

HOME AUTOMATION USING IMAGE RECOGNITION

*Dissertation submitted in partial fulfillment of the requirement for the degree
of*

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

By –

BHAVIK NAHATA (131013)

ANKIT SHARMA (131014)

ARPIT SHARMA (131015)

UNDER THE GUIDANCE OF

Prof. Dr. Sunil Bhooshan
Head of Department, ECE



JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

MAY 2017

TABLE OF CONTENTS

S. No.	Contents	Page Number
1.	DECLARATION BY THE SCHOLARS	3
2.	SUPERVISOR'S CERTIFICATE	4
3.	LIST OF FIGURES	5
4.	ACKNOWLEDGMENT	6
5.	ABSTRACT	7
6.	CHAPTER 1 : INTRODUCTION	8
7.	CHAPTER 2 : COMPONENTS	10
	2.1 Project board: Raspberry Pi 3 Model B	10
	2.2 WEBCAM	16
	2.3 Relays	17
	2.4 Other Electric Devices	20
8.	CHAPTER 3 : IMPLEMENTATION	21
	3.1 Code	21
	3.2 Code Explanation	27
9.	CHAPTER 4 : RESULTS	38
10.	CONCLUSION	39
11.	REFERENCES	40

DECLARATION BY THE SCHOLARS

We hereby declare that the work reported in the this report entitled “**Home Automation Using Image Recognition**” submitted at **Jaypee University of Information Technology, Wagnaghat** is an authentic record of our work carried out under the supervision of **Prof. Dr. Sunil Bhooshan** . We have not submitted this work elsewhere for any other degree or diploma.

BHAVIK NAHATA

ANKIT SHARMA

ARPIT SHARMA

131013

131014

131015

Department of Electronics and Communication Engineering

Jaypee University of Information Technology, Wagnaghat , India

Date -

SUPERVISOR’S CERTIFICATE

This is to certify that the work reported in this report entitled “**Home Automation Using Image Recognition**”, submitted by **Bhavik Nahata, Ankit Sharma and Arpit Sharma** at **Jaypee University of Information Technology, Wagnaghat, India** is a bonafide record of their original work carried out under my supervision. This work has not been submitted elsewhere for any other degree or diploma.

(Signature of Supervisor)

Date -

LIST OF FIGURES

Figure Number	Caption	Page Number
2.1	Raspberry Pi and components	10
2.2	GPIO header	13
2.3	Physical Appearance of RPi 3	14
2.4	Webcam to click photos	16
2.5	Relay used as interface between devices and microcontroller	17
2.6	Relay circuit linking	18
2.7	Relay input and output circuit	19
2.8	Fan image to be used	20
2.9	Bulb image to be used	20
3.1	Executing the line detection code separately	30
3.2	Executing the circle detection code separately	34
3.3	Executing the camera code separately for clicking image	36
4.1	Detection of bulb from image taken	37
4.2	Detection of fan from image taken	38
4.3	LEDs switched ON when Bulb and Fan detected	38

ACKNOWLEDGMENT

We are highly indebted to Prof. Dr. Sunil Bhooshan for his guidance and constant supervision as well as for providing necessary information regarding the project & also for his support in completing the project.

We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in this project.

ABSTRACT

When we take images from the camera, in real time, it processes the image and gives the image to the microcontroller. Then the given image is further processed and recognized by the help of MATLAB or Open CV whose source code is embedded on the microcontroller.

When the image is recognized, instruction to turn on the matched output device(s) is given by the microcontroller. And with the help of relays, the device(s) are turned on.

The main steps are interfacing the camera with the microcontroller in real time and to interface the electrical devices with the microcontroller using relays. Also, processing and recognizing the given image using proper software and turning on the device with least delay (speed) is very crucial.

Software and Tools used:

- Microcontroller (RaspberryPi)
- Open CV / MATLAB
- Camera
- Electrical Instruments (fans, lights, etc.)
- Relays

CHAPTER 1

INTRODUCTION

Today, due to industrialization, we are seeing automation of many tasks in our day to day life and in the commercial and industrial areas as well. With the ever-rising use of machines we are also seeing a shift in our lifestyles, moving from a more traditional to a western way of life. This has resulted in households that have almost all members working a job while the elderly members, kids and injured stay at home. There is also a shift to nuclear families so, there might be times when no one is available to take care of an injured, physically challenged or an old member of the family.

This shift in lifestyle has many people depending on automated machines or devices for daily tasks and chores, like the prevalence of washing machine in every household compared to 10 years before, even more to automated washing machines that automatically start up when there's water supply available every day at the same time. Similarly, automated floor cleaners are abundant in the market now too which are very useful for people that don't have much time to clean up every day because of their hectic jobs and busy schedules. This rise of use and acceptance of their usefulness in our lives has enabled even greater penetration of these products in modern households.

Home automation is the recent trend in the field of embedded systems and plays a major role in designing various smart home hardware and software designs. Its very interesting to see how home automation concepts has made the life possible for humans. Here through our project we are proposing a home automation system which runs on image recognition.

Home computerization offers you access to oversee gadgets in your home from a cell phone wherever inside the world. The term is additionally utilized for separated programmable gadgets, similar to indoor regulators and mechanical gadget frameworks, however home robotization a considerable measure of precisely depicts homes amid which almost

everything - lights, machines, electrical retailers, warming and cooling frameworks are associated with a remotely manageable system. From a home security viewpoint, this moreover incorporates your notice gadget, and each one of the entryways, windows, locks, smoke finders, police work cameras and alternate sensors that are coupled to that.

Mechanization is one among the two fundamental qualities of home computerization. Robotization alludes to the ability to program and timetable occasions for the gadgets on the system. The programming could epitomize time-related charges, such as having your lights enact or off at particular circumstances consistently. It can even typify non-planned occasions, such as turning on every one of the lights in your home once your security framework alert is activated. When you start to know the probabilities of home mechanization programming, you'll returned up with any assortment of supportive and aesthetic answers for make your life higher Plug your mechanized blinds into a brilliant outlet and program it to close at afternoon consistently. Program your home mechanization framework to open the front passage for them, and bolt it up again when they are finished.

