EXPERIMENT NUMBER : 1

EXPERIMENT NAME : GENERAL PURPOSE INPUT & OUTPUT PROGRAMMING USING RASPBERRYPI WITH SENSEHAT

DATE : 26/09/2022, MONDAY

8/10

* AIM :

To introduce Python-based basic programs using Raspberry Pi and GPIO and associated peripheral interfacing.

* INTEGRATED DEVELOPMENT ENVIRONMENT (IDE) :

Name - Thonny 4.0.1

Publisher - Aivar Annamaa

Support link - https://thonny.org

* IMPORT NECESSARY LIBRARIES :

```
from sense_hat import SenseHat # Python module to control the Raspberry Pi Sense HAT
sense = SenseHat()
sense.clear() # No arguments defaults to OFF

from time import sleep # Handle time-related tasks
```

(a) Hello World Display on SenseHAT :

→ Algorithm -

① Initialize the required colours for text and background.
② Controll the scroll speed ; Greater speed implies slower scroll time.
③ Wait for some seconds.
④ Clear everything.

→ Python Code -

```
yellow = (255, 255, 0)
blue = (0, 0, 255)
```

(b) Single character Displayed followed by Rotation and Flip:

→ Algorithm -

① Display a single text character on the LED matrix.
② Choose a suitable text and background colour
③ Perform horizontal and vertical flip.
④ Rotate the character by some degree of your choice.

→ Python Code -

```
sense.show_letter("Z",
                  text_colour = yellow, # Text Colour
                  back_colour = blue) # Background Colour

sense.flip_h() # Horizontal Flip
sense.flip_v() # Vertical Flip
sense.set_rotation(180) # Rotate character by 180 degrees
```

(c) Single Pixel activation in Scratch:

→ Algorithm -

① Choose Column Number.
② Choose Row Number
③ Choose suitable colour.

→ Python Code -

```
sense.set_pixel(1,2, blue) # First Column, Second Row, Blue Colour
```

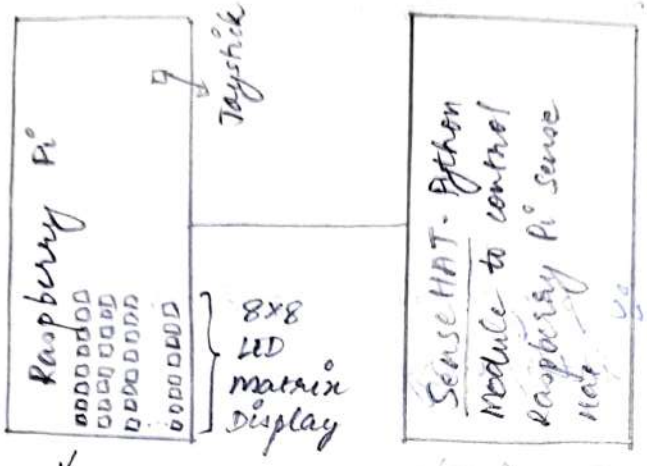
(d) Custom character Creation using LED Matrix: (Print the letter 'S')

→ Algorithm -

① Choose Column Number.
② Choose Row Number.
③ Choose suitable colour.

→ Python Code -

```
sense.set_pixel(0,0,blue); sense.set_pixel(1,0, blue); sense.set_pixel
(2,0, blue); sense.set_pixel(3,0,blue); sense.set_pixel(4,0,blue);
sense.set_pixel(5,0, blue);
sense.set_pixel(6,0, blue); sense.set_pixel(7,0, blue)
```



↓
contains other sensors
as well such as
gyroscope, humidity, temperature,
pressure, magnetometer, accelerometer, etc

sense.set_pixel(0,1,blue); sense.set_pixel(0,2,blue)
sense.set_pixel(0,3,blue); sense.set_pixel(1,3,blue); sense.set_pixel(2,3,blue); sense.set_pixel(3,3,blue); sense.set_pixel(4,3,blue); sense.set_pixel(5,3,blue); sense.set_pixel(6,3,blue); sense.set_pixel(7,3,blue)
sense.set_pixel(0,4,blue); sense.set_pixel(1,4,blue); sense.set_pixel(2,4,blue); sense.set_pixel(3,4,blue); sense.set_pixel(4,4,blue); sense.set_pixel(5,4,blue); sense.set_pixel(6,4,blue); sense.set_pixel(7,4,blue)
sense.set_pixel(7,5,blue); sense.set_pixel(7,6,blue)
sense.set_pixel(0,7,blue); sense.set_pixel(1,7,blue); sense.set_pixel(2,7,blue); sense.set_pixel(3,7,blue); sense.set_pixel(4,7,blue); sense.set_pixel(5,7,blue); sense.set_pixel(6,7,blue); sense.set_pixel(7,7,blue)

(c) Position and Action Display of Joystick:

→ Algorithm -

① Return a list of InputEvent tuples representing all events that have occurred since the last call to get_events or wait_for_event.
② Direction - The direction the joystick was moved, as a string. ("up", "down", "left", "right", "middle")
③ Action - The action that occurred, as a string ("pressed", "released", "held")
④ Print the event direction and action.

→ Python Code -

```
while True:
    for event in sense.stick.get_events():
        print(event.direction, event.action)
```

Q Character Display based on Position and Action of Joystick:

→ Algorithm -

① Return a list of InputEvent tuples representing all events that have occurred since the last call to get_events or wait_for_event.

(2) If the direction in which the joystick was moved is "middle" and the action that occurred is "held", then -
(3) Display a single text character on the LED matrix.
(4) Choose suitable text and background colour.

→ Python Code -

```
while True:
    for event in sense.stick.get_events():
        if (event.direction == "middle" and event.action == "held"):
            sense.show_letter ("z",
                               text_colour = yellow, # Text colour
                               back_colour = blue) # Background colour
```

(g) SenseHAT Led Matrix control based on Position and Action of Joystick using Function:

→ Algorithm -

(1) Define functions that sets the entire LED matrix to red, green, blue and yellow colours.
(2) Tell the program which function to associate with which direction.
(3) Return a list of InputEvent tuples representing all events that have occurred since the last call to get_events or wait_for_events.
(4) If the direction in which the joystick was moved is "middle" and the action that occurred is "held", then -
(i) Initialize the required colours.
(ii) Display a single text character on the LED matrix.
(iii) Choose suitable text and background colour.
(5) Else, clear everything.
(6) Keep the program running to receive the joystick events.

→ Python Code -

```
def red(): # Sets the entire LED matrix to red colour
    sense.clear (255, 0, 0)
```

```
def green (): # sets the entire LED matrix to green colour
    sense.clear (0, 255, 0)

def blue (): # sets the entire LED matrix to blue colour
    sense.clear (0, 0, 255)

def yellow (): # sets the entire LED matrix to yellow colour
    sense.clear (255, 255, 0)

# Tell the program which function to associate with which direction :-
sense.stick.direction_up = red
sense.stick.direction_down = blue
sense.stick.direction_left = green
sense.stick.direction_right = yellow
sense.stick.direction_middle = sense.clear # Press the enter key

# Digital Input operations using Joystick :-
while True:
    for event in sense.stick.get_events ():
        print (event.direction, event.action)

    if (event.direction == "middle" and event.action == "held"):
        yellow = (255, 255, 0); blue = (0, 0, 255) # Initialize the
colours
        sense.show_letter ("Z",
                           text_colour = yellow, # Text colour
                           back_colour = blue) # Background colour
    else:
        sense.clear () # clear everything

while True:
    pass # This keeps the program running to receive the
joystick events.
```

(4) Assignment: To implement a counter which displays the count on the led matrix. The count need to be incremented on pressing the joystick's right key, it should get decremented on pressing the joystick's left key and should get cleared to zero when the joystick's middle key is pressed.

→ Algorithm-

① Initialize the counter variable to zero.
② Initialize the required colours for text and background display.
③ Define functions to increment / decrement the counter value.
   (i) Declare the count variable globally.
   (ii) Increment / Decrement the counter value.
   (iii) Display the value with suitable text and background display.
④ Tell the program which function to associate with which direction.
⑤ Keep the program running to receive the joystick events.

→ Python Code -

```
count = 0
yellow = (255, 255, 0)
blue = (0, 0, 255)

def increment_value():
    global count
    count = count + 1
    sense.show_letter(str(count),
                      text_colour = yellow, # Text colour
                      back_colour = blue) # Background colour

def decrement_value():
    global count
    count = count + 1
    sense.show_letter(str(count),
                      text_colour = yellow, # Text colour
                      back_colour = blue) # Background colour
```

```
# Tell the program which function to associate with which direction :-
sense.stick.direction_up = increment_value
sense.stick.direction_down = decrement_value
sense.stick.direction_middle = sense.clear # Press the enter key

while True:
    pass # This keeps the program running to receive the joystick events
```

RESULT:

Thus, introduced Python basic programs using Raspberry Pi and GPIO and associated peripheral interfacing. All the simulation results were verified successfully.

10/11

7/10

EXPERIMENT NUMBER : 2

EXPERIMENT NAME: SENSOR INTERFACING AND DATA LOGGING USING RASPBERRY PI WITH SENSE HAT

DATE: 08/10/2022 , SATURDAY

* AIM:

To perform sensor interfacing and data logging using Raspberry Pi.

