

Department of Electrical and Electronics Engineering

M S Ramaiah Institute of Technology

Bangalore – 54



## CERTIFICATE

This is to certify that the following students, who were working under our guidance, have completed their work as per our satisfaction with the topic **“PATTERN RECOGNITION ON DIFFERENT PLATFORMS AND AN APPLICATION”**. To the best of our understanding the work to be submitted in this report does not contain any work, which has been previously carried out by others and submitted by the candidates for themselves for the award of any degree anywhere.

Rahul D Prabhu – 1MS09EE034

Ruthvick P – 1MS09EE040

Sunil J – 1MS09EE054

Smt. KUSUMIKA KRORI DUTTA

Assistant Professor,

EEE, MSRIT

Dr. PREMILA MANOHAR

Professor and HOD,

EEE, MSRIT

Department of Electrical and Electronics Engineering

M S Ramaiah Institute of Technology

Bangalore – 54



## **DECLARATION**

We hereby declare that the entire work embodied in this report has been carried out by us at M S Ramaiah Institute of Technology under the supervision of Smt. KUSUMIKA KRORI DUTTA, Assistant Professor, MSRIT. This report has not been submitted in part or full for the award of any diploma or degree of this or any other University.

Rahul D Prabhu – 1MS09EE034

Ruthvick P-1MS09EE040

Sunil J – 1MS09EE054

## ACKNOWLEDGEMENT

*We consider it as a great privilege to express our gratitude and respect to all those who guided us in our venture.*

*We would like to thank our principal Dr. S Y Kulkarni and Dr. Premila Manohar, HOD, Dept. of Electrical and Electronics, MSRIT for granting us the permission to carry out the project in the college.*

*We wish to express our sincere gratitude to our project guide Smt. Kusumika Krori Dutta, Assistant Professor, Dept. of Electrical and Electronics, MSRIT for her constant and ever-lasting guidance, moral support and help in editing even the smallest of mistakes. We are indebted to her for the unconditional support.*

*We would also like to thank all the teaching and non-teaching staff of the Dept. of Electrical and Electronics for their guidance and support.*

*Last but not the least we would like to thank our friends for the encouragement throughout the project.*

Rahul D Prabhu

Ruthvick P

Sunil J

## **ABSTRACT**

The present world is basically a digitized world where the computer plays a major role in almost all the fields ranging from medicine to communication to construction. And the basis of each and every field is image processing which includes feature recognition and classification, pattern recognition, projection, multi-scale signal analysis and many more. In earlier times hardware accelerated decoding of various image recognition and processing codes were present but these were recently joined by hardware accelerated encoding codes. For example the recognition and differentiation of cat and car in an image is easy to the human eye but the same is not that easy for a computer. Both have their own limitations. For example X-ray imaging and MRI scanning is not possible to the human eye. Thus image processing is the best solution to all these problems.

The processing is usually done with the help of remote servers which pose several Inconveniences like time delay, communication errors, noise and many more. Therefore we need real time on spot processing. The field of embedded systems has bridged the gap between remote and on spot processors with the help of 32-bit microcontrollers. Easy access of the same to the customers has made this even more feasible. For ever increasing image processing demand we have to incorporate both the features of control and signal processing in a single device.

In this regard we aim to carry out object: pattern recognition on a desktop PC and then implement the same on an ARM based development kit with libraries for image

extraction, FFT operations etc.. using either openCV, PYTHON or MATLAB. The ARM based computer we use is the Raspberry Pi. This has many function including network and output ports. We particularly use SimpleCV for object detection and hence face recognition, and based on this we decide on the various outputs at the general input/output ports.

# TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Literature Survey . . . . .	5
1.3	Overview . . . . .	6
1.4	Organization of Thesis . . . . .	6
<b>2</b>	<b>Face Detection</b>	<b>8</b>
2.1	Introduction . . . . .	8
2.2	Pattern Recognition . . . . .	9
2.3	Edge Detection . . . . .	10
2.3.1	Technique used . . . . .	14
2.4	Object Recognition . . . . .	15
2.5	Face Detection . . . . .	17
2.5.1	Technique used . . . . .	20
<b>3</b>	<b>The Processors</b>	<b>25</b>
3.1	The ARM Processor . . . . .	25
3.2	The X86 Processor . . . . .	28
<b>4</b>	<b>Application: Robotics</b>	<b>32</b>
4.1	Applications . . . . .	32
4.1.1	software . . . . .	33
4.1.2	Hardware . . . . .	40
<b>5</b>	<b>Results</b>	<b>43</b>
5.1	Results . . . . .	43
<b>6</b>	<b>Conclusion and Future Scope</b>	<b>46</b>
6.1	Conclusion . . . . .	46
6.1.1	Cost . . . . .	46
6.1.2	Support for operating systems . . . . .	47
6.2	Future Scope . . . . .	48
	<b>References</b>	<b>49</b>
	<b>APPENDIX</b>	<b>i</b>
.1	APPENDIX A	
	H-Bridge IC and its PIN configuration . . . . .	i

.2	APPENDIX B	
	Complete code used . . . . .	ii
.2.1	Manual control with web streaming . . . . .	ii
.2.2	Code for autonomous operation . . . . .	iv
.3	APPENDIX C	
	functions used . . . . .	x
.3.1	GPIO access . . . . .	x
.3.2	I2C . . . . .	xi
.3.3	SimpleCV . . . . .	xi
.4	APPENDIX D	
	Regulator Pin details . . . . .	xiv
.5	APPENDIX E	
	Raspberry Pi GPIO pin details . . . . .	xvi

# List of Figures

2.1	<b>canny flow chart</b>	13
2.2	<b>edge detection flow chart</b>	15
2.3	<b>Viola Jones algorithm</b>	21
2.4	<b>integral equation</b>	22
2.5	<b>process</b>	23
4.1	<b>the HTML UI</b>	36
4.2	<b>Manual control flow chart</b>	37
4.3	<b>coordinates for face detected</b>	38
4.4	<b>auto flow chart</b>	39
4.5	<b>Block diagram</b>	41
5.1	<b>Detection time in the ARM processor</b>	43
5.2	<b>Detection in the X86 Processor</b>	44
6.1	<b>Component cost comparison in a study of a typical HMI system. The study compares costs for a system based on an ARM-processor to one based on an X86-processor.</b>	47
2	<b>IC Datasheet</b>	i
3	<b>7805 pin details</b>	xiv
4	<b>Raspberry Pi pin details</b>	xvi



# **Chapter 1**

## **INTRODUCTION**

### **1.1 Introduction**

The present world is basically a digitized world where the computer plays a major role in almost all the fields ranging from medicine to communication to construction. The present day cannot even be imagined to function properly without a computer. A computer has transformed itself from a luxury to a basic necessity. And computer in term is a product of electrical and Electronics. As the usage of a product increases there has to be improvements and the computer is no exception to this. The computer has indeed come a far way from just a huge calculator or a huge storage unit to a multitasking, wide utility, highly accurate and efficient computing, storing and processing system. From a processor the size of a room to the size of a file, from hard disks of the size of shelves to the size of credit cards, from processors which used to sound like trains to soundless systems, anything and everything of a computer has undergone development and is still in the ever continuous process of progressive development. There is not a single field where the computer can be neglected. This shows that there is always scope for development.

Newer and better products are continuously being thrown into the market by companies like Intel, National Instruments, Texas Instruments, AMD etc., The competition among the companies has resulted in faster, better efficient products being rolled out like never before. A technology released today becomes obsolete almost the very next day. The super computer of today may not even stand a chance when pitted against a computer of tomorrow.

The advances are not just limited to the hardware and software components of a computer. There have also been advances in other fields i.e., The computer has widened its range of applications. It has expanded its utility. Certain functions which the computer pundits wouldnt even have dreamt off are now being done easily on a computer. One such field is Image Processing. The computer has seen leaps and bounds in this field and there is continuous work going on worldwide in this area.

An image is something that depicts or records visual perception. Images may be two-dimensional, such as a photograph, screen display or three-dimensional, such as a hologram. They may be captured by optical devices such as cameras, mirrors, lenses, telescopes, microscopes, etc. and natural objects and phenomena, such as the human eye or water surfaces. An image is an array of pixels arranged in columns and rows. It is basically a matrix of pixels where each and every pixel has a value of its own by which they can be accessed. In a (8-bit) greyscale image each

picture element has an assigned intensity that ranges from 0 to 255. A grey scale image is what people normally call a black and white image, but the name emphasizes that such an image also includes many shades of grey, to be precise 256 shades of grey. A normal greyscale image has 8 bit colour depth = 256 greyscales. Whereas a True colour image has 24 bit colour depth =  $8 \times 8 \times 8$  bits =  $256 \times 256 \times 256$  colours = 16 million colours. This shows that a pixel in a True colour image can have a value ranging from 0 to somewhere around 16 million.

After having set up the imaging system and acquired images, one can analyze and process images to extract valuable information about the objects under inspection. Image analysis combines techniques that compute statistics and measurements based on the gray-level intensities of the image pixels. Using the information gathered from analyzing the image, one may want to improve the quality of the image for inspection. One can improve the image by removing noise, highlighting features in which they are interested, and separating the object of interest from the background. This is nothing but Image Processing.

Image processing is analyzing and manipulating images with a computer in much simpler words it is just playing with images. It is any form of signal processing for which the input is an image, such as a photograph or video frame and the output is either an image or a set of characteristics or parameters related to the input image. Usually Image

Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them. The basic requirement for the processing of images is that they should be available in digitized form, that is, arrays of finite length binary words. For digitization, the given Image is sampled on a discrete grid and each sample or pixel is quantized using a finite number of bits. The digitized image is processed by a computer which includes sampling of image and quantization of sampled values. After converting the image into bit information, processing is performed. The processing techniques include Image enhancement, Image restoration, and Image compression. Digital image processing is the use of computer algorithms to perform image processing on digital images. As a subcategory or field of digital signal processing, digital image processing has many advantages over analog image processing. It allows a much wider range of algorithms to be applied to the input data and can avoid problems such as the build-up of noise and signal distortion during processing.

