societal and economic impact of road traffic accidents. In contrast to low and middle income countries, the developed countries have witnessed a significant reduction of the total number of deaths caused by traffic accidents. Good road design, superior traffic facilities, implementation of strict traffic laws, and high vehicle-safety-standards were the key factors behind this success. It is obvious that traffic authorities can tackle factors related to road design and vehicle safety criteria as they have tangible physical character. However, factors that are associated to human behavior or human limited abilities are relatively intractable.

An interesting approach to reduce the traffic accidents that result from driver associated factors is to make smart vehicles that can compensate the limited abilities of humans. A number of such smart computer-based technologies have been introduced to a very limited number of luxury classes of vehicles. In addition to their limited availability and very high cost, these technologies are only useful for ideal streets and highly sophisticated traffic systems. Therefore there is a great need to develop an affordable computer-based technology that can warn driver about possible risks such as crashing a pedestrian or a preceding vehicle. Such solution would be highly valuable if it can be adapted to new as well as old vehicle models.

1.4 Goal and objectives

Our main goal is to develop and implement a microcontroller-based system that can warn vehicle driver about possible frontal crashing risks with pedestrians.

Our specific objectives are:

1. To build a support vector machine-based classifier that can detect pedestrians in a given image using the histogram of oriented gradients (HOG) as a feature descriptor.

2. To adapt and run the trained and validated classifier on RaspberryPi microcontroller that is connected with a webcam and an audio speaker.

1.5 Project Importance

To the best of our knowledge, our project is the first of its type to implement a machine learning-based pedestrian detection system using a microcontroller.

The following points demonstrate the importance of our project:

1. It provides an affordable solution for Palestinian drivers and it can be adapted by other developing countries.
2. It can be mounted on new as well as old vehicle models and this means that a wider segment of drivers can benefit from this technology.
3. The proposed model will be tested in urban streets of the West-Bank, which would ensure its validity for streets with poor infrastructure.

1.6 Short description of the system

Our system can capture images using an inexpensive camera, yet with high resolution, which is going to be installed on top of car's dashboard to capture farms of the heads-up display. The captured images are continuously transferred to a microcontroller, which applies a pedestrian detection algorithm to classify input images as "True: a pedestrian under risk" or "False: no risk to hit a pedestrian". The "True" result will activate a warning alarm in order to prevent crashing the pedestrian.

1.7 Report Structure

In chapter two, we will talk about the analysis of our system as an introduction to make an optimal design. In chapter three, will give a background about the algorithms that will be used as well as some background information about the hardware parts. In chapter four, the hardware design and software design of the system will be described. In chapter five we will present the implementation of the system. Finally, chapter six contains the validation results of our system.

Chapter Two: System Analysis

In this chapter we will talk about the system analysis including general description, list of requirements, expected results, and constrains.

2.1 General Description

The functional operation of our system can be divided into three main phases: image acquisition, image processing, and warning. In the following paragraphs we will analyze the three phases in details.

• Phase 1: Image Acquisition

In this phase, the camera that will be mounted on the dashboard is capturing images of the head-up display and pass them to the microcontroller.

• Phase 2: Processing

In this phase, the microcontroller apply an algorithm to detect pedestrian in the risk field.

• Phase 3: Warning

In this phase, the system release a warning signal to alert the driver if there is a risk to crash a pedestrian. The warning signal be released by an audio speaker connected to the microcontroller.

2.2 List of Requirements

The requirements of our project can be categorized into two groups: hardware requirements, and software requirements.

♦ Hardware requirements include:

1. Camera.
2. Microcontroller.
3. Audio alarming system.
4. Power supply.

♦ Software requirements include:

1- Be familiar with Linux operating systems.
2- Using open CV (open source Computer Vision library).
3- Be able to write code in C++ programming language.
4- Have an enough image processing background, and its algorithms and approaches.
5- Have an enough machine learning background, and its algorithms and approaches.

2.3 Expected Results

The expected results of our project are:

1. Prototype microcontroller-based system that can be further developed to be a real business idea.
2. The long term expected output of our project is its contribution in solving the problem of road traffic accidents by minimizing their tragic societal consequences and economic burdens.

2.4 Constrains

The constrains of our system can be summarized up in the following points:

1. Day time: the system deal with images captured in the day.
2. Visual field: the system is supposed to deal with a clear visual field in front of the head-up display. That is mean it cannot deal with images captured when the weather is foggy or rainy.

Chapter Three: Background

In this chapter we will talk about the theoretical backgrounds and hardware parts background.

3.1 Theoretical Background

We will present a general idea about image processing, Histogram of Oriented Gradients, and Support Vector Machine.

- **Image processing:** is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or a set of characteristics or parameters related to the image. Most image- processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it, such as canny edge detection [6].

- **Histogram of Oriented Gradients (HOG):** The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of edge orientation histograms, scale-invariant feature transform descriptors, and shape contexts, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy. **Figure 4** shows the HOG steps while **Figure 5** shows a visualization of HOG feature.