* INTEGRATED DEVELOPMENT ENVIRONMENT (IDE):

Name - Thonny 4.0.1

Publisher - Aivar Annamaa

Support link - https://thonny.org

* IMPORT NECESSARY LIBRARIES:

```
from sense_hat import SenseHat # Python module to control the Raspberry Pi Sense HAT
sense = SenseHat()
sense.clear() # No arguments defaults to OFF

from time import sleep
```

* INITIALIZE THE COLOURS:

```
red = (255, 0, 0)
green = (0, 255, 0)
blue = (0, 0, 255)
yellow = (255, 255, 0)
```

(a) Humidity Sensor Programming:

Gets the percentage of relative humidity from the humidity sensor.

→ Python Code -

```
humidity = sense.get_humidity()
print("Relative Humidity: " + str(round(humidity, 2)) + "%")
```

(b) Pressure sensor programming:
Gets the current pressure in millibars from the pressure sensor.

→ Python Code -

```
pressure = sense.get_pressure()
print("Current Pressure: " + str(round(pressure, 2)) + " millibars")
```

(c) Temperature Sensor Programming:
temperature in degrees from the temperature sensor.

→ Python Code -

```
temperature = sense.get_temperature()
print("Temperature: " + str(round(temperature, 2)) + " degree(s)")
```

(d) Print Local Time: no section available

① Return the time in seconds since the epoch as a floating point number.

② Convert a time expressed in seconds since the epoch to a string of a form 'Sun Jun 20 23:21:05 1993' representing local time.

→ Python Code -

```
seconds = time.time()
local_time = time.ctime(seconds)
print("Local Time: " + str(local_time))
```

(e) Temperature Controller using SenseHAT:-

① check if the temperature is greater than 34 degrees. (some)

② If yes, show the message in the led matrix, control scroll speed and choose suitable text and background colours.

③ Print that the temperature limit has been crossed.

④ Clear the display.

→ Python Code -

```
if (temperature > 34):
    sense.show_message ("Temperature Limit Crossed!",
                        scroll_speed = 0.005,
                        text_colour = green,
                        back_colour = red)
    print ("Temperature Limit Crossed!")
sense.clear() # no arguments defaults to OFF
```

(4) Accelerometer Sensor Programming:

① Calls set_imu_config to disable the magnetometer and gyroscope.

② Then, gets the current orientation from the accelerometer only.

③ Print the values of pitch, roll and yaw and show the same on the LED matrix.

→ Python Code -

```
accel_only = sense.get_accelerometer()
print ("Pitch: {pitch}, Roll: {roll}, Yaw: {yaw}".format
(**accel_only))

sense.show_message ("Pitch: {pitch}, Roll: {roll}, Yaw: {yaw}".
                    format (**accel_only),
                    scroll_speed = 0.005,
                    text_colour = yellow,
                    back_colour = blue)
```

(5) Acceleration Detection using Accelerometer:

① Gets the raw x, y and z axis accelerometer data.

② Print the x, y and z values.

③ Display the same on the LED matrix.

P.T.O →

→ Python Code -

```
raw = sense.get_accelerometer_raw() # Gets the raw x, y and z
axis accelerometer data.
print ("x: {x}, y: {y}, z: {z}".format(**raw))

sense.show_message ("x: {x}, y: {y}, z: {z}".format(**raw),
                    scroll_speed = 0.005,
                    text_colour = yellow,
                    back_colour = blue)
```

(b) Acceleration Detector along with Direction Detection using Accelerometer :

① check whether the value of z is greater than 0.975 (any axis with the required value limit).

② If yes, show the message in LED matrix, control the scroll speed and choose suitable text and background colours.

③ Print the same message in shell and clear the display.

→ Python Code -

```
if (raw["z"] > 0.975) :
    sense.show_message ("Z-Axis Limit Crossed !",
                        scroll_speed = 0.005,
                        text_colour = green,
                        back_colour = red)
    print ("Z-Axis Limit Crossed !")
sense.clear() # No arguments defaults to OFF
```

(1) Assignment : To implement a multiparameter display device which will display the current temperature, humidity and pressure on the LED matrix in SenseHAT when the middle button of the joystick is pressed.

① Return a list of InputEvent tuples representing all events that have occurred since the last call to get_events or wait_for_event

② check if the direction is "middle" and action is "held".

③ If yes, display the current pressure, relative humidity and temperature on the LED matrix in SenseHAT.
④ Controll the scroll speed: Greater speed implies slower scroll time.
⑤ Choose suitable text and background colours.
⑥ Clear the display in SenseHAT.

→ Python code -

```
while True:
    for event in sense.stick.get_events():
        if (event.direction == "middle" and event.action == "held"):
            sense.show_message("Current Pressure : {}
                                Relative Humidity : {}
                                Temperature : {}".
            format(pressure, humidity, temperature),
                                scroll_speed = 0.005,
                                text_colour = yellow,
                                back_colour = blue)
    sense.clear()
    break
```

j) Datalogger for weather station using SenseHAT:

① Gets the current pressure in Millibars from the pressure sensor.
② Gets the percentage of relative Humidity from the humidity sensor.
③ Returns the time in seconds since the epoch as a floating point number
④ Calls set_imu_config to disable the magnetometer and gyroscope. Then, gets the current orientation from the accelerometer only.
⑤ Gets the raw x, y and z axis accelerometer data.
⑥ Print the message in LED matrix and append to the log file.

→ Python code -

```
with open("weather", "w") as f:
    while True:
```

PTO

```
humidity = sense.get_humidity()
pressure = sense.get_pressure()
temperature = sense.get_temperature()
seconds = time.time()
local_time = time.ctime(seconds)

print_message = str("Pressure: " + str(pressure) +
                    "Temperature: " + str(temperature) +
                    "Humidity: " + str(humidity) +
                    "Time: " + str(local_time))

sense.show_message(print_message,
                   scroll_speed=0.005,
                   text_colour=green,
                   back_colour="red")

f.writelines(print_message)
f.write("\n")
sense.clear() # No argument defaults to OFF
```

* **RESULT:**

Thus, performed sensor interfacing and data logging using Raspberry Pi. All the simulation results were verified successfully.

10/11/'22

EXPERIMENT NUMBER : 3
EXPERIMENT NAME: WEB SERVER IMPLEMENTATION USING RASPBERRY PI
DATE : 03/11/2022, THURSDAY

* AIM:
Creation of Web server and its interactions using Flask Framework on Raspberry Pi with SenseHAT.

* INTEGRATED DEVELOPMENT ENVIRONMENT:
Name - Thonny 4.0.1
Publisher - Aivar Annamaa
Support link - https://thonny.org

(a) Web server Implementation using Raspberry Pi

→ Algorithm -

① From sense_hat import SenseHAT python module to control the Raspberry Pi Sense HAT.
② No arguments defaults to OFF and toggle the LED matrix in low light mode, useful if the Sense HAT is being used in a dark environment.
③ From flask import Flask, lightweight Web Server Gateway Interface (WSGI) web application framework.
④ Flask constructor takes the name of current module (__name__) as argument.
⑤ Initialize the colours and define the modules for 'Keep Out Set', 'Ok To Come In Set' and 'Off' routes.
⑥ Map the URLs to a specific function that will handle the logic for that URL.
⑦ Run the app in the main function.

PTO

→ Python Code -

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
sense.low_light = True

from flask import Flask
app = Flask(__name__)

green = (0, 255, 0)
red = (255, 0, 0)
nothing = (0, 0, 0)

def scene_keep_out():
    R = red
    scene = [
        R, R, R, R, R, R, R, R,
        R, R, R, R, R, R, R, R,
        R, R, R, R, R, R, R, R,
        R, R, R, R, R, R, R, R,
        R, R, R, R, R, R, R, R,
        R, R, R, R, R, R, R, R,
        R, R, R, R, R, R, R, R,
        R, R, R, R, R, R, R, R]
    return scene

def scene_come_in():
    G = green
    scene = [
        G, G, G, G, G, G, G, G,
        G, G, G, G, G, G, G, G,
        G, G, G, G, G, G, G, G,
        G, G, G, G, G, G, G, G,
```

```
        G, G, G, G, G, G, G, G,
        G, G, G, G, G, G, G, G,
        G, G, G, G, G, G, G, G,
        G, G, G, G, G, G, G, G]
    return scene


def scene_off():
    N = nothing
    scene = [
        N, N, N, N, N, N, N, N,
        N, N, N, N, N, N, N, N,
        N, N, N, N, N, N, N, N,
        N, N, N, N, N, N, N, N,
        N, N, N, N, N, N, N, N,
        N, N, N, N, N, N, N, N,
        N, N, N, N, N, N, N, N,
        N, N, N, N, N, N, N, N]
    return scene

@app.route('/')
def hello_world():
    return 'Lighting Scene controller!'