Also the processors have undergone a sea change with respect to processing speed, accuracy, efficiency, power consumption, compatibility, size and aesthetic regards. The processor which was the size of an entire room when the computer was invented has now been scaled down to the size of a shoebox. The size is the only downsizing. Everything else has advanced beyond even the wildest of imagination of the computer inventors. Various companies are coming up with newer products almost on

a daily basis. So this field is always new. In this regard we thus come across two main processors making headlines everyday namely the X86 and the ARM processor. We thus thought of working in this relatively new field. With the processors and the face recognition process itself in mind we went ahead into the literature survey to start off with our project work.

## **1.2 Literature Survey**

In this regard we started a search for papers which would help us in our work. The work in this field is recent and mind boggling. We referred a lot of papers including some IEEE papers which we have mentioned in detail in the reference section. We got to know the basics of Real-time Object detection [1] and the amount of work that is presently being carried on in this field. We learnt the details of face detection programming [2] and Viola-Jones algorithm [1]. Viola Jones algorithm is at present the most efficient technique for Face detection and is being used universally by each and every one. We came to know the in depth analysis of super resolution procedures [3] which tells us how a hazy image can be extracted , the noise reduction and final image production. The final image being much better than the original one. Also we realized the real world applications of Face detection be it for security reasons or for database comparisons. We also referred to the internet constantly during the entire duration of our project.

## **1.3 Overview**

We aimed to incorporate a single technique of Face Detection involving Viola-Jones Face Detection algorithm and run the same on two kinds of processors, the X86 and the ARM processor. We basically wanted to compare the process on both the kinds of processors. Then after having done this since ARM is relatively newer we wanted to use it in our application. We have built a robot which moves based on the controls given over the network. We also have provided an autonomous override and in the process obtained an automatic vehicle.

## **1.4 Organization of Thesis**

This report has been divided into 5 main chapters. Chapter 1 deals with the introduction to our project and the reasons for us choosing this particular project. Chapter 2 speaks about the core of our project the techniques for face detection. Chapter 3 contains all the information regarding the processors we have chosen for our project and the result of Face Detection on both. In chapter 4 we will be introducing our application the Face Chaser and it will contain all the information about our application. Chapter 5 summarizes the results obtained, the working range and the limitations of our system. It also comprises of the scope for future work.



# **Chapter 2**

## **Face Detection**

### **2.1 Introduction**

In this chapter we will be covering each and every aspect regarding Image Processing as required by our project. We will be proceeding on a sequential precedence basis as to which process we have to undertake to reach our main aim: Face Detection. To the human eye detecting a face is very easy because we have got an inbuilt mega processor The Brain. We can even distinguish between different patterns, objects, colour shades, patterns, faces and even the minutest of differences just by looking at the image. But the same does not hold true for a computer. As mentioned before to a computer an image is just an arrangement of pixels in a matrix form. It cannot differentiate or identify anything in the image very easily. But with the help of certain program the computer can identify and differentiate the intricacies involved in an image. Face detection is also one of the functions which the computer can perform with a little bit of help from the programmers.



To understand Face detection we first need to understand all the basics involved and the sequential flow right from image analysis ending up with face detection. We came to know a bit on Image analysis in the previous chapter itself. So we will start off with pattern recognition.

## **2.2 Pattern Recognition**

Pattern recognition is the first most important step in a series of processes finally resulting in Face detection. Pattern recognition is an information reduction process: the assignment of visual or logical patterns to classes based on the features of these patterns and their relationships. Pattern Recognition is the science and art of giving names to the natural objects in the real world. Typical inputs to a Pattern recognition system are images or sound signals out of which the relevant objects have to be found and identified. In our project the inputs are restricted to images. The solution involves many stages such as making measurements, pre-processing and segmentation, finding a suitable numerical representation for the objects of our interest and finally classifying them based on these representations.

The main problem that the computer has to deal with while recognizing a pattern is the number of different targets or objects that it has to cope with. Thus the problem cannot be solved by straightforward computational and matching methods or database searches; but on the upside the number of classes into which the objects have to be first allotted are

finite also each and every object has to invariably be allotted into one of the classes. Based on the comparison and similarities of the classes the pattern can be detected.

The basic setting of pattern recognition mostly has one unknown object presented as a set of signals in the input of the system. At the output there will be a set of predefined classes. The purpose of the system and the process of Pattern Recognition is to assign the object on the input to one of the classes on the output. Thus each class is differentiated and the representation of different classes forms a pattern and thus a pattern can be obtained.

So basically after the process of Pattern Recognition we have differentiated each and every object in our image into different classes which makes it easier for detecting the intricacies involved. Instead of checking each and every pixel every time in the next step we can select a class and then plot out its edges and by doing so for each and every object identified we can detect the edges of the entire image.

## **2.3 Edge Detection**

Edges form the outline of an object. An edge is the boundary between an object and the background indicating the boundary between overlapping objects. This means that if the edges in an image can be identified accurately, all of the objects can be located and basic properties such as

area, perimeter, and shape can be measured relatively easily. Edge detection simplifies the object recognition to an extent. After this process all the objects are definitely identified but they are not recognized. Each and every object is identified but the computer is not able to tell which object as which in the sense it cannot differentiate between the identified objects.

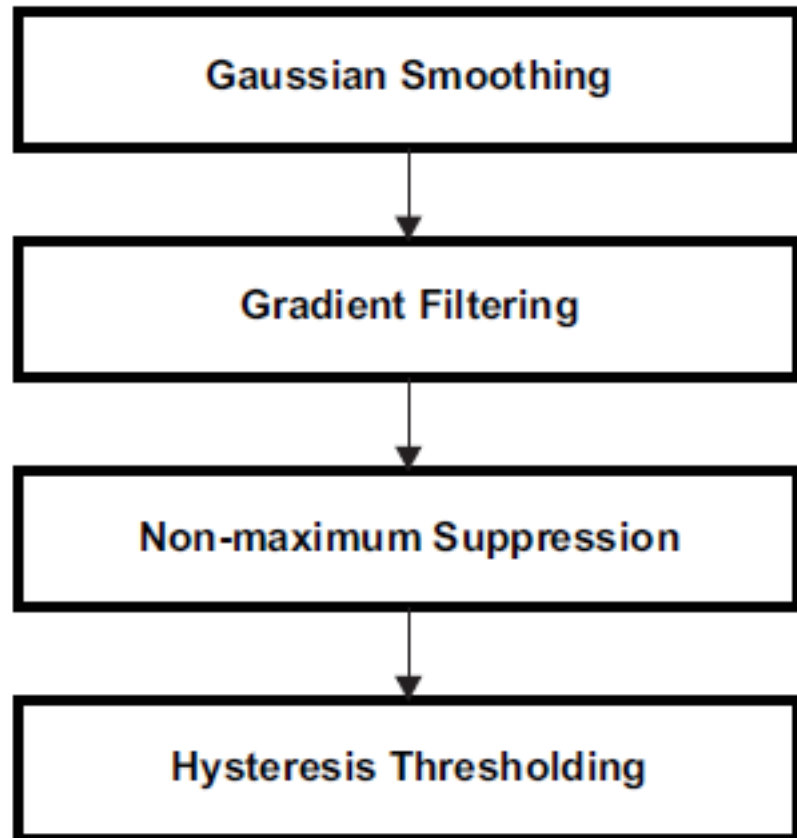
Edge Detection is basically the name for a set of mathematical methods which try to identify points in an image specifically a digital image. Not just any points but those points at which the image brightness changes sharply, or in simpler terms, has discontinuities. The points identified form a set of curved line segments termed as Edges. Edge detection is the most prominent technique in Image Processing, particularly in the fields of feature extraction and feature detection and in our project this feature itself is nothing but The Human Face.

It can be visualized on more general terms that edges mostly correspond to discontinuities in depth, surface orientation, changes in material properties and variation in scene illumination. The result of applying any Edge Detection technique to an image is usually a set of connected curves that indicate the boundaries of objects, the boundaries of surface markings as well as those corresponding to discontinuities in surface orientation. By applying Edge Detection it makes us simpler in the subsequent tasks of interpreting the information as most of the trivial data is

filtered out.

There are many methods for edge detection. They can be grouped into three categories

- Search-based/ First Order Derivative/ Gradient Methods
  - Roberts Operator
  - Sobel Operator
  - Prewitt Operator
- Zero-crossing based/ Second Order Derivative
  - Laplacian
  - Laplacian of Gaussian
  - Difference of Gaussian
- Optimal Edge detection
  - Canny Edge Detection



**Figure 1. Flow Diagram of Canny Edge Detector**

Figure 2.1: canny flow chart

The search-based methods detect edges by first computing a measure of edge strength, usually a first-order derivative expression such as the gradient magnitude, and then searching for local directional maxima of the gradient magnitude using a computed estimate of the local orientation of the edge, usually the gradient direction. The zero-crossing based methods search for zero crossings in a second-order derivative expression computed from the image in order to find edges, usually the zero-crossings of the Laplacian or the zero-crossings of a non-linear differential expression. The Canny edge detection technique will be explained in

detail in the next chapter.

### **2.3.1 Technique used**

In our project we have used the Optimal technique of Canny Edge Detection. The purpose of edge detection in general is to significantly reduce the amount of data in an image, while preserving the structural properties to be used for further image processing. Since different edge detectors work better under different conditions, it would be ideal to have an algorithm that makes use of multiple edge detectors. In this section we will be introducing the Canny technique. It is an edge detector operator which uses a multistage algorithm to detect edges in images. It was developed by John F Canny in 1986. An optimal edge detector has to incorporate good detection, good localization and minimal response.

