Sheffield Hallam University

SHARPENS YOUR THINKING

Faculty of Arts, Computing, Engineering and Sciences

Engineering Projects
Module (16 – 6077)

Final Year Project Report 2014/2015

Television Remote Controller utilizing Gesture Recognition

Student name	N.S. Silva
Student ID	EN 12523788
Course title	B.Eng(Hons) Electronic Engineering
Supervisor	Dr. Lasantha Senevirathne

SHEFFIELD HALLAM UNIVERSITY

ENGINEERING PROGRAM

Television Remote Controller utilizing Gesture Recognition

By

N.S. SILVA

B.Eng(HONS) ELECTRONIC ENGINEERING

FINAL REPORT

MODULE: 16-6077

November 2014

Abstract

The project is to develop a television remote controller utilizing image processing. This will allow the user to perform various functions (i.e. change channels, change volume) using pre-defined gestures.

The device is based on a Raspberry Pi single board computer where the software runs. The camera connected to the Raspberry Pi captures a steady video stream, which is then analyzed frame-by-frame to identify the hand gestures performed. If a gesture is detected, the necessary command is transmitted to the TV via in infra-red LED which is connected to an Arduino development board. The Raspberry Pi and the Arduino are connected to each other via their serial ports.

The device is currently able to recognize four hand gestures. Namely left movement, right movement, up and down movements. The functions that are accessed through those four gestures are channel down, channel up, volume up and volume down respectively. The hand gestures are identified through a foreground extraction process which is followed by a contour identification. This is done when the presence of the hand is detected through a cascade classifier.

This report will serve as a documentation to explain the work done throughout the year in developing this device. The workings of the device as well as the theory components for the necessary functions are also explained in this report.

Table of Contents

Table of figures

List of Tables

1 Introduction

A remote controller is a component of an electronic device which will enable the user to control the functions on the said device from a remote location. In the case of a television set, the remote control enables the user to perform various function in the television (change channels, increase/decrease volume etc) wirelessly from a short line-of-sight distance. These functions are performed by pressing one or more buttons which are present on the remote control unit. Figure 1.1 shows some of the common TV remote controls available today. The most common method of communication is via Infra-red pulses where an infra-red emitter on the remote will send out a series of pulses in correspondence with the command and the infra-red receiver on the TV will read this command and perform the necessary command.

Figure 1.1: Various TV remote controls

The aim of this project is to develop a television remote control unit which will eliminate the function of the physical remote controller unit. The necessary commands will be accomplished using simple hand gestures rather than pressing buttons on a remote control unit.

The unit developed will operate on the concept of gesture recognition. The unit will employ a camera, which act as the primary capturing device for image processing. The camera will record a continuous video stream which will be analyzed by the software. The software will first identify the hand and then the direction of movement and thereby will recognize the gesture. The necessary command will then be transmitted to the TV via an infra-red emitter.

1.1 Image Processing

Digital image processing the application of a number of a computer algorithms to process a digital image. The outcome of the process can be either images or a set of representative characteristics or a set of commands based on the images. The applications of image processing are very common in the fields of robotics, medical imaging, remote sensing, photography and forensics.

An image processing system will consist of a frame grabber that is used to collect images and a computer which processes the images using necessary software packages and possibly some output devices.

An image is a collection of pixels. Each pixel will occupy a single color which is expressed by an RGB vector. Most of the functions performed on the image will be based on the RGB values of each pixel. Many complex algorithms can be applied on images to accomplish various tasks, do various modifications, identify various objects/ features/ shapes etc. figure 2 shows such an application of image processing; finding contours in an image. The image on the left is the original image, and the image on the right is the resultant image with the contours detected.

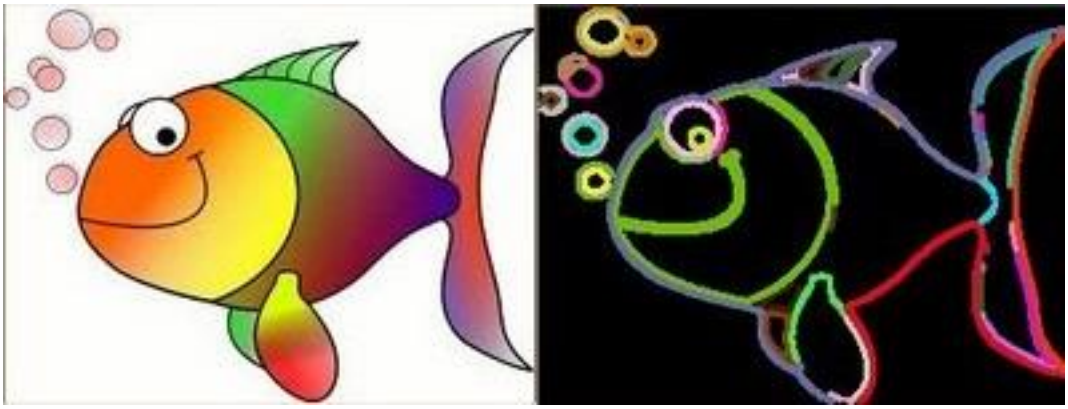

Figure 2: Contour detection

A number of functions of image processing are used in this project. Some are namely foreground subtraction, contour detection, conversion from one color space to another and many more. A functionality named Haar-like features is used for object detection. This will be further discussed in later chapters [1] [2].

1.2 Infra-red communication

Infra-red is very similar to visible light, except for its slightly higher wavelength. Due to this reason, IR light is undetectable to the human eye. However, IR light can be visually seen when looked at through a camera lens (figure 3). The invisibility to the human eye makes infra-red light perfect for wireless communication. A most common application of infra-red light for wireless transmission is the TV remote.

Figure 3: IR LED when looked at through a camera lens

When the TV remote is used to transmit a command, the IR LED in the remote repeatedly switches on and off according to the command that is transmitted. This IR signal is modulated so that the TV can pick out the IR signal which is from the remote from other sources of infra-red light. A very common modulation scheme for this application is termed as the 38 kHz modulation. There are very few natural sources that produce the regularity of a 38 kHz signal. Therefore, an IR transmitter sending data at a steady rate of 38 kHz could easily be distinguished from other ambient sources. The most common modulation scheme is 38 kHz but other frequencies can also be used.

When a button on the remote is pressed, the IR LED will blink rapidly for a very small duration. This will transmit the encoded data to the TV. Figure 4 shows a very crude example of what this stream of data may look like. The task of the receiver is to demodulate these signals and decode to a serial bit stream. In this project, an Arduino UNO module is used to transmit the necessary signals through an IR LED connected [3].

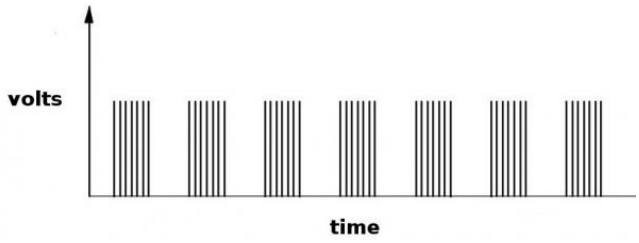

Figure 4: Transmitted signal

2 Literature review

2.1 Background

Gesture recognition interprets human gestures via mathematical algorithms. Utilizing gesture recognition to perform tasks of a TV remote is a very innovative idea as this makes it extremely easy to perform various functions on the TV. It also effectively cancels the mechanical devices which previously enabled the user to interact with the TV.

2.2 Current Systems

2.2.1 Gesture recognition remote control – MIT

This device has been developed by MIT. It utilizes gesture recognition to perform various functions on a *Sony* TV. The task is accomplished by want like instrument which is used to perform the gesture. The processor then reads this movement to map and performs the necessary actions. The user can use up to eight want movements to control various functions on the TV such as volume up/down, mute, power on/off etc.

2.2.2 Gesture Remote concept – Microchip Technology

This device too utilizes gesture recognition. It consists of a small remote like object with a sensitive surface. The user uses his thumb to make gestures in proximity to this object. These gestures include moving the thumb towards and away from the surface, motions of certain shapes and various other gestures which will control various functions of the television [4].

Figure 5: Gesture remote by microchip

2.2.3 Smart TV

The smart TV utilizes gesture recognition as well. The smart TV user will have to initiate this feature by performing a simple gesture. After this step, the user can perform any desired gestures which employ swipes, slides etc. This is quite similar to the project that is being developed. The only noticeable difference is that this function is embedded into the Smart TV while this project focuses on a more compact design which can be utilized in conjunction with different televisions [5].

