MCTR 1013 – Visual Servoing

# **Project**

|  |  |
| --- | --- |
| Team members’ names | IDs |
| Andrew Atef Thabet Mosaad | 25-0284 |
| Hussein Mohamed Mahmouod Fouad Shaker | 25-1348 |
| Ahmed Yassin Hassen Abd El Wares | 25-6169 |
| Karim Sherif Baher Botros | 25-8681 |
| Kirollos George Lamei Girgis | 25-9260 |

Contact person’s email (will be used by the TA to contact the team): andrew.atef-mosaad@student.guc.edu.eg

Contents

[Abstract 2](#_Toc450430039)

[Introduction 2](#_Toc450430040)

[1. Embedded systems. 2](#_Toc450430041)

[2. Board Comparison. 3](#_Toc450430042)

[3. Raspberry Pi and BeagleBone 4](#_Toc450430043)

[3.1 Hardware comparison 5](#_Toc450430044)

[4. Install OpenCV and Python on Raspberry Pi 2 6](#_Toc450430045)

[Image Processing using Raspberry Pi 8](#_Toc450430046)

[1. Road sign detection using OpenCV ORB 8](#_Toc450430047)

[1.1 Results 10](#_Toc450430048)

[2. Raspberry Pi Colored objects detection. 12](#_Toc450430049)

[2.1 Results 13](#_Toc450430050)

[References 15](#_Toc450430051)

# Abstract

Today image processing are used in various techniques, this report presents the implementation of image processing operations on Raspberry Pi. The Raspberry Pi is a basic embedded system and being a low cost a single-board computer used to reduce the complexity of systems in real time applications. This platform is mainly based on python. Raspberry pi is using opencv library on Raspbian OS.

In this project, we managed to track circular and colored objects on static images.

# Introduction

The image processing is a form of signal processing where the input is an image, like a photograph or video frame, the output of an image processing may be either an image or a video frame or a set of characteristics or parameters related to the image. The acquisition of digital image usually suffers from undesirable camera shakes and due to unstable random camera motions. Hence image enhancement algorithms are required to remove these unwanted camera shakes. This image processing concepts are implemented in Raspberry pi.

The Raspberry Pi is a basic embedded system having a credit card-sized single board computers developed in the UK by the Raspberry Pi Foundation. The Raspberry Pi is based on the Broadcom BCM2835 system on a chip (SOC) which includes an A 900MHz quad-core ARM Cortex-A7 CPU processor, Broadcom Video Core IV 3D GPU having 40 GPIO pins, 3.5W of power, and 1GB of RAM memory. The Raspberry Pi system has Secure SD card reader (models A and B) or Micro SD card reader (models A+ and B+) sockets for boot media and persistent storage. The system provides Raspbian Linux operating system. Python is used as main programming language for raspberry pi.

### 1. Embedded systems.

Embedded systems and supercomputers are at opposite ends of the computing spectrum, yet they share common design constraints. As the number of cores in large computers increases, the per-core power usage and cost becomes increasingly important. Luckily there is a class of processors that have already have been optimized for power and cost: those found in embedded systems. The use of embedded processors in supercomputers is not new; the various BlueGene machines use embedded-derived PowerPC chips. There is an ongoing push to continue this trend by taking ARM processors, such as those found in cellphones, and using them in supercomputing applications.

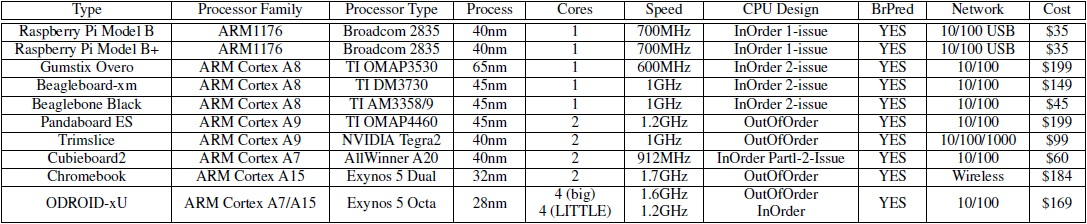
The uptake of ARM processors in supercomputers has started slowly, as vendors were waiting for the release of 64-bit processors with HPC friendly features (such as high-speed interconnects and fast memory hierarchies).