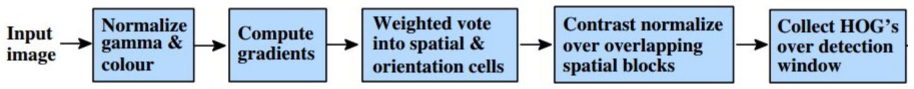

Input image
Normalize gamma & colour
Compute gradients
Weighted vote into spatial & orientation cells
Contrast normalize over overlapping spatial blocks
Collect HOG's over detection window

Figure 4: steps of HOG extraction[7]

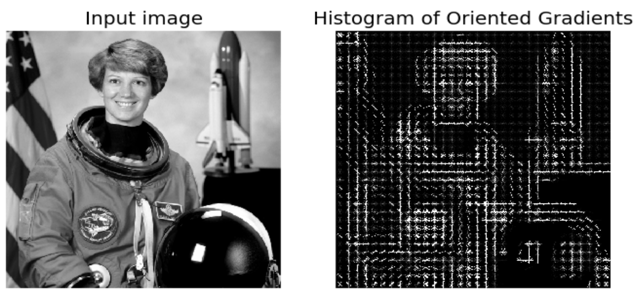


Figure 5: HOG feature visualization[7]

Navneet Dalal and Bill Triggs, researchers for the French National Institute for Research in Computer Science and Control (INRIA), first described HOG descriptors at the 2005 Conference on Computer Vision and Pattern Recognition (CVPR). In this work they focused on pedestrian detection in static images, although since then they expanded their tests to include pedestrian detection in videos, as well as to a variety of common animals and vehicles in static imagery.

The essential thought behind the histogram of oriented gradients descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is then the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing. **Figure 6** shows the process of making the histogram in HOG

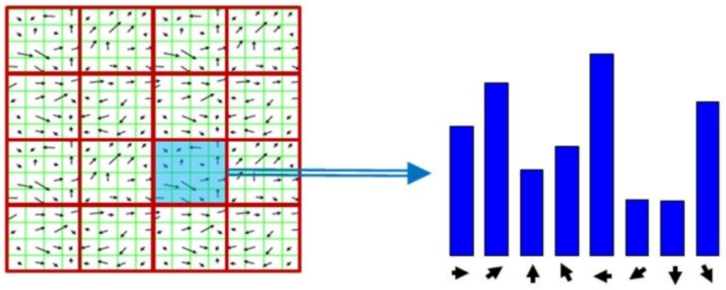


Figure 6: Histogram of Oriented Gradient explanation[7]

The HOG descriptor has a few key advantages over other descriptors. Since it operates on local cells, it is invariant to geometric and photometric transformations, except for object orientation. Such changes would only appear in larger spatial regions. Moreover, as Dalal and Triggs discovered, coarse spatial sampling, fine orientation sampling, and strong local photometric normalization permits the individual body movement of pedestrians to be ignored so long as they maintain a roughly upright position. The HOG descriptor is thus particularly suited for pedestrian detection in images.

• **Support Vector Machines (SVM):** In machine learning, support vector machines are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on. **Figure 7** shows data before and after classification using SVM

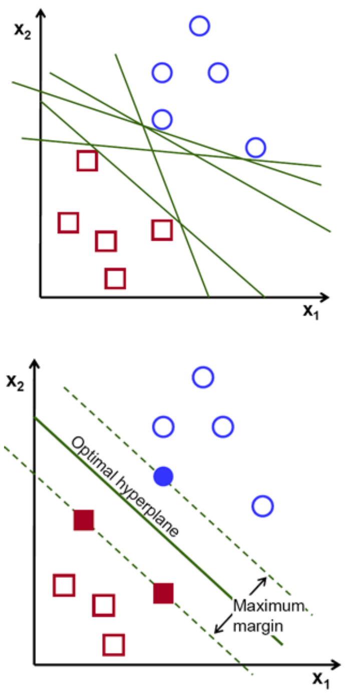


Figure 7: Example of before classification and after classification using SVM[12]

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

More formally, a support vector machine constructs a hyper-plane or set of hyper planes in a high or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyper plane that has the largest distance to the nearest training-data point of any class (called functional margin), since in general the larger the margin the lower the generalization error of the classifier. **Figure 8** shows an example of optimal hyper-plane.

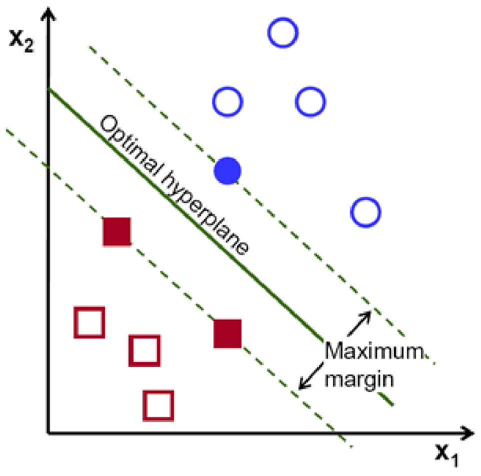


Figure 8: optimal hyper-plane

3.2 Hardware Parts Background

The parts of our project are camera, microcontroller, audio speaker, and power supply. In the following paragraphs, we will explain the basis of choosing these parts and their specifications.