CHAPTER – 2

COMPONENTS USED

2.1 Project board: Raspberry Pi 3 Model B

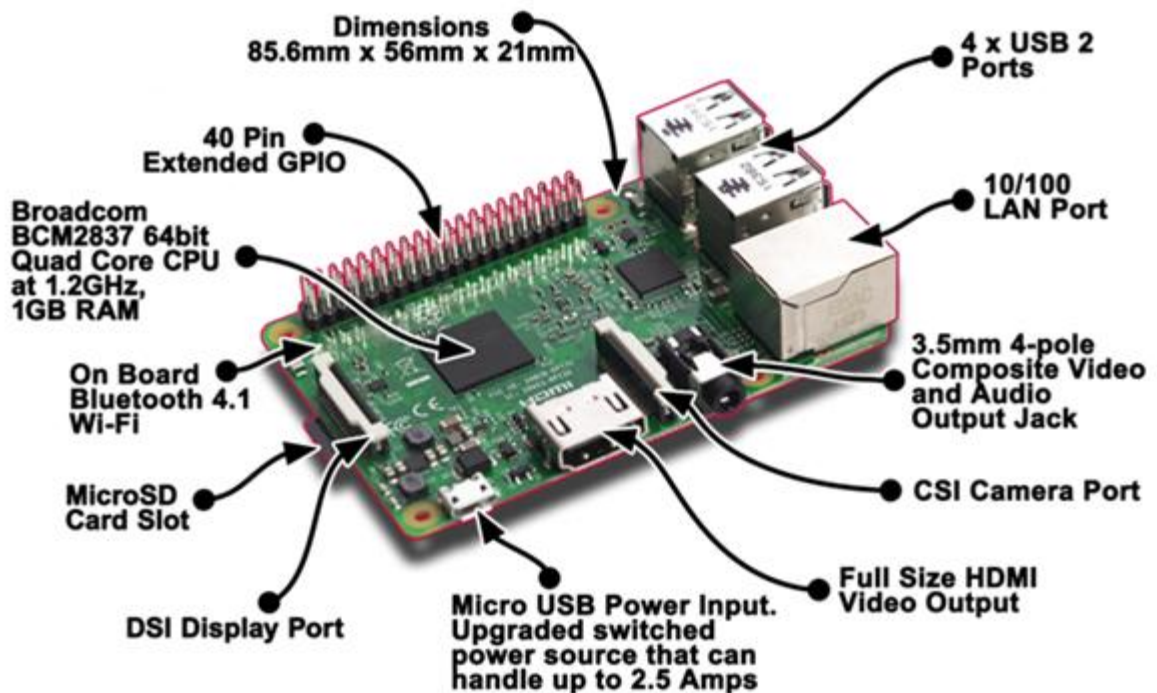


Fig 2.1 Raspberry Pi and components

2.1.1 Overview

The Raspberry Pi 3 Model B is the latest model in the Raspberry Pi family and also it's the most powerful Pi. This is because of the 64-bit quad-core ARMv8 processor. Its four ARM Cortex-A53 cores are clocked at 1.3GHz.

It's also added two of the most-requested features, an integrated 802.11n Wi-Fi, and Bluetooth 4.1 and Bluetooth Low Energy support. This means that users will no longer be

required to connect external modules to use these functions, freeing up USB ports for other devices.

In spite of these extra loaded features, the Raspberry Pi 3 is still a very good, power efficient microcontroller, running off a 5v micro USB connection with a maximum power draw of 2.5A

The Raspberry Pi 3 has been designed to be as close to previous generations as possible. This not only means that most cases and mountings will still fit the Pi 3, it also ensures that any projects designed for previous Model B versions will still work as normal, as all ports and connectors have been made intact.

The Raspberry Pi 3 is the third-generation Raspberry Pi. It replaced the Raspberry Pi 2 Model B in February 2016. Compared to the Raspberry Pi 2 it has:

- 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

Like the Raspberry Pi 2, it also has:

- 1GB RAM
- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot (now push-pull rather than push-push)
- VideoCore IV 3D graphics core

2.1.2 Specifications–

- **SoC:** Broadcom BCM2837
CPU: 4× ARM Cortex-A53, 1.2GHz
GPU: Broadcom VideoCore IV
RAM: 1GB LPDDR2 (900 MHz)
Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless
Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy
Storage: microSD
GPIO: 40-pin header, populated
Ports: HDMI, 3.5mm analogue audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

2.1.3 Configuration

- Like its previous versions, it also has four USB-2 ports, a full-sized HDMI port, a 10/100 Ethernet port, a combined 3.5mm audio/composite video out port and a MicroSD card slot. The Broadcom VideoCore IV graphics chipset from the Pi 2 also remains the same and the Pi 3 has 1GB of RAM.
- The Pi 3 also has the usual 40-pin General Purpose Input/output connector, which lets you use it to control a large variety of external devices and electronic components. There are also both camera and display connectors, which securely hook onto ribbon connectors from compatible devices.

The Raspberry Pi 3 supports a wide range of operating systems, including the recommend Raspbian OS. It also supports Windows 10 IOT, designed for those who want to create embedded devices running Windows, and a range of specialized variant operating systems, from development distros to media centers.

Raspberry Pi 3 GPIO Header

<i>Pin#</i>	<i>NAME</i>		<i>NAME</i>	<i>Pin#</i>
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)		DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)		(I ² C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Fig 2.2 GPIO header naming all the pins

As the Pi 3 hardware is still very new, not all of the operating systems popularly used with Pis are fully optimized for it yet. For example, the Ubuntu Mate team is still working on Bluetooth support, while the stable version of Windows 10 IoT dates from last November and is only designed for Pi 2 hardware, rather than the new Pi 3.



Fig. 2.3 Physical Appearance of Pi 3

Like its siblings, the Raspberry Pi 3 doesn't have a BIOS, but instead loads its system configuration information from `/boot/config.txt`, where you can not only set display properties, but also memory handling, enable camera support (assuming you have one connected to the Pi's camera port) and even overclock the CPU. Many of these features can also be configured from within Raspbian - but not other operating system distros - using the `rspi-conf` utility. These boot settings are also handy if you don't want to have to always turn on your display before the computer boots to ensure that it auto-detects the correct resolution.

2.1.4 Performance

The Raspberry Pi 3's processor upgrade instantly puts it in a different class to any previous models in the range when it comes to performance. It is around 25% faster than the Raspberry Pi 2 Model B. As ever, bear in mind that single-board computers of this kind are far slower than typical modern desktop PCs.

That's an impressive upgrade but just as importantly, the quad-core 1.2Ghz ARMv8 processor means that that Pi 3 provides a much smoother desktop experience, even if you have multiple browser windows open. It's still not ideal for heavy-duty multitasking, but we can open multiple browser tabs, edit documents and play media without any significant lag or slow-down.

2.1.5 PROS

- Low price.
- Improved performance over previous generation.
- Integrated Wi-Fi 802.11n and Bluetooth 4.1 saves hassle and USB ports.
- Upgraded processor means smoother desktop performance.

2.1.6 CONS

- Requires lots of additional hardware to function as a full PC.
- Limited operating system selection.
- Software setup may prove challenging.
- Slightly more power hungry than older Pis

2.1.7 Why did we use Raspberry pi3?

- Cheap
- It is a really powerful computer so it gives us room to include more features.

- Because of the integrated Wi-Fi, it can work on any home Wi-Fi network and can be accessed by any device within that home network
- The fast processor will let us run image processing algorithms smoothly without any slowdowns.