- Smoothing: Blurring of the image to remove noise.
- Finding gradients: The edges should be marked where the gradients of the image has large magnitudes.
- Non-maximum suppression: Only local maxima should be marked as edges.
- Double thresholding: Potential edges are determined by thresholding.
- Edge tracking by hysteresis: Final edges are determined by suppressing all edges that are not connected to a very certain (strong)

edge.

Each and every algorithm has been described in detail in the section 2.6.1

Denition of edges

- Edges are signicant local changes of intensity in an image.
- Edges typically occur on the boundary between two different regions in an image.

the flow chart for edge recognition is shown below

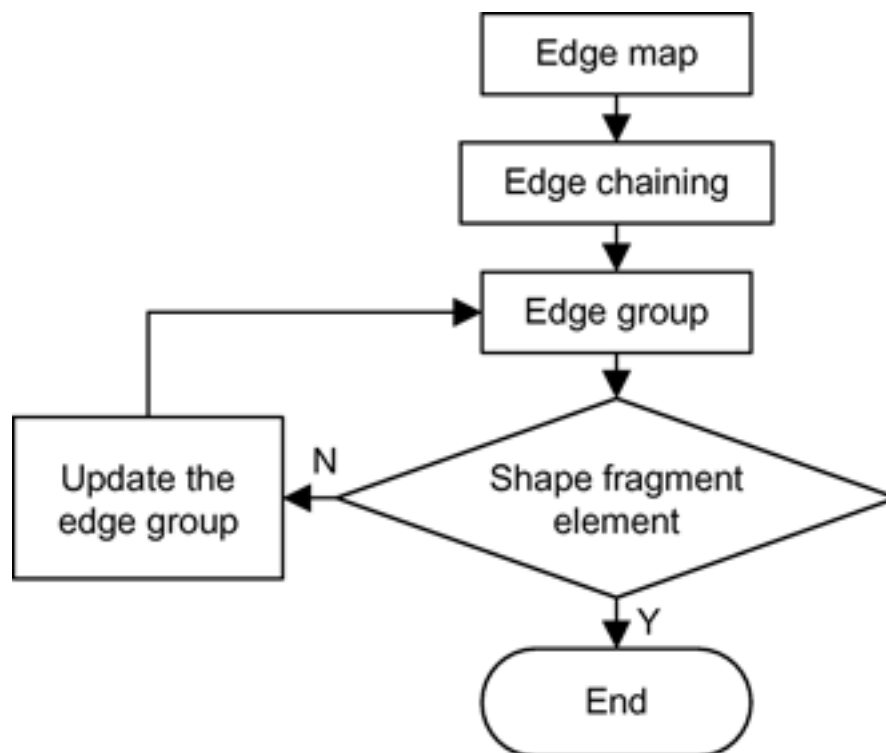


Figure 2.2: edge detection flow chart

## 2.4 Object Recognition

Object recognition has become one of the most popular tasks in computer vision. In particular, this was driven by the development of new

powerful algorithms for local appearance based object recognition. So-called smart cameras with enough power for decentralized image processing became more and more popular for all kinds of tasks, especially in the field of surveillance. Recognition is a very important tool as the robust recognition of suspicious vehicles, persons or objects is a matter of public safety. This simply makes the deployment of recognition capabilities on embedded platforms necessary.

Object recognition is the task of finding a given object in an image or video sequence. Humans recognize a multitude of objects in images with little effort, despite the fact that the image of the objects may vary somewhat in different view points, in many different sizes / scale or even when they are translated or rotated. Objects can even be recognized when they are partially obstructed from view. This task is still a challenge for computer systems in general.

Object recognition algorithms rely on matching or learning algorithms using appearance-based or feature-based techniques. Common techniques include edges, gradients, Histogram of Oriented Gradients (HOG), Haar wavelets, and linear binary patterns. Object recognition is useful in applications such as video stabilization, automated vehicle parking systems, and cell counting in bioimaging.

You can recognize objects in MATLAB with Image Processing Toolbox, Computer Vision System Toolbox, Statistics Toolbox, and Neural



Network Toolbox using a variety of models, including:

- Extracted features and boosted learning algorithms
- Bag of words models with features such as SURF and MSER
- Gradient-based and derivative-based matching approaches
- Viola-Jones algorithm
- Template matching
- Image segmentation and blob analysis

Our approach to object recognition in our project uses pixels as its geometric features. Our method is based on the gray values of the model and image itself and uses normalized cross correlation or the sum of squared or absolute differences as a similarity measure.

Thus after Object recognition the computer is able to allot the objects into different classes which makes it easier in the next and most important procedure of our project Face Detection.

## **2.5 Face Detection**

To reiterate it is very easy for us to identify a face and might even be possible to recognize it definitely if we are familiar with that particular face. But for a computer detection itself is difficult and then to compare it with a database and recognize it is easier said than done. But every object

will have its own uniqueness with the help of which we can identify them. And for a face it is the eyes, nose and lips. If at all we come across these components then most of the times we can generalize that object to be a face.

Face detection is a computer technology that determines the locations and sizes of human faces in digital images. It detects facial features and ignores anything else, such as buildings, trees and bodies. Face detection can be regarded as a specific case of object-class detection. In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class.

Face detection can be regarded as a more general case of face localization. In face localization, the task is to find the locations and sizes of a known number of faces (usually one). In face detection, one does not have this additional information

Face recognition is not so much about face recognition at all - it is much more about face detection. It is a strong belief, that the prior step to face recognition, the accurate detection of human faces in arbitrary scenes, is the most important process involved. When faces could be located exactly in any scene, the recognition step afterwards would not be so complicated.

The existing visual sensing and computing technologies are at a state where reliable, inexpensive, and accurate solutions for non-intrusive and natural means of human-computer interactions are feasible. Biometrics is an evolving application domain for face detection and is concerned with the use of physiological information to identify and verify a persons identity. In most cases, face recognition algorithms are designed to operate on images assumed to only contain frontal faces. Therefore, face detection is required to first extract faces from an image prior to the recognition step.

The identification and localization of a face or faces from either an image or video stream is a branch of computer vision known as face detection. Face detection has attracted considerable attention over recent years in part due to the wide range of applications in which it forms the preliminary stage. Some of the main application areas include: human-computer interaction, biometrics, content-based image retrieval systems (CBIRS), video conferencing, surveillance systems, and more recently, photography. Another application area that can clearly benefit from face detection is surveillance systems that would allow easier identification of criminals in public spaces.

The majority of the research work to date has primarily focused on developing novel face detection algorithms and/or improving the efficiency and accuracy of existing algorithms. A wide variety of techniques ex-

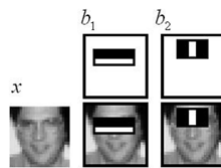
ist, ranging from simple edge-based algorithms, to complex high-level approaches using pattern recognition methods. We in our project have used the Viola-Jones face detection algorithm which is the most efficient till date

### **2.5.1 Technique used**

This section describes a visual object detection framework that is capable of processing images extremely rapidly while achieving high detection rates. Paul Viola and Michael Jones introduced a new approach for visual object detection using the principle of a boosted cascade of classifiers . It can be trained for face detection, and is capable of processing images extremely rapidly while achieving high detection rates.

To understand the Viola Jones detector, the concept of boosting needs to be explained first. Randomly answering a yes or no question with an evenly distributed answer space will yield the correct answer 50 percent of the time in the long run. If a method can improve this score by a very small amount, it is called a weak classifier. It is possible to generate weak classifiers for a great number of tasks in an automated manner by enumerating a large set of generated data on a basis of very simple rules, and then evaluating their performance on a set of samples. A heuristic that can improve the detection rate by a larger amount is called a strong classifier. By boosting, we aim to combine several of these simple, weak classifiers, and create a strong classifier. Adaboost is a well

known method to combine weak classifiers and create strong classifiers. The weak classifiers in Viola and Jones are based on three different kinds of features. The two-rectangle feature is the difference between the sum of the values in two adjacent rectangular windows. The three-rectangle feature takes three adjacent rectangles, and computes the difference between the sum of the pixels in the extreme rectangles, and the sum of the pixels in the middle rectangle. A four rectangle feature considers a 2 by 2 set of rectangles, and computes the difference between the sum of the pixels of the diagonally opposed rectangles. An example of these features is shown in figure.



**Figure 2.3: Viola Jones algorithm**

It is shown how for example, a three-rectangle classifier can return recognizable results for face sections like the nose, where the differences in pixel sums over the square are very characteristic. The minimum size of a feature is roughly comparable to the size of a face in an image, so a 16x16 section of an image can already contain hundreds of thousands of features, since a 12x12 pixel or even smaller detector is swept over each pixel. This is where the Adaboost algorithm comes in, and selects those weak classifiers to limit the selection to a few hundred weak classifiers, that will still yield good enough results. This obviously greatly increases the speed of the algorithm. Computing the rectangular features

is a straightforward operation. The algorithm then introduces the integral image. The integral image at a location  $(x, y)$  is defined as the sum of the pixel values above and to the left of  $(x, y)$ . It is therefore like an integral function over the entire image. The integral image  $ii$  is defined as a function of the original image  $i$  in equation.

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

**Figure 2.4: integral equation**

By using this representation, we can greatly increase the efficiency in calculating rectangular sums, since we only need to compute the difference in the total sum for the two corners of the rectangle. This is a contribution by Viola Jones to the original algorithm, which greatly reduces the computational complexity, since it effectively removes the needs to calculate large sums of pixels for each and every classifier, every single time. To determine whether a sample contains a face, the sum of weighted classifier scorers is taken, and compared to a previously determined threshold.