• The Camera:

We decided to choose the Logitech C310, since its image quality has enough details in order to extract the features for pedestrian detection algorithm. In addition, this camera is cheap, available, robust, and easy to use since it can be connected through a Universal Serial Bus (USB) connection port

The specifications of the Logitech C310 camera are:

♦ Camera Type: Stationary camera.

♦ Video Frame Rate (Maximum): 30 fps.

♦ Resolution Video: 1280×720 pixel.

♦ Resolution (Still): 5 Megapixels.

♦ Focus Type: Fixed.

♦ Connection: USB.

♦ Mice: Build-in Mice.

Figure 9 shows the Logitech C310 camera.

Figure 9: Logitech C310 Camera [9]

• **The Microcontroller:**

Following a trade-off between cost and performance efficiency, we decided to choose the microcontroller as it has high specifications enough for image processing. It is easy to use since it can be operated by many distributions of Linux operating system, such as Debian, Angstrom, Ubuntu, and Android. It can be programmed by several languages such as C++, Java, or Python. Moreover, it is relatively a cheap microcontroller.

The specifications of the microcontroller are:

♦ CPU: 700 MHz (ARM 1176JZF--S core (ARM11 family).

♦ Memory (SDRAM): 512 MB.

♦ Video Outputs: 3,5 mm Klinke, HDMI.

♦ Audio Outputs: 3.5 mm Jack, HDMI.

♦ Onboard Storage: Micro SD, MMC, SDIO card slot.

♦ 10/100 Ethernet RJ45 on board network.

♦ Storage via Micro SD/ MMC/ SDIO card slot.

♦ Ports: HDMI (1 port), Either-net (1 port), USB (4 ports).

♦ Software Compatibility: Debian, Android, Ubuntu.

Figure 10 shows the microcontroller board.

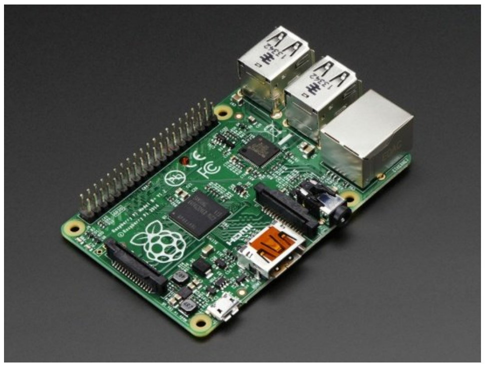


Figure 10: RaspberryPi [10]

• **Audio Alarming System:**

We decided to choose a speaker for alarming purpose in our project, since the speaker is cheap, efficient, available, and easy to connect.

Figure 11: Audio speaker[11]

Chapter Four: System Design

The design of our system can be divided into two phases, which are hardware phase and software phase. **Figure 12** shows the system diagram

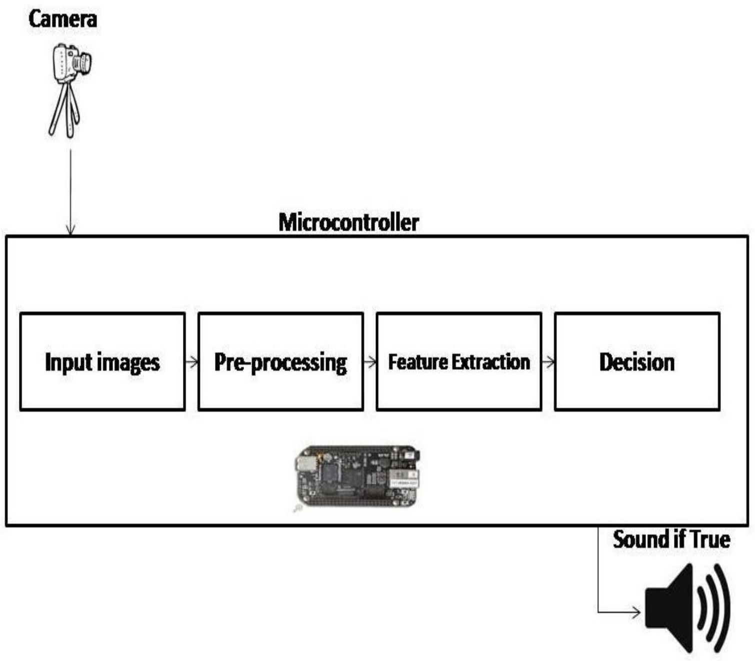


Figure 12: System diagram

4.1 Hardware Design

The hardware design of our system has three parts:

1. Connecting the camera with the microcontroller:

This part is done by connecting the Logitech C310 camera to the microcontroller through the USB connection port at the microcontroller board, as shown in **Figure 13**.

