

# **Object Tracking Implementation on Embedded Computing Platform**

By

MOHANED ESSAM MOHAMED ISMAIL

ID: 14696

Project Dissertation submitted in partial fulfillment of  
the requirements for the  
Bachelor of Engineering (Honors)  
(Electrical and Electronic Engineering)

January 2015

Universiti Teknologi PETRONAS  
Bandar Seri Iskandar  
31750 Tronoh  
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

**Object Tracking Implementation on Embedded Computing  
Platform**

By

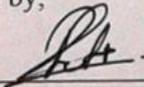
MOHANED ESSAM MOHAMED ISMAIL

ID: 14696

Project Dissertation submitted in partial fulfillment of  
the requirements for the  
Bachelor of Engineering (Honors)  
(Electrical and Electronic Engineering)

January 2015

Approved by,



---

**Mr. Patrick Sebastian**  
Electrical and Electronics Engineering,  
UNIVERSITI TEKNOLOGI PETRONAS,  
TRONOH, PERAK

## CERTIFICATION OF ORIGINALITY

Certification that I am responsible for the activities performed under this project, that this work is my own original work, unless specified otherwise or indicated in the references and the acknowledgements.

Mohamed Essam Mohamed

MOHANED ESSAM MOHAMED ISMAIL

## ABSTRACT

Embedded platforms are vital in various applications mainly Robotics and Mobile platforms. Image processing is usually done with full PC setup, which is hard to implement on mobile platforms and in other applications. This project aims to implement an image processing application on an embedded platform using structured C++ coding and OpenCV library.

The Embedded platform used in this project is Raspberry PI model B. The image processing application implemented is Object recognition and tracking; of a red circular object using the CHT algorithm. Before the image is processed using the CHT, it is passed by stages of color recognition, noise filtering and edge detection.

OpenCV and other required software were installed on the Raspberry PI, before it was able to successfully run the C++ code, and correctly detect the required object, with results similar to the same code running on Windows run PC, but with more processing time. Heat sinks were installed on specific parts of the Raspberry PI to reduce the overheating problems.

Three coding techniques were used to assess their results and compare between them in terms of application and processing time. The techniques are C++ code using Arrays, OpenCV functions and C++ code using pointers instead of Arrays. Arrays and Pointers implementations provide better robustness and accuracy, but higher time. OpenCV functions trades off these criteria for much better processing time.

Finally, the USB camera was attached to a servo motor to be controlled by the Raspberry PI. The Servo motor position is changed based on the object center in the image in order to centralize the object within the middle one third of the image. No significant delay is caused to the software by implementing the servo motor; except the delay to move the servo motor.

## **Acknowledgements**

*In the name of Allah, the most gracious, the most merciful*

I would like to extend my utmost gratitude for Mr. Patrick Sebastian, the Supervisor for this project, for his guidance and support; before and throughout the project period.

I would like to thank Professor Guillermo Sapiro of Duke University for his amazing online course which provided the needed basics in Image processing.

I want to thank UNIVERSITI TEKNOLOGI PETRONAS for providing the climate suitable for carrying out my project.

Last but not least, I would like to thank my family for their continuous support and prayers, and also my colleagues for their helpful advice and support.

## TABLE OF CONTENTS:

Certification of Approval:.....	II
Certification of Originality:.....	III
Certification of Approval:.....	II
Abstract:.....	IV
Acknowledgements:.....	V
Table of Contents:.....	VI
List of Figures: .....	VII
Abbreviations and Nomenclatures:.....	IX
Chapter 1: Introduction:	
1.1 Background:.....	10
1.2 Problem Statement:.....	10
1.3 Objectives and Scope of Study:.....	11
Chapter 2: Literature Review:	
2.1 Hough Transform:.....	12
2.2 Edge Detection: .....	12
2.3 Circular Hough Transform: .....	14
2.4 Color Detection: .....	14
Chapter 3: Methodology:	
3.1 Object Recognition: .....	15
3.2 Implementation on Raspberry-Pi: .....	16
3.3 Algorithm Implementation Using OpenCV functions: .....	20
3.4 Object Tracking by moving the camera using Servo Motor .....	21
Chapter 4: Results and Discussion:	
4.1 Program Implementation:.....	21
4.2 Discussion: .....	28
4.3 Prototype Testing:.....	30

4.4 Problems faced and Proposed Solutions: .....	32
4.5 Power Consumption measurements:.....	33
Chapter 5: Conclusion and Recommendation:	
5.1 Conclusion: .....	34
5.2 Recommendation and Future Development: .....	35
References: .....	36
Appendices:	
Appendix A: C++ Code: .....	39
Appendix B: Format of “CMakeLists.txt” file: .....	54
Appendix C: Format of “interfaces.txt” file: .....	55

## List of Figures

- Figure (1): Parametric Equation of straight line
- Figure (2): Hough Transform Accumulator
- Figure (3): Sobel Operator
- Figure (4): CHT concept for a known radius
- Figure (5): object recognition methodology
- Figure (6): Sobel Masks
- Figure (7): IP setting of the PC
- Figure (8): Using Putty software to run the tight VNC server
- Figure (9): Starting the Tightvnc viewer on PC
- Figure (10): Tightvnc viewer window on PC showing the Raspberry PI Desktop
- Figure (11): Connection Schematics
- Figure (12): GPIO P1 Header
- Figure (13): Code running on Raspberry PI.
- Figure (14): Code running on PC
- Figure (15): Program outputs
- Figure (16): OpenCV outputs
- Figure (17): Median filter block size sweeping effect on processing time
- Figure (18): Code comparison against the median filter block size
- Figure (19): effect of radius step value on processing time
- Figure (20): effect of radius step on the relative processing time
- Figure (21): effect of angle step value on processing time
- Figure (22): effect of angle step value on relative processing time
- Figure (23): Prototype
- Figure (24): Servo Motor Control based on object location
- Figure (25): Raspberry PI sources of heat
- Figure (26): Raspberry PI after installing heat sinks
- Figure (27): Power consumption measurements under normal operating conditions

## Abbreviations and Nomenclatures

CHT:	Circular Hough Transform
CPP/C++:	C plus plus
GPIO:	General Purpose Input Output
OpenCV:	Open Source Computer Vision
PC:	Personal Computer
PWM:	Pulse Width Modulation
R-PI:	Raspberry PI board
VNC:	Virtual Network Computing

# CHAPTER 1

## INTRODUCTION

### 1.1 BACKGROUND:

Image processing is becoming an important part of many fields recently. From the first Digital image processing software developed in the 1960s to study the moon surface, to the vast medical applications such as the computed tomography (CAT scan) [1], image and video processing has become an important field of research that can solve various arising problems.

Embedded platforms –such as Arduino, RaspberryPi and Beagleboard- are reasonably priced, mobile and easy to manufacture and distribute, computing platforms. Embedded platforms are mainly used for prototyping and educational purposes. However, with their ever-increasing computing capabilities, embedded platforms are becoming reliable enough to solve real-world problems.

This project aims to implement image and video processing applications on embedded platforms, which helps in increasing the areas where Image and Video processing applications are used, and enhances the performance and applications of embedded platforms.

### 1.2 PROBLEM STATEMENT:

Image and video processing applications are usually implemented through sophisticated computer software, and using full computer platforms. This limits the usage of such applications in various fields, mainly robotics and mobile platforms.

Embedded platforms are suitable to implement on mobile platforms, however, these platforms are limited in their computational capabilities, and are not usually able to run advanced image and video processing software –such as MATLAB-.

This project addresses development and performing of relatively complex image and video processing applications, namely “Object Recognition and Tracking”, using embedded platforms and simple cameras, both suitable for usage on various mobile platforms.

### **1.3 OBJECTIVES AND SCOPE OF STUDY:**

#### **1.3.1 Objectives:**

The objectives of this project are:

- 1- Development of Object Recognition and Tracking algorithm, using digital image and video processing techniques.
- 2- Development of programming code able to perform the Object Recognition and Tracking algorithm, using C++ programming language.
- 3- Implementation of the developed code on an embedded platform, namely Raspberry-Pi.

#### **1.3.2 Scope of study:**

This project is focused on the development and implementation of an Object Recognition and Tracking application on an embedded platform. However, this project is not focused on devising new “Object Recognition and Tracking” algorithms; instead, existing algorithms and techniques are used.

The object specified to be tracked is a red and circular object. The programming language used is C-language, and the embedded platform used is Raspberry-Pi.

## CHAPTER 2

### LITERATURE REVIEW

#### OBJECT RECOGNITION ALGORITHM:

##### 2.1 Hough Transform:

A suggested object recognition algorithm is based on Hough transform segmentation technique. According to [2], Hough transform uses the geometry of the recognized object, mathematical equation, to recognize the object. The Hough Transform started as a method for detecting straight lines in an image using concurrent lines rather than collinear points. However, Hough Transform used the slope intercept model to represent lines, and because of the fact that the slope and the intercept are unbounded –causing computational complications-, Hough Transform was implemented using the parametric equation of the line [3], as shown in Figure (1).

Parametric equation of a straight line:

$$\rho = x \cos \theta + y \sin \theta$$

Where: ( $\rho$ ) is the normal distance,

( $\theta$ ) is the normal angle.

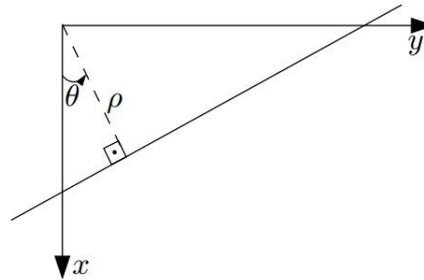


Figure (1): Parametric Equation of straight line

There are many variations of Hough Transform used to recognize various shapes, and use different techniques. One of the famous variations is the Hough Transform proposed by [3], which is mainly concerned with the line recognition. It uses an accumulator as a 2D array of parameters  $\rho$  and  $\theta$  discussed before, the parameters are practically quantized with a step size relevant to the required accuracy, lines are then detected by the most number of votes from the edge points, which means most number of collinear points on the edge. An example of a Hough Transform accumulator for line detection, is shown in Figure (2).

The accumulator shows three points of the image edge, collinear on the straight line specified by  $\rho = 80$  and  $\theta = 60$ , with the number of votes three, while all the other lines only one vote each.

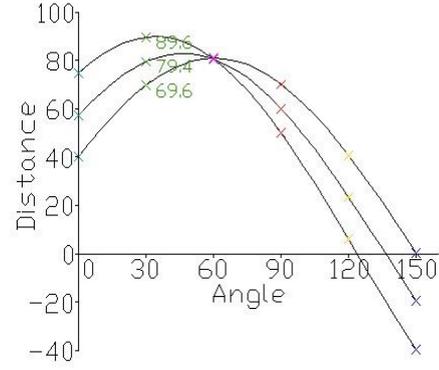


Figure (2): Hough Transform Accumulator [4].

This technique is also extended to curves and circles, using the equation of the circle:  $r^2 = (x - a)^2 + (y - b)^2$ , which should be transformed to its parametric equation:

$$x = a + r \cos \theta$$

$$y = b + r \sin \theta \quad , \text{where } \theta \text{ is the angle between } r \text{ and the } x - \text{axis}$$

Another variation of Hough Transform is proposed by [5] to detect rectangles. It's based on the relations between the parameters of the four lines forming the rectangle; the right angles between the lines and the constant distance between them.

## 2.2 Edge Detection:

Before performing the Hough Transform, the image must be passed to an edge detector [2]. There are many edge detectors that can be used to extract edges from images, and pass it to CHT. There are two categories of edge detectors; gradient and Laplacian, which uses the local maxima and minima in first derivative, and the zero crossing in the second derivative, respectively, to find the edges of the image[6]. The Sobel edge detector is under the Laplacian category. For each pixel  $[i, j]$ , whose neighbor pixels are labeled as in Figure (3), the Sobel edge is calculated as:

$$M = \sqrt{S_x^2 + S_y^2} \quad \text{Where } S_x \text{ and } S_y \text{ are:}$$

$$x - \text{direction: } S_x = (a_2 + 2a_3 + a_4) - (a_0 + 2a_7 + a_6)$$

$$y - \text{direction: } S_y = (a_0 + 2a_1 + a_2) - (a_6 + 2a_5 + a_4)$$

a <sub>0</sub>	a <sub>1</sub>	a <sub>2</sub>
a <sub>7</sub>	[i,j]	a <sub>3</sub>
a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>

Figure (3): Sobel Operator

### 2.3 Circular Hough Transform:

In the case of Circular Hough Transform (CHT), the edges, resulting from the edge detector, are used as an input to the Hough Transform algorithm. Each point of the edge is considered a center of an imaginary circle, and using the parametric equation these imaginary circles are drawn by sweeping the radius  $r$  for all possible values, and sweeping the angle  $\theta$  from  $0$  to  $2\pi$  radian. Then, for each point on the perimeter of these imaginary circles,  $x$  and  $y$  coordinates are calculated using the parametric equation, and then, a vote is added to the accumulator [2]. As any circle is defined by a center and radius, the accumulator in the case of CHT is three dimensional  $[x, y, r]$ . Finally, the accumulator is sorted with respect to the number of votes, indicating the arrangement of the triplets  $[x, y, r]$  that represent the best possible circles in the image[7]. If the case is that the radius of the required circle is known, the accumulator will be a 2D matrix of  $x$  and  $y$  coordinates. Figure (4) shows the concept of voting, and how it can yield the center of the required circle, and in this case it is the point with the highest vote count, which is three.