@app.route('/keep-out')
def keep_out():
    sense.set_pixels(scene_keep_out())
    return 'Keep out set'

@app.route('/come-in')
def come_in():
    sense.set_pixels(scene_come_in())
    return 'ok To come In Set'
```

→ OUTPUTS :

CASE I : 172.17.126.47 : 5000 /
Lighting Scene Controller !

CASE II : 172.17.126.47 : 5000 / keep out
Keep out Set } Red LED matrix

CASE III : 172.17.126.47 : 5000 / come in
Ok To Come In Set } Green LED matrix

CASE IV : 172.17.126.47 : 5000 / off
Off } Default LED Matrix (nothing)

```
@app.route('/off')
def off():
    sense.set_pixels(scene_off())
    return 'off'

# pi@raspberrypi:~ $ ifconfig
# eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
#       inet 172.17.126.47 netmask 255.255.255.0 broadcast ...
#       inet6 fe80::9d5a:cdd:c209:ebab prefixlen 64 ...
#       ...
#       ...
#       ...
#       TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

# Driver code - Main Function:-
if __name__ = '__main__':
    app.run(host='172.17.126.47') # Port Number - 5000
```

(16) Web server based Weather Station using Raspberry Pi with SenseHat

→ Algorithm -

① From sense_hat import SenseHat, python module to control the Raspberry Pi Sense HAT.

② No arguments defaults to OFF. Toggle the LED matrix in low light mode, useful if the Sense HAT is being used in a dark environment.

③ From flask import Flask, a lightweight Web Server Gateway Interface (WSGI) web application framework.

④ From flask import render_template, used to generate output from a template file that is found in the application's templates folder.

⑤ The flask constructor takes the name of current module [__name__] as argument.

PTO

(6) Import time, the python module providing various time-related function. Initialize the text and background colours for the LED matrix display.

(7) Map the URLs to a specific function that will handle the logic for that URL.

(8) Gets the current pressure in Millibars from the pressure sensor and the temperature in Degrees from the temperature sensor.

(9) Return the time in seconds since the epoch as a floating point number and convert the time in seconds since the epoch to a string of a form: 'Sun Jun 20 23:21:05 1993' representing local time.

(10) Print the message in Thonny Shell and show the same in the LED matrix with appropriate scroll speed and suitable text and background colour.

(11) Run the app in the main function.

→ Python Code -

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
sense.low_light = True

from flask import Flask
from flask import render_template
app = Flask(__name__)

import time

green = (0,255,0)
nothing = (0,0,0)

@app.route('/')
def index():
```

PTO

```
    pressure = sense.get_pressure()
    temperature = sense.get_temperature()
    seconds = time.time()
    local_time = time.ctime(seconds)

    print_message = str("Weather Monitoring System!" + "\n" +
"current Pressure: " + str(round(pressure, 2)) + "millibars" + "\n" +
"Temperature: " + str(round(temperature, 2)) + "degrees(s)" + "\n" +
"Local Time: " + str(local_time))

    sense.show_message("Current Pressure: {} Temperature: {}
Local Time: {}".format(pressure, temperature, local_time),
                       scroll_speed = 0.005,
                       text_colour = green,
                       back_colour = nothing)

    weatherData = {'weather': print_message}
    return render_template('index.html', **weatherData)

# pi@raspberrypi:~ $ ifconfig
# eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
#         inet 172.17.126.47 netmask 255.255.255.0 ...
#         inet6 fe80::9d5a:cdd:c209:eba6 prefixlen 64 ...
#         ....
#         ....
#         ....
#         TX errors 0 dropped 0 overruns 0 carrier 0 ....

# Driver Code :- Main Function :-
if __name__ == '__main__'
    app.run(host = '172.17.126.47') # Port Number - 5000
```

→ templates' Folder:- index.html -

```
<html>
    <head>
        <link rel = "stylesheet" href = "/static/style.css" />
    </head>
    <body>
        <h1> Weather: </h1>
        <h2 class = "title"> {{weather}} </h2>
    </body>
</html>
```

→ 'static' Folder :- style.css -

```
.title {
    font-family: Roboto;
    color: #FFFFFF;
}

body {
background: navy;
color: yellow;
font-family: Arial, sans-serif;
text-align: center;
}

h1 {
border-bottom: 1px solid gold;
margin-bottom: 40px;
padding: 10px;
}
```

PTO

→ OUTPUTS:

index.html -

weather

{{weather}}

c) Web Server based Remote Device Control using Raspberry Pi with Sense HAT :

→ Algorithm -

① From sense_hat import SenseHat, python module to control the Raspberry Pi Sense HAT.

② No argument defaults to OFF and toggle the LED matrix in low light mode, useful if the Sense HAT is being used in a dark environment.

③ From flask import Flask, a lightweight Web Server Gateway Interface (WSGI) web application framework.

④ From flask import render_template, used to generate output from a template file that is found in the application's templates folder.

⑤ Flask constructor takes the name of current module (__name__) as argument.

⑥ Map the URLs to a specific function that will handle the logic for that URL.

⑦ Index() definition -

(i) Set the device status to 'OFF' and show the letter 'I' in the LED matrix.

(ii) Store the device name and status in a dictionary and return the same along with the index.html as parameter in render_template().

⑧ action() definition -

(i) For the required device name and action (ON/OFF), set the corresponding device status and show the message on the LED matrix.

(ii) Store the device name and status in a dictionary and return the same along with the index.html as parameter in render_template().

⑨ clear() definition -

(i) Set the device status as 'OFF'.

(ii) Clear the Sense HAT display matrix.

(iii) Store the device name and status in a dictionary and return the same along with index.html as parameter in render_template().

⑩ Run the app in the main function.

→ Python code -

```
from sense-hat import SenseHat
sense = SenseHat()
sense.clear()
sense.low_light = True

from flask import Flask
from flask import render_template
app = Flask(__name__)

@app.route('/')
def index():
    deviceSts = "OFF"
    sense.show_letter("I")
    templateData = {'device1' : devicests}
    return render_template('index.html', **templateData)
#   return render_template('index.html', **weatherData)

@app.route("/<deviceName>/<action>")
def action(deviceName, action):
    if deviceName == 'd1' and action == 'on':
        devicests = "ON"
        sense.show_letter("O")  # Up Arrow
    if deviceName == 'd1' and action == 'off':
        deviceSts = "OFF"
        sense.show_letter("F")  # Up Arrow
    templateData = {'device1' : deviceSts}
    return render_template('index.html', **templateData)

@app.route('/off')
def clear():
    devicests = "OFF"
    sense.clear()
```

templateData = {'device1' : deviceSts}
return render_template ('index.html', **templateData)

Driver Code - Main Function :-
if __name__ == '__main__'
app.run (host = [172.17.126.47]) # Port Number - 5000
If config

→ 'static' Folder :- style.css -

```
body {
    background: blue;
    color: yellow;
}

.button {
    font: bold 15px Arial;
    text-decoration : none;
    background-color : #EEEEEE;
    color : #333333
    padding : 2px 6px 2px 6px;
    border-top : 1px solid #CCCCCC;
    border-right : 1px solid #333333;
    border-bottom : 1px solid #333333;
    border-left : 1px solid #CCCCCC;
}
```

→ 'templates' Folder :- index.html -

```
<!DOCTYPE html>
    <head>
        <title> Device Control </title>
        <link rel = "stylesheet" href = "/static/style.css" />
    </head>
    <body>
        <h1> Raspberry Pi Device Control </h1>
        <h2> Device status </h2>
        <h3> Device 1 ==> {{ device1 }} </h3>
```

* OUTPUTS :

→ index.html -

Raspberry Pi Device Control

Device Status

Device 1 ==> {{ device1 }}

Commands

Device curl ==> Device1 ON Device1 OFF

```
    <br>

    <h2> Commands </h2>
    <h3>
        Device Ctrl =>>
        <a href = "/d1/on" class = "button"> Device 1 ON </a>
        <a href = "/d1/off" class = "button"> Device 1 OFF </a>
    </h3>
  </body>

</html>
```

* **RESULT:**

Thus, implemented web server and its interactions using Flask Framework on Raspberry Pi with SenseHAT. Thus, all the simulation results were verified successfully.

EXPERIMENT NUMBER : 4

EXPERIMENT NAME: RASPBERRYPI AND IOT CLOUD SERVER INTERFACE USING MQTT PROTOCOL 97

DATE : 10/11/2022, THURSDAY

---

* AIM:

To analyze MQTT protocol used in the field of Smart devices and develop basic programming skills for deploying the protocol in hardware.

* INTEGRATED DEVELOPMENT ENVIRONMENT (IDE):

Name - Thonny 4.0.1
Publisher - Aivar Annamaa
Support link - https://thonny.org

(A) Data Transmission from Raspberry Pi to ThingSpeak IoT cloud Server :

→ Algorithm -

① Import json, can be used to work with JSON (JavaScript object Notation) data.
② Import random, implements pseudo-random number generators for various distributions.
③ Import requests, allows you to send HTTP/1.1 requests extremely easily.
④ Import threading, constructs higher-level threading interface on top of the lower level _thread module.
⑤ Import urllib.request - defines functions and classes which help in opening URLs (mostly HTTP) in a complex world.
⑥ Create a timer with interval and function as parameters.
⑦ Return an integer number selected element from the specified range.
⑧ Write a channel feed and API key.
⑨ Open the URL, which can be either a string or a Request object.