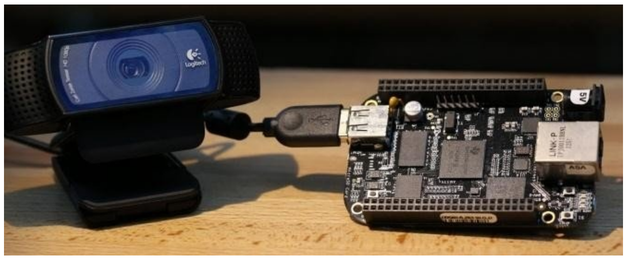


Figure 13: Connecting camera with microcontroller [10]

2. Connecting audio speaker with microcontroller:

This part is done by connecting the audio alarming system to the microcontroller board through the Jack audio port. **Figure 14** shows the jack audio port on the RaspberryPi.

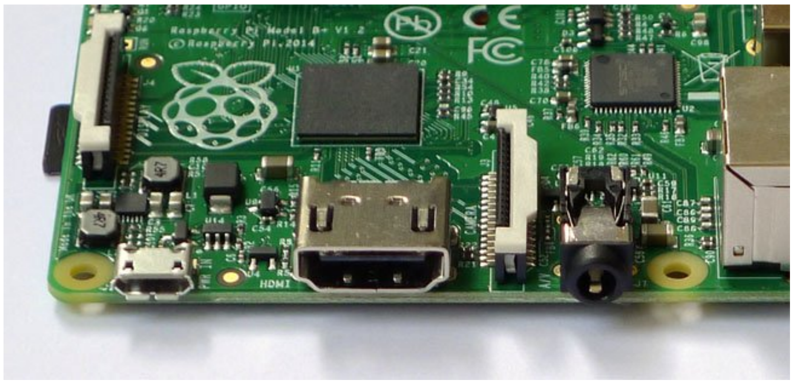


Figure 14: Audio Jack port on RaspberryPi board[10]

3. Power Supply: the power supply needed to operate the board is 5 volt, or by a USB cable, we will give it the power by USB cable. **Figure 15** shows the power supply.

Figure 15: USB car adapter[11]

4.2 Software Design

The software design is based on making an optimum algorithm for pedestrian detection considering the conditions and constraints that our system is supposed to deal with. In order to achieve an optimum algorithm the software divided into two parts training software and testing software.

4.2.1 Training Software Design

This part of the software aiming to train an SVM classifier, on a set of positive (images including pedestrians), and negative data (images without pedestrians).

The design of this software part is based on an algorithm, which extract the HOG feature of the training data. Then train an SVM classifier, based on that HOG features. **Figure 16** shows the block diagram of this software part.

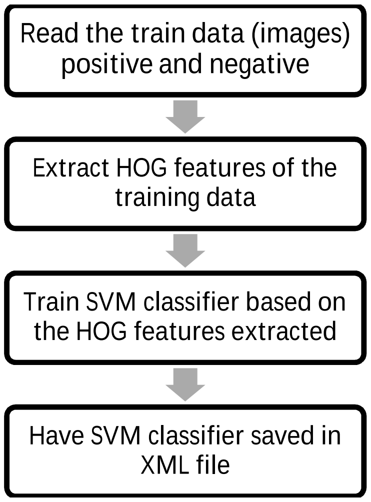

Read the train data (images) positive and negative
Extract HOG features of the training data
Train SVM classifier based on the HOG features extracted
Have SVM classifier saved in XML file

Figure 16: Training algorithm flowchart

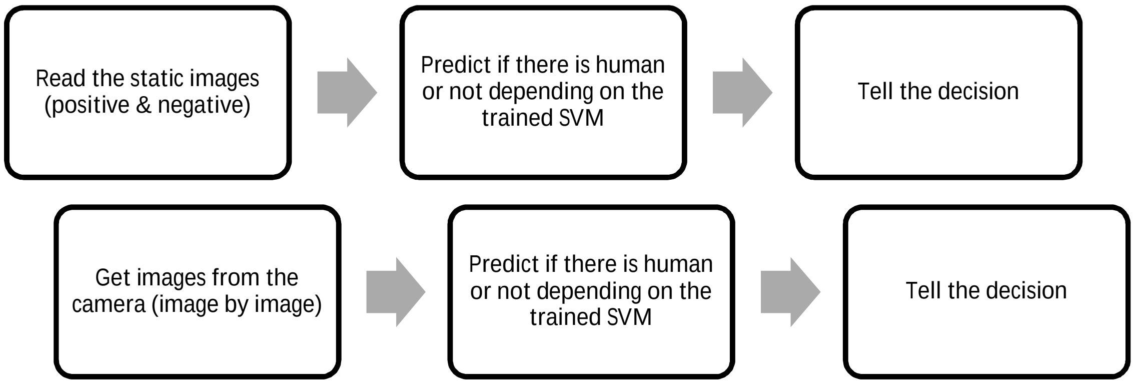
- **The pseudocode of this part is as follow:**

```
start
input: positive images
int NumberOfImages
array PositiveDescriptorsVector[NumberOfImages]
    for loop (i = 0 until i = NumberOfImages)
        vector ImageDescriptor
        extract HOG feature
        save HOG in ImageDescriptor
        PositiveDescriptorsVector[i] = ImageDescriptor
    end loop

input: negative images
array NegativeDescriptorsVector[NumberOfImages]
    for loop (i = 0 until i = NumberOfImages)
        vector ImageDescriptor
        extract HOG feature
        save HOG in ImageDescriptor
        NegativeDescriptorsVector[i] = ImageDescriptor
    end loop

SVMClassifier.train(PositiveDescriptorsVector, 1, NegativeDescriptorsVector, -1)
SVMClassifier.save("TrainedSVM.xml")
end
```