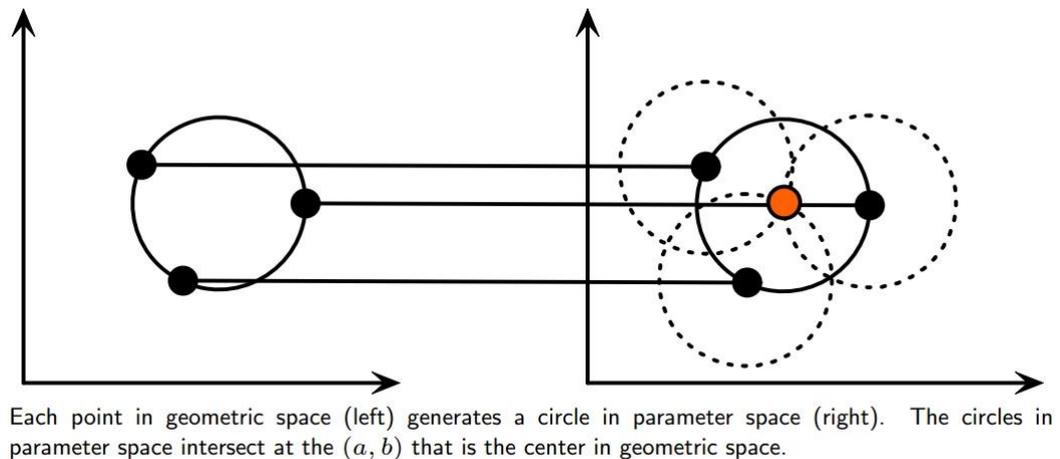


Figure (4): CHT concept for a known radius [7]

### 2.4 Color Detection:

CHT processing time is proportional to the number of the edge points in the image, as it sweeps all the parameters for each single point on the edges of the image. In order to reduce the processing time, the tested objects are reduced by fixing the color of the required object. One approach for color detection in the RGB color space is by using normalized color components, and use threshold values for the required color[8], with the formula for the normalized RED component is:  $r = R / (R+G+B)$ .

## CHAPTER 3

### METHODOLOGY

#### 3.1 OBJECT RECOGNITION:

The proposed methodology for the object recognition algorithm consists of a set of operations aimed to decrease the processing time for the CHT. The progression of these operations is as shown in

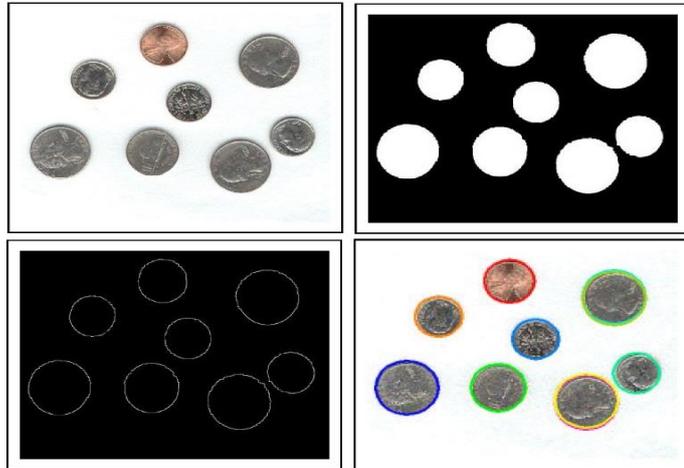


Figure (5) [7], the first image is the output of the camera. First, the image is transformed into a binary image with a color detection technique, as shown in the second image in the figure. After that, the binary image is passed to an edge detector which extracts the edges, as shown in the third image. The edges are then passed the CHT, which arranges the best circles, as shown in the fourth image.

Figure (5): object recognition methodology

Figure (5) [7], the first image is the output of the camera. First, the image is transformed into a binary image with a color detection technique, as shown in the second image in the figure. After that, the binary image is passed to an edge detector which extracts the edges, as shown in the third image. The edges are then passed the CHT, which arranges the best circles, as shown in the fourth image.

##### 3.1.1 Color Detection:

The algorithm used for color detection is by checking the normalized color components, and compare them to a range of acceptable calibrated values for the red color values. The formula used for the normalized RED component is:

$$r = R / (R+G+B)$$

##### 3.1.2 Noise Filtering:

The noise filtering is done with a median filter whose window size is five pixels. The median filter places the middle pixel with the median of its neighbor pixels in

the window of 5x5. The median is the value is obtained by arranging the neighbor values in descending order and obtain the value in the middle.

### 3.1.3 Edge Detection:

The edge detection is implemented using a Sobel operator. The Sobel operator is implemented by using convolution masks, or passing a window on the pixels of the binary image obtained from the color detector [6]. The used window used is shown in Figure (6). The edge value is obtained from the formula:

$$M = \sqrt{S_x^2 + S_y^2}$$

$S_x =$	-1	0	1
	-2	0	2
	1	0	1
$S_y =$	1	2	1
	0	0	0
	-1	-2	-1

Figure (6): Sobel Masks [6].

### 3.1.4 Circular Hough Transform (CHT):

As long as the radius of the tracked object is not fixed –size and distance to the object are not fixed -. A 3D accumulator is stored in the memory as a 3D array, and for each point on the edge, the radius is swept from 20 pixels (smallest circle to be detected) to 90 pixel (maximum radius when the object is nearest to the camera) in steps of 5 pixels, while the angle  $\theta$  is swept from 0 to  $2\pi$  radian in steps of 0.1 radian.

In CHT, there is a tradeoff between processing time, memory and accuracy, and this can be controlled by changing the step size of sweeping the circles parameters radius and the angle  $\theta$ . The less the step size, the higher the accuracy, but the more the processing time.

## 3.2 IMPLEMENTATION ON RASPBERRY PI:

### 3.2.1 Raspberry PI setup:

The setup of Raspberry PI done in this project consists of: Raspberry PI power supply, HDMI to VGA box converter, VGA screen, LAN cable, USB keyboard and mouse, and a SD Card.

The SD card is flashed using windows operated PC, with a Raspberry PI compatible OS namely “Debian”, other operating systems are available on [9]. The flashing was done using a windows application called “Win32DiskImager” available on [10], with the instructions from [11].

After that, the SD card is connected to the Raspberry PI and it is powered on. Further configurations can be done depending on the required configuration needed, as explained in [12]. Some configurations can be done through the BIOS like text file of Raspberry PI called “Config.txt”, this file can be accessed either from inside the Raspberry PI itself or directly from the SD card using any other platform. Instructions to modify this file can be found on [13].

### **3.2.2 OpenCV Installation:**

Open-CV (Open Source Computer Vision) is an open source programming library, it is used only to obtain the RGB color images from the Webcam used, although all the other parts of the code are done using normal C-language syntax.

There are many sources that describe various ways to install OpenCV library on Raspberry PI. In order to avoid confusion, only the implemented method, proven to work with this project, is introduced. The steps to install OpenCV are fully explained in [14]. However, the Cmake configuration is done as explained in [15].

### **3.2.3 Compiling the code:**

In order to compile a C++ code (the code is written in C code but compiled as C++ to make use of new OpenCV USB camera read functions written in C++) with OpenCV, there are few steps should be taken:

- 1- The CPP code is written as shown in Appendix A, and saved in the project file, here called “fyp.cpp”.
- 2- A text file named “CMakeLists.txt” is written as shown in Appendix B, and saved in the same project file.
- 3- Using the Raspberry PI terminal:
  - a. Go to the project folder
  - b. Type the command: “cmake .”
  - c. Type the command: “make”

d. The executable file is run, here as: “./fyp”

### 3.2.4 Raspberry PI Remote control on LAN network:

The aim of this section is to provide a method to control the Raspberry PI without using external peripherals except the LAN cable. There are three main steps to initiate the communication path between the PC and the Raspberry PI; Setting static IP address for the Raspberry PI, the Putty software and the VNC remote control software. These steps are explained more:

1- Assigning static IP addresses for the Raspberry PI and the PC:

To assign static IP for the Raspberry PI, the “interfaces.txt” file in the directory “\etc\network” in the Raspberry PI. The file is modified as shown in Appendix C. The corresponding IP on the PC is set as shown in Figure (7).

2- PUTTY SSH communication:

Putty software is used with SSH protocol to setup the VNC server on the Raspberry PI after the power-up. Screenshot of the opening screen of the putty software, and the command used to run the VNC server on the Raspberry PI, is shown in Figure (8).

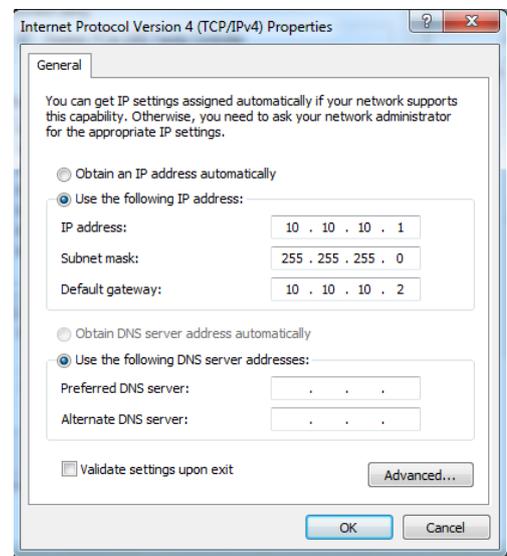


Figure 7: IP setting of the PC

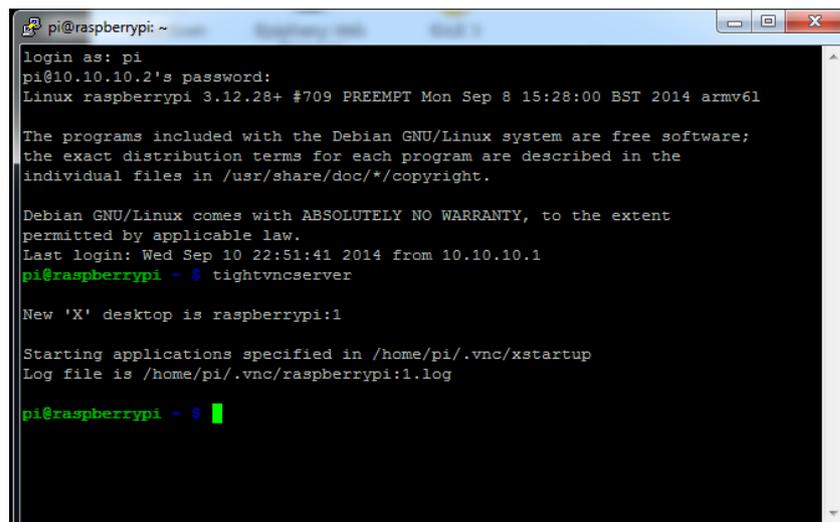


Figure 8: Using Putty software to run the tight VNC server

### 3- Installing tight VNC software on Raspberry PI and the PC:

Tightvnc software is used to remotely control the Raspberry pi in a desktop form. Keyboard, screen and mouse of the host PC can be used. The software must be installed both on Raspberry PI and the PC. The installation procedure is explained in [16]. In order to avoid connecting any peripherals to the Raspberry PI, right after power up, putty software is used to run the tightvnc server on the Raspberry PI, as shown in Figure (8). Figure (9) shows the connection window of the tightvnc software, and Figure (10) shows the usage of the software to control the Raspberry PI.