→ Python Code -

```
import urllib.request
import requests
import threading
import json
import random

def thingspeak_post():
    threading.Timer(15, thingspeak_post).start()
    val = random.randint(1, 30) # Alias for randrange(start, stop+1)
    URL = 'https://api.thingspeak.com/update?api_key=....'
    KEY = '.....XXXX.....'

    HEADER = '&field1={}&field2={}'.format(val, val)
    NEW_URL = URL + KEY + HEADER
    print(NEW_URL)

    data = urllib.request.urlopen(NEW_URL)
    print(data)

if __name__ == '__main__':
    thingspeak_post()
```

(b) Sensor Data Transmission from Raspberry Pi with SenseHAT to ThingSpeak IOT cloud Server:

→ Algorithm -

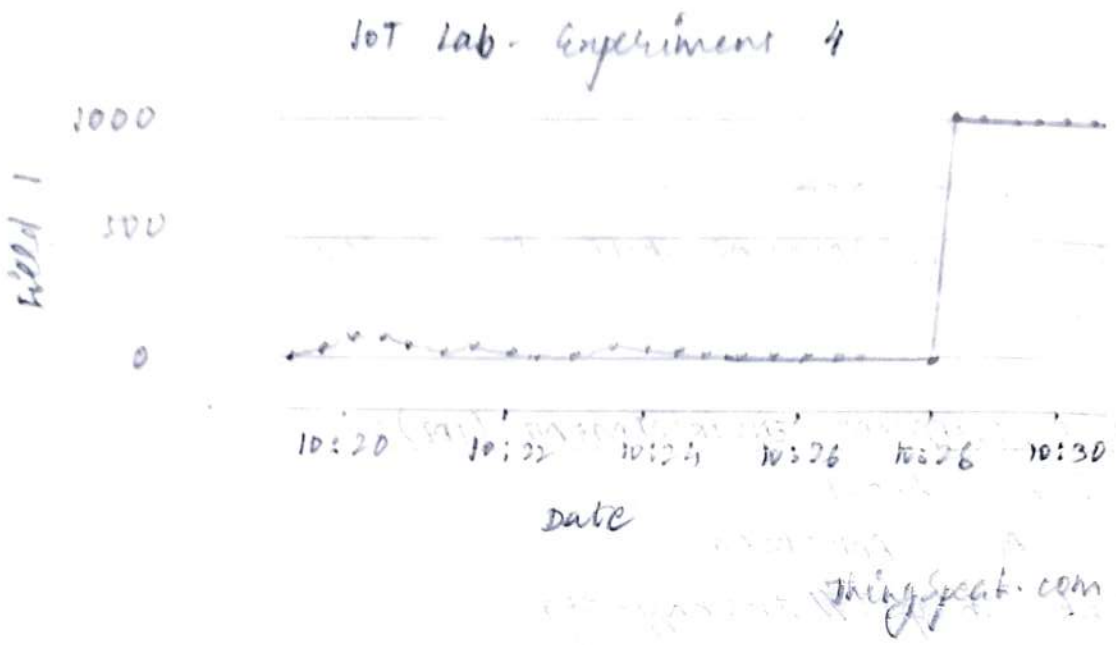
① Import json, can be used to work with JSON (JavaScript object Notation) data.

② Import requests, allows you to send HTTP/1.1 requests extremely easily.

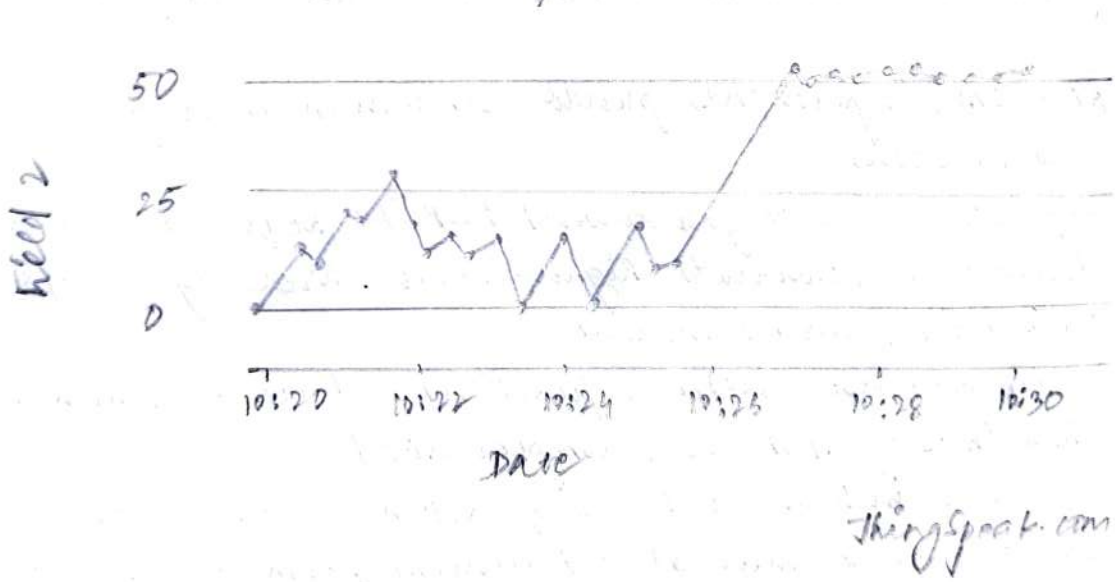
③ Import threading, constructs higher-level threading interfaces on top of the lower level _thread module.

→ OUTPUTS:

Field 1 Chart



Field 2 Chart



④ Import urllib.request, defines functions and classes which help in opening URLs (mostly HTTP) in a complex world.

⑤ Import SenseHat from sense_hat, python module to control the Raspberry Pi Sense HAT. No arguments defaults to OFF.

⑥ Create a timer with interval and function as parameters.

⑦ Get the pressure and temperature values from the corresponding sensors.

→ Python Code -

```
import urllib.request
import requests
import threading
import json

from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
sense.low_light = True

def thingspeak_post():

    threading.Timer(15, thingspeak_post).start
    pressure = sense.get_pressure()
    humidity = sense.get_humidity()

    URL = 'https://api.thingspeak.com/update?api_key=....'
    KEY = '....xxxx....'  # write API key
    HEADER = '&field1={}&field2={}'.format(pressure, humidity)

    NEW_URL = URL + KEY + HEADER
    print(NEW_URL)
    data = urllib.request.urlopen(NEW_URL)
    print(data)
```

```
if __name__ == '__main__':
    thingspeak_post()
```

(c) Sensor Data Transmission from Raspberry Pi with SenseHAT to Adafruit Iot Cloud Server :

--> Algorithm -

① From sense_hat import SenseHat, python module to control the Raspberry Pi Sense HAT.

② No arguments defaults to OFF and toggle the LED matrix in low light mode, useful if the Sense HAT is being used in dark environment.

③ Import library and create instance of REST client.

④ Import time, provides time-related functions. Import random, implements pseudo-random number generators for various distributions.

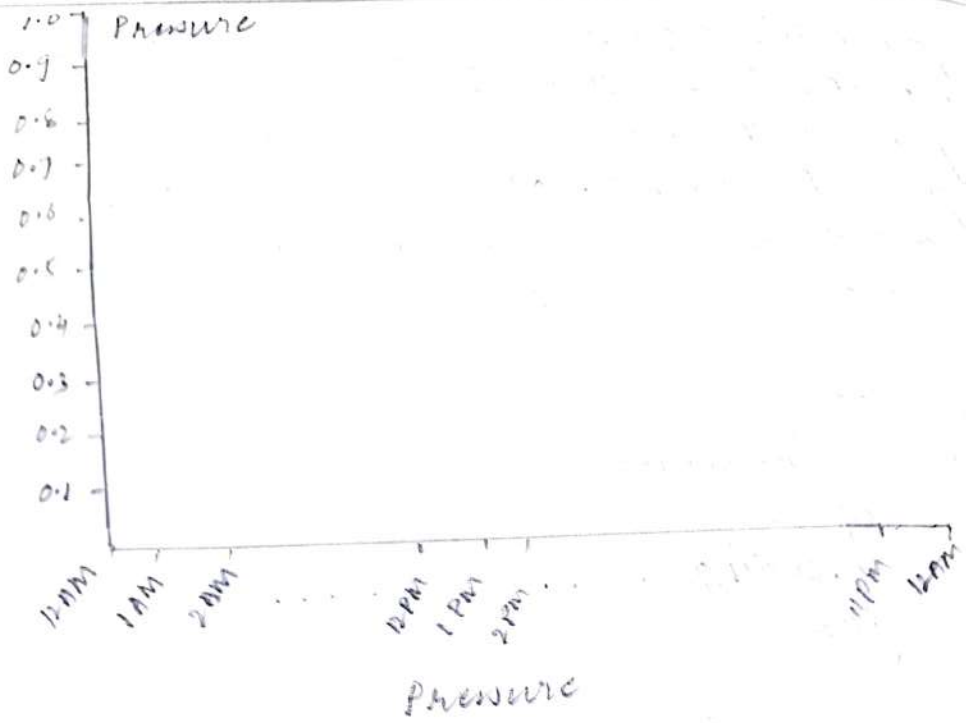
⑤ Get list of feeds :-
(i) 'Pressure' is ID Feed created in Adafruit.
(ii) 'Humidity' is ID Feed created in Adafruit.

⑥ Return an integer number selected element from the specified range.