4.2.2 Testing Software Design

This part of the software aiming to produce a software, that uses the trained SVM from the previous software part. To predict if there is pedestrian , or not in a new image (given or from camera). That new image the classifier did not trained on it.

This software can work in two ways, either by given (static) images, or by live images (dynamic) from the camera. **Figure 17** shows the block diagram of the two ways.



Read the static images (positive & negative)
Predict if there is human or not depending on the trained SVM
Tell the decision
Get images from the camera (image by image)
Predict if there is human or not depending on the trained SVM
Tell the decision

Figure 17: Testing algorithm flowchart

• Given images case:

```
start
SVMClassifier.load(trainedSVM.xml)
int NumberOfImages
int result
        for loop(i = 0, until i = NumberOfImages)
                vector ImageDescriptor
                input: TestImage
                extract HOG for TestImage
                save HOG feature in ImageDescriptor
                result = SVMClassifier.predict(ImageDescriptor)
                output the result on the screen
        end loop
end
```

• Images from camera case:

```
start
SVMClassifier.load(trainedSVM.xml)
int result
    while(1)
        do
        {
        vector ImageDescriptor
        Mat image
        image = capture frame from camera
        extract HOG for image
        save HOG feature in ImageDescriptor
        result = SVMClassifier.predict(ImageDescriptor)
            if (result == 1)
                output: warning sound
        }
        end while loop
end
```

Chapter Five: System Implementation

In this chapter we will talk about the dataset that we use, hardware implementation, and software implementation.

5.1 Dataset used

The dataset which we use in training the classifier and testing it, is the INRIAPerson dataset[8]. Which is a data set published by the National Institute for Research in Computer Science and Control (French: Institut National de Recherche en Informatique et en Automatique). This data set have been collected by the French institute while Navneet Dalal and Bill Triggs were working on their paper. Which its title was " Histograms of Oriented Gradients for Human Detection". This dataset is the most popular set for human detection researches in the world. Also there is another set by MIT (Massachusetts Institute of Technology), but it does not contain negative images. So INRIAPerson dataset is better for our project. We use 200 positive and 200 negative images from the dataset, for training purpose the size of these data are 128 pixel × 64 pixel. The positive images are containing one person only, and with some margin around his body. On the other hand, in the testing phase we use 100 mixed images (positive and negative), in variable sizes. **Figure 18** shows the content of the dataset.

Figure 18: INRIAPerson dataset[8]

5.2 Hardware Implementation

In fact the hardware implemetation of our project is simple. It can be summarized up by the following points: firstly provide power supply for the microcontroller. Then connect the Logitech C310 camera to microcontroller through the USB port on the microcontroller. And the last step is connect the speaker to the microcontroller through the jack audio port on the microcontroller. There is another step which is connect to the LCD screen through the HDMI port in order to coding and working comfortably with microcontroller. **Figure 19** shows the hardware implementation.

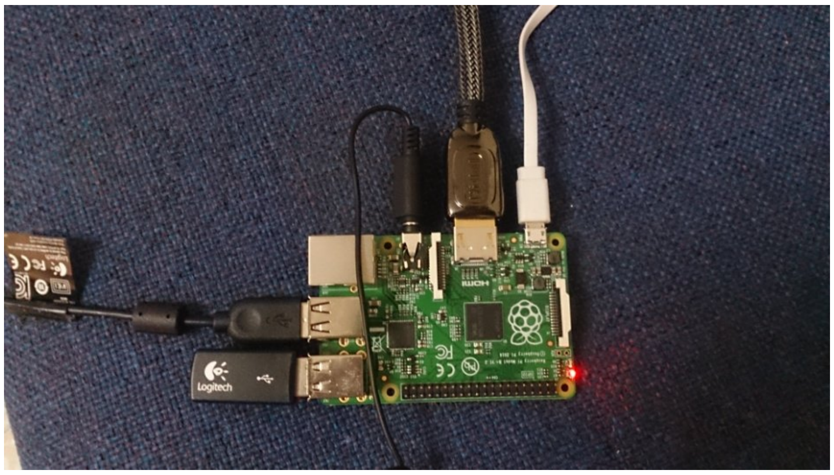


Figure 19: Hardware implementation

5.3 Software Implementation

The software part which is done on the PC, aiming to build SVM classifier by training over a set of data. This part done by training SVM classifier over the data from INRIAPerson dataset.

The implemetation of the training algorithm is done by write C++ code and using openCV software library within. The process is done as shown in the **Figure 20**.