2.2 WEBCAM



Fig. 2.4 Webcam to click photos

2.3 RELAYS

A relay is associated magnetism switch operated by a comparatively tiny electrical phenomenon which will activate or off a far larger electrical phenomenon. the center of a relay is associated magnet (a coil of wire that becomes a brief magnet once electricity flows through it). you'll think about a relay as a sort of electrical lever: switch it on with a small current and it switches on ("leverages") another appliance employing a abundant larger current. Why is that useful? because the name suggests, several sensors area

unit improbably sensitive items of equipment and manufacture solely tiny electrical currents. however usually we'd like them to drive larger items of equipment that use larger currents. Relays bridge the gap, creating it potential for little currents to activate larger ones. meaning relays will work either as switches (turning things on and off) or as amplifiers (converting tiny currents into larger ones)

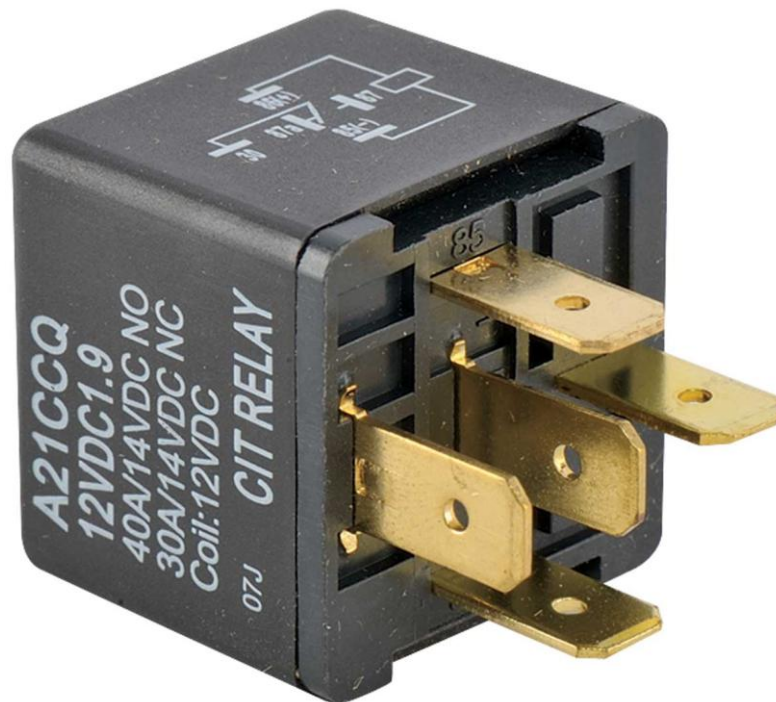


Fig. 2.5 Relay used as interface between devices and microcontroller

2.3.1 How Relays work

When power flows through the first circuit (1), it activates the electromagnet (brown), generating a magnetic field (blue) that attracts a contact (red) and activates the second circuit (2). When the power is switched off, a spring pulls the contact back up to its original position, switching the second circuit off again.

This is an example of a "normally open" (NO) relay: the contacts in the second circuit are not connected by default, and switch on only when a current flows through the magnet. Other relays are "normally closed" (NC; the contacts are connected so a current flows through them by default) and switch off only when the magnet is activated, pulling or pushing the contacts apart. Normally open relays are the most common.

Here's another animation showing how a relay links two circuits together. It's essentially the same thing drawn in a slightly different way. On the left side, there's an input circuit powered by a switch or a sensor of some kind. When this circuit is activated, it feeds current to an electromagnet that pulls a metal switch closed and activates the second, output circuit (on the right side). The relatively small current in the input circuit thus activates the larger current in the output circuit:

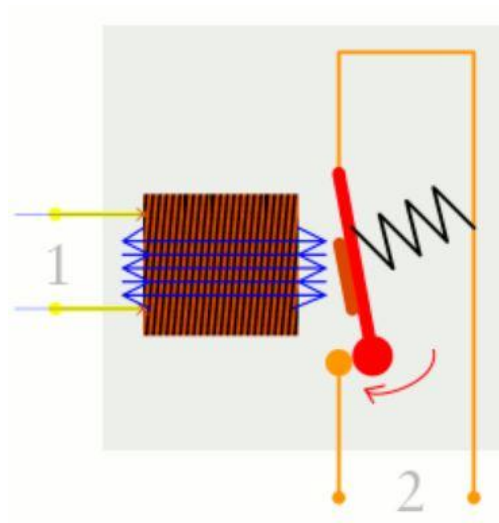


Fig. 2.6 Relay circuit linking

1. The input circuit (black loop) is switched off and no current flows through it until something (either a sensor or a switch closing) turns it on. The output circuit (blue loop) is also switched off.
2. When a small current flows in the input circuit, it activates the electromagnet (shown here as a red coil), which produces a magnetic field all around it.
3. The energized electromagnet pulls the metal bar in the output circuit toward it, closing the switch and allowing a much bigger current to flow through the output circuit.
4. The output circuit operates a high-current appliance such as a lamp or an electric motor.