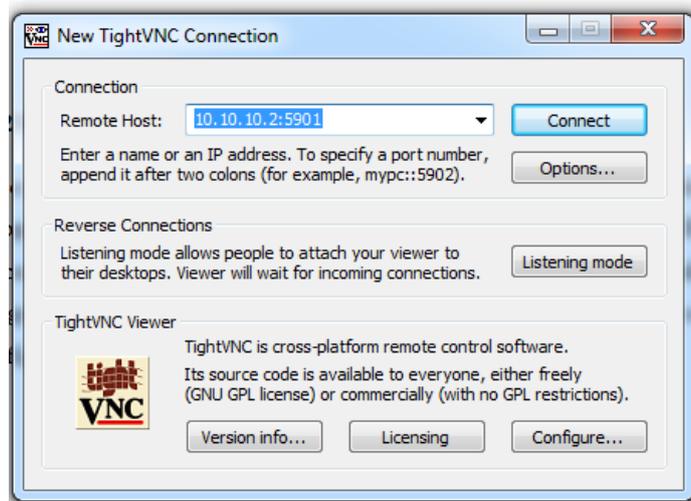


Figure 9: Starting the Tightvnc viewer on PC

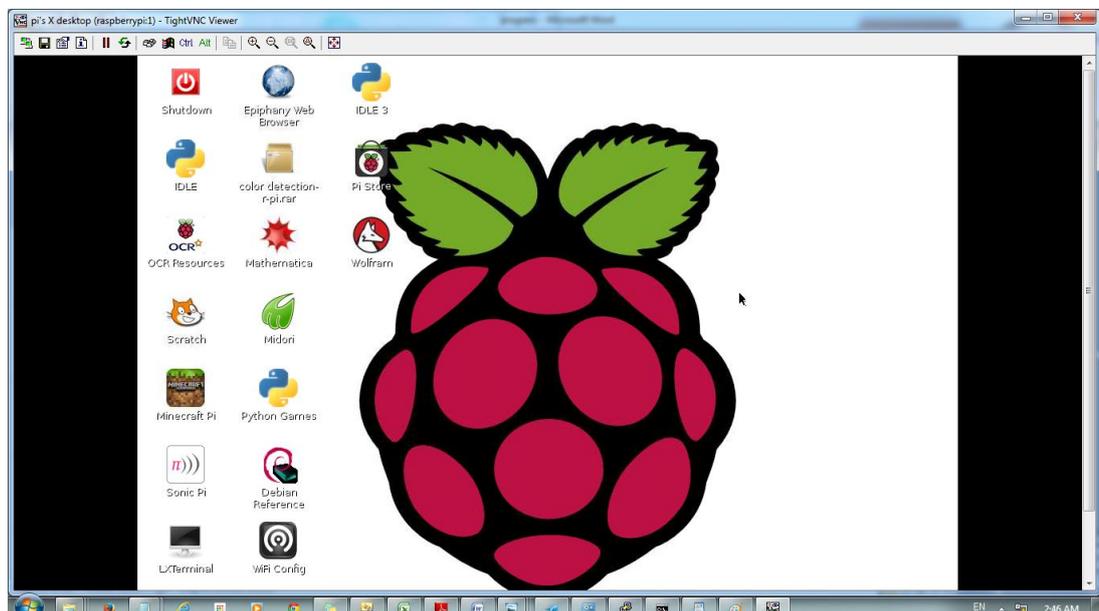


Figure 10: Tightvnc viewer window on PC showing the Raspberry PI Desktop

### 3.3- ALGORITHM IMPLEMENTATION USING OPENCV FUNCTIONS:

As shown in the C++ code in Appendix A under the OpenCV section of the code, three out of four of the Algorithm stages can be modified to be done using the OpenCV functions [17], while maintaining almost the same parameters. The functions used to implement each stage of the algorithm are as follows:

#### 1- Color Detection:

The color detection stage is obtained from the normal code; as the color detected image is assumed as an input to the whole algorithm.

#### 2- Median Filtering:

The Median filtering stage of the algorithm is implemented using the OpenCV function “medianblur” with the syntax as:

```
medianBlur(img_gray, image_filtered, block_size);
```

#### 3- Edge Detection:

The edge detection stage of the algorithm is done with the “Canny” function, as follows:

```
Canny(img_filtered, img_edge, edge_min_threshold, edge_max_threshold, 3);
```

#### 4- Hough Transform:

Hough transform is done in OpenCV using the “HoughCircles” function, with the following syntax:

```
vector<Vec3f> circles;  
HoughCircles(img_show_edge_opencv, circles, CV_HOUGH_GRADIENT, 1, 1,200,2*3*r_min/8,  
r_min, r_thresh);
```

And the following syntax for highlighting the recognized circle:

```
Vec3i c = circles[0];  
circle( img_show_opencv, Point(c[0], c[1]), c[2], Scalar(0,0,255), 3, CV_AA);  
circle( img_show_opencv, Point(c[0], c[1]), 2, Scalar(0,255,0), 3, CV_AA);
```

### 3.4- OBJECT TRACKING CAMERA CONTROL USING SERVO MOTOR:

#### 3.4.1 Schematics:

The Servo Motor is connected to the Raspberry PI through the GPIO P1 header pin 18, which can be used for PWM output; the schematic of the connection is shown in Figure 11, and the GPIO P1 header pin configuration is shown in Figure 12.

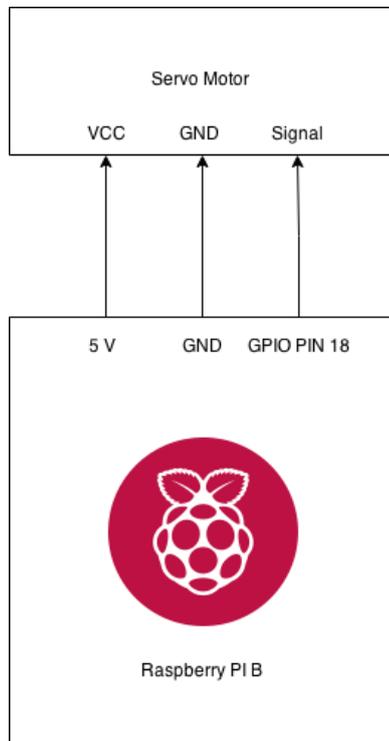


Figure 11: Connection Schematics, Done using: <https://www.draw.io/>



Figure 12: GPIO P1 Header, from [18]

#### 3.4.2 Servo Motor Control on Raspberry PI using C language:

The C code used for controlling the Servo motor is fully obtained from [19], which uses [18] for GPIO configuration. This code is merged with the detection software to move the camera based on the Object centre location. The Image is divided into three sections; if the Object centre is in the middle, the servo keeps its position. However, if the Object center falls under the left or the right sections, the Servo motor position is decreased or increased by 18 degrees, as shown in the code in Appendix A (the code shows right and left turns by 10 which is 10 % where  $10\% * 180 = 18$  degrees).

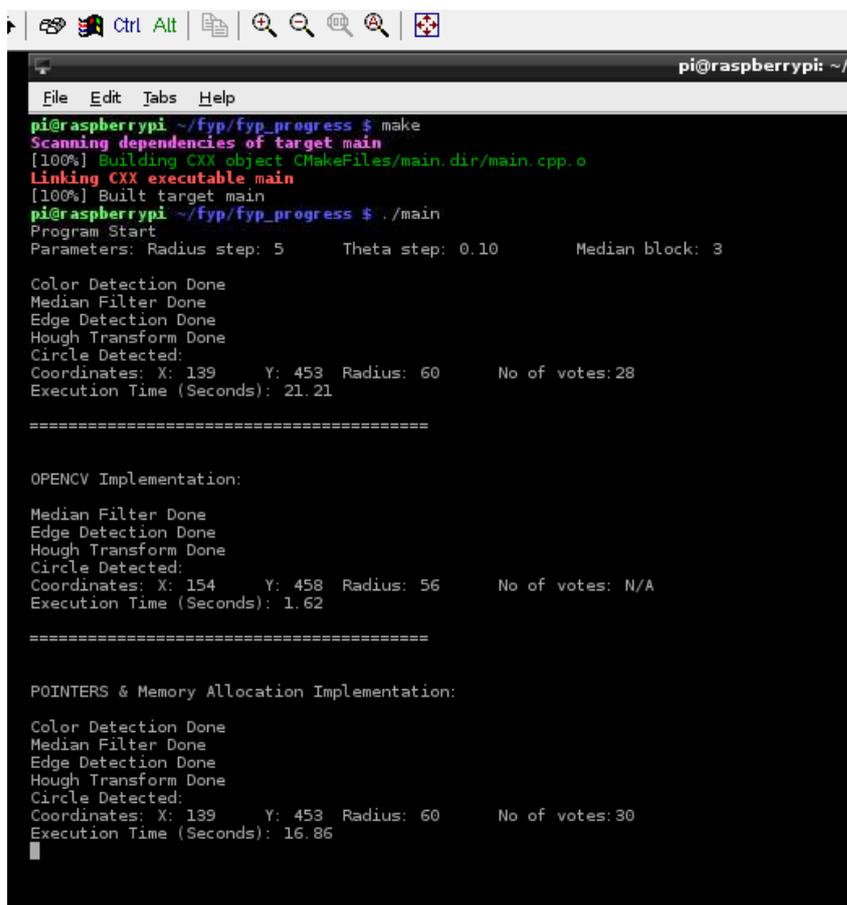
# CHAPTER 4

## RESULTS AND DISCUSSION

### 4.1 PROGRAM IMPLEMENTATION:

The C++ code was written using Microsoft Visual Studio 2010 compiler, with OpenCV library installed for testing and debugging purposes. After that the code was transferred and compiled on Raspberry PI. The complete C++ code used is shown in Appendix A.

A screen shot of the code running on Raspberry PI is shown in Figure (13). A screen shot of the same code running on PC for the same camera input is shown in Figure (14). The output images of the code are shown in Figure (15), the images are shown in a sequence similar to the code sequence; Image taking, Color detection, Median Filtering, Edge detection and Hough Transform.



```
pi@raspberrypi: ~/fyp/fyp_progress $ make
Scanning dependencies of target main
[100%] Building CXX object CMakeFiles/main.dir/main.cpp.o
Linking CXX executable main
[100%] Built target main
pi@raspberrypi ~/fyp/fyp_progress $ ./main
Program Start
Parameters: Radius step: 5      Theta step: 0.10      Median block: 3

Color Detection Done
Median Filter Done
Edge Detection Done
Hough Transform Done
Circle Detected:
Coordinates: X: 139      Y: 453      Radius: 60      No of votes:28
Execution Time (Seconds): 21.21

=====

OPENCV Implementation:

Median Filter Done
Edge Detection Done
Hough Transform Done
Circle Detected:
Coordinates: X: 154      Y: 458      Radius: 56      No of votes: N/A
Execution Time (Seconds): 1.62

=====

POINTERS & Memory Allocation Implementation:

Color Detection Done
Median Filter Done
Edge Detection Done
Hough Transform Done
Circle Detected:
Coordinates: X: 139      Y: 453      Radius: 60      No of votes:30
Execution Time (Seconds): 16.86
```

Figure (13): Code running on Raspberry PI.

```

Program Start
Parameters: Radius step: 5   Theta step: 0.10   Median Block: 3

Color Detection Done
Median Filter Done
Edge Detection Done
Hough Transform Done
Circle Detected:
Coordinates: X: 139   Y: 453   Radius: 60   No of votes:28
Execution Time (Seconds): 1.73
=====

OPENCU Implementation:
Median Filter Done
Edge Detection Done
Hough Transform Done
Circle Detected:
Coordinates: X: 154   Y: 458   Radius: 56   No of votes: N/A
Execution Time (Seconds): 0.35
=====

POINTERS & Memory Allocation Implementation:
Color Detection Done
Median Filter Done
Edge Detection Done
Hough Transform Done
Circle Detected:
Coordinates: X: 139   Y: 453   Radius: 60   No of votes:30
Execution Time (Seconds): 1.20
-