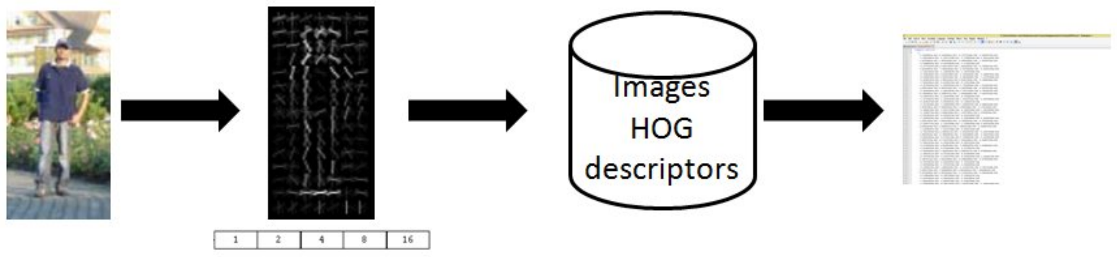


Figure 20: Training algorithm implementation

As shown in the figure, the first step done in the code is reading the images, this is done using the following command:

```
for (int i = 0; i< pFileNum; i++)
        {
        Mat img;
        img = imread(pFullFileName);
        }
```

Then the code will extract the HOG feature for each image, and this is done by the following command:

```
HOGDescriptor d(Size(32, 16), Size(8, 8), Size(4, 4), Size(4, 4), 9);
vector< float> descriptorsValues;
vector< Point> locations;
d.compute(img_gray, descriptorsValues, Size(0, 0), Size(0, 0), locations);
```

After that the code train and save an SVM classifier using the extracted descriptors, using the following command:

```
svm.train(PN_Descriptor_mtx, labels, Mat(), Mat(), params);
svm.save("trainedSVM.xml");
```

The other software part which is done on the microcontroller, aiming to take images captured from the camera connected to, and decide if these images contains pedestrian or not depending on the SVM classifier trained in the PC software part.

The implemetation of the testing algorithm is done by write C++ code and using openCV software library within. The process is done as shown in the **Figure 21**.

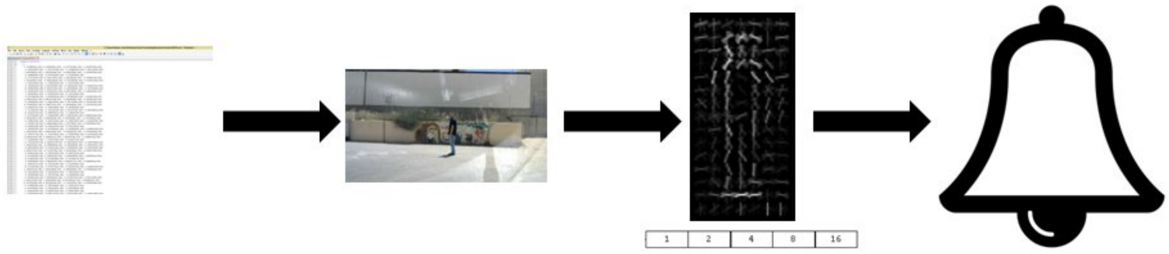


Figure 21: Detecting algorithm implementation

As shown in the figure the very first step is to load the XML file. That contains the trained SVM details, this is done by the following command

```
svm.load("trainedSVM.xml");
```

Then the code will capture an image from the camera, this is done by the following command

```
std::string arg = "video0";
VideoCapture capture(arg);
capture >> img;
```

And then compute the HOG descriptor using the same commands that were used in the training code

```
HOGDescriptor d(Size(32, 16), Size(8, 8), Size(4, 4), Size(4, 4), 9);
vector< float> descriptorsValues;
vector< Point> locations;
d.compute(img, descriptorsValues, Size(0, 0), Size(0, 0), locations);
```

Then the code send the descriptor to the SVM to predict the if there is pedestrian or not

```
Mat fm = Mat(descriptorsValues);
int result = svm.predict(fm);
```

Chapter Six: Validation and Testing

In this chapter we will talk about the validation of our system and the testing

6.1 Validation

We use fivefold cross validation technique to validate our working approach, and the results were awesome. Fivefold cross validation done on 500 images (data set), half of them positive (contain pedestrian) and half negative (does not contain pedestrian). We divide these images into five groups each group contains 400 images for training (200 positive, and 200 negative) and 100 images for testing. and the results were as shown in **Table 3**.

Table 3. Validation Results: This table shows the results of the five-fold cross validation. In each training round, 4 sets apart from the indicated validation set were used to train the model and the fifth set was used to calculate the accuracy measures (TP: True Positive, TN: True Negative, FP: False Positive, FN: False Negative).

Validation Sets	Validation Results	
Set 1	TP: 49	FP: 1
	TN: 47	FN: 3
Set 2	TP: 47	FP: 3
	TN: 49	FN: 1
Set 3	TP: 48	FP: 2
	TN: 49	FN: 1
Set 4	TP: 49	FP: 1
	TN: 50	FN: 0
Set 5	TP: 48	FP: 2
	TN: 50	FN: 0

6.2 Testing

When running the testing algorithm on a real-time case (images from camera), the system worked well, however the accuracy was obviously lower. We noticed that the accuracy of our classifier, using static images, is highly correlated with the number of images used for training.