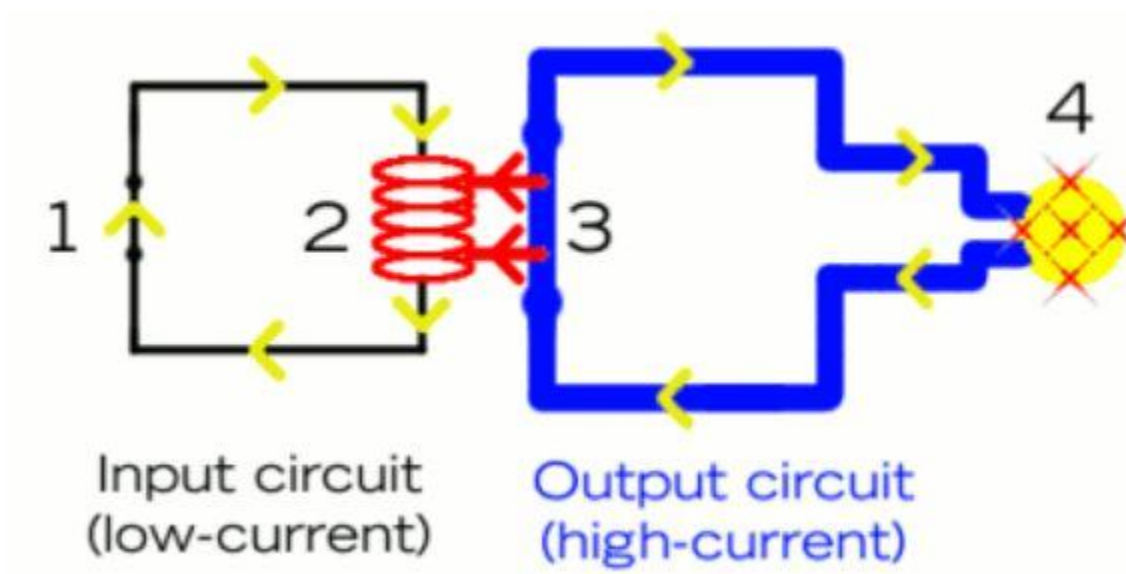


Fig. 2.7 Relay input and output circuit

2.4 OTHER ELECTRONIC DEVICES

- Fan

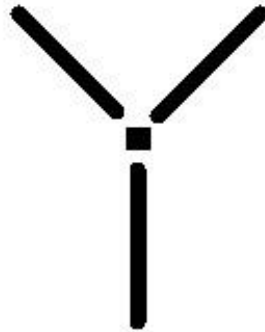


Fig. 2.8 Fan image to be used

- Bulb / Led

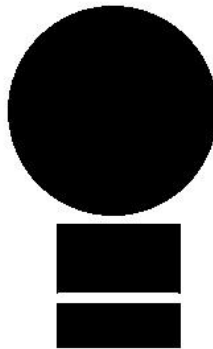


Fig. 2.9 Bulb image to be used

CHAPTER – 3

IMPLEMENTATION

3.1 Code Used

```
#!/usr/bin/python

import cv2
import cv2.cv as cv
import numpy as np
import sys
import math
import RPi.GPIO as GPIO
print __doc__

# Camera 0 is the integrated web cam on my netbook
camera_port = 0

#Number of frames to throw away while the camera adjusts
to light levels
ramp_frames = 30

# Now we can initialize the camera capture object with
the cv2.VideoCapture class.
# All it needs is the index to a camera port.
camera = cv2.VideoCapture(camera_port)
```

```
# Captures a single image from the camera and returns it
in PIL format
def get_image():
    # read is the easiest way to get a full image out of a
VideoCapture object.
    retval, im = camera.read()
    return im

# Ramp the camera - these frames will be discarded and
are only used to allow v4l2
# to adjust light levels, if necessary
for i in xrange(ramp_frames):
    temp = get_image()
print("Taking image...")
# Take the actual image we want to keep
camera_capture = get_image()
file = "/home/pi/opencv-3.0.0/samples/data/result.jpg"

# A nice feature of the imwrite method is that it will
automatically choose the
# correct format based on the file extension you
provide. Convenient!
cv2.imwrite(file, camera_capture)

#cv2.imshow("captured image",ile)
```

```

# You'll want to release the camera, otherwise you won't
be able to create a new
# capture object until your script exits
del(camera)

'''circle'''
print __doc__
try:
    fn = sys.argv[1]
except:
    fn = "/home/pi/opencv-3.0.0/samples/data/result.jpg"
src = cv2.imread(fn, 1)
img = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
img = cv2.medianBlur(img, 5)
cimg = src.copy() # numpy function
#circles = cv2.HoughCircles(gray_img, cv2.HOUGH_GRADIENT,
2, 10, np.array([]), 20, 60, m/10)[0]
circles = cv2.HoughCircles(img, cv.CV_HOUGH_GRADIENT, 1,
10, np.array([]), 110 , 30, 1, 100)
if circles is not None:
    a, b, c = circles.shape
    for i in range(b):
        cv2.circle(cimg, (circles[0][i][0],
circles[0][i][1]), circles[0][i][2], (0, 0, 255), 2)
        cv2.circle(cimg, (circles[0][i][0],
circles[0][i][1]), 2, (0, 255, 0), 3) # draw center of
circle

```

```

        x=1
        print("Bulb Detected  ")

GPIO.setmode(GPIO.BCM)

led=27

GPIO.setup(led, GPIO.OUT)
if x==1:
    GPIO.output(led,True)
    print("Light on")
    cv2.imshow("source", src)
    cv2.imshow("detected circles", cimg)
    cv2.waitKey(0)

try:
    fn = sys.argv[1]
except:
    fn = "/home/pi/opencv-3.0.0/samples/data/result.jpg"
print __doc__
src = cv2.imread(fn)
dst = cv2.Canny(src, 50, 200)
cdst = cv2.cvtColor(dst, cv2.COLOR_GRAY2BGR)

if True: # HoughLinesP
    lines = cv2.HoughLinesP(dst, 1, math.pi/180.0, 40,
np.array([], 50, 10)

```



```

a,b,c = lines.shape

for i in range(a):
    cv2.line(cdst, (lines[i][0][0], lines[i][0][1]),
(lines[i][0][2], lines[i][0][3]), (0, 0, 255), 3)
    i=i+1
else:    # HoughLines
    lines = cv2.HoughLines(dst, 1, math.pi/180.0, 50,
np.array([]), 0, 0)

a,b,c = lines.shape
for i in range(a):
    rho = lines[i][0][0]
    theta = lines[i][0][1]
    a = math.cos(theta)
    b = math.sin(theta)
    x0, y0 = a*rho, b*rho
    pt1 = ( int(x0+1000*(-b)), int(y0+1000*(a)) )
    pt2 = ( int(x0-1000*(-b)), int(y0-1000*(a)) )
    cv2.line(cdst, pt1, pt2, (0, 0, 255), 3,
cv2.LINE_AA)
    i=i+1

GPIO.setmode(GPIO.BCM)

led=23

GPIO.setup(led, GPIO.OUT)
if i==1:

```

```
GPIO.output(led,True)
print("Fan Detected")
cv2.imshow("source", src)
cv2.imshow("detected lines", cdst)
```

```
cv2.waitKey(0)
```

3.2 Code Explanation

Hough transform

The Hough redesign might be an element extraction method used in picture examination, portable PC vision, and computerized picture handle. the point of the system is to look out blemished cases of articles at interims a correct classification of shapes by an alternative method. This alternative method is circulated in an exceptionally parameter territory, from that protest competitors range unit acquired as local maxima in an extremely charged collector zone that is explicitly made by the control for figuring the Hough rebuild.

The traditional Hough rebuild was included with the ID of lines inside the picture, however later the Hough redesign has been stretched out to trademark places of optional shapes, most commonly circles or ovals. The Hough redesign in light of the fact that it is all around utilized nowadays was manufactured by Richard Duda and Peter Hart in 1972, UN organization alluded to as it a "summed up Hough change" when the associated 1962 patent of Paul Hough. The redesign was advanced inside the portable PC vision group by Dana H. Ballard through a 1981 diary article titled "Summing up the Hough redesign to watch optional shapes".

Theory

In robotized investigation of computerized pictures, a subproblem regularly emerges of recognizing basic shapes, for example, straight lines, circles or ovals. By and large an edge locator can be utilized as a pre-handling stage to get picture focuses or picture pixels that are on the coveted bend in the picture space. Because of flaws in either the picture information or the edge identifier, in any case, there might miss focuses or pixels on the coveted bends and in addition spatial deviations between the perfect line/circle/oval and the uproarious edge focuses as they are acquired from the edge indicator. Consequently, it is regularly non-minor to bunch the removed edge components to a fitting arrangement of lines, circles or ovals. The reason for the Hough change is to address this issue by making it conceivable to perform groupings of edge focuses into protest competitors by playing out an express voting strategy over an arrangement of parameterized picture objects (Shapiro and Stockman, 304).

The least complex instance of Hough change is recognizing straight lines. When all is said in done, the straight line $y = mx + b$ can be spoken to as a point (b, m) in the parameter house. Be that as it may, vertical lines make a drag. they'd make to boundless estimations of the slant parameter m . In this way, for system reasons, Duda and Hart arranged the use of the creator customary kind where r is that the separation from the birthplace to the highest

reason on hold, and θ (theta) is that the point between the x hub and furthermore the line associating the root therewith highest reason.

Implementation

The direct Hough revise algorithmic program utilizes a two-dimensional cluster, alluded to as Associate in Nursing aggregator, to locate the presence of a line spoken to by $r = x \cos \theta + y \sin \theta$. The measurement of the aggregator rises to the measure of obscure parameters, i.e., two, considering amount estimations of r and θ inside thetry (r, θ) . for each constituent at (x, y) and its neighborhood, the Hough revise algorithmic program decides whether there's sufficient confirmation of a line at that constituent. Provided that this is true, it'll figure the parameters (r, θ) of that line, so chase for the aggregator's canister that the parameters constitute, and increase the value of that receptacle. By finding the canisters with the absolute best values, more often than not by desiring for local maxima inside the collector house, the premier conceivable lines will be extricated, and their (estimated) geometric definitions check off. (Shapiro and sodbuster, 304) the best way of finding these pinnacles is by applying some sort of edge, however elective procedures may yield higher winds up in very surprising conditions – determinative that lines square measure discovered correspondingly as what number. Since the lines came don't contain any length data, it's commonly essential, inside the following stride, to search out that segments of the picture coordinate with that lines. Also, owing to state mistakes inside the edge location step, there'll now and then be blunders inside the aggregator house, which can fabricate it non-unimportant to search out the appropriate pinnacles, and in this manner the reasonable lines.