```

Figure (14): Code running on PC

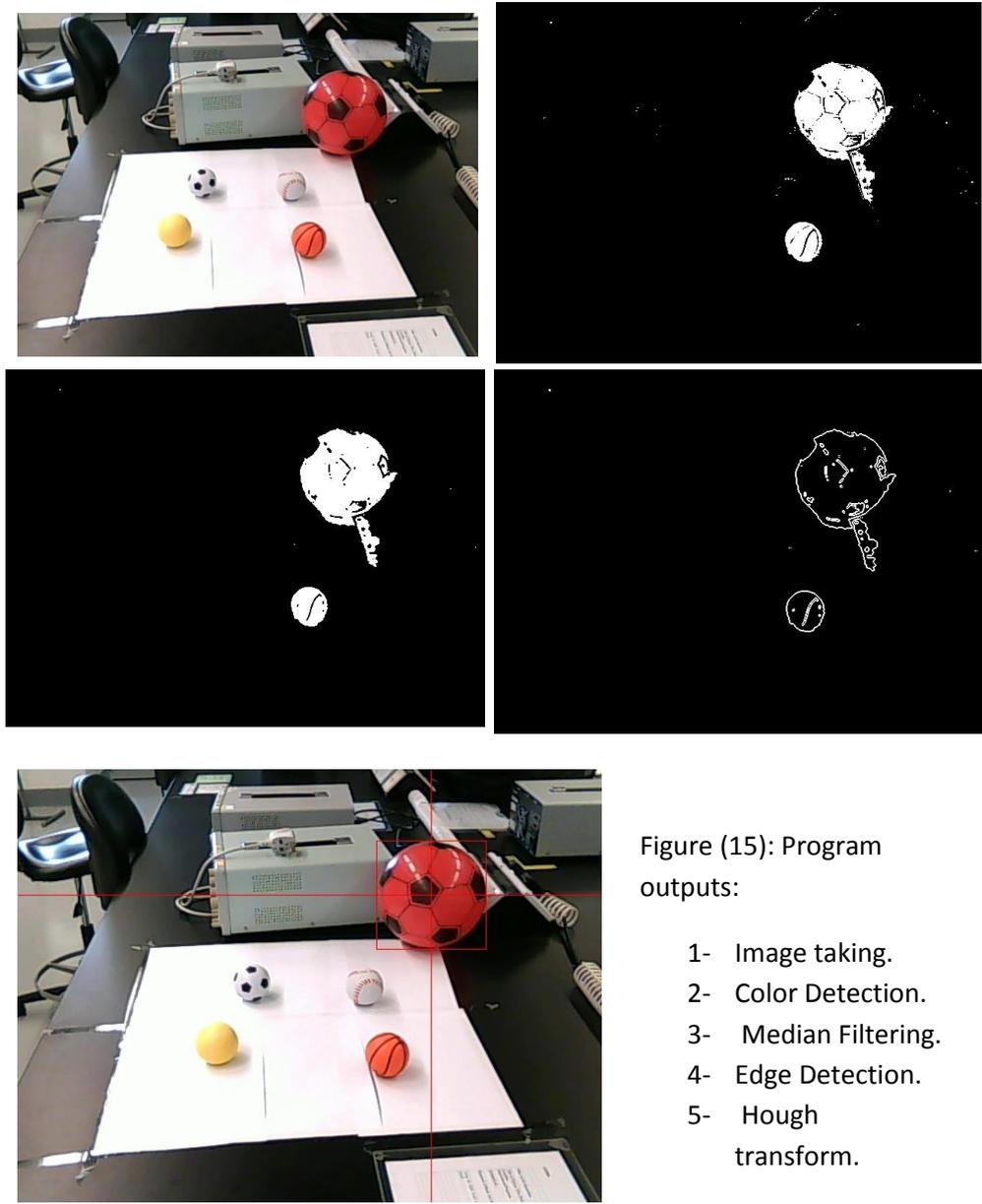


Figure (15): Program outputs:

- 1- Image taking.
- 2- Color Detection.
- 3- Median Filtering.
- 4- Edge Detection.
- 5- Hough transform.

#### 4.1.1 Output using OpenCV functions:

The output shown in Figure (15) is done using the first section of the code shown in Appendix A, which uses the devised methodology, done using arrays. However, the second section of the code is done using the OpenCV functions for median filtering, edge detection, and Hough transform. The output of this section of the code is shown in Figure (16).

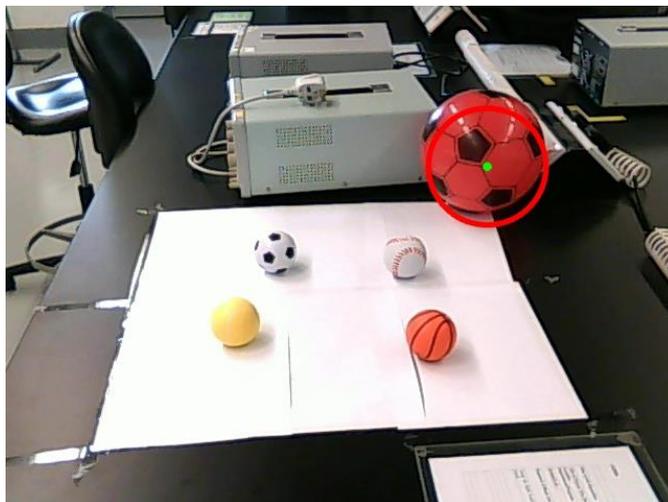
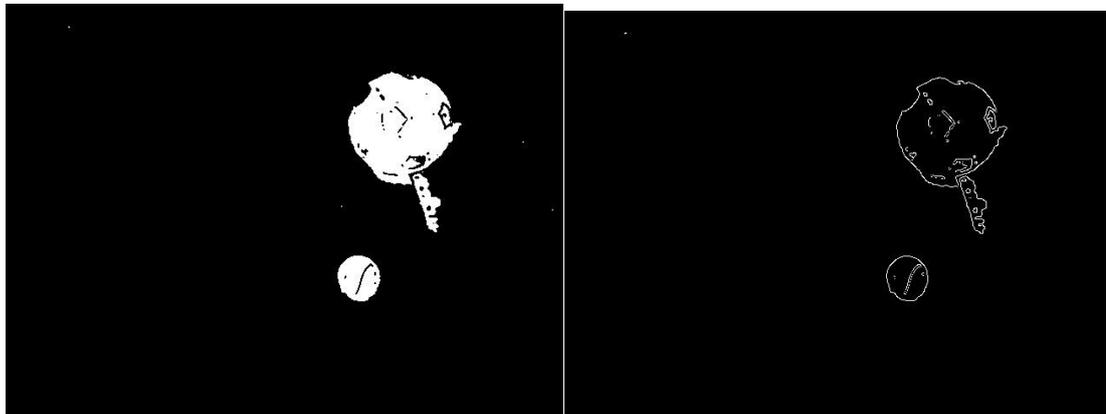


Figure (16): OpenCV outputs:

- 1- Median Filtering.
- 2- Edge Detection.
- 3- Hough transform.

#### 4.1.2 Output using Pointers:

The third section of the code shown in Appendix A, implements the same methodology but using pointers instead of arrays. As expected, the outcomes of all the stages are exactly the same as the arrays section; however, the execution time is different.

### 4.1.3 Parameters Sweep in Raspberry PI:

Various parameters of the code sections were swept and the effect on the processing time of the arrays, OpenCV and pointers are recorded. Also, the reference program is considered having the parameters; median filter block size 3x3, Hough transform radius step of 5 pixels and angle 0.1 Rad. The processing time in the following section are normalized to the processing time of this special reference parameters, except for the technique comparison where the pointers processing time is normalized to the arrays processing time at each point, and the OpneCV processing time is normalized to the pointers processing times. The results are as follows:

#### 4.1.3.1 Median Filter window block size:

By sweeping the median filter window block size over the values from 1x1 pixels to 9x9 pixels, we get the processing time values as shown in Figure (17). The window block size is shown on the horizontal axis while the percentage of the normalized processing time is shown on the vertical axis. (Note the zero crossing at the reference median filter block size of 3)

Figure (18) shows comparison of the processing times of the three techniques with their percentage normalized processing times on the vertical access, and the median filter window size on the horizontal axis.

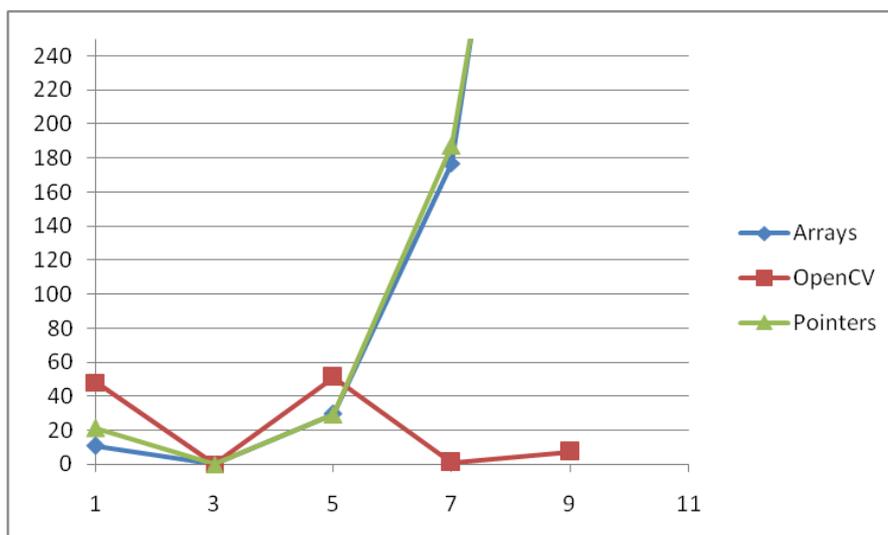
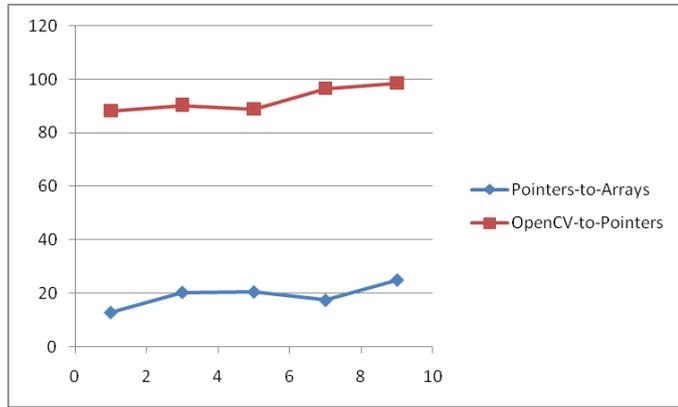


Figure (17): Median filter block size sweeping effect on processing time

Figure (18): Code comparison against the median filter block size



#### 4.1.3.2 Hough Transform Radius step change:

The Hough Transform radius step is swept over 1,3,5,7,9 and 11 pixels, and results are shown in Figure (19).

The radius step value is shown on the horizontal axis while the time processing percentage normalized to the reference parameters processing time (note the zero crossing at the reference code radius step).

Figure (20) shows the percentage relative processing time for the three code implementations verses radius step value.

Figure (19): effect of radius step value on processing time

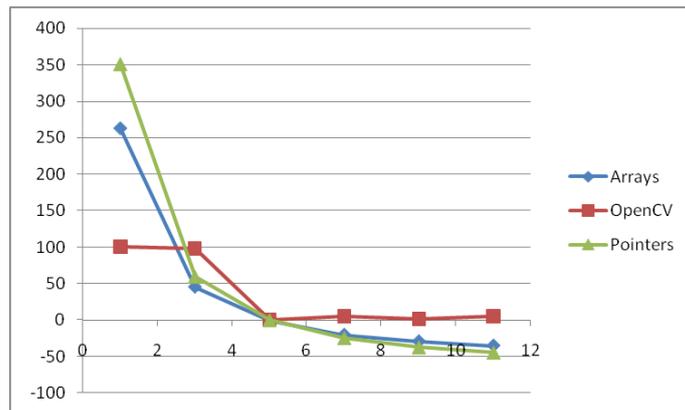
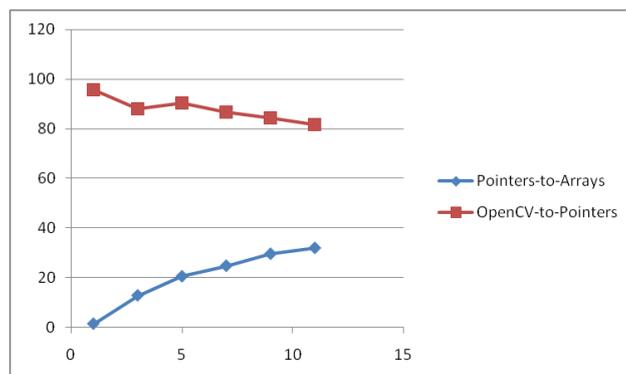


Figure (20): effect of radius step on the relative processing time



### 4.1.3.3 Hough Transform Angle step change:

The Hough transform angle step change sweeping results are shown in Figure (21).

The angle is swept from 0.05 to 0.3 Radian with 0.1 radian steps. The angle step change value is shown on the horizontal axis, and corresponding processing time is shown on the vertical axis. The curves cross the horizontal axis at the reference image to which the processing time is normalized).

Figure (22) shows the relative processing time of the three techniques verses the angle step change value.