We believe that the accuracy of our system on real-time images from camera can be significantly improved using a large number of training images. In fact, we attempted to increase the number of training images. However, the memory limitation of our personal PCs was the major problem to achieve this objective. Therefore, we would recommend using a more powerful computer for training classifier purposes.

The main problems that we faced in running the project can be summarized up in the following points:

1- Difficulty of configuring the openCV software library with the RaspberryPi microcontroller. In order to solve this problem we have contact an Italian openCV expert who help us configuring it.

2- Accessing the camera in order to obtain images from it. In order to solve this problem we read about the Raspberry port, and how to access it.

3- Filling the RAM by the images came from the camera. In order to solve this problem we have use the release command which release each image from the RaspberryPi RAM immediately after processing it.

Conclusion

We implement successfully an SVM based classifier, which used HOG as descriptor feature to detect pedestrians. The RaspberryPi microcontroller is a very good platform for such applications, since it gives a satisfied processing speed. The key point to increase the accuracy of our system, is to expand the number of training images.

We recommend using RaspberryPi microcontroller for such applications, also we recommend to use FPEG approach for doing this project. And try to use ultrasonic sensor to detect the distance between the car and the pedestrians.

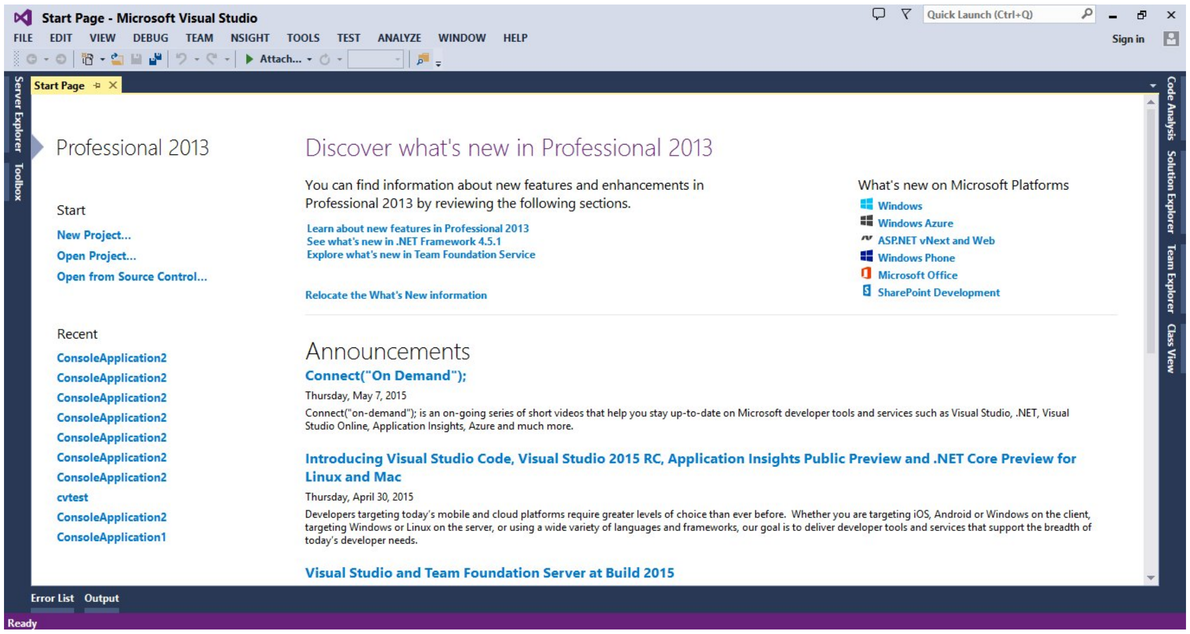
References

1. WHO Report, *Global status report on road safety 2013: Supporting a decade of action*. 2013, world health organization WHO: Geneva, Switzerland.
2. WHO Fact sheet, *Road traffic injuries.* Fact sheet N°358 2013. "http://www.who.int/mediacentre/factsheets/fs358/en/".
3. Gresser, E. *Traffic accidents kill 1.24 million people a year worldwide; wars and murders, 0.44 million.* Progressive Economy 2014]; Available from: http://progressive-economy.org/.
4. Jaradat, F., *Road traffic accidents in the West-Bank, Palestine. Report of 2013.* . 2013, Ministry of Transportation- the Higher Council of Traffic Ramallah.
5. Lum, H. and J.A. Reagan, *Interactive Highway Safety Design Model: Accident Predictive Module*, in *Public Roads Magazine*. 1995.
6. Wikipedia-page. *http://en.wikipedia.org/wiki/Image_processing*.
7. Dalal, N. and B. Triggs. *Histograms of oriented gradients for human detection*. in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. 2005: IEEE.
8. *INRIAPerson dataset*; Available from: http://pascal.inrialpes.fr/data/human/.
9. *Logitech Inc.*; Available from: http://www.logitech.com/.
10. *Raspberry Pi Organization;* Available from: https://www.raspberrypi.org/
11. *Conrad Electronic Inc;* Available from: *https:// www.conrad.com*
12. *Introduction to Support Vector Machines;* Available from: http://docs.opencv.org

Appendix

This appendix includes the setup of our project

1- We install Visual Studio 2013 IDE on PC, which is an IDE produced by Microsoft. It's an IDE compatible with C++ programming language and openCV software library.