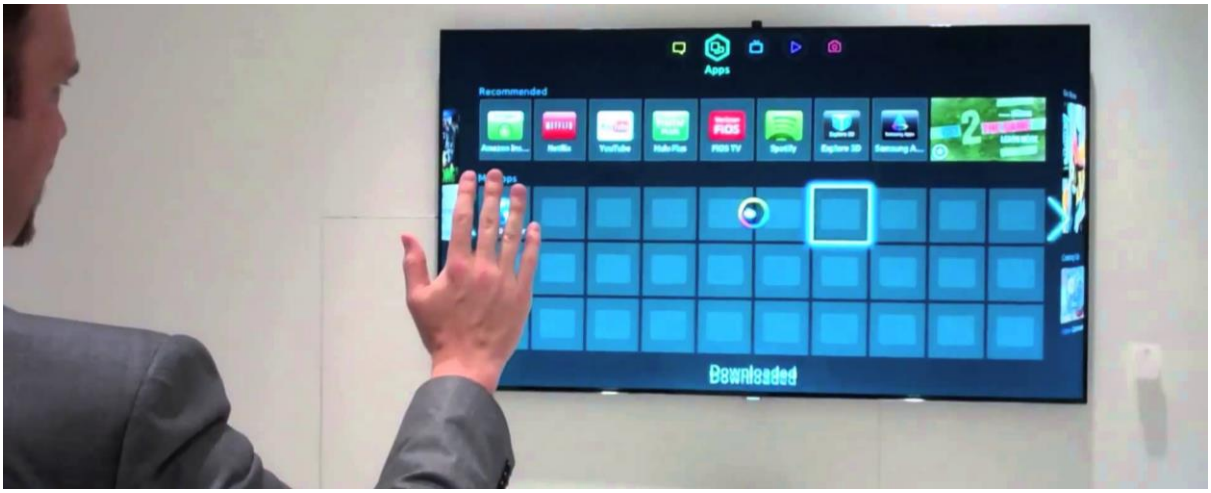

Figure 6: Controlling a Smart TV with a simple hand gesture

3 Approach

The unit runs a software written using C++ and OpenCV image processing libraries. The software was initially tested using a computer and webcam. And it was later transferred to the Raspberry Pi module. Figure 4 shows a simple block diagram of the program. An explanation of the flow chart will proceed the figure.

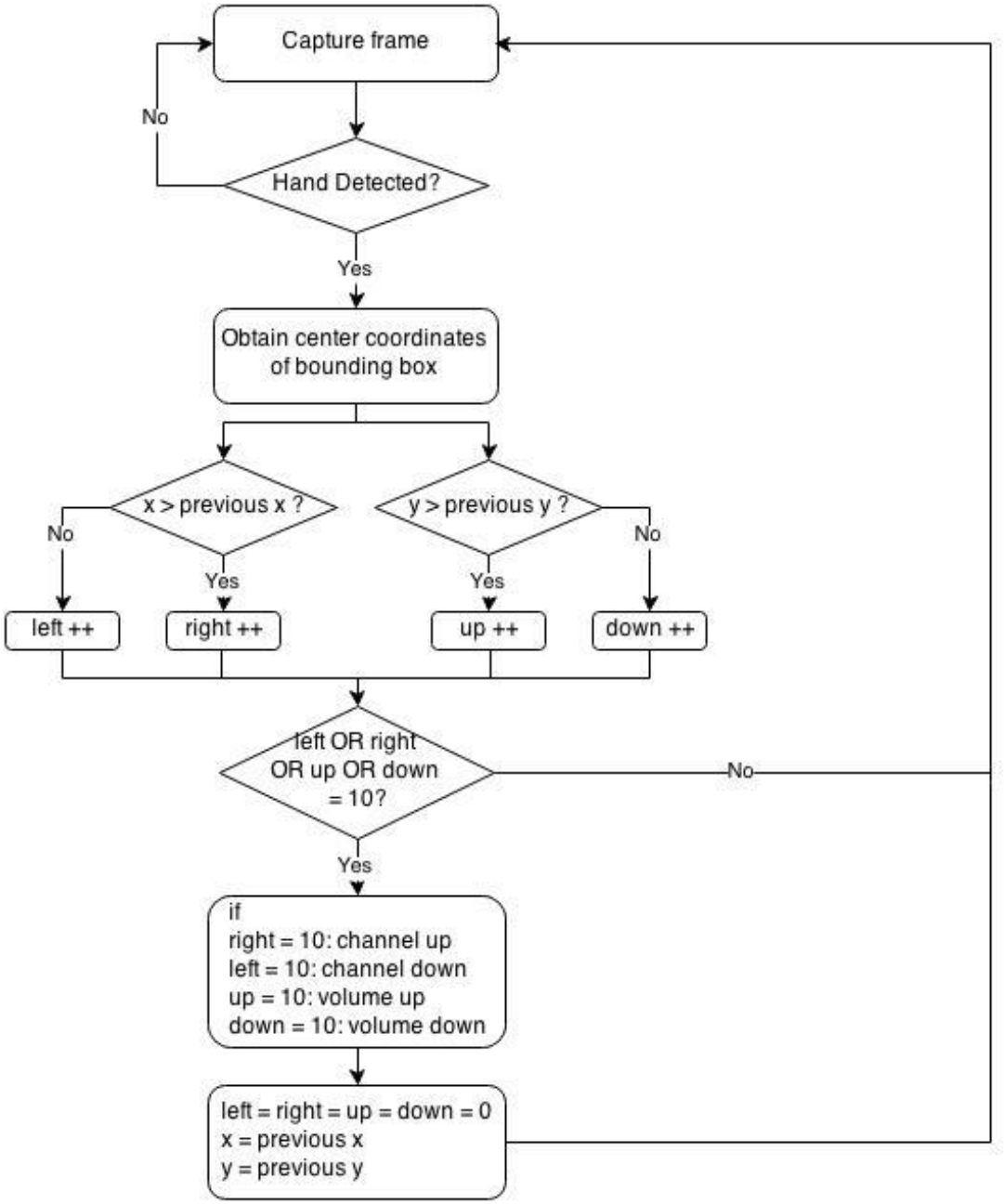

Figure 7: Flow chart of the program

3.1 Image Acquisition

As it is clearly seen from the flow chart, the first step is the acquisition of a single frame. Certain transformations are applied to this captured frame (refer appendix A for source code). These transformations are namely color conversion, histogram equalization and blurring. The operations are explained in detail in the Theory chapter.

The color conversion is used to convert the frame to a grayscale image. This is done to ensure that each pixel value is a single sample i.e. carrying only the intensity value. Following to this step, a histogram equalization is performed. This is used to increase the contrast of the image. Through this adjustment, the intensities can be better distributed on the histogram which allows for areas of lower contrast to gain a higher contrast.

The next step is the blurring of the captured frame. This is done to facilitate an easier edge detection. Figure 6 shows an example of the operations color conversion, histogram equalization and blur. The top right image is the original while the top left is the grayscale variant. The bottom left and right images are of the histogram equalization and blurring respectively.

Figure 8: Image transformations

3.2 Hand Detection and Motion Tracking

The next step is the detection of the hand. This may be said to be the most crucial step in the entire program. The initial approach used a contour detection, and assumed that the largest moving contour to be the hand. This proved to be ineffective as the program would recognize other moving objects as the largest contour and malfunction. To correct this, a cascade classifier was used to detect the palm of the hand. The software will begin tracking only when the palm is detected. Refer appendix B for the source code for this detection.

The gray scale converted frame is passed onto the function. Then a cascade classifier is used to detect the presence of a hand in the frame. The identified objects (the palm of the hand in this case) is returned as a list of rectangles. Figure 7 shows such an instance where the palm of the hand is identified. The bounding rectangle is also shown.

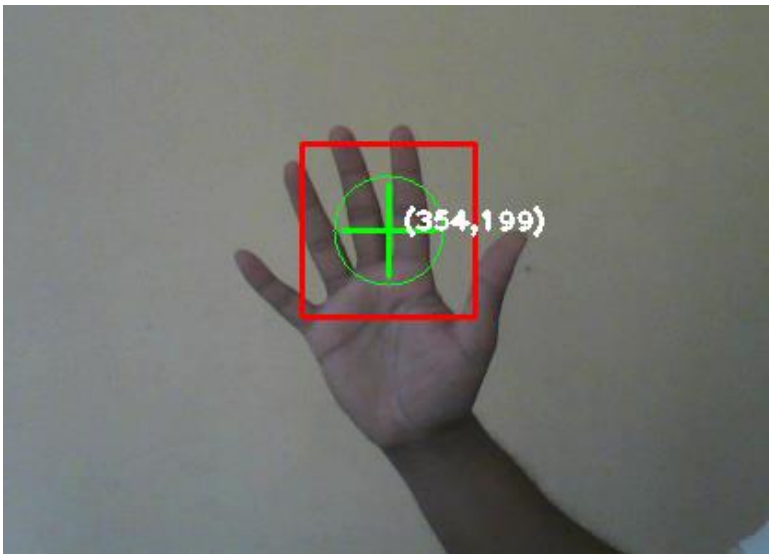

Figure 9: The palm of the hand is identified

The motion tracking can only begin when the presence of the hand is detected. Contour detection is used to accomplish this task. The program detects the largest contour and identifies it to be the hand. This task is accomplished under the assumption that the only moving object is the hand.