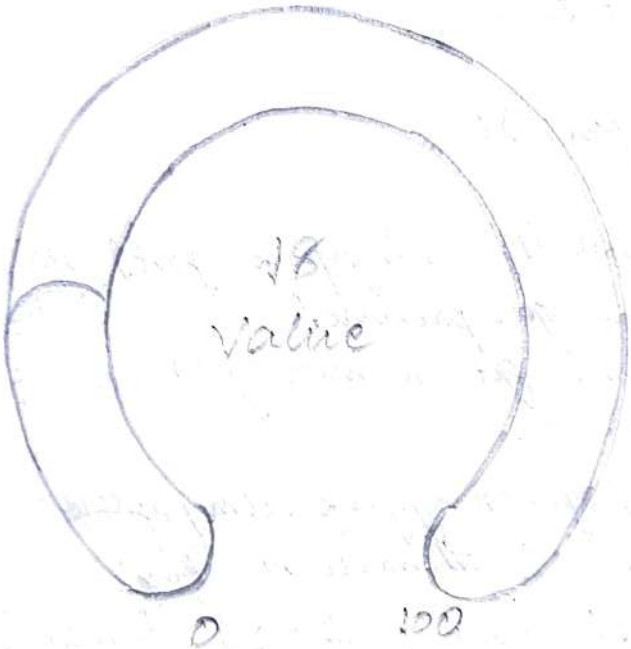
⑦ If humidity and pressure values are not None, send pressure and Humidity feeds to Adafruit IO.

⑧ Else, print that 'Failed to get Pressure and Humidity Data from SenseHat!'

⑨ Sleep (wait) for some time to avoid flooding Adafruit IO.


Pressure
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
12AM
1AM
2AM
8PM
1PM
2PM
11PM
12AM
Pressure

Humidity


78
Value
0
100

→ Python Code -

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
Sense.low_light = True

from Adafruit_IO import client, feed
import time
import random

READ_TIMEOUT = 5
ADAFRUIT_IO_KEY = '.... xxxx ....' # Active Key
ADAFRUIT_IO_USERNAME = '.... xxxx ....' # Username
aio = client (ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

pressure_feed = aio.feeds ('pressure')
humidity_feed = aio.feeds ('humidity')

while True:

    val = random.randint (1, 30) #Alias for randrange (start, stop))
    pressure = val
    humidity = val

    if humidity is not None and pressure is not None:
        print ('Pressure = {0:0.1f} Pascal \n Humidity = {1:0.1f}%'
               .format (pressure, humidity))

    pressure = '%.2f' % (pressure)
    humidity = '%.2f' % (humidity)
    aio.send (pressure_feed.key, str (pressure))
    aio.send (humidity_feed.key, str (humidity))
```

```
else:
    print ('Failed to get Pressure and Humidity Data from SenseHat!')
    time.sleep(READ_TIMEOUT)
```

(d) **Remote Device Control using Adafruit IOT Cloud Server on Raspberry Pi:**

→ Algorithm -

① From sense-hat import SenseHat, python module to control the Raspberry Pi Sense HAT.

② No argument defaults to OFF and toggle the LED matrix in low light mode, useful if the Sense HAT is being used in a dark environment.

③ Import sys, provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.

④ From Adafruit_IO import MQTTClient
- Username, Active key, Key from Feeds

⑤ Define callback functions which will be called when certain events happen :-

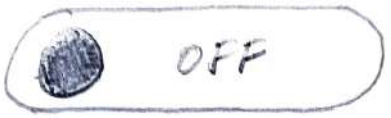
(i) connected() - Will be called when the client connects
Subscribe to a feed

(ii) disconnected() - Will be called when the client disconnects
optional argument arg can be an integer giving the exit or another type of object; Zero is considered "successful termination".

(iii) message() - Will be called when a subscribed feed has a new value. The feed_id parameter identifies the feed, and the payload parameter has the new value.
If payload = 1 ; print "Light ON"
Display a single text character on the LED mat
If payload = 0 ; print "Light OFF"
No arguments defaults to OFF

⑥ Open a new MQTT connection to the specified broker

Remote_Device_Control



→ Python code-

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
sense.low_light = True

import sys
from Adafruit_IO import MQTTClient

ADAFRUIT_IO_KEY = '....xxxx....' # Active key
ADAFRUIT_IO_USERNAME = 'S...xxxx....' # username
FEED_ID = '....xxx....' # key from Feeds

def connected(client):
    print('Connected to Adafruit IO! Listening for {0} changes...'
format(FEED_ID))
    client.subscribe(FEED_ID)

def disconnected(client)
    print('Disconnected from Adafruit IO!')
    sys.exit(1)

def message(client, feed_id, payload):
    print('Feed {0} received new value: {1}'.format(feed_id,
payload))

    if payload == "1":
        print("Light ON")
        sense.show_letter("O")

    if payload == "0":
        print("Light OFF")
        sense.clear()
```

```
client = MQTT Client (ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)
client.on_connect = connected
client.on_disconnect = disconnected
client.on_message = message
client.connect()
client.loop_blocking()
```

* **RESULT:**

Thus, analysed MQTT protocol in the field of Smart devices and developed basic programming skills for deploying the protocol in hardware. All the simulation results were verified successfully.

EXPERIMENT NUMBER : 5
EXPERIMENT NAME: MACHINE LEARNING USING IRIS DATASET
DATE : 17/11/2022, THURSDAY

* AIM:

To implement various machine learning techniques using Raspberry Pi.

* INTEGRATED DEVELOPMENT ENVIRONMENT (IDE):

Name - Thonny 4.0.1 Python 3.10.6
Publisher - Aivar Annamaa
Support link - https://thonny.org

* IMPORT NECESSARY LIBRARIES:

```
import sys #Provides access to some variables used or maintained
           by the interpreter and to functions that interact
           strongly with the interpreter.
print ('Python : {}'.format(sys.version))

import scipy #Includes modules for statistics, optimization,
             integration, linear algebra, Fourier transforms,
             signal and image processing, and more
print ('scipy : {}'.format(scipy.__version__))

import numpy #support for large, multi-dimensional arrays
             and matrices
print ('numpy : {}'.format(numpy.__version__))

import matplotlib #Provides a MATLAB-like plotting framework
print ('matplotlib : {}'.format(matplotlib.__version__))

import pandas #Library for working with data sets
print ('pandas : {}'.format(pandas.__version__))
```

```
import sklearn #Provides various tools for model fitting,
                data preprocessing, model selection,
                model evaluation, and many other utilities
print ('sklearn : {}'. format (sklearn. __version__)
```

(a) Univariate and Multivariate Data Plots using IRIS dataset :

→ Algorithm -

① Import pandas, library for working with data sets.
② From pandas. plotting import scatter_matrix, to draw a matrix of scatter plots.
③ Import matplotlib. pyplot as plt, collection of command style functions that make matplotlib work like MATLAB.
④ Load the dataset and read comma-seperated values (csv) file into DataFrame.
⑤ Return a tuple representing the dimensionality of the DataFrame and return the first n rows.
⑥ Generate descriptive statistics that summarizes the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.
⑦ Group DataFrame using a mapper or by a Series of columns and make plots of Series or DataFrame.
⑧ Make a histogram of the DataFrame's columns and visualize data with scatter plots.

→ Python Code -

```
import pandas
from pandas. plotting import scatter_matrix
import matplotlib. pyplot as plt

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/
        master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length',
         'petal-width', 'class']
dataset = pandas. read_csv (url, names = names)
```

```
print ('Dataset Dimensions: ' + str(dataset.shape))
print ('\nHead of the Data: \n' + str(dataset.head(20)))
print ('\nData statistics: \n' + str(dataset.describe))
print ('\nClass Distribution: \n' + str(dataset.groupby('class').size()))

dataset.plot(kind='box', layout=(2,2), sharex=False, sharey=
False); plt.show().
dataset.hist(); plt.show()
scatter_matrix(dataset); plt.show()
```

(b) Analysis of Machine Learning using IRIS Dataset:

→ Algorithm -

① Import matplotlib.pyplot as plt, a collection of command style functions that make matplotlib work like MATLAB.

② Import pandas, library for working with data sets. Import scatter_matrix from pandas.plotting to draw a matrix of scatter plots.

③ From sklearn import model_selection, split arrays or matrices into random train and test subsets.

④ Import metrics for classification technique and other model building libraries.

⑤ Load dataset and read the comma-seperated values (csv) file into DataFrame.

⑥ Create training and validation datasets and return a Numpy representation of the DataFrame. Assume independent (data) variable and dependent (target) variable.

⑦ Control the shuffling applied to the data before applying the split and set the scoring criteria.

⑧ Build all the models first, and then evaluate each model.

⑨ Provide train/test indices to split data into train/test sets and evaluate a score by cross-validation.