Figure 2.5: process





# Chapter 3

## The Processors

### 3.1 The ARM Processor

The ARM architecture describes a family of RISC-based computer processors. It was first developed in the 1980s by Acorn Computers Ltd to power their desktop machines. The official Acorn RISC Machine project started in October 1983. A key design goal was achieving low-latency input/output. Globally as of 2013 it is the most widely used 32-bit instruction set architecture. The name was originally an acronym for Acorn RISC Machine. After the name Acorn was dropped, then it became Advanced RISC Machine. ARM architecture forms the basis for every ARM processor. Using a RISC based approach to computer design, ARM processors require significantly fewer transistors than processors that would typically be found in a traditional computer.

The benefits of this approach are lower costs, less heat, and less power usage, traits that are desirable for use in light, portable, battery-powered

devices such as smart phones and tablet computers. The reduced complexity and simpler design allows companies to build a low-energy system on a chip for an embedded system incorporating memory, interfaces, radios, etc. Over time, the ARM architecture has evolved to include architectural features to meet the growing demand for new functionality, high performance and the needs of new and emerging markets. The ARM architecture supports implementations across a wide range of performance points, establishing it as the leading architecture in many market segments. The ARM architecture supports a very broad range of performance points leading to very small implementations of ARM processors, and very efficient implementations of advanced designs using state of the art micro-architecture techniques.

. Implementation size, performance, and low power consumption are key attributes of the ARM architecture. ARM developed architecture extensions to provide support for Java acceleration security, SIMD, and Advanced SIMD technologies. The ARMv8-architecture adds a Cryptographic extension as an optional feature. The ARM architecture is similar to a Reduced Instruction Set Computer (RISC) architecture, as it incorporates these typical RISC architecture features.

A uniform register file load/store architecture, where data processing operates only on register contents, not directly on memory contents. Simple addressing modes, with all load/store addresses determined from regis-

ter contents and instruction fields only. Enhancements to a basic RISC architecture enable ARM processors to achieve a good balance of high performance, small code size, low power consumption and small silicon area.

As of 2005, about 98 pc of the more than one billion mobile phones sold each year used at least one ARM processor. As of 2009, due to low power consumption the ARM architecture is the most widely used 32-bit RISC architecture in mobile devices and embedded systems. ARM offers several microprocessor core designs, including the ARM7, ARM9, ARM11, Cortex-A8, Cortex-A9, and Cortex-A15. The ARM core has remained essentially the same size throughout these changes. The ARM architecture specifies several CPU modes, depending on architecture.

At any moment in time, the CPU can be in only one mode, but it can switch modes due to external events (interrupts) or programmatically. The ARM supports add, subtract, and multiply instructions. The ARM architecture is supported by a large number of embedded and real-time operating systems, including Windows CE, Symbian. The very first ARM-based Acorn Archimedes personal computers ran an interim operating system called Arthur, which evolved into RISC OS, used on later ARM-based systems from Acorn and other vendors.

## 3.2 The X86 Processor

X 86 processor is mainly desktop processor. An x86 processor is similar to those used in many windows machines. x86 or 80x86 is the generic name of a microprocessor architecture first developed and manufactured by Intel. The x86 architecture was first used for the Intel 8086 Central Processing Unit (CPU) released during 1978, a fully 16-bit design based on the earlier 8-bit based 8008 and 8080. Although not binary compatible, it was designed to allow assembly language programs written for these processors to be mechanically translated into equivalent 8086 assembly. The 32-bit generation of this architecture is also called "x86". The 8086 was introduced in 1978 as a fully 16-bit extension of Intel's 8-bit based 8080 microprocessor and also introduced memory segmentation to overcome the 16-bit addressing barrier of such designs. The term x86 derived from the fact that early successors to the 8086 also had names ending with "86". Many additions and extensions have been added to the x86 instruction set over the years.

The architecture has been implemented in many processors. Most of the x86 processors used in new personal computers and servers have 64-bit capabilities, to avoid compatibility problems with older computers or systems, the term x86-64 (or x64) is often used to denote 64-bit software, with the term x86 implying only 32-bit. Although the 8086 was primarily developed for embedded systems and small single-user

computers, largely as a response to the successful 8080-compatible the x86 line soon grew in features and processing power. Today, x86 is used effectively in both stationary and portable personal computers and has replaced midrange computers and Reduced instruction set computer (RISC) based processors in a majority of servers and workstations as well. A large amount of software, including operating systems (OSs) such as DOS, Windows, Linux, BSD, Solaris and Mac OS X, functions with x86-based hardware.

Modern x86 is relatively uncommon in embedded systems, however, and small low power applications (using tiny batteries) as well as low-cost microprocessor markets, such as home appliances and toys, lack any significant x86 presence. The x86 architecture is a variable instruction length, primarily "CISC" design with emphasis on backward compatibility. The instruction set is not typical CISC, however, but basically an extended version of the simple eight-bit 8008 and 8080 architectures. Byte-addressing is enabled and words are stored in memory with little-endian byte order. Memory access to unaligned addresses is allowed for all valid word sizes. During execution, current x86 processors employ a few extra decoding steps to split most instructions into smaller pieces (micro-operations). These are then handed to a control unit that buffers and schedules them in compliance with x86-semantics so that they can be executed, partly in parallel, by one of several (more or less specialized) execution units. These modern x86 designs are thus superscalar, and also

capable of out of order and speculative execution , which means they may execute multiple x86 instructions simultaneously, and not necessarily in the same order as given in the instruction stream.



# Chapter 4

## Application: Robotics

### 4.1 Applications

Manual control of Robot and streaming of live images, control conditions of a robot using the recognized images

Manual control of the robot here is done using the Raspberry Pi ( the ARM based computer ) wherein using the GPIO (general purpose input/output) pins are used for controlling the motors. These pin values are controlled using various libraries available for different languages. We in particular have used the python language. The controls can either be invoked from the terminal or through JavaScript from an HTML webpage.

The testing was done through the terminal and the final control is through the HTML page. For the automatic control the image is first processed ie., the face is first detected in the frame using the viola jones algorithm and we have applied the haar classifier for frontal face. This detected image is then judged as to where exactly on the screen is it



present. Based on the position of the detected image the robot is given control signals so as to move in the set direction.

#### **4.1.1 software**

Since the entire image processing is done on board the robot itself we have used python 2.0 as there are various libraries. The python program has included many libraries including the GPIO access, webiopi ( a custom library for integration of python and JavaScript).

In the beginning the modes of the GPIO pins are set. Then the server for the HTML page is started. Here in the python program a loop is started where in it checks for the input of keystrokes and clicks from the mouse. Then suitable Macros are assigned for the starting and stopping the motors of the wheels on the robot. These control signals from the GPIO pins are given to the H-bridge (in our case we use the L239D dual H-Bridge npn transistor model). The H-bridge working is explained in detail in the hardware part.

As for the streamer from the robot, we use the generic MJPG streamer available for UNIX

The libraries are shown below

```
1 import RPi.GPIO as GPIO
2 import webiopi
3 GPIO.setmode(GPIO.BOARD)
4 L1=11 # H-Bridge 1
5 L2=12 # H-Bridge 2
6 R1=22 # H-Bridge 3
7 R2=23 # H-Bridge 4
8 t=13
9 def left_stop():
10     GPIO.output(L1, GPIO.LOW)
11     GPIO.output(L2, GPIO.LOW)
12
13 #def left_stop():
14 #    GPIO.output(L1, GPIO.HIGH)
15 #    GPIO.output(L2, GPIO.HIGH)
16
17 def left_forward():
18     GPIO.output(L1, GPIO.HIGH)
19     GPIO.output(L2, GPIO.LOW)
20
21 def left_backward():
22     GPIO.output(L1, GPIO.LOW)
23     GPIO.output(L2, GPIO.HIGH)
24
25 def right_stop():
26     GPIO.output(R1, GPIO.LOW)
27     GPIO.output(R2, GPIO.LOW)
```

project/manual1.py

The server and the macros are shown below

```
1
2 GPIO.setup(L1, GPIO.OUT)
3 GPIO.setup(L2, GPIO.OUT)
4 GPIO.setup(R1, GPIO.OUT)
5 GPIO.setup(R2, GPIO.OUT)
6 stop()
7 server = webiopi.Server(port=8000, login="msrit", password="
    electrical")
8 server.addMacro(go_forward)
9 server.addMacro(go_backward)
10 server.addMacro(turn_left)
11 server.addMacro(turn_right)
12 server.addMacro(stop)
13
14 webiopi.runLoop()
15 server.stop()
16
17 #GPIO.setmode(L1, GPIO.IN)
18 #GPIO.setmode(L2, GPIO.IN)
19 #GPIO.setmode(t, GPIO.IN)
20
21 #GPIO.setmode(R1, GPIO.IN)
22 #GPIO.setmode(R2, GPIO.IN)
```