The last aftereffects of the direct Hough improve could be a two-dimensional cluster (grid) like the collector—one measurement of this lattice is that the amount point θ and subsequently the option measurement is that the amount remove r . all aspects of the lattice joins a value satisfactory the aggregate of the focuses or pixels that square measure situated out and about depict by amount parameters (r, θ) . that the part with the absolute best cost demonstrates the line that is most depict inside the info picture.

3.2.1 Algorithm to detect lines

Corner or edge detection: The resultant binary/grey image can have 0s indicating non-edges and 1s or on top of indicating edges. this can be our input image. Rho vary and alphabetic character vary creation. ρ ranges from $-\text{max_dist}$ to max_dist wherever max_dist is that the diagonal length of the input image. θ ranges from -90° to 90° you'll be able to have additional or less bins within the ranges to trade-off accuracy, area and speed. E.g. each third angle in -90° to 90° to cut back from a hundred and eighty to sixty values.

Hough accumulator of θ vs ρ . it's a 2nd array with variety|the amount|the quantity} of rows up to the amount of ρ values and therefore the number of columns up to the amount of θ values. Voting within the accumulator. for every edge purpose and for every θ price, realize the closest ρ price and increment that index within the accumulator. every component tells what number points/pixels contributed “votes” for potential line candidates with parameters (ρ, θ) .

Peak finding. native maxima within the accumulator indicates the parameters of the foremost outstanding lines within the input image. Peaks are often found most simply by applying a threshold or a relative threshold (values up to or bigger than some fastened proportion of the world most value).

minLineLength - Minimum length of line. Line segments shorter than this are rejected.

maxLineGap - Maximum allowed gap between line segments to treat them as single line.

void HoughLines(InputArray image, OutputArray lines, double rho, double letter of the alphabet, int threshold, double srn=0, double stn=0)

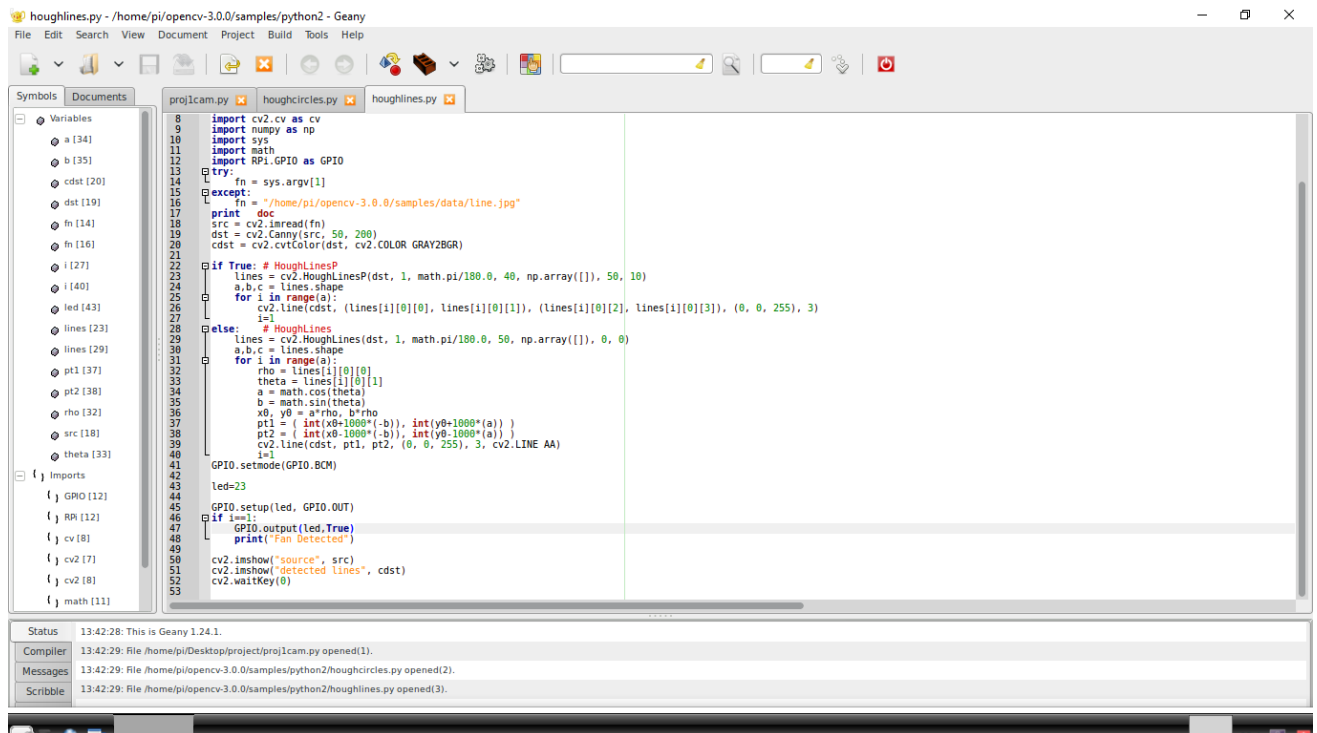
Parameters:

- picture – 8-bit, mono twofold supply picture (utilize edge finders)
- lines – Output vector of lines. each line is diagrammatic by a two-component vector . is that the separation from the facilitate inception (upper left corner of the picture). is that the line revolution edge in radians ().
- rho – Distance determination of the aggregator in pixels.
- theta – Angle determination of the aggregator in radians.
- limit – Accumulator edge parameter. exclusively those lines square measure returned that get enough votes ($>$ threshold).
- srn – For the multi-scale Hough revamp, it's a divisor for the space determination letter . The coarse collector separate determination is letter and furthermore the right aggregator determination is ρ/srn . In the event that each $\text{srn}=0$ and $\text{stn}=0$, the traditional Hough improve is utilized. Something else, each these parameters should be certain.
- stn – For the multi-scale Hough modify, it's a divisor for the space resolutiontheta.

A decent case for Hough Line adjust is given in OpenCV Documentation.

Steps:

1. Stack picture and change over to dim scale.
2. Apply the Hough adjust to search out the lines.(HoughLines)
3. Draw the recognized lines.(line)
4. Demonstrate the outcome



```
8 import cv2,cv as cv
9 import numpy as np
10 import sys
11 import math
12 import RPi.GPIO as GPIO
13
14 try:
15     fn = sys.argv[1]
16 except:
17     fn = "/home/pi/opencv-3.0.0/samples/data/Line.jpg"
18 print doc
19 src = cv2.imread(fn)
20 dst = cv2.canny(src, 50, 200)
21 cdst = cv2.cvtColor(dst, cv2.COLOR_GRAY2BGR)
22
23 if True: # HoughLinesP
24     lines = cv2.HoughLinesP(dst, 1, math.pi/180.0, 40, np.array([],), 50, 10)
25     a,b,c = lines.shape
26     for i in range(a):
27         cv2.line(cdst, (lines[i][0][0], lines[i][0][1]), (lines[i][0][2], lines[i][0][3]), (0, 0, 255), 3)
28         i+=1
29 else: # HoughLines
30     lines = cv2.HoughLines(dst, 1, math.pi/180.0, 50, np.array([],), 0, 0)
31     a,b,c = lines.shape
32     for i in range(a):
33         rho = lines[i][0][0]
34         theta = lines[i][0][1]
35         a = math.cos(theta)
36         b = math.sin(theta)
37         x0, y0 = a*rho, b*rho
38         pt1 = ( int(x0-1000*(-b)), int(y0-1000*(a)) )
39         pt2 = ( int(x0+1000*(-b)), int(y0+1000*(a)) )
40         cv2.line(cdst, pt1, pt2, (0, 0, 255), 3, cv2.LINE_AA)
41         i+=1
42
43 GPIO.setmode(GPIO.BCM)
44 led=23
45 GPIO.setup(led, GPIO.OUT)
46 if i==1:
47     GPIO.output(led,True)
48     print("Fan Detected")
49
50 cv2.imshow("source", src)
51 cv2.imshow("detected Lines", cdst)
52 cv2.waitKey(0)
53
```