P.T.O

Dataset Dimension
(150, 5)
Head of Data :

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa

statistics

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Class Distribution

class	
Iris-setosa	50
Iris-versicolor	50
Iris-virginica	50

dtype: int64

Figure 1 - Descriptive statistics of IRIS

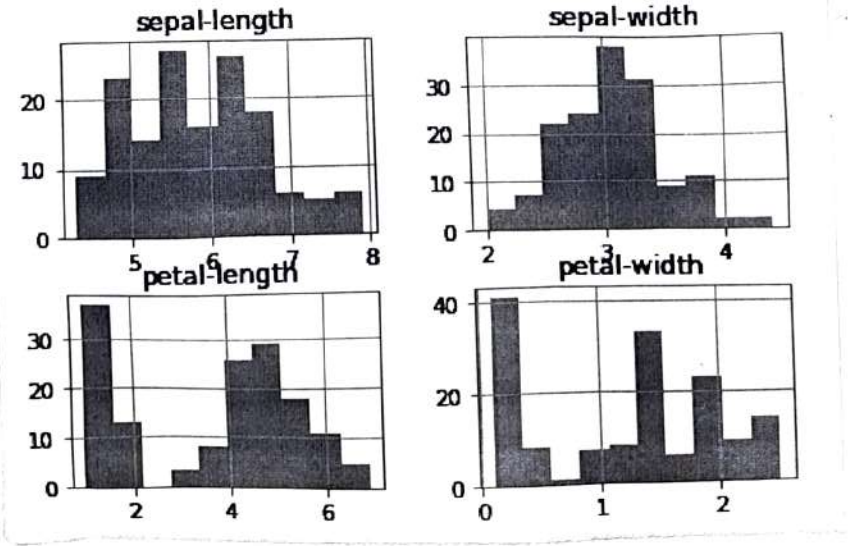


Figure 2 - Histogram of IRIS columns

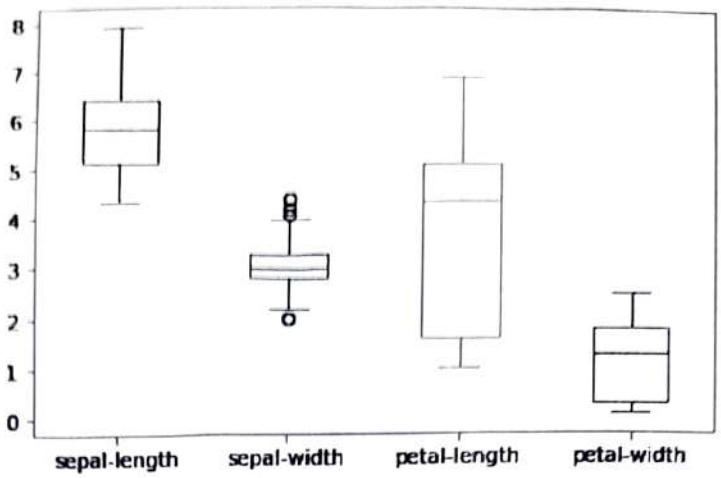


Figure 3 - Box Plot

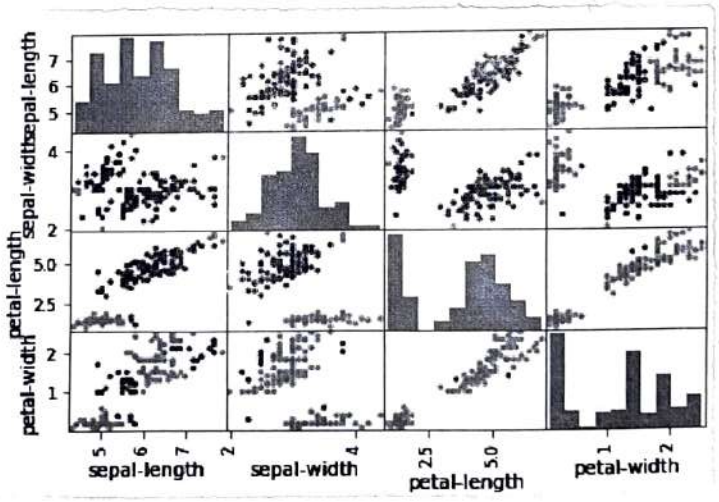


Figure 4 - Matrix of scatter plots

→ Python Code –

```
import matplotlib.pyplot as plt
import pandas
from pandas.plotting import scatter_matrix
from sklearn import model_selection

from sklearn.metrics import accuracy_score # Accuracy
classification score
from sklearn.metrics import classification_report # Build a text
report showing the main classification metrics
from sklearn.metrics import confusion_matrix # compute
confusion matrix to evaluate the accuracy of a classification

from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis as LDA # Linear Discriminant Analysis
from sklearn.linear_model import LogisticRegression # Logistic
Regression (aka logit, MaxEnt) classifier
from sklearn.naive_bayes import GaussianNB # Gaussian Naive
Bayes (GaussianNB)
from sklearn.neighbors import KNeighborsClassifier # Classifier
implementing the k-nearest neighbors vote
from sklearn.tree import DecisionTreeClassifier # A decision
tree classifier
from sklearn.svc import SVC # C-Support Vector Classification

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/
master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',
'class']
dataset = pandas.read_csv(url, names=names)
```

PTO

```
array = dataset.values
x = array[:, 0:4]
y = array[:, 4]
validation_size = 0.20
seed = 7

X_train, X_validation, Y_train, Y_validation =
        model_selection.train_test_split(x, y,
                                         test_size = validation_size,
                                         random_state = seed)

models = results = names = []
scoring = 'accuracy'

models.append(('LR', LogisticRegression(solver = 'liblinear',
multi_class = 'ovr')))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(kernel = 'linear', random_state = 0)))
models.append(('LDA', LDA(n_components = 1)))

for name, model in models:
    kfold = model_selection.KFold(n_splits = 10, random_state = seed,
shuffle = True)
    cv_results = model_selection.cross_val_score(model, X_train,
Y_train, cv= kfold, scoring = scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(),
cv_results.std())
    print(msg)
```

(c) Analysis of Machine Learning Models using Box and whisker diagrams:

① Import matplotlib.pyplot as plt, a collection of command style functions that make matplotlib like MATLAB.

② Import pandas, library for working with data sets. Import scatter_matrix from pandas plotting to draw a matrix of scatter plots.

③ From sklearn import model_selection, split arrays or matrices into random train and test subsets.

④ Import metrics for classification technique and other model building libraries.

⑤ Load dataset and read the comma-seperated values (csv) file into DataFrame.

⑥ Create training and validation datasets and return a NumPy representation of the DataFrame. Assume data (independent) and target (dependent) variable.

⑦ Control the shuffling applied to the data frame before applying the split and set the scoring criteria.

⑧ Build all the models first, and then evaluate each model.

⑨ Provide train/test indices to split data into train/test sets and evaluate a score by cross-validation.

⑩ Plot Model Results-

(i) Create a new figure, or activate an existing figure.
(ii) Add a centered suptitle to the figure.
(iii) Add an axes to the current figure or retrieve an existing axes
(iv) Make a box and whisker plot
(v) Set the xaxis' labels with list of string labels.
(vi) Display all open figures.

⑪ Machine Learning Model Analysis -

(i) Call the classifier with required parameters.
(ii) Fit model according to the given training data and paramsters.
(iii) Predict class labels for samples in X.
(iv) Print accuracy score and classification report.

→ Python Code -

```
import matplotlib.pyplot as plt
import pandas
from pandas.plotting import scatter_matrix
from sklearn import model_selection

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
as LDA
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/
master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-
width', 'class']
dataset = pandas.read_csv(url, names=names)

array = dataset.values
X = array[:,0:4]
Y = array[:,4]
validation_size = 0.20
seed = 7

X_train, X_validation, Y_train, Y_validation =
    model_selection.train_test_split(X, Y,
                                     test_size = validation_size,
                                     random_state = seed)
```

```
models= results = names = []
scoring = 'accuracy'

models.append (('LR', LogisticRegression(solver = 'liblinear',
multi_class = 'ovr')))
models.append (('KNN', KNeighborsClassifier()))
models.append (('CART', DecisionTreeClassifier()))
models.append (('NB', GaussianNB()))
models.append (('SVM', SVC(kernel = 'linear', random_state = 0)))
models.append (('LDA', LDA(n_components = 1)))

for name, model in models:
    kfold = model_selection.KFold(n_splits = 10, random_state = seed,
shuffle = True)
    cv_results = model_selection.cross_val_score(model, X_train,
Y_train, cv = kfold, scoring = scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(),
cv_results.std())
    print(msg)

figure = plt.figure()
figure.suptitle('Algorithm Comparison')
algplot = figure.add_subplot(1, 1, 1)
plt.boxplot(results)
algplot.set_xticklabels(names)
plt.show()

print("\n Linear Discriminant Analysis Classifier: \n")
LDA = LDA(n_components = 1)
LDA.fit(X_train, Y_train)
predictions = LDA.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
print ("\n Logistic Regression classifier : \n")
LR = LogisticRegression (solver = 'liblinear', multi_class = 'ovr')
LR.fit (X_train, Y_train)
predictions = LR.predict (X_validation)
print (accuracy_score (Y_validation, predictions))
print (classification_report (Y_validation, predictions))

print ("\n K-Neighbors classifier : \n")
knn = KNeighborsClassifier ()
knn.fit (X_train, Y_train)
predictions = knn.predict (X_validation)
print (accuracy_score (Y_validation, predictions))
print (classification_report (Y_validation, predictions))

print ("\n Decision Tree classifier : \n")
DT = DecisionTreeClassifier ()
DT.fit (X_train, Y_train)
predictions = DT.predict (X_validation)
print (accuracy_score (Y_validation, predictions))
print (classification_report (Y_validation, predictions))

print ("\n Naive Bayes classifier : \n")
NB = GaussianNB ()
NB.fit (X_train, Y_train)
predictions = NB.predict (X_validation)
print (accuracy_score (Y_validation, predictions))
print (classification_report (Y_validation, predictions))

print ("\n Support Vector Machine (svm) classifier : \n")
SVM = SVC (kernel = 'linear', random_state = 0)
SVM.fit (X_train, Y_train)
predictions = SVM.predict (X_validation)
print (accuracy_score (Y_validation, predictions))
print (classification_report (Y_validation, predictions))
```

```
LR: 0.958333(0.055902)
KNN: 0.983333(0.033333)
CART: 0.950000(0.076376)
GNB: 0.966667(0.040825)
LDA: 0.975000(0.038188)
SVM: 0.983333(0.033333)
0.9
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	0.85	0.92	0.88	12
Iris-virginica	0.90	0.82	0.86	11
accuracy			0.90	30
macro avg	0.92	0.91	0.91	30
weighted avg	0.90	0.90	0.90	30

Figure 5 - Classifier Results

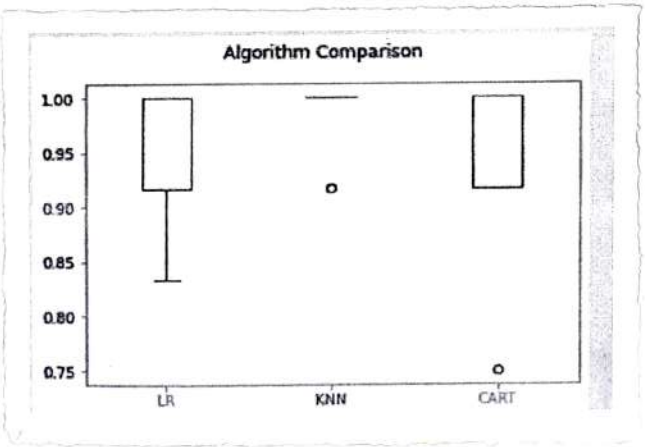


Figure 6 - Comparison of LR, KNN and CART results

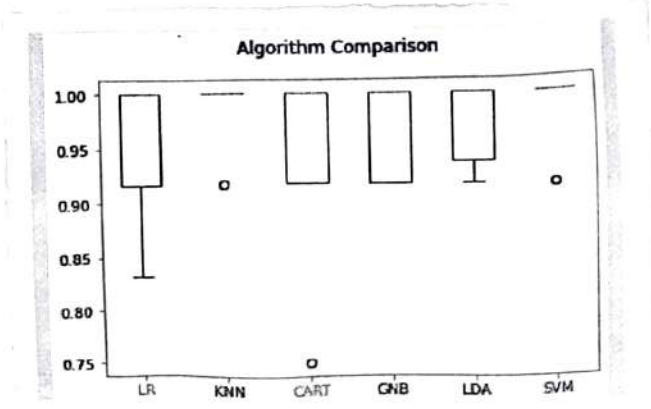


Figure 7 - Comparison of all model results.

RESULT :

Thus, implemented various machine learning techniques using Raspberry Pi. All the simulation results were verified successfully.

9/10

EXPERIMENT NUMBER - 6

EXPERIMENT NAME - IMAGE PROCESSING USING RASBERRY PI

DATE - 24/11/2022, THURSDAY

---

* AIM:

Introduce basic image processing techniques using Raspberry Pi.

* INTEGRATED DEVELOPMENT ENVIRONMENT (IDE):

Name - Thonny 4.0.1

Publisher - Aivar Annamaa

support link - https://thonny.org

* IMPORT NECESSARY LIBRARIES:

① Install Pillow library using the following command -