To identify moving objects, foreground subtraction must be performed. The method used in this is MOG2 which abbreviates for Method of Gaussian. The operation of the foreground subtraction method is explained in the Theory section (refer appendix A for the source code).

After the application of foreground subtraction, the resultant image will contain only the moving objects. Figure 8 shows an example for this process. The image on the left is the webcam feed, where the motion is tracked and illustrated using a crosshair. The image on the right is the foreground subtraction. It can clearly be seen that the background has been subtracted and the moving objects remain on the image.

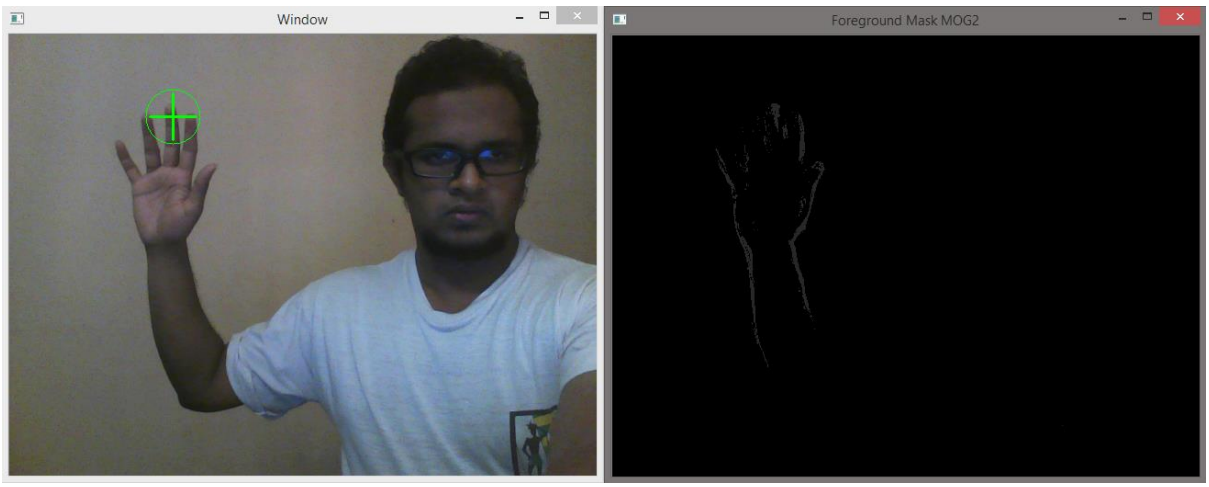

Figure 10: Foreground Subtraction, motion tracking

The motion is tracked using a contour detection (Appendix C). The program operates under the assumption that the hand will be the only moving object. The largest contour is identified from the foreground subtracted image, and a bounding box is drawn surrounding the largest contour. The center coordinates of this rectangle are then identified which will be used to deduce the direction of motion.

The direction of the motion is identified using the coordinates obtained. To accomplish this, 4 counters have been implemented. These are names left, right, up and down. The operation of this is shown in the flowchart at figure 5. The explanation of this process is as follows.

1. At startup, all four counters are set to 0
2. When the hand is detected, and the subsequent motion is detected, the two coordinates of the bounding rectangle are stored.
3. The coordinates are identified in the next frame, and these are compared with those from the previous coordinates obtained. The current coordinates are named x and y, and the previous coordinates are names prev_x and prev_y for ease of explanation.

4. If x > prev_x, then the counter ‘right’ is incremented by one.
5. If x < prev_x, then the counter ‘left’ is incremented by one.
6. If y > prev_y, then the counter ‘up’ is incremented by one
7. If y < prev_y, then the counter ‘down’ is incremented by one.
8. The current x and y coordinates are saved before scanning the next frame (x=prev_x) and y=prev_y).
9. If any of the counters reach the value 10, then it is identified as a motion in that particular direction, and the necessary command is transmitted to the TV via the infra-red LED connected to the Arduino UNO. If motion is in the same direction for 10 frames, then it it identified as a command.
10. After identifying a command, the four counters are set to 0.

3.3 Infra-red communication

The infra-red communication is accomplished using an Arduino UNO development board. The Arduino and PC/ Raspberry Pi communicates via their two serial ports. The Ardunio libraries developed by Ken Sherrif [4] is used for this.

Initially, the necessary command from the TV remote are captured and saved. This is accomplished through the Arduno as well (refer appendix E for the source code). Each key on the remote has a particular code which is transmitted when pressed. The IR LED is turned on and off periodically at a frequency at 38 kHz according to the code of the pressed key. To obtain these codes, an IR receiver was connected to pin 3 of the Arduino UNO. When the remote is pointed at the receiver and the necessary button is pressed, the bit stream transmitted is recorded and saved.

To transmit a necessary code, another Arduino program was written (appendix F). This program continuously scans for any data sent through the serial port. The main program was written to transmit a single 8 bit value through the serial port in correspondence with the command. If the command is ‘left’, an integer ‘l’ would be sent. Similarly, for the commands right, up and down; the integers ‘r’, ‘u’ and ‘d’ would be transmitted respectively.

The Arduino receives this data and identifies the function that is to be performed,

- If the received data is ‘l’, the function is channel down

- If the received data is ‘r’, the function is channel up
- If the received data is ‘u’, the function is volume up
- If the received data is ‘d’, the function is volume down

The Arduino then transmits the necessary code via the IR LED.

4 Theory

4.1 Color conversion

An image is composed of a large number of pixels, each pixel is represented by a RGB vector. Each having 3 elements. A grayscale image however is represented by single value pixels, where the pixel value shows the color intensity. This is again different from black and white images where the pixel values are of only two distinct levels. 0 for white, and 1 for black. Grayscale images have the pixel values representing various shades of gray, where the intensity is weakest at black, and strongest at white.

Grayscale pixel intensities can be expressed by using rational numbers, or percentages. However, in modern computing, an 8 bit binary value is used re represent the pixel intensity. This allows 256 different levels of intensity (shades of gray) to be stored. This is advantageous in image processing as a single pixel can be represented by a value occupying a single byte [5]. Figure 11 shows a full color RGB image, and the conversion of the same image to grayscale.

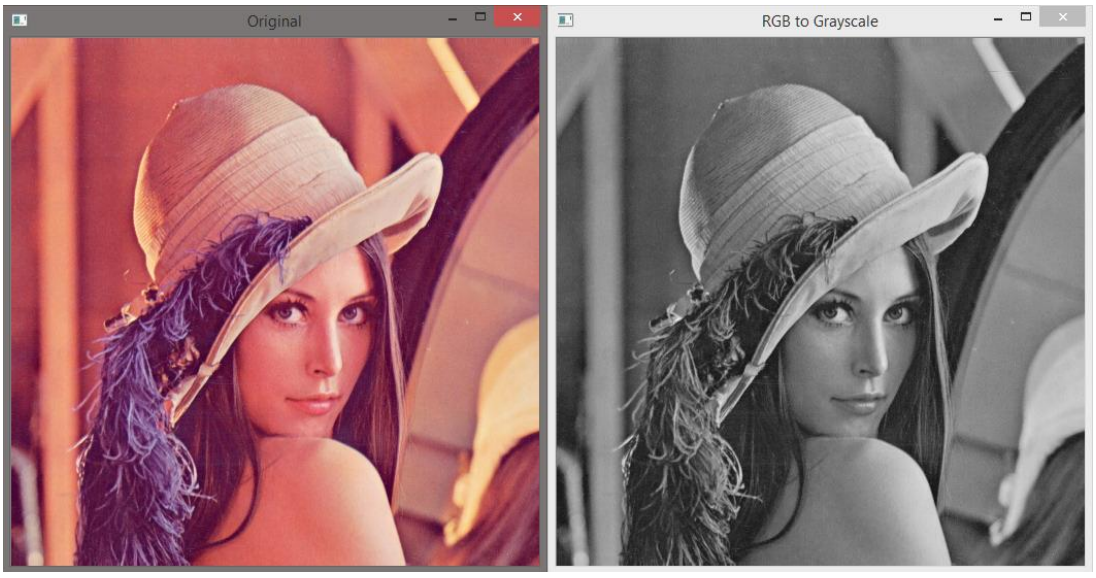

Figure 11: color and grayscale images

4.2 Histogram Equalization

Histogram equalization is a method used to adjust the contrast of an image by using its histogram. An image histogram is a graphical representation of the intensity distribution of an image. The histogram will quantify the number of pixels for each intensity value considered. Figure 12 shows an image and its histogram. By observing the histogram, it can be seen that a very small number of pixels exist with zero intensity or maximum intensity.