Figure (21): effect of angle step value on processing time

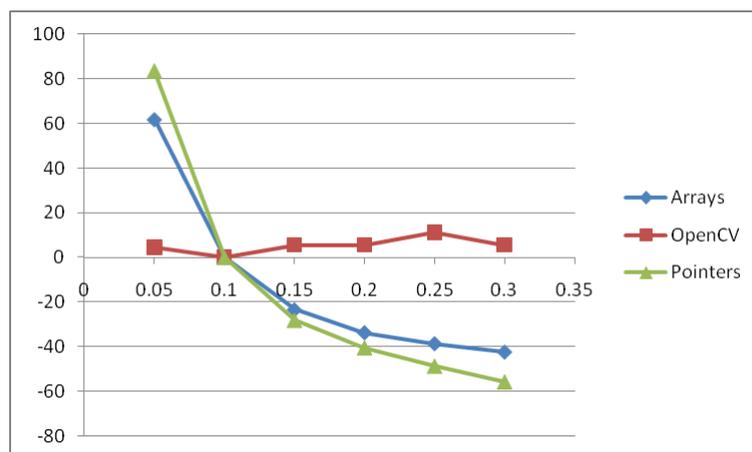
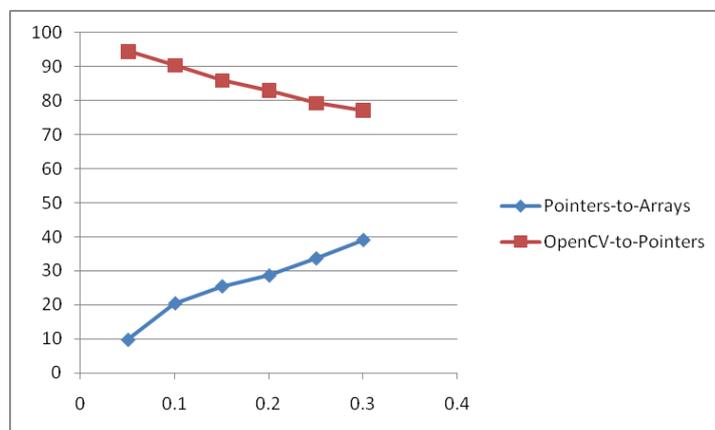


Figure (22): effect of angle step value on relative processing time



## **4.2 DISCUSSION:**

### **4.2.1 General aspects:**

The same image was used to run the code on both Raspberry PI and the PC, for all three coding techniques. As shown in Figures (13) and (14), the output of the code running on the Raspberry PI is consistent with the results obtained from the PC. As the center point of the detected circle in the Raspberry PI, in the case of the arrays and pointers section, is (139, 453) with radius of 60 pixels, and 30 votes. In the case of PC, the circle center is exactly the same (291, 360) with radius of 60, and 30 votes.

As shown in Figures (13) and (14) also, the Processing time of the Raspberry PI is generally slower than the processing time of the PC. The processing time of the Raspberry PI is 21.21 seconds for the arrays section, 1.62 seconds for the OpenCV section and 16.86 seconds for the pointers section. For the PC, the processing time values are 1.73, 0.31 and 1.2 respectively.

It can also be noted that for this case the OpenCV functions does not return the number of votes, instead it reads the min required number of votes for the circle to be detected (it detects all the circles above the limit).

The best processing time for both Raspberry PI and PC is the OpenCV functions, followed by the pointers implementation and finally the arrays implementation. The OpenCV functions are faster by 74.17% than the pointers in the PC, and by 92.5% in the Raspberry PI. The pointers implementation is faster than the arrays implementation by 30.64 % in the PC and by 20.51 % in the Raspberry PI.

### **4.2.2 Effect of parameter change on the processing time:**

The processing time depends on the parameter changes, however, not all parameter changes has the same effect on processing time. First, it can be concluded from Figure (17) that increasing the median filter block size increases the processing time exponentially, this can be related to the sorting algorithm used in the sort array function shown in Appendix A, as the sorting algorithm implemented is just a simple array sorting algorithm that successively swap the array elements based on their values. However, this is not the case for the OpenCV functions implementation

as it is expected that it implemented a more time efficient algorithm for obtaining the median value calculation. Also, it can be seen from Figure (17) that increasing the median filter block size, increases the advantage of the pointers implementation compared to the arrays implementation to around 25%, and also the OpenCV implementation compared to the pointers implementation until it almost reaches 100%.

Second, the effect of increasing the radius step change in the Hough Transform is shown in Figure (18). As shown, the processing time of both pointers and arrays decreases rapidly with increasing the radius step until they reach about 50 % percent less time at 11 pixels. This is because more radius step size means less processing and less iterations in the program. However, the OpenCV implementation decreases a bit until it reaches the reference image (image with parameters; median block size=3, radius step=5 and angle step= 0.1) processing time and it becomes constant after that. One important observation in Figure (19) is that by increasing the radius step change, the pointers implementation is proving to become much better than the arrays implementation from 0% at 1 pixel radius step to about 30% at 11 pixels radius step. It can also be seen that the OpenCV implementation advantage over pointers implementation is decreasing to about 80%. This is due to the fact that the OpenCV processing time is constant while the pointers processing time is improving.

Finally, the effect of the angle step size of the Hough Transform on the processing time is shown in Figures (20) and (21). It should be emphasized that in the OpenCV implementation the angle step size cannot be changed by the user. Changing the angle step size shows similar results as changing the radius step size however the results are exaggerated in the angle case.

### 4.3 Prototype Testing:

The Prototype implemented for this project is, as mentioned in the methodology, to attach the USB camera on a Servo motor controlled by the PWM pin of the Raspberry PI board. Figure 23, shows the prototype while running.

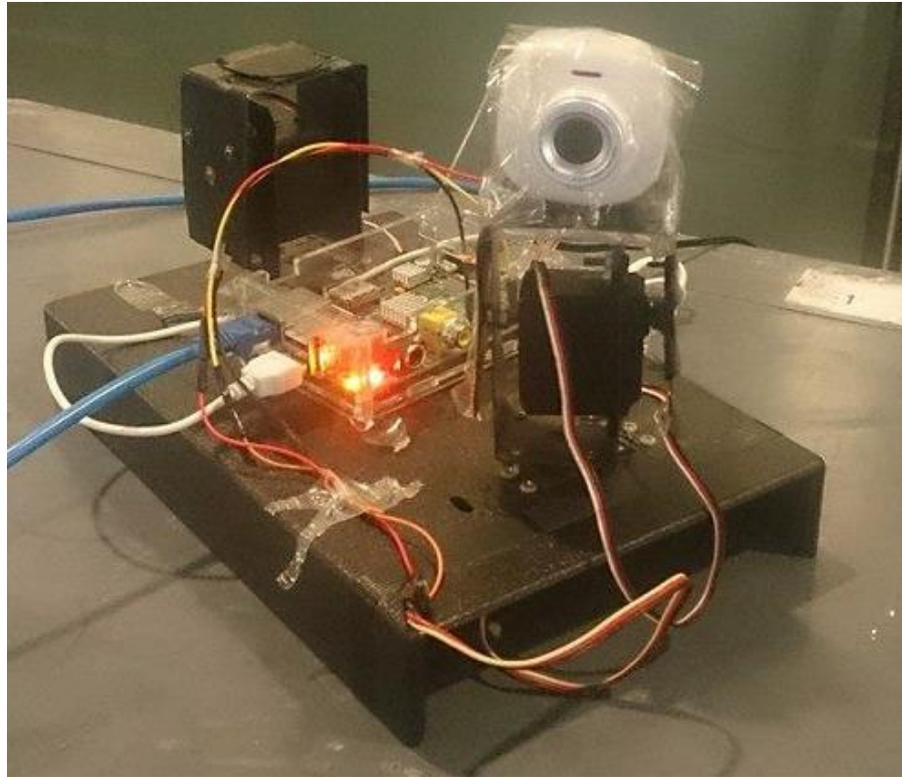


Figure 23: Prototype

The prototype was tested the camera is able to turn in 18 degrees steps tracking the object. Based on the object position in the image the Raspberry PI determines to whether turn the servo to the right by 18 degrees, left 18 degrees or to keep its current position; if the object is located in the left one third of the image (based on the pixel location), the right one third of the image or the middle one third. This is further illustrated in Figure 24, which shows an image taken by the Raspberry PI, where the ball is recognized and the position of the Servo is changed based on the Circle center location in the image, so in this case shown in Figure 24 the servo will turn to the left by 18 degrees.

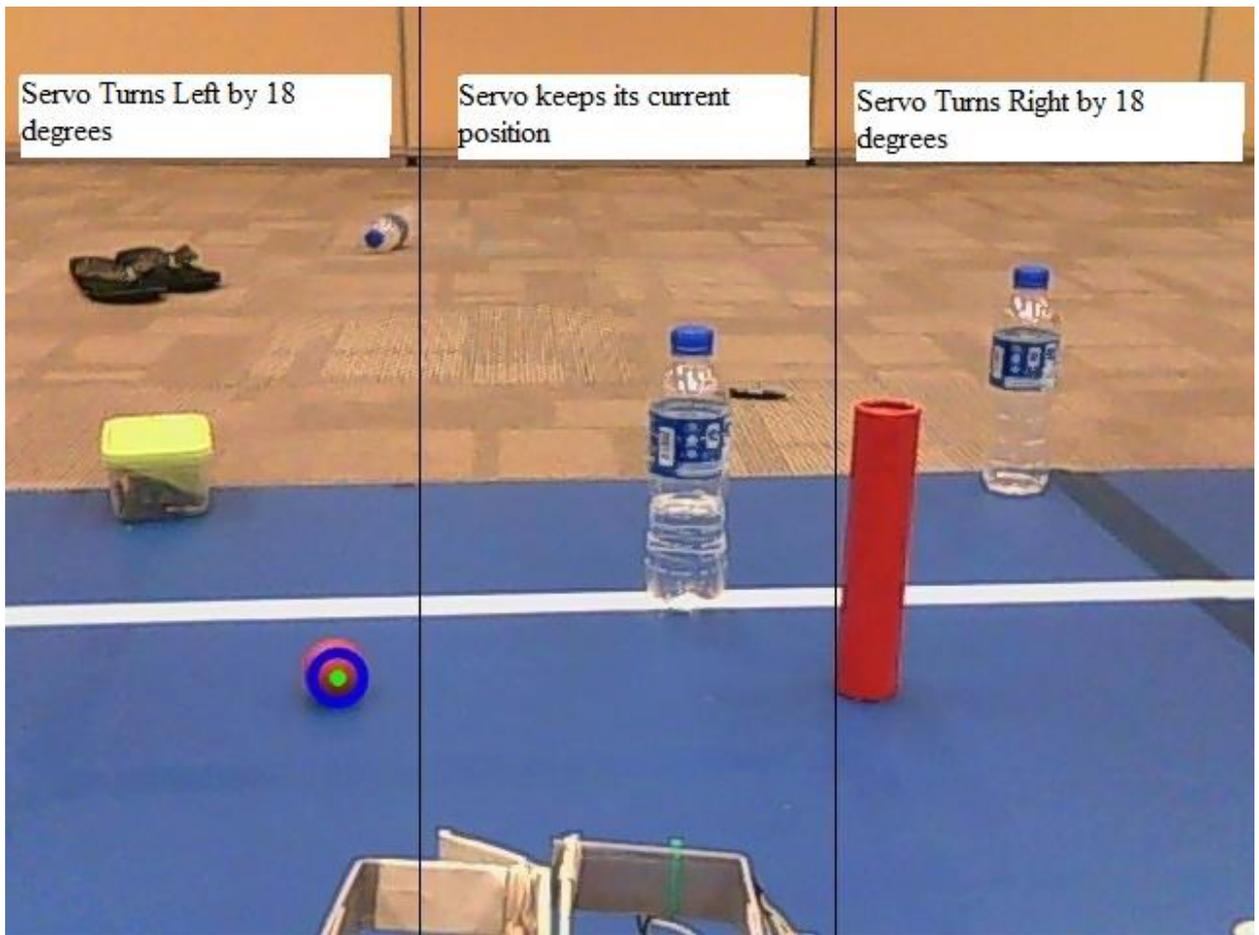


Figure 24: Servo Motor Control based on object location (Recognized Circle is shown in Blue with green Center)

The prototype was run under several conditions, and was also presented during ELECTREX exhibition in UTP. The output of the code shows no overhead delay after connecting the servo, except for the time to move the servo which takes around one second (manually measured).

More testes are being carried out to measure the power consumption during various conditions of the software runs.

#### 4.4 PROBLEMS FACED AND PROPOSED SOLUTIONS:

The main problems faced with Raspberry PI were:

- 1- Low USB power output:
  - Solution was to use a Powered USB-Hub.
- 2- Blanking of HDMI screen (Screen blanks with any processing):
  - HDMI current in boot options (config.txt) is set to the lowest value
  - Resolution is changed to a less blanking option
- 3- Overheating:
  - Heat sinks installed to avoid IC over current. According to [20], main sources of heat in the Raspberry PI are; the SOC (System On Chip) (center), the USB/Ethernet controller (right), and the voltage regulator (left), as shown in Figure (23). The board after installing the Heat sinks is shown in Figure (24).