```
sudo pip3 install pillow
```

② Install an image viewing utility called xv for the built-in function show() to work.

```
sudo apt-get install xli -y
cd /usr/local/bin
sudo ln -s /usr/bin/xli xv
```

③ Import Tkinter library for GUI support in Python.

```
sudo apt-get install python-3-pil.imagetk
```

(a) Display Image and its Properties:

→ ALGORITHM -

① From PIL import Image, provides a class with the same name which is used to represent a PIL image.

② From PIL import ImageTk, contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL images.

③ Import tkinter as tk, standard Python interface to the Tcl/Tk GUI toolkit.

④ Open and identify the given image file.

Figure 1. Sample Image from web

⑤ Print the image characteristics for the following functionalities :-
(i) mode - string specifying the pixel format used by the image
(ii) format - File format of the source file
(iii) size - size in pixels is given as a 2-tuple (width, height)
(iv) Info - A dictionary holding data associated with the image
(v) Getbands() - Returns a tuple containing the name of each band in the image

⑥ Construct a toplevel Tk widget and a Tkinter-compatible photo image. Widget that is used to implement display boxes where you can place text or images.

⑦ Organize widgets in blocks before placing them in the parent widget and loop forever, waiting for events from the user, until the user exits the program.

→ PYTHON CODE -

```
from PIL import Image
from PIL import ImageTk
import tkinter as tk
im = Image.open('Sample_Image_Web.tiff')

print(im.mode)
print(im.format)
print(im.size)
print(im.info)
print(im.getbands())

root = tk.Tk()
root.title("Display Image")
photo = ImageTk.PhotoImage(im)

l = tk.Label(root, image = photo)
l.pack()
l.photo = photo
root.mainloop()
```

Figure 2 - Display Image and its Properties

(6) Splitting and merging Image channels:

→ ALGORITHM -

① From PIL import Image. provides a class with the same name which is used to represent a PIL image.

② From PIL import ImageTk, contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL images.

③ Import tkinter as tk, standard Python interface to the Tcl/Tk GUI toolkit.

④ Opens and identify the given image file. Construct a toplevel Tk widget and split the image into individual bands.

⑤ Create a Tkinter-compatible photo image and widget that is used to implement display boxes where you can place text or images.

⑥ Organize widgets in blocks before placing them in the parent widget and merge a set of single band images into a new multiband image.

⑦ Loop forever, waiting for events from the user, until the user exits the program.

→ PYTHON CODE -

```
from PIL import Image
from PIL import ImageTk
import tkinter as tk
im = Image.open('sample_Image-web.tiff')

root = tk.Tk()
root.title("RED channel Demo")
r,g,b = im.split()

photo_1 = ImageTk.PhotoImage(im)
l-1 = tk.Label(root, image = photo_1)
l-1.pack()
l-1.photo = photo_1
```

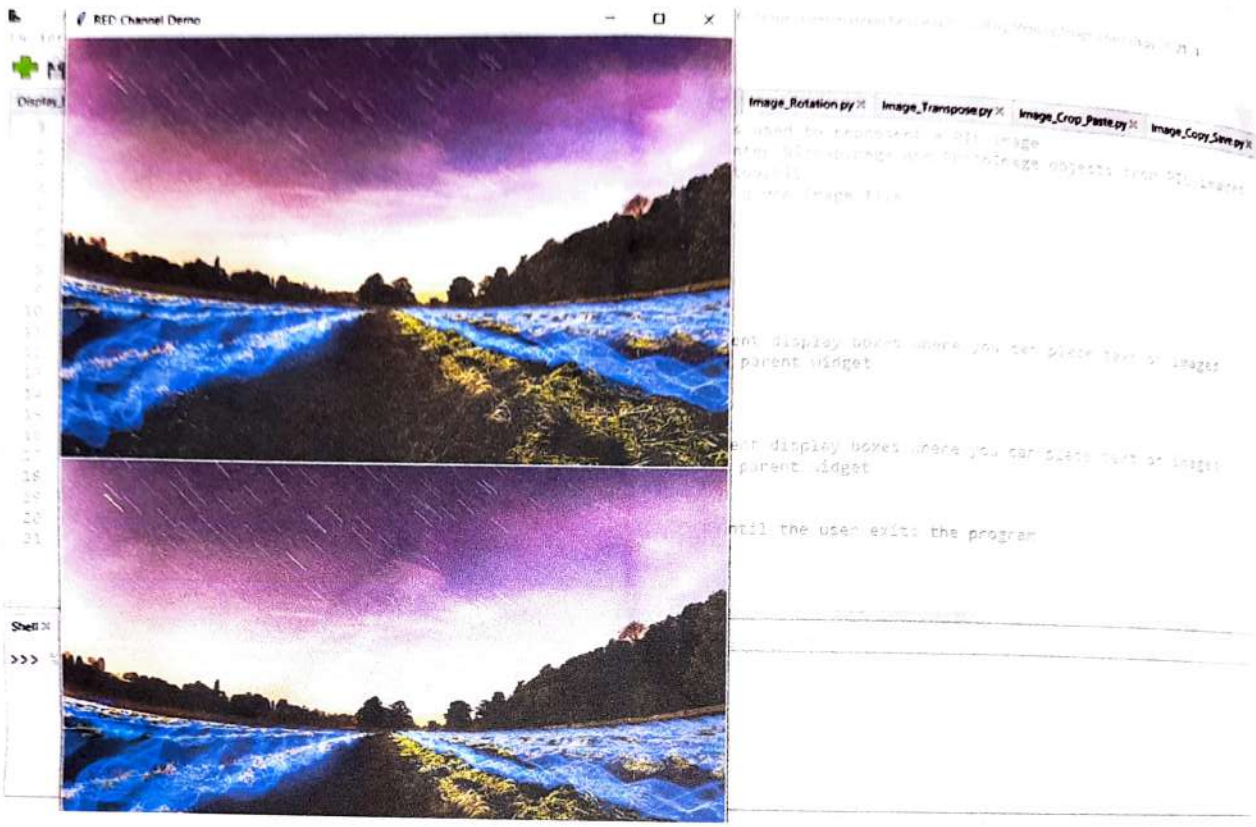


Figure 3 - Splitting and Merging Image Channels.