project/manual2.py

The UI for the manual control

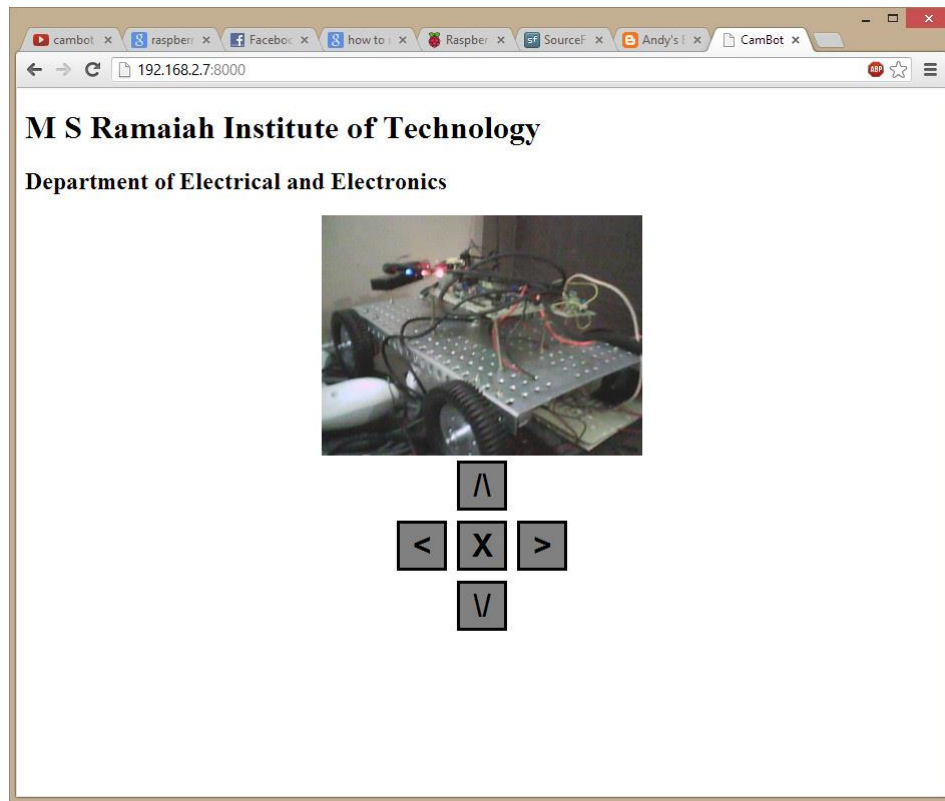


Figure 4.1: the HTML UI

The flowchart of the Manual control is given below

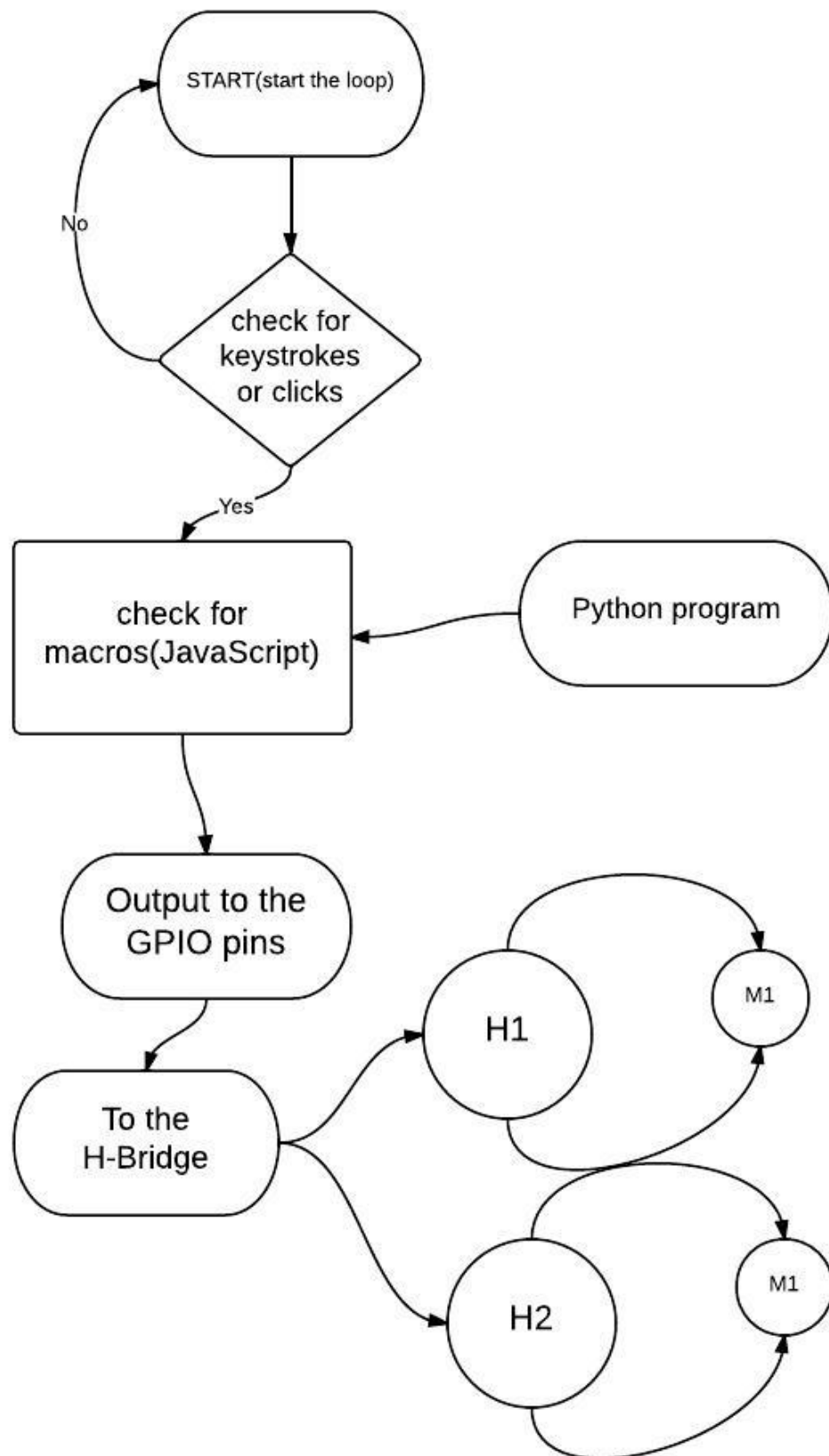


Figure 4.2: Manual control flow chart

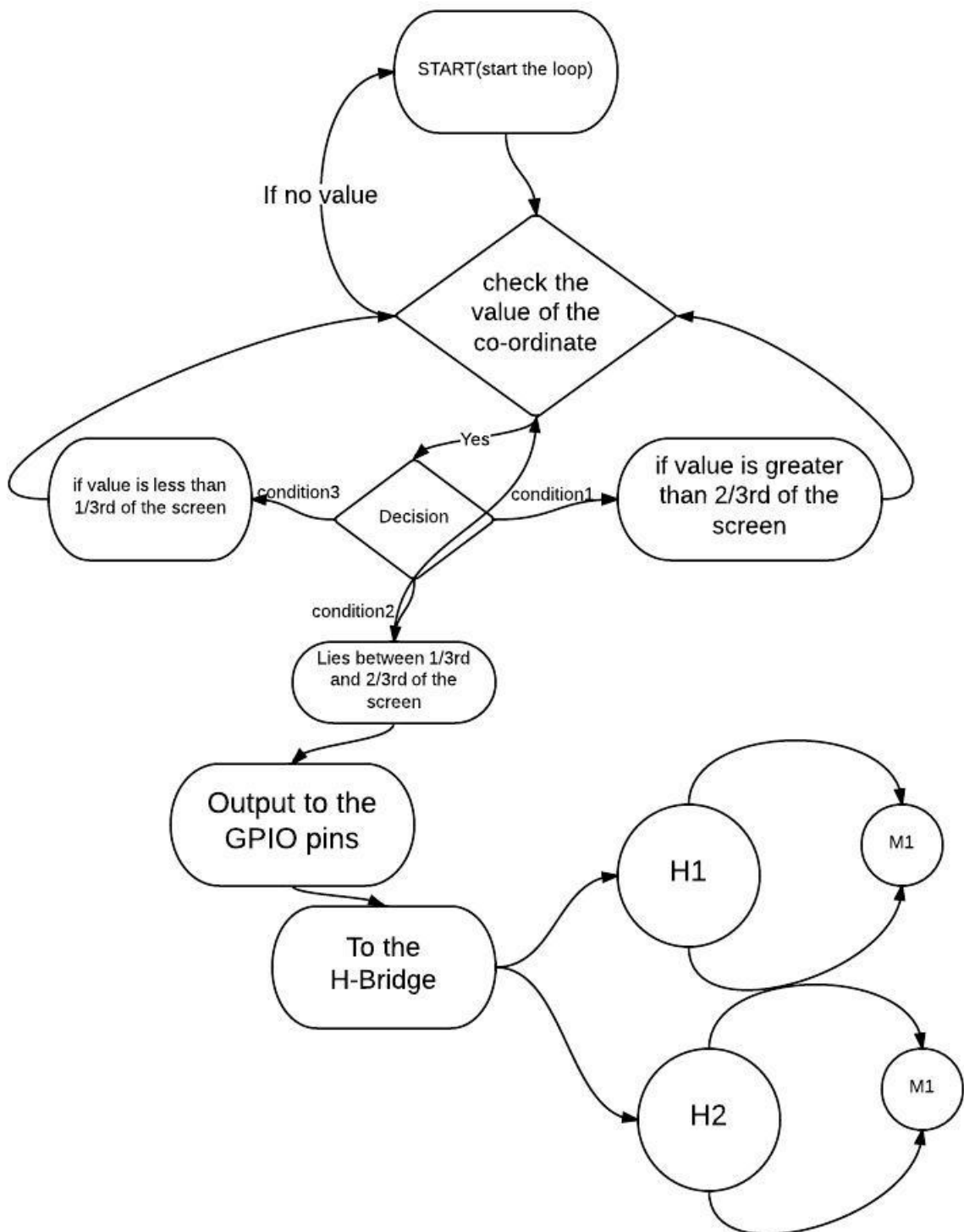
For the auto control we have the procedure wherein we detect the face and we draw a rectangle around the detected face. And based on one of the co-ordinates of the corners of the rectangle we decide on the direction of the robot

Below is the picture showing the value of the co-ordinate of the corner of the rectangle