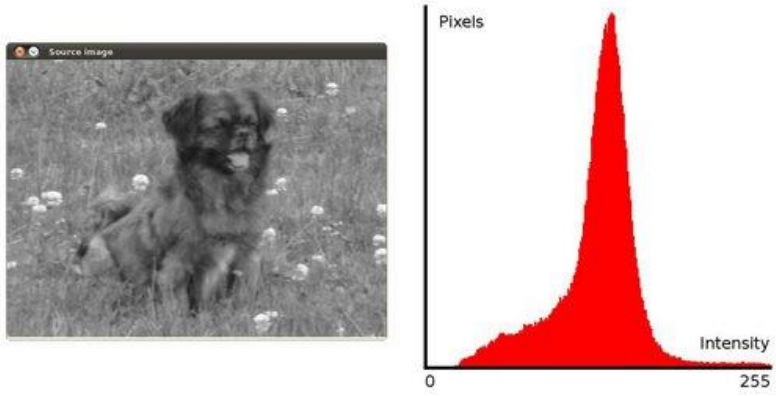

Figure 12: Grayscale image and its histogram

When a histogram equalization is performed, the intensity range is stretched out to improve the contrast in the image. Considering figure 12, it can be seen that a large number of pixels seem to be clustered around the middle intensity levels. Histogram equalization will stretch out this range (figure 13). By observing figure 13, it can be seen that the underpopulated intensity regions are now filled after stretching of the histogram. The resultant image is also shown in figure 13 [6].

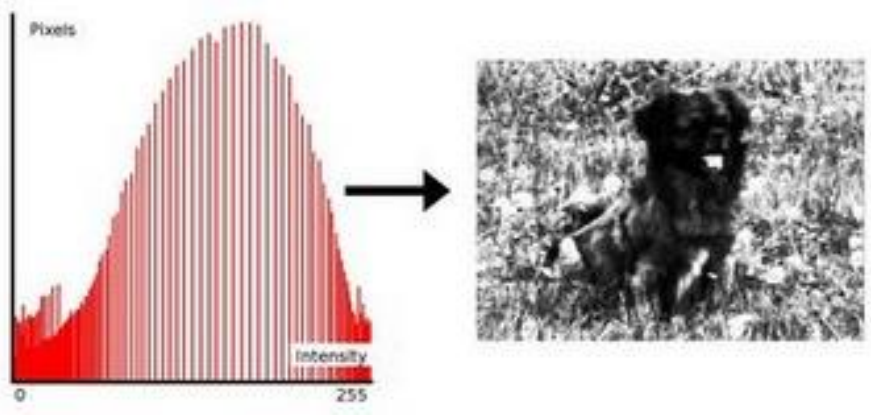

Figure 13: Histogram equalization and the resultant image

4.3 Blur

The blur technique used in this project is accomplished by using a normalized box. This technique implies to the blurring of an image by a Gaussian function. This technique was used to reduce the noise level and the level of detail of the image for easier identification of contours.

To perform this operation, a filter needs to applied to the image. The filter used in this project is a normalized box filter. When applied, the output pixel's value will be determined as a weighted sum of input pixel values. The normalized box filter is the simplest filter where each output pixel is the mean of its kernel neighboring pixels. All the pixels contribute with equal weights. Figure 14 shows a blur which is applied to the image shown in figure 11 [7].

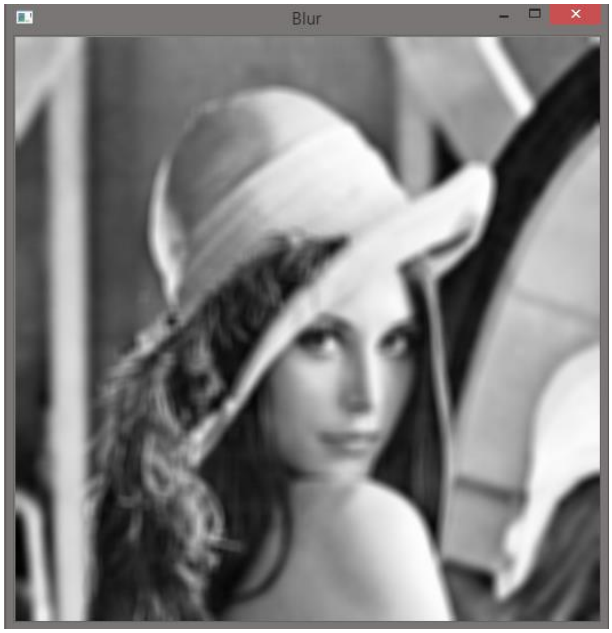

Figure 14: Smoothened image

4.4 Foreground Extraction

When the camera is recording a continuous stream of frames, the background will remain mostly unchanged. This is true when the camera is stationary. Foreground extraction is the extraction of moving objects. A model of the background should be built to extract moving objects by comparison of each frames with the foreground model.

The method used for foreground extraction in this project is MOG2. Figure 15 shows the extraction of a foreground object by using the background mask. We can clearly see the background model from which the current is subtracted. After applying a threshold, the foreground mask is obtained.

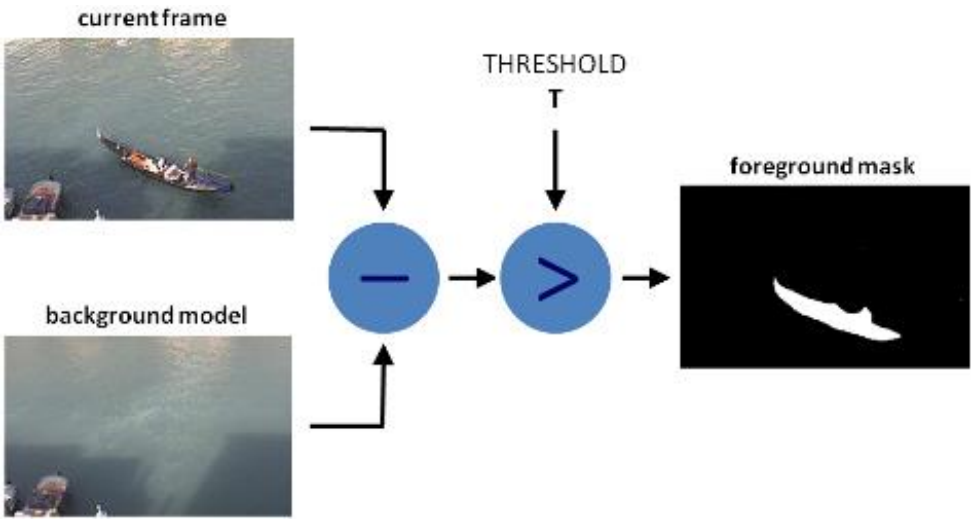

Figure 15: Foreground extraction

MOG2 is a Gaussian mixture based background/ foreground segmentation algorithm. This method uses a method to model each background pixel by a mixture of Gaussian distributions. The weights of the mixture will represent the time proportion each color will remain. The probable background colors are ones which stay for long durations and are static. The importance of MOG2 algorith over MOG algorithm is that MOG2 selects the appropriate number of Gaussian distributions for each pixel whereas a fixed number of distributions are used in MOG algorithm. MOG2 provides better adaptability when the illumination tends to change rapidly and every frame is used for calculation of the foreground mask and updating of the background model[8] [9].

A background objet needs to be established to use this algorithm. This is accomplished by the OpenCV function cv::createBackgroundSubtractorMOG2(). There are some optional parameters which can be set manually. Some of these parameters are the length of history, number of Gaussian

mixtures, threshold etc. When the background subtractor object is created. The option of detecting shadows can be toggled on or off as desired. Figures 16 and 17 show a random frame from a video stream and the foreground extraction performed on it using the MOG2 algorithm. It can be seen that the background is represented in black and the moving objects are represented in white. This method was used in the project to identify the motion of the hand and then to facilitate tracking of the motion.