In the meantime it is possible to take readily-available commodity 32-bit ARM boards and use them to build computing clusters. We look at the performance and power characteristics of ten different 32-bit ARM development boards. Our goal is to find a low-cost, low-power, yet high-performance board for use in constructing an educational compute cluster.

After weighing the various tradeoffs, we chose the Raspberry Pi as the basis for our cluster. We built a prototype 32-node Raspberry Pi cluster with per-node energy measurement capabilities, and compare the power and performance tradeoffs with various x86 machines. Our cluster is flexible and can be easily expanded to 64 (or more) nodes.

## 2. Board Comparison.

Here we present the power and performance tradeoffs of ten different commodity 32-bit ARM boards, as listed in Table 1.The boards, all running Linux based system operations, span a wide variety of speeds, cost, and processor types.



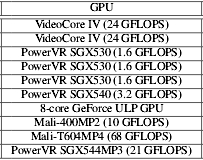


Table 1 :The ten 32-bit ARM boards comparison.

## 3. Raspberry Pi and BeagleBone



Figure 1 : Raspberry Pi 2 Model B

The Raspberry Pi is a small credit card sized computer that is able to run a range of Linux distributions. The board is developed by the [Raspberry Pi foundation](https://www.raspberrypi.org/about/) and then licensed through several manufacturing agreements.

It features GPIO pins, 4 USB ports, Ethernet, HDMI, audio out and a few other hardware slots you can connect devices to. We will go more into this in a bit.  
The Pi also requires an operating system in order to function. This is stored on an external micro SD card that needs to be connected to the Pi. The Pi also supports networking via a wireless USB device or over Ethernet.



Figure 2 : Beaglebone black

The Beaglebone black like the Raspberry Pi is roughly about the size of a credit card. The board is developed by [beagleboard.org foundation](http://beagleboard.org/about) a non-profit corporation based in the US.

This board features a large range of GPIO pins, 1 USB port and a HDMI port. We will go into more about the hardware in the section.

This board like the Pi requires an operating system in order to function. However this board is shipped with an operating system already pre-installed on its internal memory. Making it a little easier to setup over the Pi.

## 3.1 Hardware comparison



Table 2 : Raspberry Pi 2 and BeagleBone black Hardware comparison

## 4. Install OpenCV and Python on Raspberry Pi 2

Step 1 : start by updating and upgrading installed packages

$ sudo apt-get update

$ sudo apt-get upgrade

$ sudorpi-update

Step 2 : Install the required developer tools and packages

$ sudo apt-get install build-essential cmakepkg-config

Step 3 : Install the necessary image I/O packages

$ sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev libpng12-dev

Step 4 : Install the GTK development library

$ sudo apt-get install libgtk2.0-dev

Step 5 : Install the necessary video I/O packages

$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev

Step 6 : Install libraries that are used to optimize various operations within OpenCV

$ sudo apt-get install libatlas-base-dev gfortran

Step 7 : Install pip

$ wget https://bootstrap.pypa.io/get-pip.py

$ sudo python get-pip.py

Step 8 : Install  virtualenv  and virtualenvwrapper

$ sudo pip install virtualenvvirtualenvwrapper

$ sudorm -rf ~/.cache/pip

Then, update your ~/.profile

# virtualenv and virtualenvwrapper

export WORKON\_HOME=$HOME/.virtualenvs

source /usr/local/bin/virtualenvwrapper.sh

Reload your .profile  file

$ source ~/.profile

Create your computer vision virtual environment

$ mkvirtualenv cv

Step 9 : install the Python 2.7 development tools

$ sudo apt-get install python2.7-dev

$ pip install numpy

Step 10 : Download OpenCV and unpack it

$ wget -O opencv-2.4.10.zip http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.10/opencv-2.4.10.zip/download

$ unzip opencv-2.4.10.zip

$ cd opencv-2.4.10

Setup the build

$ mkdir build

$ cd build

$ cmake -D CMAKE\_BUILD\_TYPE=RELEASE -D CMAKE\_INSTALL\_PREFIX=/usr/local -D BUILD\_NEW\_PYTHON\_SUPPORT=ON -D INSTALL\_C\_EXAMPLES=ON -D INSTALL\_PYTHON\_EXAMPLES=ON  -D BUILD\_EXAMPLES=ON ..