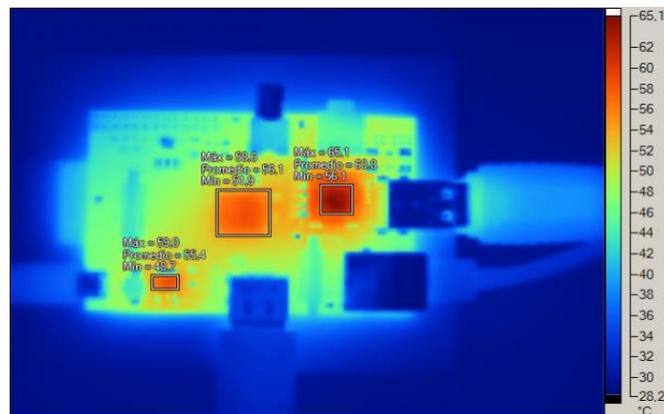


Figure (25): Raspberry PI sources of heat



Figure (26): Raspberry PI after installing heat sinks

#### 4.5 POWER CONSUMPTION MEASUREMENTS:

For Power consumption measurements the Raspberry PI is powered up from GPIO pins: 5 Volt and GND.

- 1- under normal Operating condition (Peripherals connected: monitor, USB keyboard and mouse, USB Camera), as shown in Figure (27):
  - The Raspberry PI draws 0.95 A
  - Average Power = 4.75 W

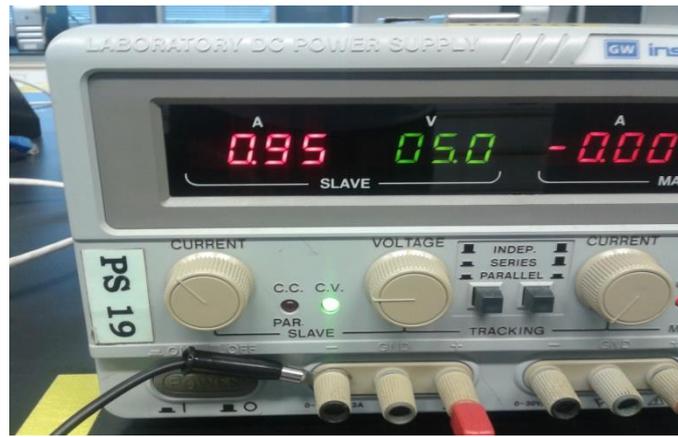


Figure (27): Power consumption measurements under normal operating conditions

- 2- Under TightVNC Remote Control while idle:
  - The Raspberry PI draws 0.44 A
  - Average Power = 2.2 W
- 3- Remote Control while compiling or code running without Servo movement:
  - The Raspberry PI draws 0.55 A
  - Average Power = 2.75 W
- 4- Remote Control while code running with Servo movement:
  - The Raspberry PI draws 0.67 A
  - Average Power = 3.35 W

## CHAPTER 5

### CONCLUSION AND RECOMMENDATION

#### 5.1 CONCLUSION:

Digital image processing is becoming a vital part of many applications, specially navigation and medical application. In order to make digital image processing applicable to more fields, embedded platforms are used to implement specific image processing tasks, which is the main aim of this project.

The project uses a Raspberry-Pi, which is a relatively powerful and reasonably priced embedded platform, which is becoming more used in educational and research applications. The Raspberry-Pi is used with a webcam and an object recognition script, to recognize and track a specified object, namely a red circular object.

The Object recognition algorithm is based on Circular Hough Transform (CHT). However, some preprocessing operations must be performed on the image, through image processing, before it becomes suitable to undergo CHT. These preprocessing operations are; Color detection (thresholding), Image restoration (noise filtering) and edge detection.

A C++ code was developed to implement the algorithm along with its preprocessing operations. The code was developed using Microsoft Visual Studio compiler and OpenCV library on the PC, and implemented on Raspberry PI using Cmake and OpenCV library.

The code was successfully compiled and run on Raspberry PI, giving results consistent was the results obtained from the PC, however, the processing time is much higher.

Three implementations of the algorithm were done, they make use of; Arrays, Pointers and OpenCV functions. All three implementations were assessed in terms of their capabilities and processing times.

The USB camera was attached to a Servo Motor that is controlled by the Raspberry PI PWM pin in a way to centralize the Object in the middle one third of the image. The prototype was tested under various conditions.

There are some problems that the Raspberry PI suffers from; overheating, Low USB power and screen blanking. Some solutions were implemented to solve these arising problems as adding heat sinks to the board IC units and voltage regulator.

Further improvements can be done to the algorithm and the code to increase the accuracy and decrease the processing time; such as improving the tracking algorithm, and overclocking of the board SOC.

## **5.2 RECOMMENDATION AND FUTURE DEVELOPMENT:**

As this project does not just address the problem of implementing image processing applications on Raspberry PI, it also examines three of the possible ways to implement these applications and comparison between them for different applications.

So it can be concluded that generally, Arrays and Pointers implementations provides control on the algorithm and robustness of the program, however, they have more processing time, which is also exponentially proportional to median filter block size, radius and angle step sizes in the Hough Transform. OpenCV functions provides less processing time that also is not much affected by changing these parameters, but it lacks control on some of the code aspects such as angle step size and the number of votes achieved by the recognized circles.

There are many ways to carry this project forward; the first and most important initiative is by replacing the OpenCV functions used only to acquire the image and write it back to the PC or Raspberry PI with C or C++ syntax, so that there will be no need to install or use the OpenCV library. One other initiative is to find ways to create the code into an independent executable file that can be portable to platforms without the need to install OpenCV or even compile the code on the platform.

## REFERNCES:

- [1] Space Foundation, “Digital Image Processing - Medical Applications”, Internet: <http://www.spacefoundation.org/programs/space-technology-hall-fame/inducted-technologies/digital-image-processing-medical>, 1994.
- [2] S. Liangwongsan *et al.* “Extracted Circle Hough Transform and Circle Defect Detection algorithm”, *International Science Index*, World Academy of Science Engineering and Technology, Vol. 5, No. 12, 2011.
- [3] R. Duda. and P. Hart. Use of the Hough transform to detect lines and curves in pictures. *Communications of the ACM*,15(1):11–15, January 1972.
- [4] Wikipedia, “Hough space plot example”, Internet: [http://en.wikipedia.org/wiki/Hough\\_transform](http://en.wikipedia.org/wiki/Hough_transform), February 19, 2006.
- [5] Jung, C.R.; Schramm, R., "Rectangle detection based on a windowed Hough transform," *Computer Graphics and Image Processing*, 2004. Proceedings. 17th Brazilian Symposium on, vol., no., pp.113, 120, 17-20 Oct. 2004
- [6] G. T. Shrivakshan, C. Chandrasekar, “A Comparison of various Edge Detection Techniques used in Image Processing”, *IJCSI International Journal of Computer Science Issues*, Vol. 9, Issue 5, No 1, September 2012.
- [7] H. Rhody. (2005, Oct. 11). Lecture 10: Hough Circle Transform [Online]. Available: [https://www.cis.rit.edu/class/simg782/lectures/lecture\\_10/lec782\\_05\\_10.pdf](https://www.cis.rit.edu/class/simg782/lectures/lecture_10/lec782_05_10.pdf)
- [8] T. Gevers, J. Weijer, and H. Stokman.“Color feature detection”. *Color Image Processing: Emerging Applications*.
- [9] The Raspberry Pi Foundation,” DOWNLOADS”, Internet: <http://www.raspberrypi.org/downloads/> , [Dec. 20, 2014]
- [10] SourceForge, Internet: [http://sourceforge.net/projects/win32diskimager/?source=typ\\_redirect](http://sourceforge.net/projects/win32diskimager/?source=typ_redirect) , [Dec. 20, 2014]

- [11] The Raspberry Pi Foundation, “INSTALLING OPERATING SYSTEM IMAGES USING WINDOWS”, Internet:  
<http://www.raspberrypi.org/documentation/installation/installing-images/windows.md> , [Dec. 20, 2014]
- [12] The Raspberry Pi Foundation, “RASPI-CONFIG”, Internet:  
<http://www.raspberrypi.org/documentation/configuration/raspi-config.md> , [Dec. 20, 2014]
- [13] The Raspberry Pi Foundation, “CONFIG.TXT”, Internet:  
<http://www.raspberrypi.org/documentation/configuration/config-txt.md> , [Dec. 20, 2014]
- [14] B. Gaines. “Instructions for setting up a New Raspberry Pi OS and OpenCV”, Internet:  
[https://docs.google.com/document/d/1wmcaBUogffbxD78IghNyag4jl2gAZp1d\\_9HLkBGriLA/edit](https://docs.google.com/document/d/1wmcaBUogffbxD78IghNyag4jl2gAZp1d_9HLkBGriLA/edit) , March, 2014 [Dec. 20, 2014].
- [15] R. Castle. “Installing OpenCV on a Raspberry Pi”, Internet:  
<http://robertcastle.com/2014/02/installing-opencv-on-a-raspberry-pi/> , Feb. 22, 2014 [Dec. 20, 2014].
- [16] The Raspberry Pi Foundation, “VNC (VIRTUAL NETWORK COMPUTING)”, Internet:  
<http://www.raspberrypi.org/documentation/remote-access/vnc/> [Feb. 20, 2014]
- [17] opencv dev team, “Hough Circle Transform”, Internet:  
[http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough\\_circle/hough\\_circle.html](http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html), Feb 25, 2015 [April 4, 2015].
- [18] B. Traynor. “RPi Low-level peripherals”, Internet:  
[http://elinux.org/RPi\\_Low-level\\_peripherals](http://elinux.org/RPi_Low-level_peripherals) , [Jan. 14, 2015 [April 4, 2015].

- [19] F. Buss. “PWM example”, Internet:  
<http://www.frank-buss.de/raspberrypi/pwm.c>, 2012 [April 4, 2015].
- [20] M. Dornisch. “DIY Raspberry Pi Heat Sink”, Internet:  
<http://www.michaeldornisch.com/2012/06/diy-raspberry-pi-heat-sink.html> ,  
June 25, 2012 [Dec. 20, 2014].

## APPENDIX A: C++ Code

```
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <stdio.h>
#include "opencv2/opencv.hpp"

#include <stdlib.h>
#include <math.h>
//===== Servo Settings start =====//
#define BCM2708_PERI_BASE 0x20000000
#define GPIO_BASE (BCM2708_PERI_BASE + 0x200000) /* GPIO controller */
#define PWM_BASE (BCM2708_PERI_BASE + 0x20C000) /* PWM controller */
#define CLOCK_BASE (BCM2708_PERI_BASE + 0x101000)

#define PWM_CTL 0
#define PWM_RNG1 4
#define PWM_DAT1 5

#define PWMCLK_CNTL 40
#define PWMCLK_DIV 41

//#include <stdio.h>
#include <string.h>
//#include <stdlib.h>
#include <dirent.h>
#include <fcntl.h>
#include <assert.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>

#include <unistd.h>

#define PAGE_SIZE (4*1024)
#define BLOCK_SIZE (4*1024)

// I/O access
volatile unsigned *gpio;
volatile unsigned *pwm;
volatile unsigned *clk;

// GPIO setup macros. Always use INP_GPIO(x) before using OUT_GPIO(x) or SET_GPIO_ALT(x,y)
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |= (1<<(((g)%10)*3))
#define SET_GPIO_ALT(g,a) *(gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?3:2)<<(((g)%10)*3))

#define GPIO_SET *(gpio+7) // sets bits which are 1 ignores bits which are 0
#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which are 0

volatile unsigned *mapRegisterMemory(int base);
void setupRegisterMemoryMappings();
void setServo(int percent);
void initHardware();
//===== Servo Settings end =====//
using namespace cv;
using namespace std;

#define x 480
#define y 640
```

```

#define block 3
#define r_step 5
#define sobel_max 3*255
#define edge_max 4*255
#define edge_min 255
int r_thresh=240;
int r_min=30;
double theta_step=0.35;
int wait;
unsigned char img[x][y][3];
int accum[240][x][y];
unsigned char img_gray[x][y];
unsigned char img_median[x][y];
unsigned char img_filtered[x][y];
int img_edge[x][y];

int red_max_perc=80;
int red_min_perc=40;
int green_max_perc=30;
int green_min_perc=10;
int blue_max_perc=30;
int blue_min_perc=7;

long long starttime,
endtime,endtime_opencv_1,endtime_opencv_2,starttime_opencv_2,endtime_pointer_1,starttime_pointer_2,
endtime_pointer_2;
double sec;

void sort_array(int array[],int index);
void zero_array(int array[],int index);
void sort_array_pointer(unsigned char*,int index);
void zero_array_pointer(unsigned char*,int index);

int main(){
    // init PWM module for GPIO pin 18 with 50 Hz frequency
    initHardware();

FILE *fi,*fo;
printf("Program Start\nParameters: Radius step:%2d\tTheta step: %2.2f\tMedian block: %d\n",r_step,
theta_step, block);
Mat bugz(x,y,CV_8UC3);
int sz[3] = {x,y,3};
Mat img_cap(3,sz, CV_8UC1, Scalar::all(1));
int sez[3] = {x,y,3};
Mat img_show(3,sez, CV_8UC1, Scalar::all(0));
Mat img_show_opencv(3,sez, CV_8UC1, Scalar::all(0));
Mat img_show_pointer(3,sez, CV_8UC1, Scalar::all(0));
Mat img_show_gray(x,y, CV_8UC1, Scalar::all(0));
Mat img_show_filtered(x,y, CV_8UC1, Scalar::all(0));
Mat img_show_edge(x,y, CV_8UC1, Scalar::all(0));
Mat img_show_filtered_opencv(x,y, CV_8UC1, Scalar::all(0));
Mat img_show_edge_opencv(x,y, CV_8UC1, Scalar::all(0));
Mat img_show_gray_pointer(x,y, CV_8UC1, Scalar::all(0));
Mat img_show_filtered_pointer(x,y, CV_8UC1, Scalar::all(0));
Mat img_show_edge_pointer(x,y, CV_8UC1, Scalar::all(0));
VideoCapture cap;

try
{
    setServo(servo_angle);