Figure 16: Random frame from a video stream

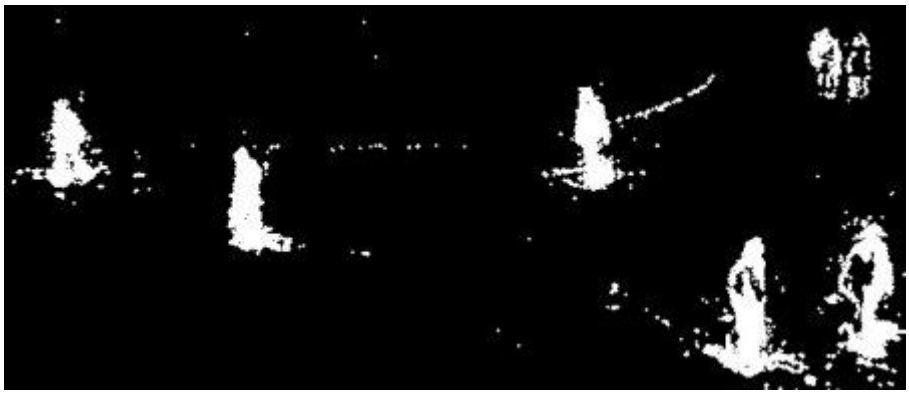

Figure 17: MOG2 foreground extraction

4.5 Cascade Classifier

A cascade classifier is form of ensemble learning, where the concentration of several classifiers are used for learning and detection. In operation, all the information obtained from the output of a given classifier are considered as additional information for the next classifier in the cascade. This method was proposed by Paul Viola and Michael Jones.

This project uses a Haar feature based cascade classifier. This is a machine learning technique where the cascade function is trained from a number of positive and negative images. The classifier is used to detect objects (hands in this case) in other images. For the training of the classifier, about 600 positive images and about 1000 negative images were used. To extract features from each image, Haar-like features are used. These features are shown in figure 18. Each feature is a single value which is obtained by subtracting the sun of pixels under the white rectangle from the sum of pixels under that black rectangle.

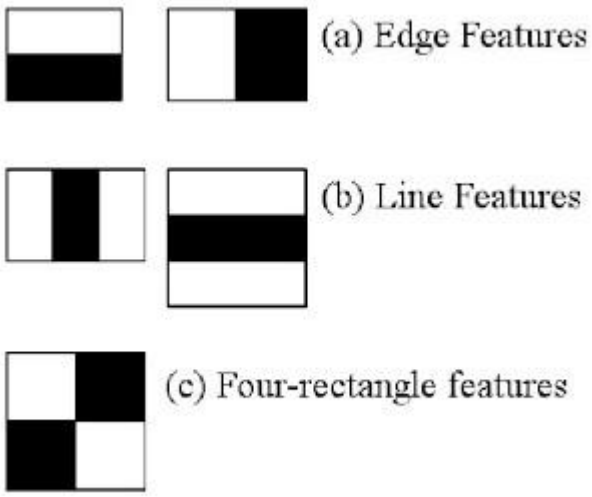

Figure 18: Haar features

All possible sizes and locations of each kernel are used to calculate a number of features. For each feature calculation, the sum of pixels under white and black rectangles needs to be calculated. However, it can be observed that some features obtained are irrelevant. This can be explained using a face-recognition example shown in figure 19. The first feature shows that the region of the eyes are generally darker than the area of the nose and cheeks. The second feature shows that the eyes are generally darker than the bridge of the nose. If these windows were applied in other places, the results obtained would be irrelevant.

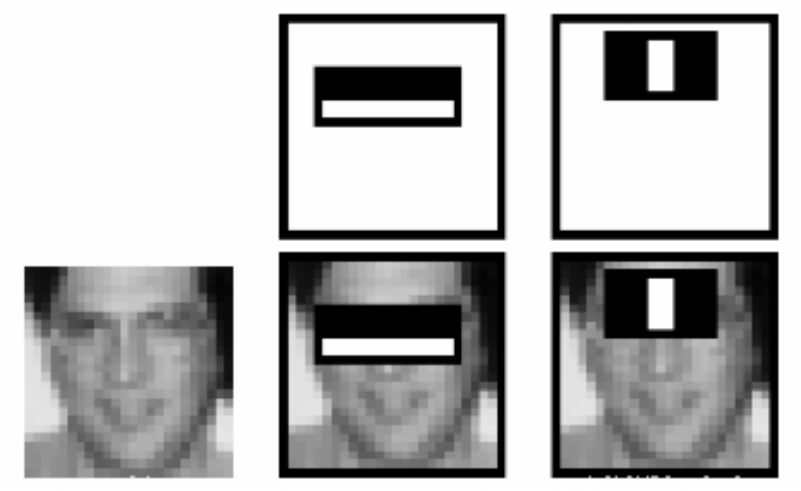

Figure 19: Haar features

To minimize these, each feature is applied on all training images initially. The best threshold is found for each feature where the objects can be classified into positives and negatives. To reduce any errors, the features with a minimum error rate are selected. To elaborate this process; each image will have equal weights initially, weights of misclassified images are increased after each classification. The same process is repeated and error rates and new weights are calculated. This process is repeated until the required error rate is obtained.

For ease of explanation, the face-recognition example is used for further explanation of Haar-like features. If a random image is selected. The region of the face is a very small area compared to the non-face region. Therefore more prominence is given to regions where there is a possibility of finding a face. Cascade classifiers play a crucial rose in this aspect. Instead of applying all extracted features on the image, the features are grouped into different stages of classifiers and applied. If the window in an image fails the first stage, it is discarded. If it passes, it is passed on to the second stage. The window which passes all stages is recognized as the object (in this example, the face) [10] [11].

4.6 Thresholding

Thresholding is the separation of an image into several section based on the threshold value set. The intensity of each pixel is compared with the threshold value. If it is greater than the threshold value it is assigned to a certain value corresponding to the thresholding type. If it is less than that of the threshod value, it is again assigned to a different value corresponding to the threshold type. Figure 20 shows a simple thresholding applied to a test image. OpenCV offers several thresholding types [12].

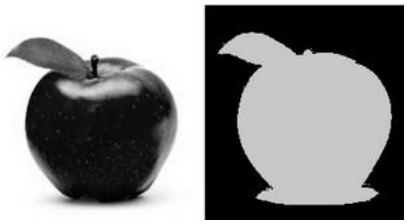

Figure 20: Thresholding

4.7 Contours

A contour can be defined as a list of points the represents a curve in an image which joins all the continuous points with the same color or intensity. These points are joined along the boundary. Contours are extremely useful in shape analysis and object detection/recognition. Blurring the image and applying a threshold makes it easier to find a contour on an image. Figure 20 shows a test image which is passed through a contour detection. The figure shows the two types of contours; exterior contours (dashed lines) and holes (dotted lines) [12].

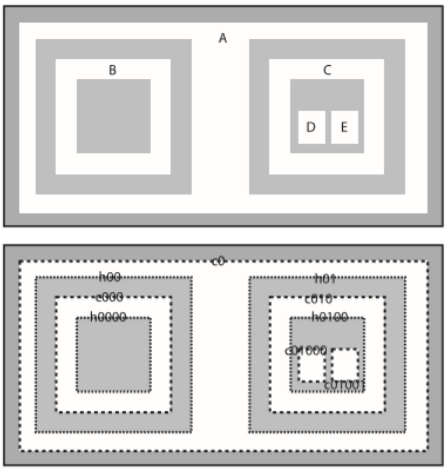

Figure 21: Contours in a test image

5 Results

The operation of the unit is as follows.

1. Performing a hand gesture
2. Recognition of the hand
3. Recognition of the gesture
4. Transmitting the relevant command via IR

The initial step was to set the Arduino up with the proper IR codes. These codes were obtained by using the actual remote from the TV being used and a SM0038 IR decoder. The IR decoder detects the data stream passed by the IR LED and converts them into a series of electrical pulses. The Arduino then reads them and displays them for further use. Figure 22 shows a screenshot of this process. Here the OFF column represents the duration for which the signal was low, and the ON column represents the duration for which the signal is high.

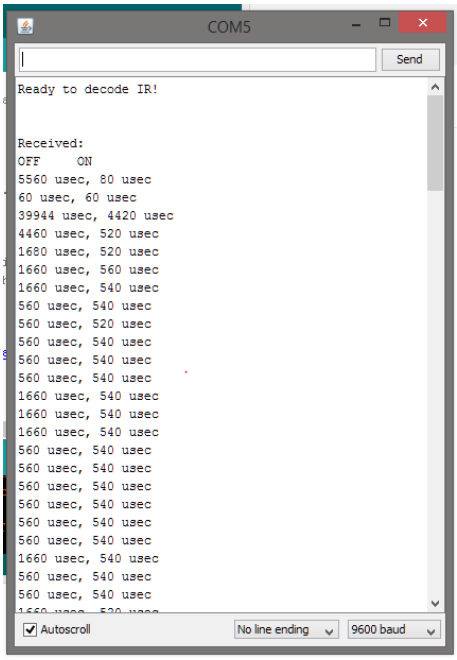

Figure 22: Decoding the IR signal

Initially to test the motion tracking, the program was modified to print out the identified gesture instead of transmitting to the Arduino. The program was found to be working in this instance. Figure 23 shows a screen capture of the operation of this program.