Compile OpenCV

$ make

Step 11 :sym-link OpenCV into site-packages  directory

$ cd ~/.virtualenvs/cv/lib/python2.7/site-packages/

$ ln-s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so

$ ln -s /usr/local/lib/python2.7/site-packages/cv.py cv.py

Step 12 : test

$ workon cv

$ python

>>> import cv2

>>> cv2.\_\_version\_\_

'2.4.10'

# Image Processing using Raspberry Pi

## 1. Road sign detection using OpenCV ORB

The objective achieved in this part was to be able to write python code to detect road signs for roundabouts using Thedetect\_roundabouts method which obtains an image from live streaming video attached to our Raspberry Pi. It then uses our classifier file to detect any roundabout signs in said image.

Now, there are many OpenCV techniques we can use to inspect these candidate objects. Here we used [OpenCV ORB](http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_orb/py_orb.html), which is one of a number of [feature detection and description algorithms](http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html) that we can match images on.

Let us explain the following code we used:

Import numpy as py

import cv2

IMAGE\_SIZE =200.0

MATCH\_THRESHOLD =3

cap = cv2.VideoCapture(0)

cv2.namedWindow("Video")

convert\_rgb = True

while True:

status, img = cap.read()

roundabout\_cascade =cv2.CascadeClassifier('/home/pi/Desktop/haarcascade\_roundabout.xml')

gray =cv2.cvtColor(img,cv2.COLOR\_RGB2GRAY)

roundabouts =roundabout\_cascade.detectMultiScale(

     gray,

     scaleFactor=1.4,

     minNeighbors=3

     )

orb =cv2.ORB\_create()

bf =cv2.BFMatcher(cv2.NORM\_HAMMING,crossCheck=True)

roadsign =cv2.imread('/home/pi/Desktop/roundabout.jpg',0)

kp\_r,des\_r =orb.detectAndCompute(roadsign,None)

for(x,y,w,h) inroundabouts:

     obj =gray[y:y+h,x:x+w]

     ratio =IMAGE\_SIZE /obj.shape[1]

     obj =cv2.resize(obj,(int(IMAGE\_SIZE),int(obj.shape[0]\*ratio)))

     kp\_o, des\_o =orb.detectAndCompute(obj,None)

     iflen(kp\_o) ==0ordes\_o ==None: continue

     matches =bf.match(des\_r,des\_o)

     if(len(matches) >=MATCH\_THRESHOLD):

         cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

cv2.imshow("Video", img)

k = 0xFF &cv2.waitKey(1)

if k == 27:

break

elif k == ord("g"):

convert\_rgb = not convert\_rgb

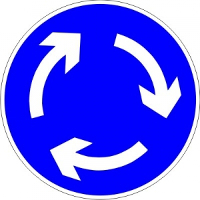
cap.set(cv2.CAP\_PROP\_CONVERT\_RGB, convert\_rgb)

First, we have to capture live stream with camera then Capture frame-by-frame to apply our processes on.

We load the roundabout cascade. we use the aforementioned OpenCV*detectMultiScale* function to detect all roundabout objects in the street snap.

Now we get to the OpenCV ORB stuff. we create an instance of ORB and a matcher.

Next, We load an image of a roundabout road sign (for matching each detected object with). I use ORB to obtain the keypoints and descriptors of the roundabout image. Here’s the roundabout image:



We are ready to loop through all the objects detected by our cascade…

WE use the object’s coordinates to obtain its image from the street photograph. we resize the object image, to bring it inline with our roundabout image.

We use ORB to obtain the keypoints and descriptors of the object. Now we can match the descriptors of our object with our roundabout image.