**Figure 4.3: coordinates for face detected**

The flow chart for the auto control is shown below



**Figure 4.4: auto flow chart**

### 4.1.2 Hardware

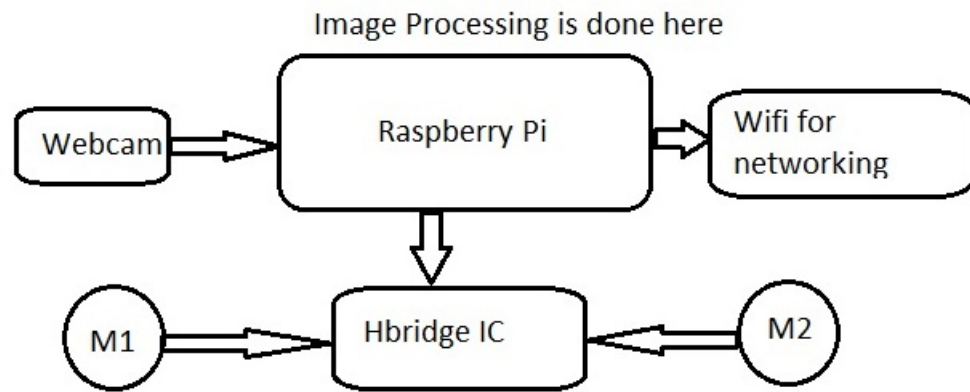
The core component of our project is the Raspberry pi (the ARM based computer). The image processing and the control inputs to the H-Bridge is give from the Raspberry Pi itself

The other basic components involved are

- The motors for the wheels: we have used a 12V, 300 rpm DC motor
- L239D( H-bridge) : for the direction control of the motors, mainly consists of npn transistors
- Battery pack: We have used a 1.2 ah 12V lead acid battery which is maintenance free and can be recharged
- Regulator IC 7805: for powering the Raspberry pi and the H-Bridge IC
- Wi-Fi modem: we have use the Asus N13 modem because of its compatibility with the Raspberry Pi
- Webcam: we have used the Logitech C110 web camera

The block diagram for the Hardware components connected are shown below





**Figure 4.5: Block diagram**

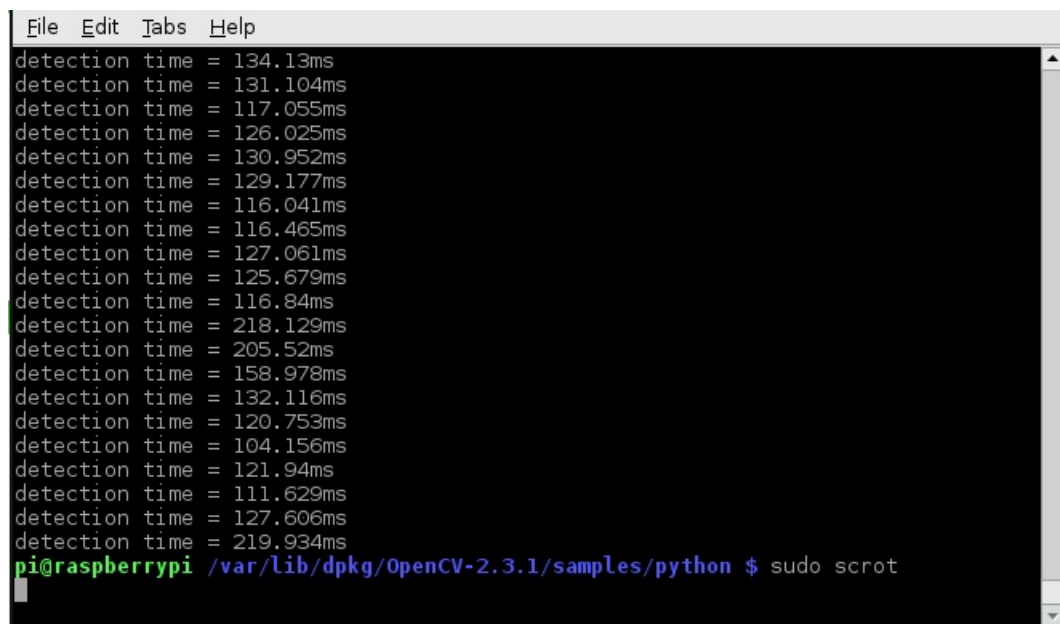


# Chapter 5

## Results

### 5.1 Results

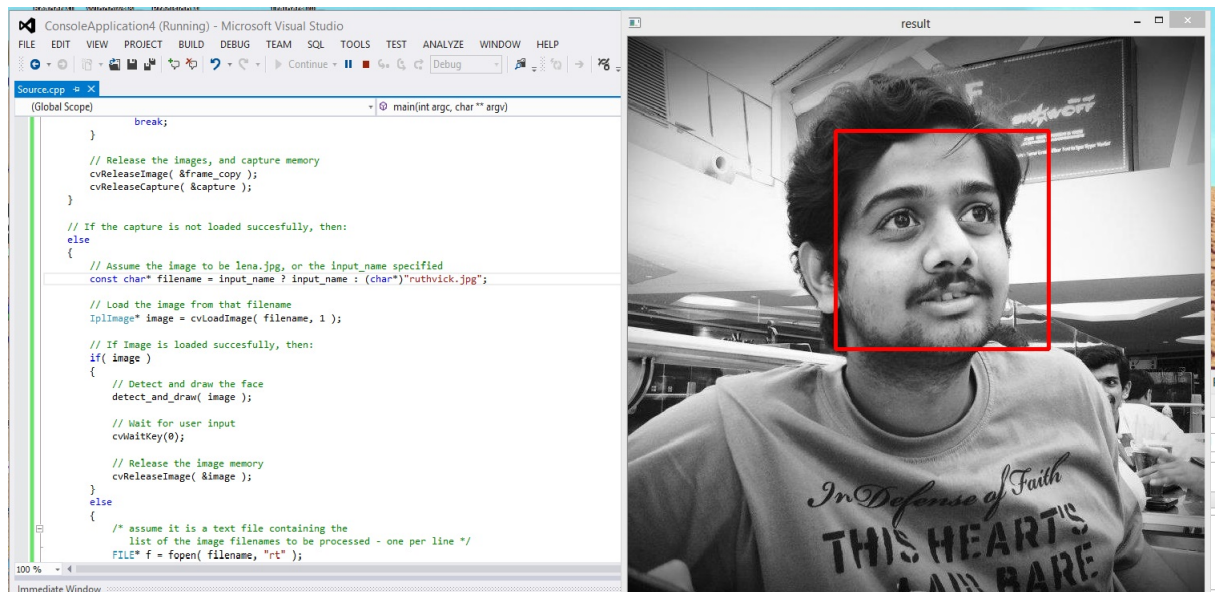
We have tested the face detection algorithm using openCV on the X86 (the Desktop proocessor) and we have found that the detection time using the viola-jones algorithm is around 3ms and when we applied the same algorithm using simpleCV on the ARM processor we have obtained results of around 200-300 ms Detection time

A screenshot of a terminal window with a menu bar (File, Edit, Tabs, Help) and a list of 20 detection times in milliseconds. The times range from approximately 104ms to 219ms. The terminal prompt shows the user is 'pi' on a 'raspberrypi' machine, in the directory '/var/lib/dpkg/OpenCV-2.3.1/samples/python', and has just run the command 'sudo scrot'.

```
File Edit Tabs Help
detection time = 134.13ms
detection time = 131.104ms
detection time = 117.055ms
detection time = 126.025ms
detection time = 130.952ms
detection time = 129.177ms
detection time = 116.041ms
detection time = 116.465ms
detection time = 127.061ms
detection time = 125.679ms
detection time = 116.84ms
detection time = 218.129ms
detection time = 205.52ms
detection time = 158.978ms
detection time = 132.116ms
detection time = 120.753ms
detection time = 104.156ms
detection time = 121.94ms
detection time = 111.629ms
detection time = 127.606ms
detection time = 219.934ms
pi@raspberrypi /var/lib/dpkg/OpenCV-2.3.1/samples/python $ sudo scrot
```

Figure 5.1: Detection time in the ARM processor

on the Desktop or the X86 processor we have obtained a time of around 3 ms. It is shown below



**Figure 5.2: Detection in the X86 Processor**



# Chapter 6

## Conclusion and Future Scope

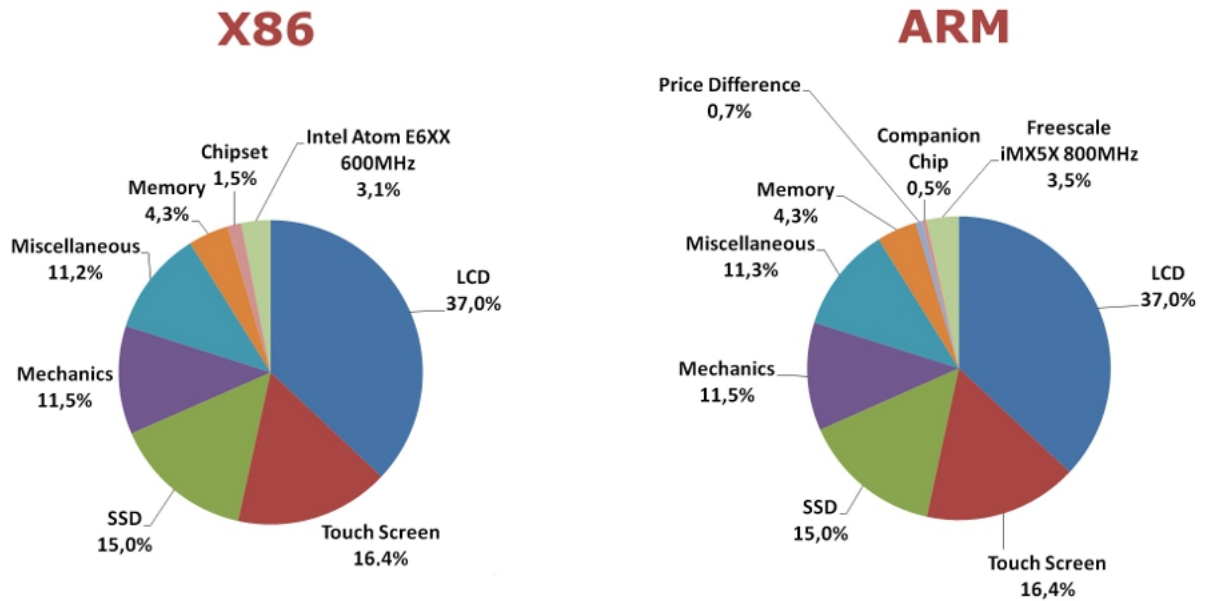
### 6.1 Conclusion

#### 6.1.1 Cost

The cost structure of the two alternative systems is shown in the picture below. It turns out that the ARM platform (Processor and companion chip) is the 14pc cheaper alternative. It's a fact tempting to put in focus.