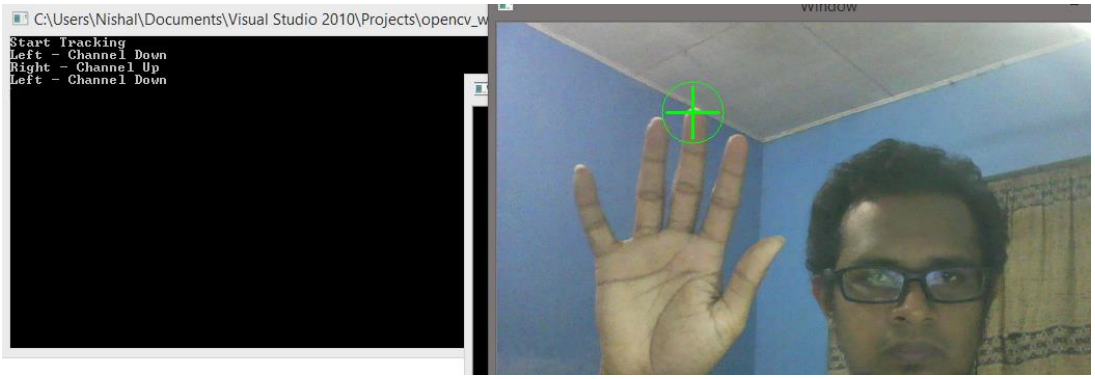

Figure 23: Testing of the program

After testing, the program was run along with the Arduino containing the infra-red code transmitting program. The desired results were obtained.

- Swipe hand to the right – channel up
- Swipe hand to the left – channel down
- Swipe hand upwards – volume up
- Swipe hand downwards – volume down

There were a few issues with the operation. One of the main issues was the hand recognition. The Cascade classifier would sometime recognize different objects as the hand. This problem could be resolved by training the classifier with more positive and negative images.

Another small issue arose with the contour detection. As the system assumes that the largest contour as the hand, if different larger object movement occurs, the system would recognize that as the hand movement. These issues are further discussed in the Critique section.

6 Critique and Further Development

The following is an excerpt from the initial description of the project (source – Project Proposal)

The objective of this project is to develop a gesture recognition system that will be used to perform various functions on a TV. A camera, mounted on the TV will record the images and these images will be continuously scanned by the control unit.

To start up the unit the user will perform a simple wave gesture. When this gesture is detected, the system is kept alert for further gestures.

The user will then perform the desired gesture. The gestures will consist of swipes in all 4 directions, finger patterns and various other gestures which are assigned to various functions.

The control unit will perform the necessary algorithms and deduce the action that is to be done for the gesture performed. The system will then send a binary data stream to the TV via an Infra-Red transmitter to the TV. This data stream will be decoded by the microprocessor inside the TV and the necessary action will be executed.

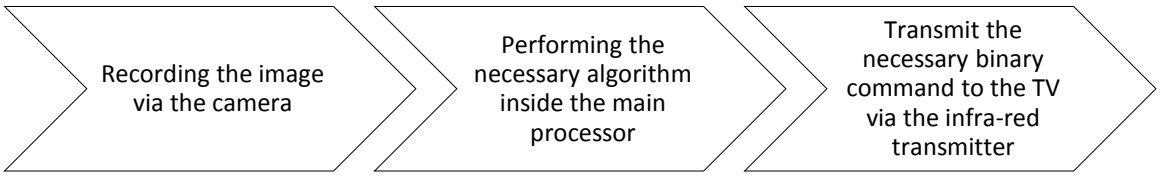

Figure 6.1: Basic bock diagram of the process

The flow of the block diagram can be explained as follows:

- Recording the image via the camera:

  The camera which is mounted on top of the TV will be transmitting images to the control module from the moment it powers on. The number of such images that can be captured depends on the camera. A camera with a decent resolution and a good frame rate will be selected for this process. The resolution too is very important since the user will be some distance away from the TV at all times.

- Performing the algorithms:

  The images which are sent to the control module are scanned continuously. If a pre-defined gesture is performed and captured, it will be identified. Then processor will then identify the function assigned to this gesture.

- Transmit the necessary binary command to the TV via the infra-red transmitter:

  The processor will identify the binary code corresponding to the function that is needed to be performed. This binary code will then be transmitted to the TV via an infra-red transmitter. As shown in figure 1.2, this message signal will consist of 4 parts. Initially at the gesture detection, a sequence of data sent to inform the device that a command is oncoming. Then the actual command and the device address are sent. Finally the stop sequence is sent to the TV to inform that the action is now over.

  The binary codes corresponding to the same action by different TV manufacturers could be different. Therefore the codes of the actions which are to be performed should be stored in a database. Initially the user will select the make of the TV, and the processor will select the binary codes corresponding to that make from the database and use them.

It can be seen that most of the initial expectations have been met, and that the device functions as initially proposed.

The initial plan consisted of a finger pattern identification method as well. This functionality was successfully obtained in a separate program where the program could count the number of fingers held. This was accomplished using convexity defects and contour detection. Some difficulties arose in merging the two programs. Also the foreground extraction algorithms used were different in the two programs.

The algorithm used for foreground extraction in this instance was called Codebook. Codebook operates by studying the background for a previously specified number of frames [16]. If any moving objects occur in the training period, the program would not function properly. Although

the foreground extraction was better than the currently used algorithm, practical difficulties arose in using the codebook method. The finger pattern recognition was discarded due to the above mentioned reasons. Figure 23 shows some screenshots of this program where the number of fingers were successfully counted. Figure 24 shows the foreground extraction by codebook method.

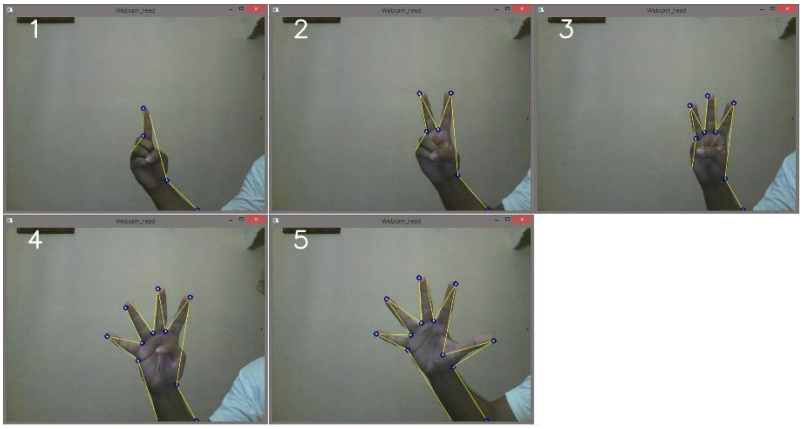

Figure 2: Counting the number of fingers held

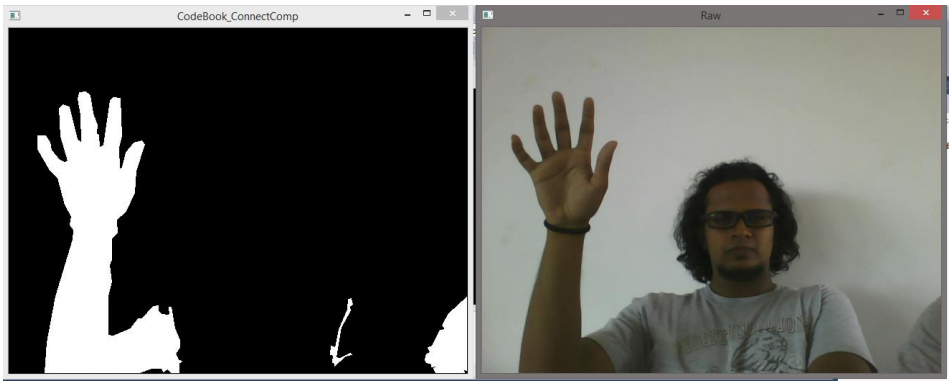

Figure 3: Codebook foreground extraction

Another practical error that arose was with the distance. Currently, for perfect operation, the user should be in close proximity to the camera (5 – 6 feet distance). This is a main issue that is expected to be developed.

Another possible development is to include some more functionalities i.e. two handed gestures, so that more functions may be controlled with hand gestures. Further work is hoped to be done on the finger pattern recognition and incorporate that into the device as well.

Currently, the device works with a Samsung TV. It could be programmed to work with any TV set, but this would involve plugging the device into a computer and modifying the program. This is a serious drawback in presenting this as a consumer product. Suggested solutions for this issue are;

- To save a database of remote control code sets where the user can select the necessary once. This functionality will be similar to that of a current universal remote control unit.
- To incorporate an IR detector into unit as well. Thereby, the user can use the current remote control unit to transmit the necessary codes to the device and save them. This will enable the initial setup to be extremely simple, and also to use the device with any television set which used an infra-red remote controller.