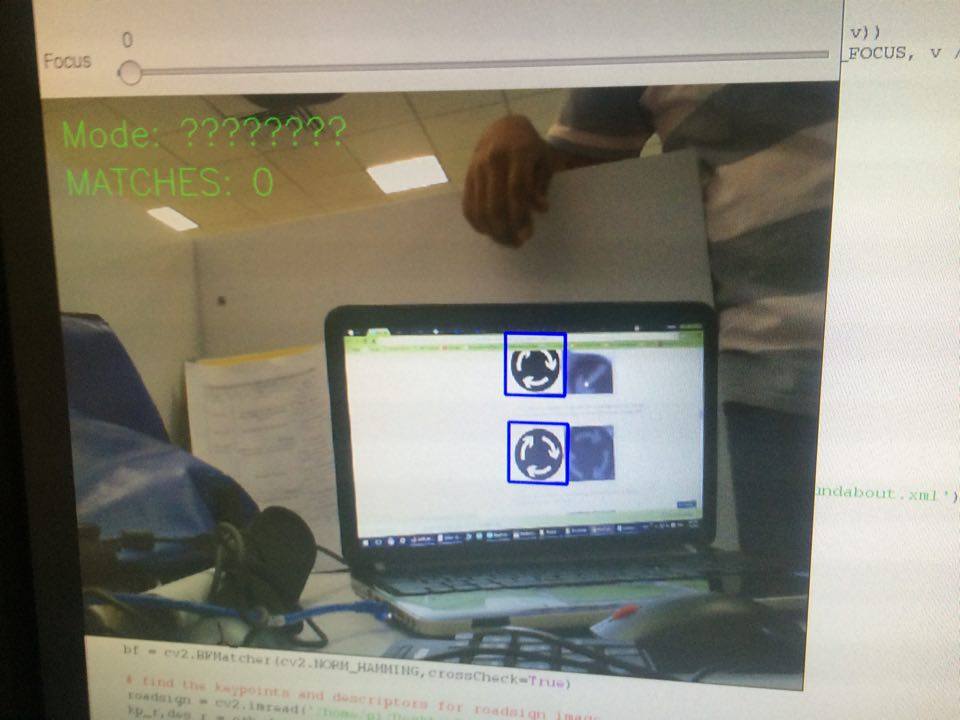
All that’s left is to draw a rectangle around our detected object, on the captured frame. But the object needs to pass the match threshold for this to happen!

So we see, OpenCV ORB has determined which of our candidate objects are actually roundabout signs. If fewer than 3 matches are found between our object and the roundabout image, the object is discarded.

## 1.1 Results







## 2. Raspberry Pi Red Colored objects detection.

In this part we made a program capable of detecting red objects usingcv.inrange and color boundries in dynamic images.

The following is analysis of the code used in this program line by line:

We’ll start by importing our necessary packages

#include <iostream>

#include <unistd.h>

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include "GPIOClass.h"

//#include "rpiPWM1.h"

#include "wiringPi.h"

#include "softPwm.h"

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

using namespace cv;

using namespace std;

boolrottate;

bool move;

int main( intargc, char\*\* argv )

{

VideoCapturecap(0); //capture the video from webcam

if ( !cap.isOpened() ) // if not success, exit program

{

cout<< "Cannot open the web cam" <<endl;

return -1;

}

namedWindow("Control", CV\_WINDOW\_AUTOSIZE); //create a window called "Control"

intiLowH = 170;

intiHighH = 179;

intiLowS = 150;

intiHighS = 255;

intiLowV = 60;

intiHighV = 255;

//Create trackbars in "Control" window

createTrackbar("LowH", "Control", &iLowH, 179); //Hue (0 - 179)

createTrackbar("HighH", "Control", &iHighH, 179);

createTrackbar("LowS", "Control", &iLowS, 255); //Saturation (0 - 255)

createTrackbar("HighS", "Control", &iHighS, 255);

createTrackbar("LowV", "Control", &iLowV, 255);//Value (0 - 255)

createTrackbar("HighV", "Control", &iHighV, 255);

intiLastX = -1;

intiLastY = -1;

//Capture a temporary image from the camera

Mat imgTmp;

cap.read(imgTmp);

//Create a black image with the size as the camera output

Mat imgLines = Mat::zeros(imgTmp.size(), CV\_8UC3 );;

//------------------------------------------GPIO------------------------------------------

GPIOClass\* Motor1PWM = new GPIOClass("2"); //create new GPIO object to be attached to PIN2

GPIOClass\* Motor3PWM = new GPIOClass("3"); //create new GPIO object to be attached to PIN3

Motor1PWM->export\_gpio(); //export PIN2

Motor3PWM->export\_gpio(); //export PIN3

Motor1PWM->setdir\_gpio("out");

Motor3PWM->setdir\_gpio("out");

// rpiPWM1 pwm(1000.0, 256, 50.0, rpiPWM1::MSMODE);