But a fact is also that 14 pc turns into a price difference of merely 0.7 pc for the complete system. The cost of processor and chipset aren't really that a great percentage of the overall component cost. In this case the display and touch screen make up more than half the cost of components in the complete system. A decisive effort on evaluation of optional displays and touch screens for instance would have a better chance of lowering the cost of components.

According to studies the costs for an HMI system are shown below



**Figure 6.1: Component cost comparison in a study of a typical HMI system. The study compares costs for a system based on an ARM-processor to one based on an X86-processor.**

### 6.1.2 Support for operating systems

Among the listed operating systems Linux and Windows are the more uncomplicated to work with in your embedded development project. Historically Linux support has generally been good in the X86 architecture and excellent for Intel platforms. Linux support for the ARM architecture varies depending on the processor. Search for the more widely used processors to ensure proper Linux support.

If you are striving for the really safe start in software development a Windows OS on an X86 processor is probably your first choice. A Windows OS in full version always (at least really close to always) is up and running instantly. When things are working properly its time to downscale to an embedded Windows version. If you are not up for

it yourself there are a lot of companies and professionals out there that knows Windows and are willing to offer their services.

## **6.2 Future Scope**

- As we have show in one of our application we have used the processor for controlling robots over the network and hence the internet. The Image is streamed live and the processing can be done on the remote X86 processor, as the processing time is much better than the ARM processor
- If remote Desktops are not feasible like in the villages in India then the processing can be done on board the robot itself as shown in the autonomous control
- other applications or the future scope for our project include Home Automation where in the Images can be recognised and the face detected and it can be compare to the database and hence can be used to control switches or doors





## References

- [1] *Robust Real-time Object Detection*; Paul Viola, Michael Jones
- [2] *Face Detection on Embedded Systems*; Abbas Bigdeli et al.,
- [3] *Single Image Super-Resolution using Gaussian process*; Wan Chi Sui, Kwok Hai Hung
- [4] <http://www.hecronic.se/website1/embedded/arm-versus-x86/arm-versus-x86.php>



## APPENDIX A

### H-Bridge IC and its PIN configuration

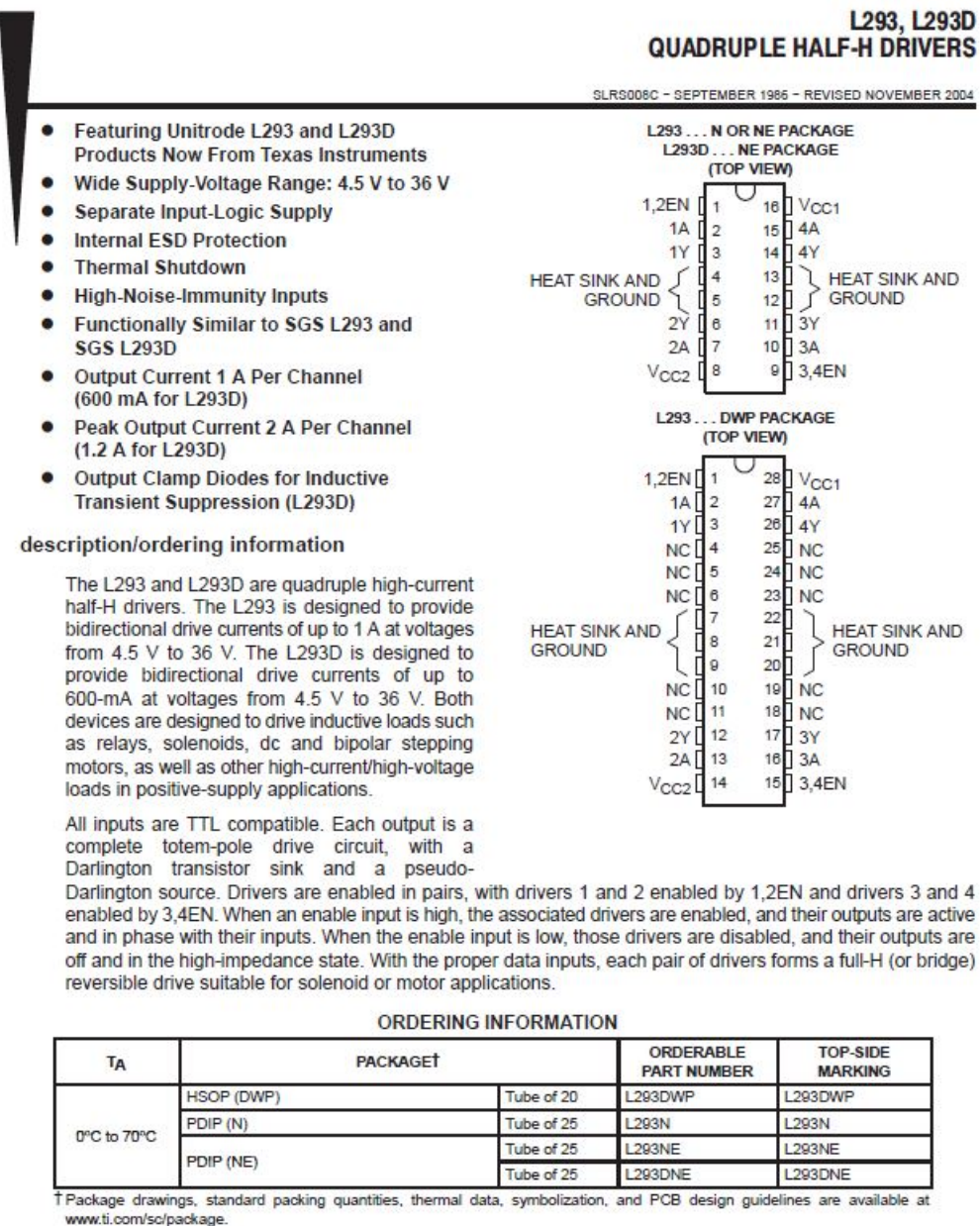


Figure 2: IC Datasheet

## .2

# APPENDIX B

## Complete code used

### .2.1 Manual control with web streaming

```
1 import RPi.GPIO as GPIO
2 import webiopi
3 GPIO.setmode(GPIO.BOARD)
4 L1=11 # H-Bridge 1
5 L2=12 # H-Bridge 2
6 R1=22 # H-Bridge 3
7 R2=23 # H-Bridge 4
8 t=13
9 def left_stop():
10     GPIO.output(L1, GPIO.LOW)
11     GPIO.output(L2, GPIO.LOW)
12
13 #def left_stop():
14 #    GPIO.output(L1, GPIO.HIGH)
15 #    GPIO.output(L2, GPIO.HIGH)
16
17 def left_forward():
18     GPIO.output(L1, GPIO.HIGH)
19     GPIO.output(L2, GPIO.LOW)
20
21 def left_backward():
22     GPIO.output(L1, GPIO.LOW)
23     GPIO.output(L2, GPIO.HIGH)
24
25 def right_stop():
26     GPIO.output(R1, GPIO.LOW)
27     GPIO.output(R2, GPIO.LOW)
28
```

```

29 #def right_stop():
30     # GPIO.output(R1, GPIO.HIGH)
31     # GPIO.output(R2, GPIO.HIGH)
32
33 def right_forward():
34     GPIO.output(R1, GPIO.HIGH)
35     GPIO.output(R2, GPIO.LOW)
36
37 def right_backward():
38     GPIO.output(R1, GPIO.LOW)
39     GPIO.output(R2, GPIO.HIGH)
40
41 def go_forward():
42     left_forward()
43     right_forward()
44
45 def go_backward():
46     left_backward()
47     right_backward()
48
49 def turn_left():
50     left_backward()
51     right_forward()
52
53 def turn_right():
54     left_forward()
55     right_backward()
56
57 def stop():
58     left_stop()
59     right_stop()
60 GPIO.setup(L1, GPIO.OUT)
61 GPIO.setup(L2, GPIO.OUT)
62 GPIO.setup(R1, GPIO.OUT)
63 GPIO.setup(R2, GPIO.OUT)

```

```

64 stop()
65 server = webiopi.Server(port=8000, login="msrit", password="
    electrical")
66 server.addMacro(go_forward)
67 server.addMacro(go_backward)
68 server.addMacro(turn_left)
69 server.addMacro(turn_right)
70 server.addMacro(stop)
71
72 webiopi.runLoop()
73 server.stop()
74
75 #GPIO.setmode(L1, GPIO.IN)
76 #GPIO.setmode(L2, GPIO.IN)
77 #GPIO.setmode(t, GPIO.IN)
78
79 #GPIO.setmode(R1, GPIO.IN)
80 #GPIO.setmode(R2, GPIO.IN)

```

project/manual.py

## .2.2 Code for autonomous operation

```

1 import time
2 import sys
3 import cv2.cv as cv
4 from optparse import OptionParser
5 import RPi.GPIO as GPIO
6 GPIO.setmode(GPIO.BOARD)
7 GPIO.setup(11, GPIO.OUT)
8 GPIO.setup(12, GPIO.OUT)
9 GPIO.setup(22, GPIO.OUT)
10 GPIO.setup(23, GPIO.OUT)
11 #GPIO.setup(13, GPIO.OUT)
12 #GPIO.setup(16, GPIO.OUT)