Another long term development idea is to incorporate more wireless communication methods (i.e. Bluetooth) to the unit to broaden its usage.

7 Costing

The costing for the components were as follows. Since all the programming and circuit assembly was done in-house, no extra costs were incurred.

Table 1: Costing

Components	**Price (Rs.)**
Raspberry Pi single board computer	7000.00
IR sensor + LED	250.00
Camera	4000.00
Arduino UNO development board	1700.00
Miscellaneous components	1000.00
Total estimated cost	13950.00

If the product is to be marketed, it is recommended to use a programmable microcontroller instead of the Raspberry Pi and the Arduino boards. This will reduce the cost significantly and the unit could be marketed for a competitive price.

8 Conclusion

The development of this project was a tremendous experience to me, both academically and professionally. The ability to divide a project into sub-projects so as to make maximum use of the allocated time was an extremely important experience to me.

Image processing was a totally new topic to me. I have gathered a fair amount of knowledge on the subject and its sub sections. I hope to gather more knowledge and pursue in the field of Image Processing.

Another one of my main intentions was to develop my project to a complete and marketable product. I believe I can achieve this goal very soon with some modifications and improvements and I hope that this product will attract much interest with the public as this will revolutionize the TV-user interaction.

9 References

[1] H. Zhou, J. Wu and J. Zhang, Digital Image Processing -part 1, Jianguo Zhang and Ventus Publishing ApS, 2010.

[2] "Finding contours in your images," [Online]. Available: http://docs.opencv.org/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html.

[3] "IR Communicatoin," [Online]. Available: https://learn.sparkfun.com/tutorials/ir-communication.

[4] "Gesture Remote," [Online]. Available: http://www.gesture-remote.com/.

[5] "Samsung Smart TV- Guide Book," [Online]. Available: http://www.samsung.com/global/microsite/tv/common/guide_book_5p_vi/move.html.

[6] "Multi Protocol infrared remote library," [Online]. Available: http://www.righto.com/2009/08/multi-protocol-infrared-remote-library.html.

[7] "Grayscale," [Online]. Available: http://en.wikipedia.org/wiki/Grayscale.

[8] "Histogram Equalization," [Online]. Available: http://docs.opencv.org/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html.

[9] "Smoothing Images," [Online]. Available: http://docs.opencv.org/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html.

[10] R. Laganière, OpenCV 2 Computer Vision Application Programming Cookbook, Mumbai: Packt Publishing, 2011.

[11] Z.Zivkovic, "Improved adaptive Gausian mixture model for background subtraction," 2004.

[12] "Face Detection using Haar cascades," [Online]. Available: http://docs.opencv.org/trunk/doc/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html.

[13] P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features," 2001.

[14] "Basic Thresholding Operations," [Online]. Available: http://docs.opencv.org/doc/tutorials/imgproc/threshold/threshold.html.

[15] G. Bradski and A. Kaehler, Learning OpenCV: Computer Vison wth the OpenCV Library, O'Reilly, 2008.

[16] K. Kim, T. H, D. Harwood and L. Davis, "Real Time foreground-background segmentation using codebook method".

11 Appendices

11.1 Appendix A

```
	while(1){

		cap>>frame;

		cvtColor(frame, frame_gray, CV_RGB2GRAY );
		equalizeHist( frame_gray, frame_gray );
		blur(frame_gray, frame_gray, Size(10,10));

		pMOG2->operator()(frame_gray, fgMaskMOG2, 0.04);

		threshold(fgMaskMOG2, fgMaskMOG2, 40, 100, 2);


//		imshow("Grayscale", frame_gray);
		imshow("Foreground Mask MOG2", fgMaskMOG2);

		if(trackingEnabled){
//			palmDetect(frame_gray, frame, fgMaskMOG2);
			searchForMovement(fgMaskMOG2, frame);
		}

		imshow("Window", frame);

		if(object[0]>prev_object[0])	r=r+1;
		if(object[0]<prev_object[0])	l=l+1;
		if(object[1]>prev_object[1])	u=u+1;
		if(object[1]<prev_object[1])	d=d+1;

//---------------------------------------
		if(r>=10){
			WriteComPort("COM3", "u");
			cout<<"Left - Channel Down"<<endl;
			l=0; r=0; d=0; u=0;
		}

		if(l>=10){
			WriteComPort("COM3", "d");
			cout<<"Right - Channel Up"<<endl;
			l=0; r=0; d=0; u=0;
		}

		if(u==20){
			WriteComPort("COM3", "Down");
			cout<<"Down"<<endl;
			l=0; r=0; d=0; u=0;
		}

		if(d==20){
			WriteComPort("COM3", "Up");
			cout<<"Up"<<endl;
			l=0; r=0; d=0; u=0;
		}
```

```
//---------------------------------------
		switch(waitKey(10)){
			case 116: //start tracking the biggest contour at key press 't'
				trackingEnabled = !trackingEnabled;
				if(trackingEnabled == true) cout<<"Start Tracking."<<endl;
				break;

			case 033: //stop tracking at key press 'esc'
				trackingEnabled = !trackingEnabled;
				if(trackingEnabled == false) cout<<"Tracking
disabled."<<endl;
				break;
		}

		waitKey(10);

	}			//end while loop

}				//end main
```

11.2 Appendix B

```
void palmDetect(Mat frame, Mat &cameraFeed, Mat fgMaskMOG2){

	Mat temp;
	frame.copyTo(temp);
	vector<Rect> palm;

		palm_cascade.detectMultiScale(temp, palm, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE,
Size(30, 30));

	for(size_t i=0; i<palm.size(); i++){

//		x = palm[i].x + palm[i].width/2;
//		y = palm[i].y + palm[i].height/2;
//		Point center(x,y);
		Point v1(palm[i].x, palm[i].y);
		Point v2(palm[i].x+palm[i].width, palm[i].y+palm[i].height);

//		ellipse(cameraFeed, center, Size( faces[i].width*0.5,
faces[i].height*0.5), 0, 0, 360, Scalar(0,0,255), 4, 8, 0);
		rectangle(cameraFeed, v1, v2, Scalar(0,0,255), 2, 8, 0);

	int xpos = palm[i].x + palm[i].width/2;
	int ypos = palm[i].y + palm[i].height/2;

	object[0] = xpos;
	object[1] = ypos;

	//make some temp x and y variables so we dont have to type out so much
	int x = object[0];
	int y = object[1];
```

```
        //draw crosshairs
        circle(cameraFeed, Point(x,y), 30, Scalar(0,255,0), 1);
        line(cameraFeed, Point(x,y), Point(x,y-25), Scalar(0,255,0), 2);
        line(cameraFeed, Point(x,y), Point(x,y+25), Scalar(0,255,0), 2);
        line(cameraFeed, Point(x,y), Point(x-25,y), Scalar(0,255,0), 2);
        line(cameraFeed, Point(x,y), Point(x+25,y), Scalar(0,255,0), 2);

        putText(cameraFeed," (" +
intToString(x)+","+intToString(y)+")",Point(x,y),1,1,Scalar(255,255,255),2);


//            searchForMovement(fgMaskMOG2, cameraFeed);

        if(object[0]>prev_object[0])      r=r+1;
        if(object[0]<prev_object[0])      l=l+1;
        if(object[1]>prev_object[1])      u=u+1;
        if(object[0]<prev_object[0])      d=d+1;

        prev_object[0] = object[0];
        prev_object[1] = object[1];
        }


}
```

11.3 Appendix C

```
void searchForMovement(Mat thresholdImage, Mat &cameraFeed){

        Mat temp;
        thresholdImage.copyTo(temp);

        vector< vector<Point>> contours;
        vector<Vec4i> hierarchy;

//      findContours(temp, contours, hierarchy, CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE
);
        findContours(temp, contours, hierarchy, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE );

        vector< vector<Point>> largestContourVec;
        largestContourVec.push_back(contours.at(contours.size()-1));

        objectBoundingRectangle = boundingRect(largestContourVec.at(0));

        int xpos = objectBoundingRectangle.x + objectBoundingRectangle.width/2;
        int ypos = objectBoundingRectangle.y + objectBoundingRectangle.height/2;

        object[0] = xpos;
        object[1] = ypos;
```

```
        //make some temp x and y variables so we dont have to type out so much
        int x = object[0];
        int y = object[1];


        //draw crosshairs
        circle(cameraFeed, Point(x,y), 30, Scalar(0,255,0), 1);
        line(cameraFeed, Point(x,y), Point(x,y-25), Scalar(0,255,0), 2);
        line(cameraFeed, Point(x,y), Point(x,y+25), Scalar(0,255,0), 2);
        line(cameraFeed, Point(x,y), Point(x-25,y), Scalar(0,255,0), 2);
        line(cameraFeed, Point(x,y), Point(x+25,y), Scalar(0,255,0), 2);

}
```