Status 13:42:28: This is Geany 1.24.1.
Compiler 13:42:29: File /home/pi/Desktop/project/proj1cam.py opened(1).
Messages 13:42:29: File /home/pi/opencv-3.0.0/samples/python2/houghcircles.py opened(2).
Scrubble 13:42:29: File /home/pi/opencv-3.0.0/samples/python2/houghlines.py opened(3).

Fig. 3.1 Executing the line detection code separately

3.2.2 Algorithm to detect circles

`cv2.HoughCircles(image, technique, dp, minDist)`

- image: 8-bit, single channel picture. On the off chance that working with a shading picture, change over to grayscale introductory.

- method: Defines the methodology to watch hovers in pictures. At present, the sole upheld strategy is `cv2.HOUGH_GRADIENT`, that relates to the Yuen et al. paper.

- dp: This parameter is that the opposite greatness connection of the collector determination to the picture determination (see Yuen et al. for extra points of interest). fundamentally, the bigger the displaced person gets, the littler the gatherer exhibit gets.

- minDist: Minimum separation between the center (x, y) directions of identified circles. On the off chance that the `minDist` is quite recently too little, numerous circles inside a similar neighborhood in light of the fact that the first is additionally (dishonestly) recognized. In the event that the `minDist` is quite recently excessively monster, then a few circles won't not be identified in the scarcest degree.

- param1: Gradient worth usual handle edge recognition inside the Yuen et al. technique.

- param2: Accumulator edge worth for the `cv2.HOUGH_GRADIENT` method. The littler the edge is, the extra circles will be identified (counting false circles). The bigger the edge is, the extra circles can most likely be came.

- minRadius: Minimum size of the range (in pixels).

- maxRadius: most size of the span (in pixels).

If this technique appears difficult, don't worry. It's really not unfortunate.

Be able to manipulate with the parameter values from image to image. The `minDist`

parameter is particularly vital to urge right. while not associate degree optimum minDist worth, you will find yourself missing out on some circles, otherwise you might detection several false circles.

We'll utilize NumPy for numerical process, argparse for parsing command arguments, and cv2 for our OpenCV bindings.

Then, on Lines 7-9 we have a tendency to analyze our command arguments.

We'll want solely one switch, --image, that is that the path to the image we would like to notice circles in.

Let's go ahead and load the image:

We stack our picture off plate on Line 12 and make a duplicate of it on Line 13 so we can draw our distinguished circles without annihilating the first picture.

As we'll see, the cv2.HoughCircles work requires a 8-bit, single channel picture, so we'll simply ahead and change over from the RGB shading space to grayscale on Line 14.

Approve, time to identify the circles:

Recognizing the circles is dealt with by the cv2.HoughCircles work on Line 17. We go in the picture we need to recognize hovers as the main contention, the circle identification technique as the second contention (as of now, the cv2.cv.HOUGH_GRADIENT strategy is the main circle location strategy bolstered by OpenCV and will probably be the main strategy for quite a while), an aggregator estimation of 1.5 as the third contention, lastly a minDist of 100 pixels.

A check is made on Line 20 to guarantee no less than one circle was found in the picture. Line 22 then handles changing over our circles from skimming point (x, y) directions to whole numbers, enabling us to draw them on our yield picture. From that point, we begin circling once again the middle (x, y) organizes and the sweep of the hover on Line 25.

We draw the real distinguished hover on Line 28 utilizing the cv2.circle work, trailed by drawing a rectangle at the focal point of the hover on Line 29.

At long last, Lines 32 and 33 show our yield picture. So there you have it — recognizing hovers in pictures utilizing OpenCV. Be that as it may, how about we simply ahead and investigate a few outcomes. Start up a shell, and exeIn this blog entry I demonstrated to you generally accepted methods to utilize the `cv2.HoughCircles` work in OpenCV to distinguish hovers in pictures.

Not at all like identifying squares or rectangles in pictures, distinguishing circles is significantly harder since we can't answer on approximating the quantity of focuses in a shape.

To help us distinguish hovers in pictures, OpenCV has provided the `cv2.HoughCircles` work.

While the `cv2.HoughCircles` technique may appear to be muddled at to start with, I would contend that the most critical parameter to play with is the `minDist`, or the base separation between the inside (x, y) directions of recognized circles.

In the event that you set `minDist` too little, you'll end up with a few inaccurately recognized circles. On the inverse hand, if `minDist` is recently excessively goliath, then you'll end up missing a few circles. Setting this parameter without a doubt takes some fine institutionalization