Professional 2013

Start

New Project...
Open Project...
Open from Source Control...

Recent

ConsoleApplication2
ConsoleApplication2
ConsoleApplication2
ConsoleApplication2
ConsoleApplication2
ConsoleApplication2
ConsoleApplication2
cvtest
ConsoleApplication2
ConsoleApplication1

Discover what's new in Professional 2013

You can find information about new features and enhancements in Professional 2013 by reviewing the following sections.

Learn about new features in Professional 2013
See what's new in .NET Framework 4.5.1
Explore what's new in Team Foundation Service

Relocate the What's New information

What's new on Microsoft Platforms

Windows
Windows Azure
ASP.NET vNext and Web
Windows Phone
Microsoft Office
SharePoint Development

Announcements

Connect("On Demand");
Thursday, May 7, 2015
Connect("on-demand"); is an on-going series of short videos that help you stay up-to-date on Microsoft developer tools and services such as Visual Studio, .NET, Visual Studio Online, Application Insights, Azure and much more.

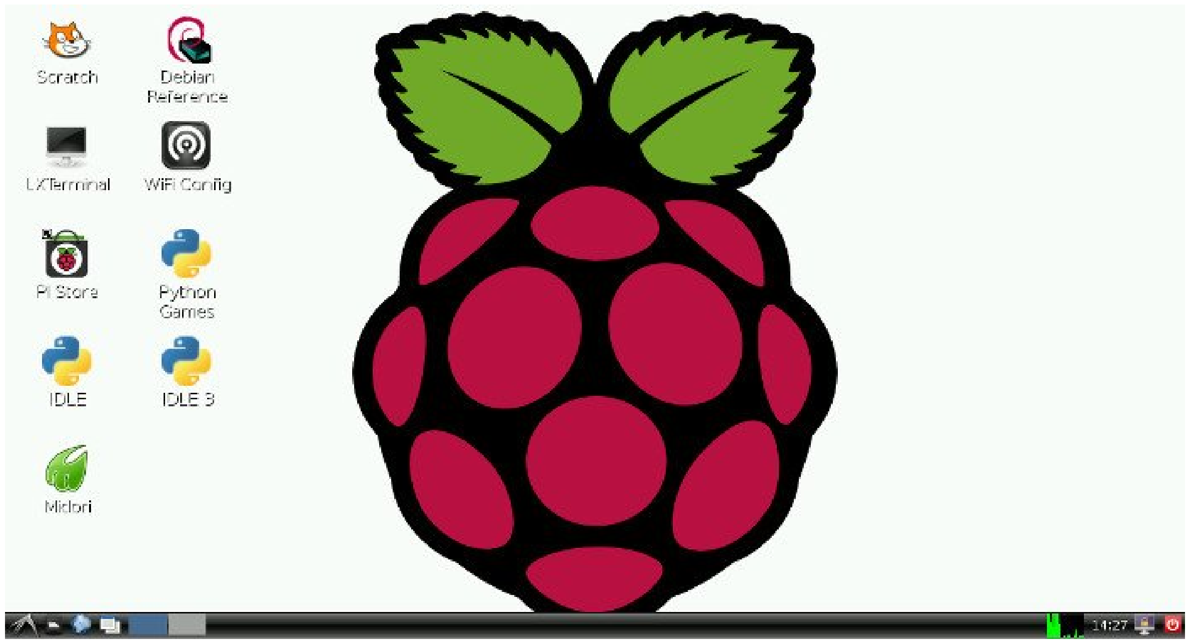
Introducing Visual Studio Code, Visual Studio 2015 RC, Application Insights Public Preview and .NET Core Preview for Linux and Mac
Thursday, April 30, 2015
Developers targeting today's mobile and cloud platforms require greater levels of choice than ever before. Whether you are targeting iOS, Android or Windows on the client, targeting Windows or Linux on the server, or using a wide variety of languages and frameworks, our goal is to deliver developer tools and services that support the breadth of today's developer needs.

Visual Studio and Team Foundation Server at Build 2015

2- We install openCV 2.4.10 version on PC and make all its configuration to use it in C++ coding. OpenCV (Open Source Computer Vision Library): is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 7 million. The library is used extensively in companies, research groups and by governmental bodies. It has C++, C, Python, Java and

MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications.

3- We install Raspbian operating system on the microcontroller. Which is a distribution from Debian. This process takes around 2 hours.



4- We install openCV 2.4.10 version on microcontroller and make it's configuration. This process takes around 22 hours.