11.4 Appendix D

```
bool WriteComPort(CString PortSpecifier, CString data){

        DCB dcb;
        DWORD byteswritten;
        HANDLE hPort = CreateFile(
                PortSpecifier,
                GENERIC_WRITE,
                0,
                NULL,
                OPEN_EXISTING,
                0,
                NULL
                );

        if (!GetCommState(hPort,&dcb)){
                cout<<"Communication port not found"<<endl;
                return false;
        }

        dcb.BaudRate = CBR_9600;  //9600 Baud
        dcb.ByteSize = 8;   //8 data bits
        dcb.Parity = NOPARITY;  //no parity
        dcb.StopBits = ONESTOPBIT;  //1 stop

        if (!SetCommState(hPort,&dcb)){
                cout<<"Communication not successful"<<endl;
                return false;
        }

        bool retVal = WriteFile(hPort, data, 1, &byteswritten, NULL);
        CloseHandle(hPort);   //close the handle
        return retVal;
}
```

11.5 Appendix E

```
#define IRpin_PIN PIND
#define IRpin 2

#define MAXPULSE 65000

#define RESOLUTION 20


uint16_t pulses[100][2];
uint8_t currentpulse = 0;

void setup(void) {
  Serial.begin(9600);
  Serial.println("Ready to decode IR!");
}

void loop(void) {
  uint16_t highpulse, lowpulse; // temporary storage timing
  highpulse = lowpulse = 0; // start out with no pulse length


    while (IRpin_PIN & (1 << IRpin)) {
     // pin is still HIGH

     // count off another few microseconds
     highpulse++;
     delayMicroseconds(RESOLUTION);

     // If the pulse is too long, we 'timed out' - either nothing
     // was received or the code is finished, so print what
     // we've grabbed so far, and then reset
     if ((highpulse >= MAXPULSE) && (currentpulse != 0)) {
       printpulses();
       currentpulse=0;
       return;
     }
  }
  // we didn't time out so lets stash the reading
  pulses[currentpulse][0] = highpulse;

  // same as above
  while (! (IRpin_PIN & _BV(IRpin))) {
     // pin is still LOW
     lowpulse++;
     delayMicroseconds(RESOLUTION);
     if ((lowpulse >= MAXPULSE) && (currentpulse != 0)) {
       printpulses();
       currentpulse=0;
       return;
     }
  }
  pulses[currentpulse][1] = lowpulse;

  // we read one high-low pulse successfully, continue!
  currentpulse++;
}
```

```
void printpulses(void) {
  Serial.println("\n\r\n\rReceived: \n\rOFF \tON");
  for (uint8_t i = 0; i < currentpulse; i++) {
    Serial.print(pulses[i][0] * RESOLUTION, DEC);
    Serial.print(" usec, ");
    Serial.print(pulses[i][1] * RESOLUTION, DEC);
    Serial.println(" usec");
  }

  // print it in a 'array' format
  Serial.println("int IRsignal[] = {");
  Serial.println("// ON, OFF (in 10's of microseconds)");
  for (uint8_t i = 0; i < currentpulse-1; i++) {
    Serial.print("\t"); // tab
    Serial.print(pulses[i][1] * RESOLUTION / 10, DEC);
    Serial.print(", ");
    Serial.print(pulses[i+1][0] * RESOLUTION / 10, DEC);
    Serial.println(",");
  }
  Serial.print("\t"); // tab
  Serial.print(pulses[currentpulse-1][1] * RESOLUTION / 10, DEC);
  Serial.print(", 0};");
}
```

11.6 Appendix F

```
#include <LiquidCrystal.h>

//-------------------------------------------------------------------------------------------
int IRledPin =  13;    // LED connected to digital pin 13
const int ledPin =  7;      // the number of the LED pin
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
//-------------------------------------------------------------------------------------------
void setup(){

  pinMode(IRledPin, OUTPUT);
  pinMode(ledPin, OUTPUT);

  lcd.begin(16, 2);
  Serial.begin(9600);
}
//-------------------------------------------------------------------------------------------
void loop(){

    if (Serial.available() > 0){
      int inByte = Serial.read();

      switch (inByte){
        case 'd':
```

```
            Channel_Up();
            break;

            case 'u':
            Channel_Down();
            break;

        }
    }

 //       Serial.println("Sending IR signal");
 //       VolumeUp();

}
//-----------------------------------------------------------------------------------------
-----------------------------------------
void pulseIR(long microsecs) {
  cli();  // this turns off any background interrupts

  while (microsecs > 0) {
    // 38 kHz is about 13 microseconds high and 13 microseconds low
   digitalWrite(IRledPin, HIGH);  // this takes about 3 microseconds to happen
   delayMicroseconds(10);         // hang out for 10 microseconds, you can also change
this to 9 if its not working
   digitalWrite(IRledPin, LOW);   // this also takes about 3 microseconds
   delayMicroseconds(10);         // hang out for 10 microseconds, you can also change
this to 9 if its not working

   // so 26 microseconds altogether
   microsecs -= 26;
  }

  sei();  // this turns them back on
}
//-----------------------------------------------------------------------------------------
----------------------------------------
void Channel_Up() {

  digitalWrite(ledPin, HIGH);
  lcd.clear();
  lcd.write("Channel up");
  Serial.println("Sending IR signal for Channel Up");

  pulseIR(160 );  delayMicroseconds(57652 );
  pulseIR(4440 ); delayMicroseconds(4440 );
  pulseIR(520 );  delayMicroseconds(1660 );
  pulseIR(540 );  delayMicroseconds(1660 );
  pulseIR(540 );  delayMicroseconds(1660 );
  pulseIR(540 );  delayMicroseconds(560 );
  pulseIR(540 );  delayMicroseconds(560 );
  pulseIR(540 );  delayMicroseconds(560 );
  pulseIR(540 );  delayMicroseconds(560 );
  pulseIR(540 );  delayMicroseconds(560 );
  pulseIR(540 );  delayMicroseconds(1660 );
  pulseIR(540 );  delayMicroseconds(1660 );
  pulseIR(540 );  delayMicroseconds(1660 );
  pulseIR(540 );  delayMicroseconds(560 );
  pulseIR(540 );  delayMicroseconds(560 );
  pulseIR(520 );  delayMicroseconds(580 );
```

```
 pulseIR(520 );  delayMicroseconds(580 );
 pulseIR(520 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(520 );  delayMicroseconds(1680 );
 pulseIR(540 );  delayMicroseconds(540 );
 pulseIR(540 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(1680 );
 pulseIR(520 );  delayMicroseconds(580 );
 pulseIR(520 );  delayMicroseconds(1680 );
 pulseIR(520 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(46100 );
 pulseIR(4460 );

// delay(65);

 pulseIR(160 );  delayMicroseconds(57652 );
 pulseIR(4440 );  delayMicroseconds(4440 );
 pulseIR(520 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(1660 );
 pulseIR(540 );   delayMicroseconds(1660 );
 pulseIR(540 );   delayMicroseconds(560 );
 pulseIR(540 );   delayMicroseconds(560 );
 pulseIR(520 );  delayMicroseconds(580 );
 pulseIR(520 );   delayMicroseconds(580 );
 pulseIR(520 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(540 );  delayMicroseconds(560 );
 pulseIR(520 );  delayMicroseconds(1680 );
 pulseIR(540 );  delayMicroseconds(540 );
 pulseIR(540 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(1680 );
 pulseIR(520 );  delayMicroseconds(580 );
 pulseIR(520 );  delayMicroseconds(1680 );
 pulseIR(520 );   delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(1660 );
 pulseIR(540 );  delayMicroseconds(46100 );
 pulseIR(4460 );
```

```
  digitalWrite(ledPin, LOW);
}
//--------------------------------------------------------------------------------------
----------------------------------------
void Channel_Down(){

  digitalWrite(ledPin, HIGH);
  lcd.clear();
  lcd.write("Channel down");
  Serial.println("Sending IR signal for Channel Down");

  pulseIR(140 );    delayMicroseconds(41656 );
  pulseIR(4420 );    delayMicroseconds(4440) ;
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(520 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(540 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(520 );    delayMicroseconds(1680 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );

  //delayMicroseconds(46080 );

  pulseIR(4420 );    delayMicroseconds(4440 );
  pulseIR(540 );    delayMicroseconds(1680 );
  pulseIR(520 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
```

```
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(520 );    delayMicroseconds(580 );
  pulseIR(520 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(520 );    delayMicroseconds(580 );
  pulseIR(520 );    delayMicroseconds(1680 );
  pulseIR(520 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(560 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(1660 );
  pulseIR(540 );    delayMicroseconds(17324 );
  pulseIR(80 );

  digitalWrite(ledPin, LOW);
}
```