```

```

        sleep(1);
        printf("\nServo at %d degree", servo_angle)

while(1)
{

starttime=getTickCount();
cap.open(0);
waitKey(10);
cap>>bugz;
cap>>img_cap;
cap>>img_show;
cap>>img_show_pointer;
cap>>img_show_opencv;
cap.release();
endtime_pointer_1=getTickCount();

int i=0,j=0,h=0;
char cd;
for(i=0;i<240;i++)for(j=0;j<x;j++)for(h=0;h<3;h++)img[i][j][h]=0;
for(i=0; i<480; i++)
{

    for(j=0; j<640; j++)
    {
        for(h=0; h<3; h++)
        {
            img[i][j][2-h]=(*(img_cap.data+(y*3*i)+(3*j)+(h)));
        }
    }
}
//----- Color Detection -----
-----//

for(i=0;i<240;i++)for(j=0;j<x;j++)img_gray[i][j]=0;
double red, green, blue;
for(i=1;i<x;i++)
{
    for(j=1;j<y;j++)
    {
        red=(100.0*(img[i][j][0])/((img[i][j][0])+(img[i][j][1])+(img[i][j][2])));
        green=(100.0*(img[i][j][1])/((img[i][j][0])+(img[i][j][1])+(img[i][j][2])));
        blue=(100.0*(img[i][j][2])/((img[i][j][0])+(img[i][j][1])+(img[i][j][2])));
        // if(red<=red_max_perc && red>=red_min_perc && green<=green_max_perc &&
green>=green_min_perc && blue<=blue_max_perc && blue>=blue_min_perc &&
img[i][j][0]>=edge_thresh)img_gray[i][j]=255;
        if(red<=red_max_perc && red>=red_min_perc && green<=green_max_perc &&
green>=green_min_perc && blue<=blue_max_perc && blue>=blue_min_perc)img_gray[i][j]=255;

        else img_gray[i][j]=0;
    }
}

printf("\nColor Detection Done");
//----- Write Mat Color Detection -----
-----//

for(int i=0; i<x; i++)
{

```

```

        for(int j=0; j<y; j++)
        {

                (*(img_show_gray.data+(y*i)+(j)))=img_gray[i][j];

        }
}

endtime_opencv_1=getTickCount();
//----- Median Filter -----
-----//

int counter=0,max=0,k=0,l=0;
int m=block/2;
int n=m;
if ((block%2)==0)n=m+1;
double total=0;
for(i=0;i<x;i++)for(j=0;j<y;j++)img_median[i][j]=0;
for(i=0;i<x;i++)for(j=0;j<y;j++)img_filtered[i][j]=0;
int hold[block*block];

for(i=m; i<x-n-1; i++)
{
        for(j=m; j<y-n-1; j++)
        {
                counter=0;
                max=0;
                zero_array(hold,block*block);
                for(k=i-m;k<=i+n;k++)for(l=j-m;l<=j+n;l++){
                        hold[counter]=img_gray[k][l];
                        counter++;
                }
                sort_array(hold,block*block);
                img_filtered[i][j]=hold[counter/2];
        }
}
printf("\nMedian Filter Done");
//----- Write Mat Median Filter -----
-----//

for(int i=0; i<x; i++)
{
        for(int j=0; j<y; j++)
        {

                (*(img_show_filtered.data+(y*i)+(j)))=img_filtered[i][j];

        }
}

//----- Edge Detection -----
-----//
int dx=0,dy=0;
for(i=0;i<240;i++)for(j=0;j<x;j++)img_edge[i][j]=0;
for(i=1;i<x-1;i++)
{

```

```

for(j=1;j<y-1;j++)
{
    dx = (img_filtered[i+1][j+1])+(2*img_filtered[i+1][j])+ (img_filtered[i+1][j-1])-
(img_filtered[i-1][j-1])-(2*img_filtered[i-1][j])-(img_filtered[i-1][j+1]);
    dy = (img_filtered[i+1][j+1])+(2*img_filtered[i][j+1])+ (img_filtered[i-1][j+1])-
(img_filtered[i-1][j-1])-(2*img_filtered[i][j-1])-(img_filtered[i+1][j-1]);

    img_edge[i][j]=((abs(dx)+abs(dy))>=sobel_max)?255:0;

}
}
//for(i=1;i<x-1;i++)for(j=1;j<y-1;j++)if((img_edge[i-1][j-1]+img_edge[i-1][j]+img_edge[i-
1][j+1]+img_edge[i][j-1]+img_edge[i][j]+img_edge[i][j+1]+img_edge[i+1][j-
1]+img_edge[i+1][j]+img_edge[i+1][j+1])>=4*255)img_edge[i][j]=0;
printf("\nEdge Detection Done");

//-----Write Mat Edge-----//

for(int i=0; i<x; i++)
{
    for(int j=0; j<y; j++)
    {

        (*(img_show_edge.data+(y*i)+(j)))=img_edge[i][j];

    }
}

//----- Hough Transform -----
-----//
for(i=0;i<240;i++)for(j=0;j<x;j++)for(k=0;k<y;k++)accum[i][j][k]=0;
double theta=0,pi=22.0/7.0;
int x_max=0,y_max=0,r_max=1,r=0;
m=0;
n=0;
int max_accum=0;
for(i=1;i<x;i++)
{
    for(j=1;j<y;j++)
    {

        if(img_edge[i][j]== 255)
        {

            for(r=(r_min);r<(r_thresh);r+=r_step)
            {

                for(theta=0;theta<=(2*pi);theta+=theta_step)
                {

                    m=ceil(i+(r*(cos(theta))));
                    n=ceil(j+(r*(sin(theta))));

                    if((m>=0) && (m<x) && (n>=0) && (n<y))
                    {

                        accum[r][m][n]++;
                    }
                }
            }
        }
    }
}

```

```

        if((accum[r][m][n])>max_accum)
        {
            max_accum=accum[r][m][n];
            x_max=m;
            y_max=n;
            r_max=r;
        }
    }
}
}
}
}
}
printf("\nHough Transform Done");
printf("\nCircle Detected:\nCoordinates: X: %d\tY: %d\tRadius: %d\tNo of
votes:%d",x_max,y_max,r_max,max_accum);
//----- Hough Transform -----
-----//
endtime=getTickCount();
sec=((endtime-starttime)/(getTickFrequency()));
printf("\nExecution Time (Seconds): %3.2f\n", sec);

//----- Highlight Circle-----
-----//

for(j=1;j<y;j++)
{
    if(x_max>=0 && x_max<x)
    {
        img[x_max][j][0]=255;
        img[x_max][j][1]=0;
        img[x_max][j][2]=0;
    }
}
for(i=1;i<x;i++)
{
    if(y_max>=0 && y_max<y)
    {
        img[i][y_max][0]=255;
        img[i][y_max][1]=0;
        img[i][y_max][2]=0;
    }
}

for(i=x_max-r_max;i<=x_max+r_max;i+=(2*r_max))
{
    for(j=y_max-r_max;j<=y_max+r_max;j+=(2*r_max))
    {
        for(l=-r_max;l<=r_max;l++)
        {
            for(k=-r_max;k<=r_max;k++)
            {
                if((i>=0) && (i<x) && (j>=0) && (j<y))
                {
                    img[i][y_max+k][0]=255;
                    img[i][y_max+k][1]=0;
                }
            }
        }
    }
}

```

```

        img[i][y_max+k][2]=0;
    }
    if((i>=0) && (i<x) && (j>=0) && (j<y))
    {
        img[x_max+1][j][0]=255;
        img[x_max+1][j][1]=0;
        img[x_max+1][j][2]=0;
    }
}
}
}

//----- Write Highlighted Image-----//

for(int i=0; i<x; i++)
{
    for(int j=0; j<y; j++)
    {
        for(int h=0; h<3; h++)
        {
            *(img_show.data+(y*3*i)+(3*j)+(h))=img[i][j][2-h];
        }
    }
}

imwrite("output_c_img_array.jpg",img_show);
imwrite("gray_c_img_array.jpg",img_show_gray);
imwrite("filtered_c_img_array.jpg",img_show_filtered);
imwrite("edge_c_img_array.jpg",img_show_edge);

//===== OPENCV =====//
starttime_opencv_2=getTickCount();

printf("\n===== \n\n\nOPENCV Implementation:\n");
medianBlur(img_show_gray, img_show_filtered_opencv, block);
printf("\nMedian Filter Done");
Canny( img_show_filtered_opencv, img_show_edge_opencv, edge_min, edge_max, 3 );
printf("\nEdge Detection Done");
vector<Vec3f> circles;
HoughCircles(img_show_edge_opencv, circles, CV_HOUGH_GRADIENT, 1, 1,200,2*3*r_min/8, r_min,
r_thresh);

Vec3i c = circles[0];
printf("\nHough Transform Done");
printf("\nCircle Detected:\nCoordinates: X: %d\tY: %d\tRadius: %d\tNo of votes: N/A",c[1],
c[0],c[2]);
    circle( img_show_opencv, Point(c[0], c[1]), c[2], Scalar(0,0,255), 3, CV_AA);
    circle( img_show_opencv, Point(c[0], c[1]), 2, Scalar(0,255,0), 3, CV_AA);
imwrite("filtered_c_img_opencv.jpg",img_show_filtered_opencv);
imwrite("edge_c_img_opencv.jpg",img_show_edge_opencv);
imwrite("output_c_img_opencv.jpg",img_show_opencv);

endtime_opencv_2=getTickCount();

sec((((endtime_opencv_1-starttime)/(getTickFrequency()))+((endtime_opencv_2-
starttime_opencv_2)/(getTickFrequency())));
printf("\nExecution Time (Seconds): %3.2f\n", sec);

```

```

//===== Pointers =====//
printf("\n===== \n\nPOINTERS & Memory Allocation
Implementation:\n");

starttime_pointer_2=getTickCount();
unsigned char* img_gray_pointer;
img_gray_pointer=(unsigned char*)malloc(x*y*sizeof(unsigned char));
for(i=0;i<x;i++)
{
    for(j=0;j<y;j++)
    {
        red=(100.0*(*(img_cap.data+(y*3*i)+(3*j)+(2)))/((*(img_cap.data+(y*3*i)+(3*j)+(0)))+(*(img_c
ap.data+(y*3*i)+(3*j)+(1)))+(*(img_cap.data+(y*3*i)+(3*j)+(2)))));

        green=(100.0*(*(img_cap.data+(y*3*i)+(3*j)+(1)))/((*(img_cap.data+(y*3*i)+(3*j)+(0)))+(*(img_cap.d
ata+(y*3*i)+(3*j)+(1)))+(*(img_cap.data+(y*3*i)+(3*j)+(2)))));

        blue=(100.0*(*(img_cap.data+(y*3*i)+(3*j)+(0)))/((*(img_cap.data+(y*3*i)+(3*j)+(0)))+(*(img_cap.da
ta+(y*3*i)+(3*j)+(1)))+(*(img_cap.data+(y*3*i)+(3*j)+(2)))));

        if(red<=red_max_perc && red>=red_min_perc && green<=green_max_perc && green>=green_min_perc
&& blue<=blue_max_perc && blue>=blue_min_perc)*(img_gray_pointer+(y*i)+j)=255;

        else *(img_gray_pointer+(y*i)+j)=0;
    }
}

printf("\nColor Detection Done");
//----- Write Mat Color Detection -----//

for(int i=0; i<x; i++)
{
    for(int j=0; j<y; j++)
    {

        *(img_show_gray_pointer.data+(y*i)+(j))=*(img_gray_pointer+(y*i)+j);

    }
}

endtime_opencv_1=getTickCount();
//----- Median Filter -----//
unsigned char* img_filtered_pointer;
img_filtered_pointer=(unsigned char*)calloc(x*y,sizeof(unsigned char));
if(img_filtered_pointer == NULL)
{
    printf("Error: img_filtered_pointer");
    getchar();
    exit(0);
}

counter=0,max=0,k=0,l=0;

m=block/2;