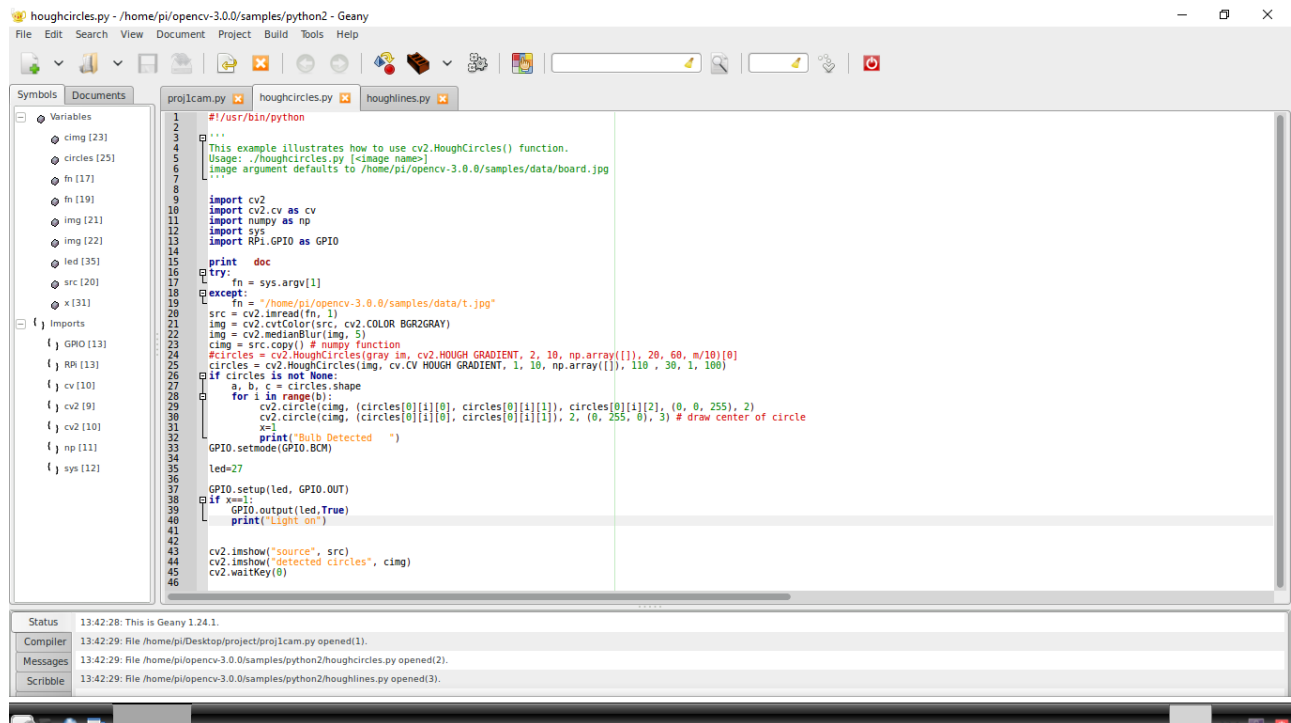


Fig. 3.2 Executing the circle detection code separately

3.2.3 Accessing a image in Raspberry Pi using Python and OpenCV

```
1 # import the necessary packages
2 from picamera.array import PiRGBArray
3 from picamera import PiCamera
4 import time
5 import cv2
6
7 # initialize the camera and grab a reference to the raw camera capture
8 camera = PiCamera()
9 rawCapture = PiRGBArray(camera)
10
11 # allow the camera to warmup
12 time.sleep(0.1)
13
14 # grab an image from the camera
15 camera.capture(rawCapture, format="bgr")
16 image = rawCapture.array
17
18 # display the image on screen and wait for a keypress
19 cv2.imshow("Image", image)
20 cv2.waitKey(0)
```

We'll begin by importation our necessary packages on Lines 2-5. From there, we tend to initialize our PiCamera object on Line eight and grab a relation to the raw capture part on Line nine. This rawCapture object is particularly helpful since it provides North American nation direct access to the camera stream and avoids the high-priced compression to JPEG format, that we'd then ought to take and decipher to OpenCV format anyway. I extremely advocate that you just use PiRGBArray whenever you wish to access the Raspberry Pi camera — the performance gains area unit well worthwhile. From there, we tend to sleep for a tenth of a second on Line twelve — this enables the camera detector to heat up.

Finally, we tend to grab the particular ikon from the rawCapture object on Line fifteen wherever we tend to take special care to make sure our image is in BGR format instead of RGB. OpenCV represents pictures as NumPy arrays in BGR order instead of RGB — this tiny nuisance is delicate, however important to recollect because

it will result in some confusing bugs in your code down the road. Finally, we tend to show our image to screen on Lines nineteen and twenty.

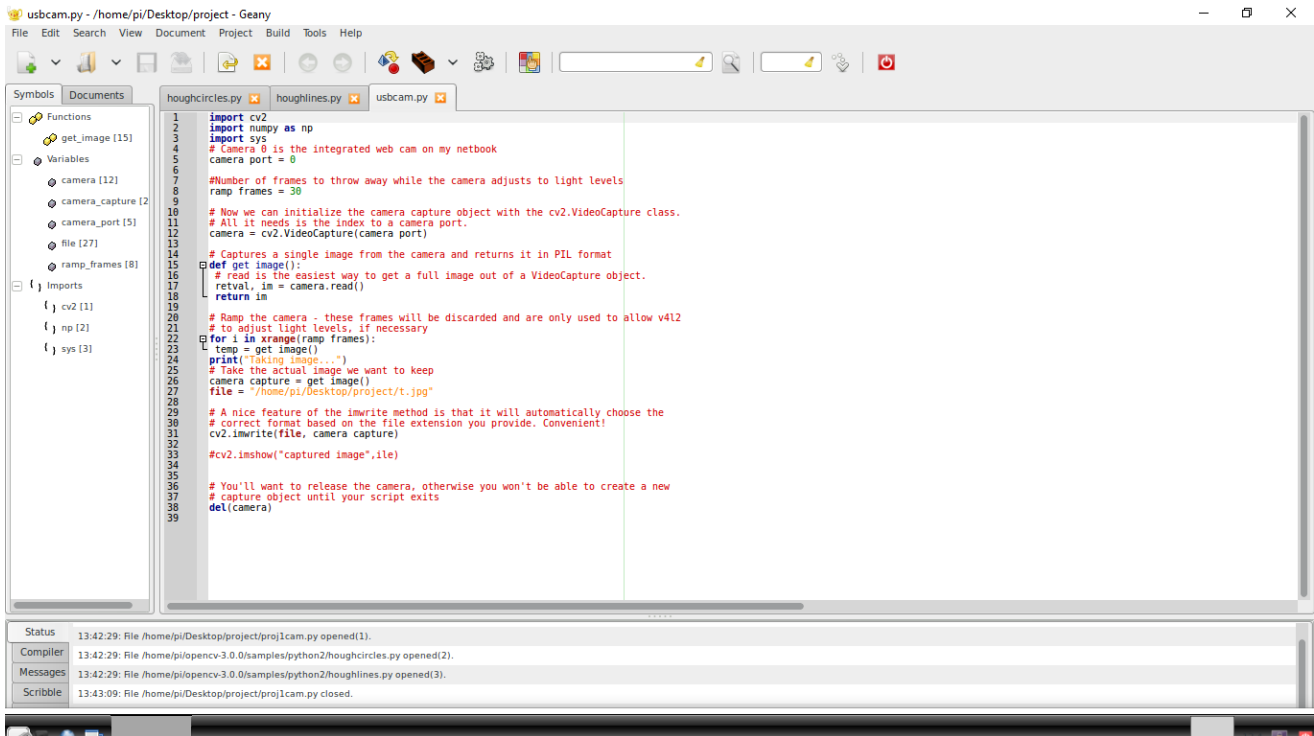


Fig. 3.3 Executing the camera code separately for clicking image

CHAPTER 4

RESULTS

Here are the screenshots of the results of the project which we have performed.

There is a terminal on which the commands are being run to execute the code. This is the LX Terminal of Raspberry Pi. Also there are two images from the result obtained on the execution of the code. The image on the left is the image which is taken by the camera and image on right is when the bulb is detected from the taken image