// softPwmCreate(18,50,100);

//--------------------------------------------------------------------------------------------

/\* while(1)

{

Mat imgOriginal;

cap.read(imgOriginal);

imshow("Original", imgOriginal); //show the original image

if (waitKey(30) == 27)

break;

}

\*/

while (true)

{

Mat imgOriginal;

boolbSuccess = cap.read(imgOriginal); // read a new frame from video

if (!bSuccess) //if not success, break loop

{

cout<< "Cannot read a frame from video stream" <<endl;

break;

}

Mat imgHSV;

cvtColor(imgOriginal, imgHSV, COLOR\_BGR2HSV); //Convert the captured frame from BGR to HSV

Mat imgThresholded;

inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV), imgThresholded); //Threshold the image

//morphological opening (removes small objects from the foreground)

erode(imgThresholded, imgThresholded, getStructuringElement(MORPH\_ELLIPSE, Size(5, 5)) );

dilate(imgThresholded, imgThresholded, getStructuringElement(MORPH\_ELLIPSE, Size(5, 5)) );

//morphological closing (removes small holes from the foreground)

dilate(imgThresholded, imgThresholded, getStructuringElement(MORPH\_ELLIPSE, Size(5, 5)) );

erode(imgThresholded, imgThresholded, getStructuringElement(MORPH\_ELLIPSE, Size(5, 5)) );

//Calculate the moments of the thresholded image

Moments oMoments = moments(imgThresholded);

double dM01 = oMoments.m01;

double dM10 = oMoments.m10;

doubledArea = oMoments.m00;

// if the area <= 10000, I consider that the there are no object in the image and it's because of the noise, the area is not zero

if (dArea> 10000)

{

//calculate the position of the ball

intposX = dM10 / dArea;

intposY = dM01 / dArea;

if (iLastX>= 0 &&iLastY>= 0 &&posX>= 0 &&posY>= 0)

{

//Draw a red line from the previous point to the current point

// line(imgLines, Point(posX, posY), Point(iLastX, iLastY), Scalar(0,0,255), 2);

}

iLastX = posX;

iLastY = posY;

}

imshow("Thresholded Image", imgThresholded); //show the thresholded image

imgOriginal = imgOriginal + imgLines;

cout<< "Area of red object : " <<dArea/1000 << " Position : [" <<iLastX<<","<<iLastY<<"]"<<endl;

intAreaMax = 3000;

intAreaMin = 50;

if(iLastX> 340 || iLastX< 300 && (dArea/1000) >AreaMin)

{

rottate = true; move = false;

}

else

{

rottate = false; move = true;

}

if(iLastX> 340 &&rottate&& (dArea/1000) >AreaMin)

{

Motor1PWM->setval\_gpio("1");

Motor3PWM->setval\_gpio("0");

cout<< "ROTATE RIGHT" <<endl;

}

if(iLastX< 300 &&rottate&& (dArea/1000) >AreaMin)

{

Motor1PWM->setval\_gpio("0");

Motor3PWM->setval\_gpio("1");

cout<< "ROTATE LEFT" <<endl;

}

if((dArea/1000) <AreaMax&& move && (dArea/1000) >AreaMin )

{

Motor1PWM->setval\_gpio("1");

Motor3PWM->setval\_gpio("1");

cout<< "MOVE FORWARD" <<endl;

}

if( (dArea/1000) >AreaMax || (dArea/1000) <AreaMin )

{

rottate = false; move = false;

}

if(!move && !rottate)

{

Motor1PWM->setval\_gpio("0");

Motor3PWM->setval\_gpio("0");

cout<< " !!!!!!STOOOOOOP!!!!!" <<endl;

//usleep(5000000);

}

cout<< " Move : " << move << " Rotate : " <<rottate<<endl;

// softPwmWrite(18,50);

// unsigned intdcyccount = 50;

// pwm.setDutyCycleCount(dcyccount);

imshow("Original", imgOriginal); //show the original image

if (waitKey(30) == 27) //wait for 'esc' key press for 30ms. If 'esc' key is pressed, break loop

{

cout<< "esc key is pressed by user" <<endl;

break;

}

}

return 0;