```

```

n=m;
if((block%2)==0)n=m+1;

//for(i=0;i<240;i++)for(j=0;j<x;j++)img_filtered[i][j]=0;

unsigned char* hold_pointer;
hold_pointer=(unsigned char*)calloc(block*block,sizeof(unsigned char));
for(i=m; i<x-n-1; i++)
{
    for(j=m; j<y-n-1; j++)
    {
        counter=0;
        zero_array_pointer(hold_pointer,block*block);
        for(k=i-m;k<=i+n;k++)
        {
            for(l=j-m;l<=j+n;l++)
            {
                (*(hold_pointer+counter))=*(img_gray_pointer+(y*k)+(l));
                counter++;
            }
        }
        sort_array_pointer(hold_pointer,block*block);

        (*(img_filtered_pointer+(y*i)+j))=*(hold_pointer+(counter/2));
    }
}
printf("\nMedian Filter Done");
//----- Write Mat Median Filter -----
//-----//

for(int i=0; i<x; i++)
{
    for(int j=0; j<y; j++)
    {

        (*(img_show_filtered_pointer.data+(y*i)+(j)))=*(img_filtered_pointer+(y*i)+j));
    }
}

//----- Edge Detection -----
//-----//
unsigned char* img_edge_pointer;
img_edge_pointer=(unsigned char*)calloc(x*y,sizeof(unsigned char));
if(img_edge_pointer == NULL)
{
    printf("Error: img_filtered_pointer");
    getchar();
    exit(0);
}
dx=0,dy=0;
for(i=1;i<x-1;i++)

```

```

{
    for(j=1;j<y-1;j++)
    {

        dx =
        (*(img_filtered_pointer+y*(i+1)+(j+1)))+(2*(*(img_filtered_pointer+y*(i+1)+(j))))+(*(img_filtered_p
ointer+y*(i+1)+(j-1)))-(*(img_filtered_pointer+y*(i-1)+(j+1)))-(2*(*(img_filtered_pointer+y*(i-
1)+(j))))-(*(img_filtered_pointer+y*(i-1)+(j-1)));
        dy =
        (*(img_filtered_pointer+y*(i+1)+(j+1)))+(2*(*(img_filtered_pointer+y*(i)+(j+1))))+(*(img_filtered_p
ointer+y*(i-1)+(j+1)))-(*(img_filtered_pointer+y*(i-1)+(j-1)))-(2*(*(img_filtered_pointer+y*(i)+(j-
1))))-(*(img_filtered_pointer+y*(i-1)+(j-1)));

        (*(img_edge_pointer+(y*i)+j))=((abs(dx)+abs(dy))>=sobel_max)?255:0;

    }
}
printf("\nEdge Detection Done");

//-----Write Mat Edge-----//

for(int i=0; i<x; i++)
{
    for(int j=0; j<y; j++)
    {

        (*(img_show_edge_pointer.data+(y*i)+(j)))=(*(img_edge_pointer+(y*i)+j));

    }
}

//----- Hough Transform -----//
-----//

unsigned char* accum_pointer;
accum_pointer=(unsigned char*)calloc(240*x*y,sizeof(unsigned char));
if(accum_pointer == NULL)
{
    printf("Error: img_filtered_pointer");
    getchar();
    exit(0);
}
theta=0,pi=22.0/7.0;
x_max=0,y_max=0,r_max=1,r=0;
m=0;
n=0;
max_accum=0;
int holder=0;
for(i=1;i<x;i++)
{
    for(j=1;j<y;j++)
    {

        if(*(img_edge_pointer+(y*i)+j)== 255)
        {

            for(r=(r_min);r<(r_thresh);r+=r_step)

```

```

{
    for(theta=0;theta<=(2*pi);theta+=theta_step)
    {
        m=ceil(i+(r*(cos(theta))));
        n=ceil(j+(r*(sin(theta))));

        if((m>=0) && (m<x) && (n>=0) && (n<y))
        {
            holder=++(*(accum_pointer+((x*y)*r)+(y*m)+n));

            if(holder>max_accum)
            {
                max_accum=holder;
                x_max=m;
                y_max=n;
                r_max=r;
            }
        }
    }
}
printf("\nHough Transform Done");
printf("\nCircle Detected:\nCoordinates: X: %d\tY: %d\tRadius: %d\tNo of
votes:%d",x_max,y_max,r_max,max_accum);
//----- Hough Transform -----
-----//
endtime_pointer_2=getTickCount();
sec=((endtime_pointer_2-starttime_pointer_2)/(getTickFrequency()))+((endtime_pointer_1-
starttime)/(getTickFrequency()));
printf("\nExecution Time (Seconds): %3.2f\n", sec);
//..... Set Servo
Angle.....//
if (y_max <= y/3) servo_angle-=10; //10 % which translates to 18 degrees.
else if (y_max <= 2*y/3) servo_angle-=0;
else if (y_max <= y) servo_angle+=10;

if (servo_angle > 100) servo_angle=100;
if (servo_angle < 0) servo_angle=0;
//----- Highlight Circle-----
-----//

for(j=1;j<y;j++)
{
    if(x_max>=0 && x_max<x)
    {
        (*(img_show_pointer.data+(y*3*(x_max))+(3*(j))+(2-0)))=0;
        (*(img_show_pointer.data+(y*3*(x_max))+(3*(j))+(2-1)))=0;
        (*(img_show_pointer.data+(y*3*(x_max))+(3*(j))+(2-2)))=255;
    }
}
for(i=1;i<x;i++)
{
    if(y_max>=0 && y_max<y)
    {

```

```

                (*(img_show_pointer.data+(y*3*i)+(3*(y_max))+(2-0)))=0;
                (*(img_show_pointer.data+(y*3*i)+(3*(y_max))+(2-1)))=0;
                (*(img_show_pointer.data+(y*3*i)+(3*(y_max))+(2-2)))=255;
            }
        }

for(i=x_max-r_max;i<=x_max+r_max;i+=(2*r_max))
{
    for(j=y_max-r_max;j<=y_max+r_max;j+=(2*r_max))
    {
        for(l=-r_max;l<=r_max;l++)
        {
            for(k=-r_max;k<=r_max;k++)
            {
                if((i>=0) && (i<x) && (j>=0) && (j<y))
                {
                    (*(img_show_pointer.data+(y*3*i)+(3*(y_max+k))+(2-0)))=0;
                    (*(img_show_pointer.data+(y*3*i)+(3*(y_max+k))+(2-1)))=0;
                    (*(img_show_pointer.data+(y*3*i)+(3*(y_max+k))+(2-2)))=255;
                }
                if((i>=0) && (i<x) && (j>=0) && (j<y))
                {
                    (*(img_show_pointer.data+(y*3*(x_max+1))+(3*j)+(2-0)))=0;
                    (*(img_show_pointer.data+(y*3*(x_max+1))+(3*j)+(2-1)))=0;
                    (*(img_show_pointer.data+(y*3*(x_max+1))+(3*j)+(2-2)))=255;
                }
            }
        }
    }
}
//----- Write Highlighted Image-----//

imwrite("gray_c_img_pointer.jpg",img_show_gray_pointer);
imwrite("filtered_c_img_pointer.jpg",img_show_filtered_pointer);
imwrite("edge_c_img_pointer.jpg",img_show_edge_pointer);
imwrite("output_c_img_pointer.jpg",img_show_pointer);

setServo(servo_angle);
sleep(1);
printf("\nServo at %d degree", servo_angle)

}

}
catch (Exception& e)
{
    const char* err_msg = e.what();
    std::cout << "exception caught: imshow:\n" << err_msg << std::endl;
}

cout << "\n* End of Program *";
getchar();

return 0;

}
void sort_array(int array[],int index)
{
    int i=0,j=0,mid=0;

```

```

for(i=1;i<index;i++)
{
    for(j=1;j<index;j++)
    {
        if(array[j-1]<array[j])
        {
            mid=array[j];
            array[j]=array[j-1];
            array[j-1]=mid;
        }
    }
}

void zero_array(int array[],int index)
{
    int j=0;
    for(j=0;j<index;j++)array[j]=0;
}

void sort_array_pointer(unsigned char* poin,int index)
{
    int i=0,j=0,mid=0;
    for(i=1;i<index;i++)
    {
        for(j=1;j<index;j++)
        {
            if((*poin+j-1)<(*poin+j))
            {
                mid=(*poin+j);
                (*poin+j)=(*poin+j-1);
                (*poin+j-1)=mid;
            }
        }
    }
}

void zero_array_pointer(unsigned char* poin,int index)
{
    int j=0;
    for(j=0;j<index;j+)(*poin+j)=0;
}
//=====
===== servo function declaration
// map 4k register memory for direct access from user space and return a user space pointer to it
volatile unsigned *mapRegisterMemory(int base)
{
    static int mem_fd = 0;
    char *mem, *map;

    /* open /dev/mem */
    if (!mem_fd) {
        if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
            printf("can't open /dev/mem \n");
            exit (-1);
        }
    }
}

```

```

/* mmap register */

// Allocate MAP block
if ((mem = static_cast<char*>(malloc(BLOCK_SIZE + (PAGE_SIZE-1)))) == NULL) {
    printf("allocation error \n");
    exit (-1);
}

// Make sure pointer is on 4K boundary
if ((unsigned long)mem % PAGE_SIZE)
    mem += PAGE_SIZE - ((unsigned long)mem % PAGE_SIZE);

// Now map it
map = (char *)mmap(
    (caddr_t)mem,
    BLOCK_SIZE,
    PROT_READ|PROT_WRITE,
    MAP_SHARED|MAP_FIXED,
    mem_fd,
    base
);

if ((long)map < 0) {
    printf("mmap error %d\n", (int)map);
    exit (-1);
}

// Always use volatile pointer!
return (volatile unsigned *)map;
}

// set up a memory regions to access GPIO, PWM and the clock manager
void setupRegisterMemoryMappings()
{
    gpio = mapRegisterMemory(GPIO_BASE);
    pwm = mapRegisterMemory(PWM_BASE);
    clk = mapRegisterMemory(CLOCK_BASE);
}

void setServo(int percent)
{
    int bitCount;
    unsigned int bits = 0;

    // 32 bits = 2 milliseconds
    bitCount = 16 + 16 * percent / 100;
    if (bitCount > 32) bitCount = 32;
    if (bitCount < 1) bitCount = 1;
    bits = 0;
    while (bitCount) {
        bits <<= 1;
        bits |= 1;
        bitCount--;
    }
    *(pwm + PWM_DAT1) = bits;
}

// init hardware
void initHardware()
{
    // mmap register space
    setupRegisterMemoryMappings();
}

```

```

// set PWM alternate function for GPIO18
SET_GPIO_ALT(18, 5);

// stop clock and waiting for busy flag doesn't work, so kill clock
*(clk + PWMCLK_CNTL) = 0x5A000000 | (1 << 5);
usleep(10);

// set frequency
// DIVI is the integer part of the divisor
// the fractional part (DIVF) drops clock cycles to get the output frequency, bad for servo
motors
// 320 bits for one cycle of 20 milliseconds = 62.5 us per bit = 16 kHz
int idiv = (int) (19200000.0f / 16000.0f);
if (idiv < 1 || idiv > 0x1000) {
    printf("idiv out of range: %x\n", idiv);
    exit(-1);
}
*(clk + PWMCLK_DIV) = 0x5A000000 | (idiv<<12);

// source=osc and enable clock
*(clk + PWMCLK_CNTL) = 0x5A000011;

// disable PWM
*(pwm + PWM_CTL) = 0;

// needs some time until the PWM module gets disabled, without the delay the PWM module
crashes
usleep(10);

// filled with 0 for 20 milliseconds = 320 bits
*(pwm + PWM_RNG1) = 320;

// 32 bits = 2 milliseconds, init with 1 millisecond
setServo(0);

// start PWM1 in serializer mode
*(pwm + PWM_CTL) = 3;
}

```

## **APPENDIX B: Format of “CMakeLists.txt” file.**

```
cmake_minimum_required(VERSION 2.8)
project( fyp )
find_package( OpenCV REQUIRED )
add_executable( fyp fyp.cpp )
target_link_libraries( fyp ${OpenCV_LIBS} )
```

## **APPENDIX C: Format of “interfaces.txt” file.**

auto lo

iface lo inet loopback

iface eth0 inet static

address 10.10.10.2

netmask 255.255.255.0

gateway 10.10.10.1

allow-hotplug wlan0

iface wlan0 inet manual

wpa-roam /etc/wpa\_supplicant/wpa\_supplicant.conf

iface default inet dhcp