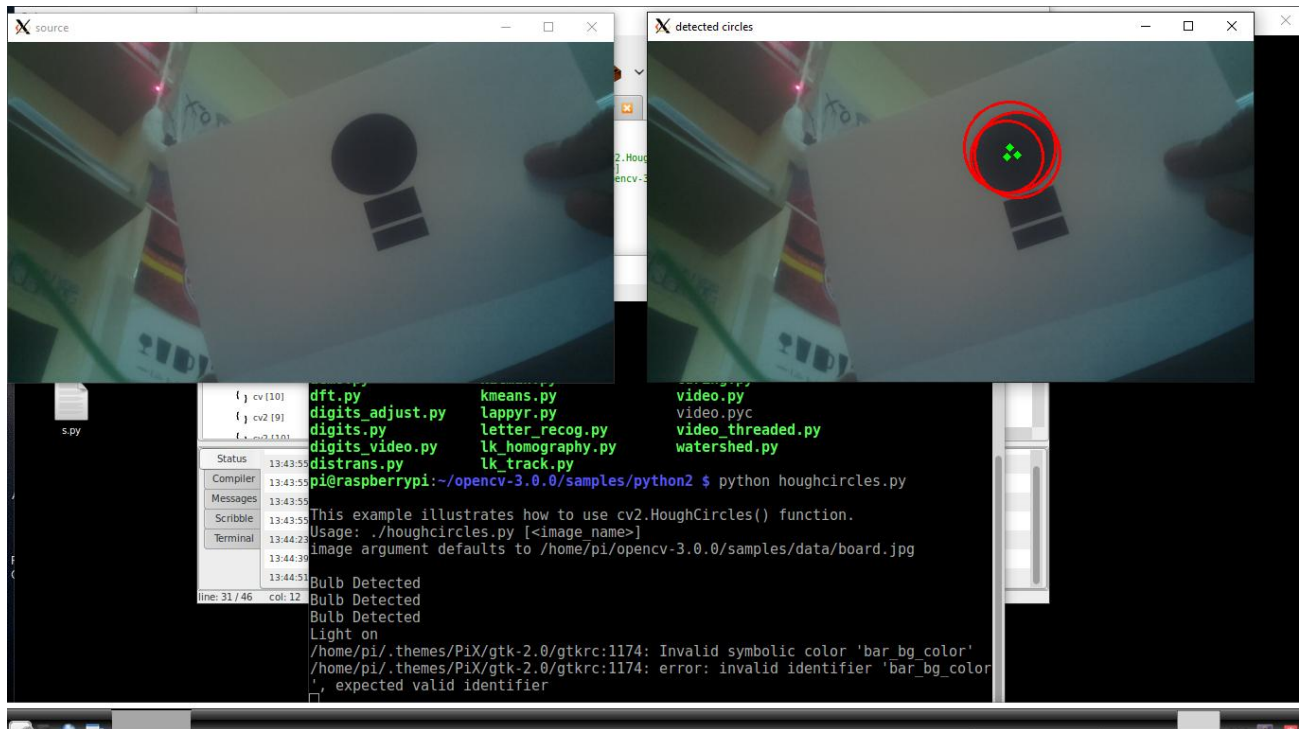


Fig. 4.1 Detection of bulb from image taken

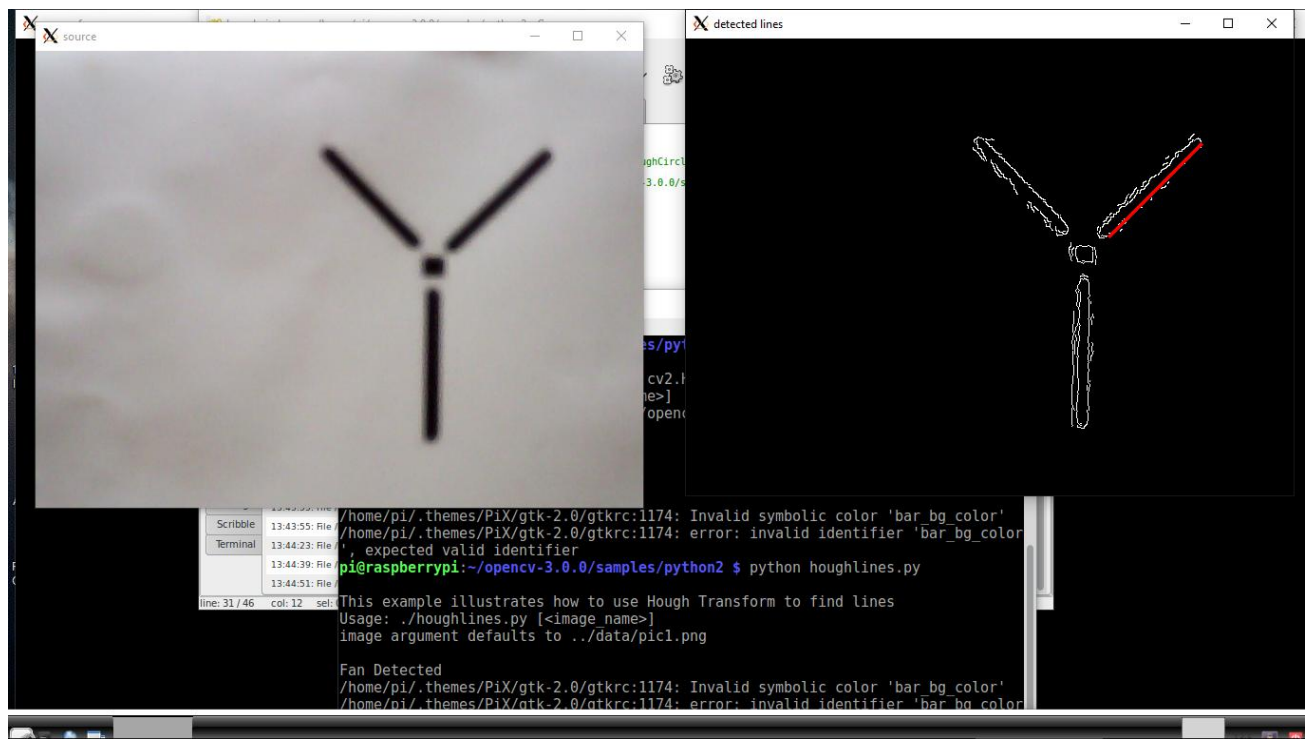


Fig. 4.2 Detection of fan from image taken



Fig. 4.3 LEDs switched ON when Bulb and Fan detected

CONCLUSION

This project presents the overall design of Home Automation System (HAS) with low cost and wireless system. This system is designed to assist and provide support in order to fulfill the needs of elderly and disabled in home. Also, the smart home concept in the system improves the standard of living at home.

This project is based on the Raspberry pi and Python. These platforms are Free Open Source Software. So the overall implementation cost is low and can be easily configured.

The Raspberry pi acts as a server, analyses the data and activates the GPIO (General Purpose Input Output) Pins. The GPIO Pins are connected to the relays switch which activated the required home appliances.

In this way, automation process is carried out. This is a simple prototype. Using this as a reference further it can be expanded to many other programs such as face recognition or safety measures.

REFERENCES

- M. Gamba, A. Gonella, C.E. Palazzi, "Design issues and solutions in a modern home automation system", *Proceedings of International Conference on Computing Networking and Communications (ICNC 2015)*, pp. 1111-1115, Feb. 2015.
- Charles Severance, "Eben Upton: Raspberry Pi", vol. 46, no. 10, pp. 14-16, 2013.
- V. Patchava, H. B. Kandala, P R. Babu, "A Smart Home Automation technique with Raspberry Pi using IoT", *Smart Sensors and Systems (IC-SSS 2015)*, Dec. 2015.
- Wen-Chiang Huang, Chwan-Hwa Wu, "Adaptive color image processing and recognition for varying backgrounds and illumination conditions" *IEEE Transactions on Industrial Electronics* (Volume: 45, Issue: 2, Apr 1998) pp. 351 – 357, 1998.
- Rafael C. Gonzalez, Richard E. Woods, *Digital Image Processing 3 Edition*, Pearson Education, 2016.
- <http://opencv.org/documentation.html>
- <https://in.mathworks.com>