}

**Results**

**Code Simulation from Raspberry Pi**

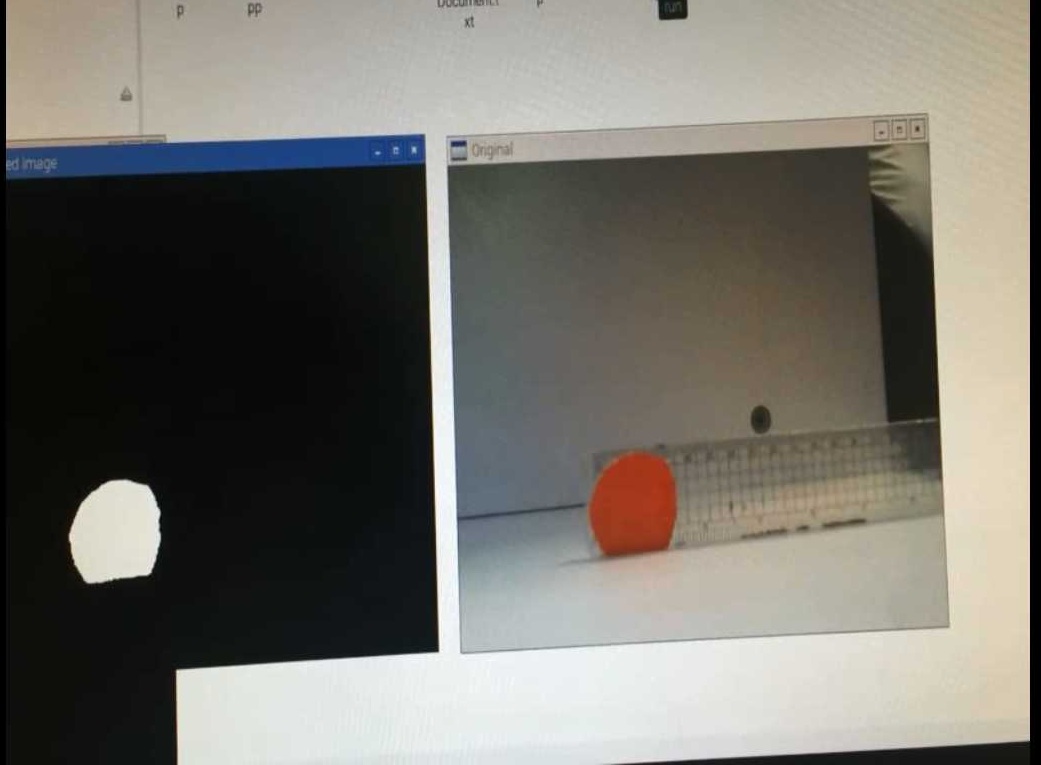
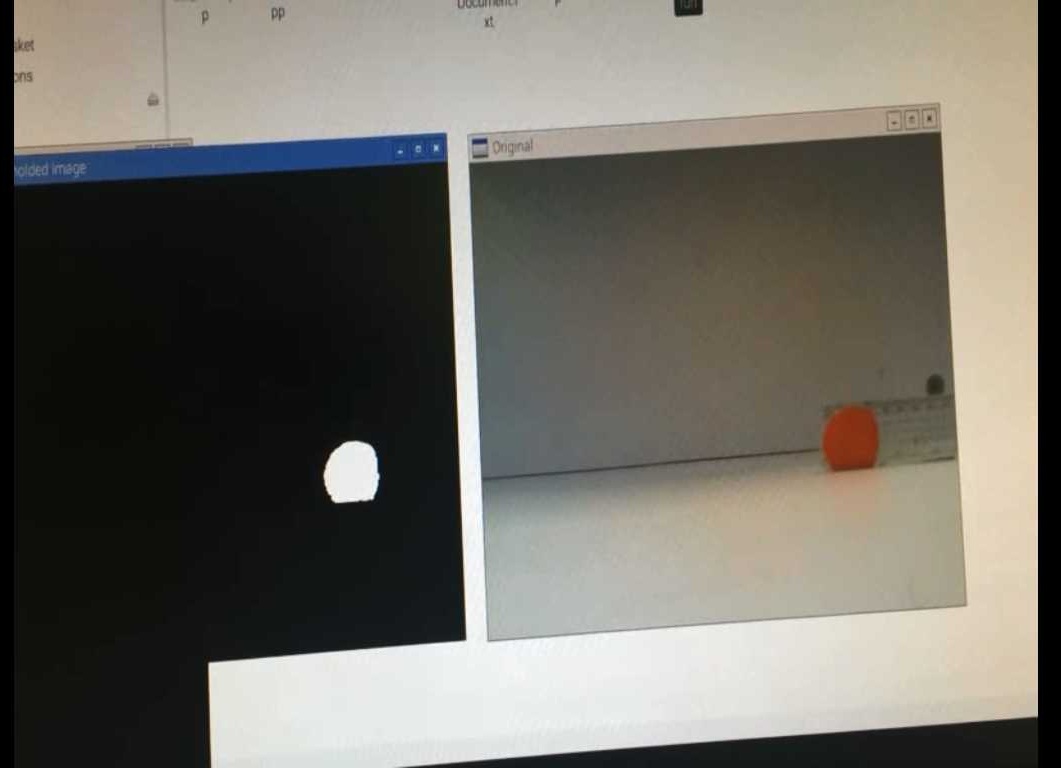
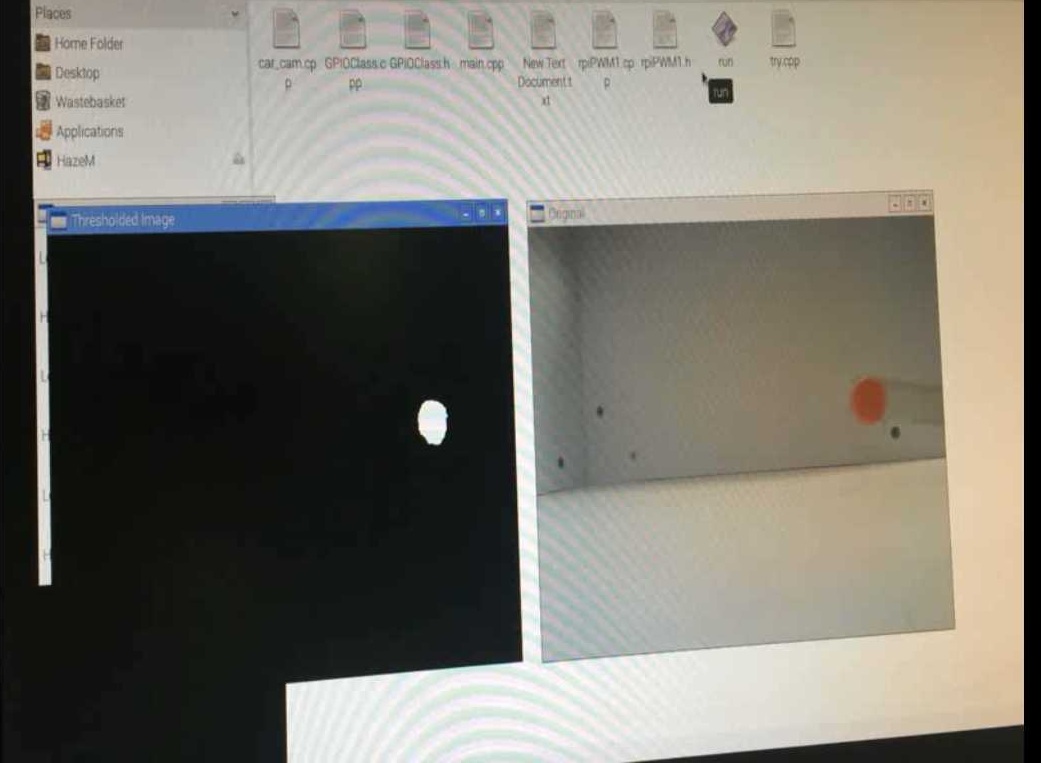


Figure 3: Those pictures show how the processing happen on Raspberry Pi code

**Real-time car model**



Figure 4: Those pictures show the car while approaching the red circle (Fire).

# References

[1] Beagle Bone Detailed Information Available at: http://beagleboard.org/, https://en.wikipedia.org/wiki/BeagleBoard.

[2] Raspberry Pi Detailed Information Available at: https://www.raspberrypi.org/, https://en.wikipedia.org/wiki/Raspberry\_Pi

[3] http://opencv.org/wp-content/