```

```

13 #GPIO.setup(15, GPIO.OUT)
14 min_size = (20, 20)
15 image_scale = 2
16 haar_scale = 1.2
17 min_neighbors = 2
18 haar_flags = 0
19
20 def detect_and_draw(img, cascade):
21     # allocate temporary images
22     gray = cv.CreateImage((img.width, img.height), 8, 1)
23     small_img = cv.CreateImage((cv.Round(img.width / image_scale),
24                                cv.Round (img.height / image_scale)), 8, 1)
25
26     # convert color input image to grayscale
27     cv.CvtColor(img, gray, cv.CV_BGR2GRAY)
28
29     # scale input image for faster processing
30     cv.Resize(gray, small_img, cv.CV_INTER_LINEAR)
31
32     cv.EqualizeHist(small_img, small_img)
33
34     if(cascade):
35         t = cv.GetTickCount()
36         faces = cv.HaarDetectObjects(small_img, cascade, cv.
37                                     CreateMemStorage(0),
38                                     haar_scale, min_neighbors,
39                                     haar_flags, min_size)
40
41         t = cv.GetTickCount() - t
42         print "detection time = %gms" % (t/(cv.GetTickFrequency()
43                                     *1000.))
44     if faces:
45         for ((x, y, w, h), n) in faces:
46             # the input to cv.HaarDetectObjects was resized, so
47             scale the

```



```

43 # bounding box of each face and convert it to two
    CvPoints
44 pt1 = (int(x * image_scale), int(y * image_scale))
45 pt2 = (int((x + w) * image_scale), int((y + h) *
    image_scale))
46 cv.Rectangle(img, pt1, pt2, cv.RGB(255, 0, 0), 3, 8,
    0)
47 z = int(x * image_scale)
48 print "z= %d" %z
49 if z<79:
50     GPIO.output(11, GPIO.HIGH)
51     GPIO.output(12, GPIO.LOW)
52     GPIO.output(22, GPIO.LOW)
53     GPIO.output(23, GPIO.LOW)
54     #GPIO.output(11, GPIO.HIGH)
55     #GPIO.output(12, GPIO.LOW)
56     #GPIO.output(13, GPIO.LOW)
57     time.sleep(0.05)
58 if z>80 & z<179:
59     GPIO.output(11, GPIO.HIGH)
60     GPIO.output(12, GPIO.LOW)
61     GPIO.output(22, GPIO.HIGH)
62     GPIO.output(23, GPIO.LOW)
63     #GPIO.output(11, GPIO.LOW)
64     #GPIO.output(12, GPIO.HIGH)
65     #GPIO.output(13, GPIO.LOW)
66     time.sleep(0.05)
67 if z>180 & z<260:
68     GPIO.output(11, GPIO.LOW)
69     GPIO.output(12, GPIO.LOW)
70     GPIO.output(22, GPIO.HIGH)
71     GPIO.output(23, GPIO.LOW)
72     #GPIO.output(11, GPIO.LOW)
73     #GPIO.output(12, GPIO.LOW)
74     #GPIO.output(13, GPIO.HIGH)

```

```

75         time.sleep(0.05)
76     else:
77         GPIO.output(11, GPIO.HIGH)
78         GPIO.output(12, GPIO.HIGH)
79         GPIO.output(22, GPIO.HIGH)
80         GPIO.output(23, GPIO.HIGH)
81
82
83
84 if __name__ == '__main__':
85
86     parser = OptionParser(usage = "usage: %prog [options] [filename |
            camera_index]")
87     parser.add_option("-c", "--cascade", action="store", dest="
            cascade", type="str", help="Haar cascade file, default %
            default", default = "../data/haarcascades/
            haarcascade_frontalface_alt.xml")
88     (options, args) = parser.parse_args()
89
90     cascade = cv.Load(options.cascade)
91
92     if len(args) != 1:
93         parser.print_help()
94         sys.exit(1)
95
96     input_name = args[0]
97     if input_name.isdigit():
98         capture = cv.CreateCameraCapture(int(input_name))
99     else:
100         capture = None
101
102
103
104     width = 320 #leave None for auto-detection
105     height = 240 #leave None for auto-detection

```

```

106
107     if width is None:
108         width = int(cv.GetCaptureProperty(capture, cv.
109             CV_CAP_PROP_FRAME_WIDTH))
110     else:
111         cv.SetCaptureProperty(capture, cv.CV_CAP_PROP_FRAME_WIDTH, width)
112
113     if height is None:
114         height = int(cv.GetCaptureProperty(capture, cv.
115             CV_CAP_PROP_FRAME_HEIGHT))
116     else:
117         cv.SetCaptureProperty(capture, cv.CV_CAP_PROP_FRAME_HEIGHT, height)
118
119     if capture:
120         frame_copy = None
121         while True:
122             frame = cv.QueryFrame(capture)
123             if not frame:
124                 cv.WaitKey(0)
125                 break
126             if not frame_copy:
127                 frame_copy = cv.CreateImage((frame.width, frame.height
128                     ),
129                     cv.IPL_DEPTH_8U, frame.
130                     nChannels)
131
132             # frame_copy = cv.CreateImage((frame.width, frame.
133             height),
134             # cv.IPL_DEPTH_8U, frame.
135             nChannels)
136
137             if frame.origin == cv.IPL_ORIGIN_TL:
138                 cv.Copy(frame, frame_copy)
139             else:

```

```
135         cv.Flip(frame , frame_copy , 0)
136
137         detect_and_draw(frame_copy , cascade)
138
139         if cv.WaitKey(10) >= 0:
140             break
141     else:
142         image = cv.LoadImage(input_name , 1)
143         detect_and_draw(image , cascade)
144         cv.WaitKey(0)
```

project/autol.py

## **.3**

### **APPENDIX C**

#### **functions used**

##### **.3.1 GPIO access**

In order to access the GPIO pins, you will need to import the RPi.GPIO library. To do this, insert `import RPi.GPIO` at the top of your Python source code. You will then need to prepend any method or value names from the library with `RPi.GPIO..`

`setmode(mode)` Set the numbering scheme for the GPIO pins. For all tutorials on this website, BCM is used as mode.

`setup(pin, direction, pullupdown PUDOFF )` Configure a particular GPIO pin as an input or output, optionally with a pull up/down resistor. direction is either IN or OUT. pullupdown can be PUDOFF, PUDUP or PUD-DOWN. Our modified version of RPi.GPIO allows pin 4 to be set up as a general purpose clock using `setup(4, ALT0)`.

`setclock(pin, frequency)` Use the chosen pin as a clock of the given frequency. Only works with pin 4.

`input(pin)` Receive the current state of the chosen pin.

`output(pin, state)` Write the given state to a pin. state can be HIGH, LOW, 1, 0, True or False.

### **.3.2 I2C**

In order to communicate with servos (amongst other things), you will need to import the I2C library. To do this, insert `import i2c` at the top of your Python source code. You will then need to prepend any method or value names from the library with `i2c`.

- `I2C(frequency=50)` Create an I2C instance which communicates at a given frequency. 50Hz is good for controlling servos, but anything between 40 and 1000Hz can be used. Higher frequencies give more control and avoid flicker.
- `I2C.setSpeeds(left, right)` When combined with our robot chassis, set the speeds of the two motors. Values range from -100 to 100, with 0 being stationary.
- `I2C.setPWM(pin, dutycycle)` Used to set servos to particular speeds, or LEDs to particular brightnesses. A dutycycle of 0 will give no brightness on an LED, 2047 will give half brightness, and 4095 will give full brightness.

### **.3.3 SimpleCV**

SimpleCV is a library used for efficient computer vision tasks. Documentation can be found [here](#). For the rest of the methods described here, you will need to import `imgproc`.

- `delay(msec)` Delay execution for msec milliseconds.
- `handleEvents()` Continually poll for events. Returns True on success, and False on quit.
- `class Viewer` Viewer class, handles the creation and manipulation of a window; you may only have one window open at a time.
- `Viewer(width, height, title)` Open an SDL window, ready to display an image.
- `Viewer.displayImage(image)` Display an image in the window.
- `del viewer` Close the view.
- `class Image` A container for image data, with pixel level access as well as high level functionality.
- `Image(imageptr)` Initialises an image based on a C pointer.
- `Image.copy()` Create a copy of an image container and its data, and return it.
- `Image.drawRect(x, y, w, h, r, g, b)` Draw a unfilled coloured rectangle on an image.
- `Image.chromaKey(rkey, gkey, bkey, threshold)` Test the image against a Chroma key, setting non-passing pixels to black.
- `Image.detectBlobs()` Detect blobs in the image, returning a list of tuples of (x, y, width, height) of the rectangle bounds of the discovered blob.

- `Image.detectFaces()` Detect faces in the image, returning a list of tuples of (x, y, width, height) of the rectangle bounds of the discovered face.
- `Image[index]` Get a tuple of the red, green and blue values of a pixel. Co-ordinates are provided as a tuple of their x and y position. Example: `red, green, blue = img[x, y]` will set red, green and blue to the respective intensities of the pixel.
- `Image[key] = value` Set the value of a pixel at a given co-ordinate provided in the tuple key. key is a tuple of the x and y position, and value is a 3 component tuple of the red, green and blue intensities. Example: `img[64, 32] = 255, 128, 0` sets the pixel at position x: 64 and y: 32 to a bright orange colour.
- `del image` Handles the destruction of the image data.
- `class Camera` Camera capture device: interfaces with a webcam allowing the user to grab images.
- `Camera(width, height)` Initialise the camera, setting the size of the capture to width pixels wide and height pixels high.
- `Camera.grabImage()` Grab a single image from the camera.
- `del camera` Free the camera capture device.



.4

## APPENDIX D

### Regulator Pin details



Figure 3: 7805 pin details



.5

## APPENDIX E

### Raspberry Pi GPIO pin details



Figure 4: Raspberry Pi pin details