```
photo_2 = ImageTk.PhotoImage(Image.merge("RGB", (r,g,b)))
l_2 = tk.Label(root, image = photo_2)
l_2.pack()
l_2.photo = photo_2

root.mainloop()
```

(c) Image Mode Conversion:

→ ALGORITHM -

① From PIL import Image, provides a class with the same name which is used to represent a PIL image.

② From PIL import ImageTk, contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL images.

③ Import tkinter as tk, standard Python interface to the Tcl/Tk GUI toolkit.

④ Open and identify the given image file. Return a converted copy of the image, translating a color image to greyscale.

⑤ Construct a toplevel Tk widget and a Tkinter-compatible photo image widget that is used to implement display boxes where you can place text or images.

⑥ Organizes widgets in blocks before placing them in the parent widget. Loop forever, waiting for events from the user, until the user exits the program.

→ PYTHON CODE -

```
from PIL import Image
from PIL import ImageTk
import tkinter as tk
im = Image.open('Sample_Image_Web.tiff')

res = im.convert("L")
root = tk.Tk()
root.title("Colorspace Conversion Demo")
photo = ImageTk.PhotoImage(res)
```

Figure 4 - Image Mode Conversion

```
l = tk.Label(root, image = photo)
l.pack()
l.photo = photo
root.mainloop()
```

(d) Image Resizing:

→ ALGORITHM -

① From PIL import Image, provides a class with the same name which is used to represent a PIL image.

② From PIL import ImageTk, contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL images.

③ Import tkinter as tk, standard Python interface to the Tcl/Tk GUI toolkit.

④ In a function "show_value", return a resized copy of the image and a Tkinter-compatible photo image.

⑤ Construct a toplevel Tk widget. Open and identify the given image file.

⑥ Widget that is used to implement display boxes where you can place text or images. Organize widgets in blocks before placing them in the parent widget.

⑦ Provide a graphical slider object that allows you to select values from a specific scale.

(i) label - You can display a label within the scale widget by setting this option to the label's text.

(ii) from_ - A float or integer value that defines one end of the scale's range.

(iii) to - A float or integer value that defines one end of the scale's range. Can be either greater than or less than the from_ value.

(iv) resolution

(v) command - A procedure to be called every time the slider is moved.

(vi) orient - Scale runs along the x dimension.

⑧ Pack widgets relative to the earlier widget and loop forever, waiting for events from the user, until the user exits the program.

Figure 5 - Image resizing

→ PYTHON CODE -

```
from PIL import Image
from PIL import ImageTk
import tkinter as tk

def show_value (size):
    print ('Resize: ', size)
    img = im.resize (int (size), int (size)))
    photo = ImageTk.PhotoImage (img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk ()
root.attributes ('-fullscreen', True)
im = Image.open ('Sample_Image_Web.tiff')
photo = ImageTk.PhotoImage (im)

l = tk.Label (root, image = photo)
l.pack ()
l.photo = photo

w = Tk.Scale (root,
              label = "Resize",
              from_ = 128,
              to = 512,
              resolution = 1,
              command = show_value,
              orient = tk.HORIZONTAL)
    )

w.pack ()
root.mainloop ()
```

(c) Image Rotation :

→ ALGORITHM -

① Import tkinter as tk, standard Python interface to the Tcl/Tk GUI toolkit

② From PIL import Image, provides a class with the same name which is used to represent a PIL image.

③ From PIL import ImageTk, contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL images.

④ In a function "show_value", return a rotated copy of the image and create a Tkinter-compatible photo image.

⑤ construct a toplevel Tk widget and a Tkinter-compatible photo image. Refer to the name provided to the window.

⑥ Open and identify the given image file. Widget that is used to implement display boxes where you can place text or images.

⑦ Organize widgets in blocks before placing them in the parent widget.

⑧ Provide a graphical slider object that allows you to select values from a specific value.

(i) label - You can display a label within the scale widget by setting this option to the label's text.

(ii) from_ - A float or integer value that defines one end of the scale's range.

(iii) to - A float or integer value that defines one end of the scale's range. Can be either greater than or less than the from_ value.

(iv) resolution

(v) command - A procedure to be called every time the slider is moved.

(vi) orient - Scale runs along the x dimension.

⑨ Packs widgets relative to the earlier widget and loop forever, waiting for events from the user, until the user exits the program.

PTO

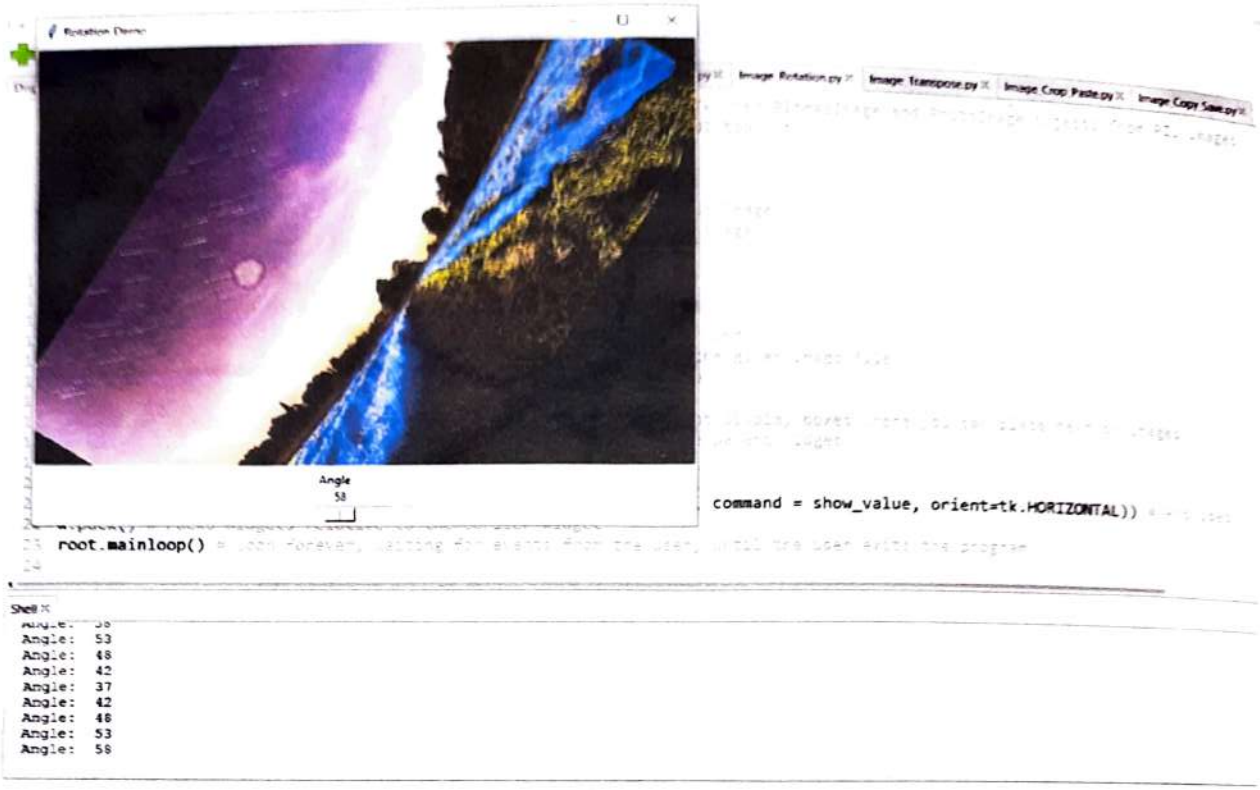


Figure 6 - Image Rotation

→ PYTHON CODE -

```
from PIL import Image
from PIL import ImageTk
import tkinter as tk

def show_value (angle):
    print ('Angle : ', angle)
    img = im.rotate (float (angle))
    photo = ImageTk.PhotoImage (img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk ()
root.title ('Rotation Demo')
im = Image.open ('Sample_Image_Web.tiff')
photo = ImageTk.PhotoImage (im)

l = tk.Label (root, image = photo)
l.pack ()
l.photo = photo

w = (tk.Scale (root,
               label = "Angle",
               from_ = 0,
               to = 360,
               resolution = 1,
               command = show_value,
               orient = tk.HORIZONTAL)
    )

w.pack ()
root.mainloop ()
```

(f) Image Transpose :

→ ALGORITHM -

① Import tkinter as tk, standard Python interface to the Tcl/Tk GUI toolkit.

② From PIL import Image, provides a class with the same name which is used to represent a PIL image.

③ From PIL import ImageTk, contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL images.

④ Construct a toplevel Tk widget. Refer to the name provided to the window. Open and identify the given image file.

⑤ Transpose image (flip or rotate in 90-degree steps).
PARAMETERS : method - one of
Transpose.FLIP_LEFT_RIGHT
Transpose.FLIP_TOP_BOTTOM
Transpose.ROTATE_90
Transpose.TRANSVERSE
Transpose.ROTATE_180
Transpose.ROTATE_270
Transpose.TRANSPOSE

⑥ Create a Tkinter-compatible photo image. Widget that is used to implement display boxes where you can place text or images.

⑦ Organizes widgets in blocks before placing them in the parent widget.

→ PYTHON CODE -

```
from PIL import Image
from PIL import ImageTk
import tkinter as tk

root = tk.Tk()
root.title('Transpose Demo')
im = Image.open('Sample_Image_Web.tiff')
out = im.transpose(Image.Transpose.ROTATE_180)

photo = ImageTk.PhotoImage(out)
l = tk.Label(root, image=photo)
l.pack()
l.photo = photo
```

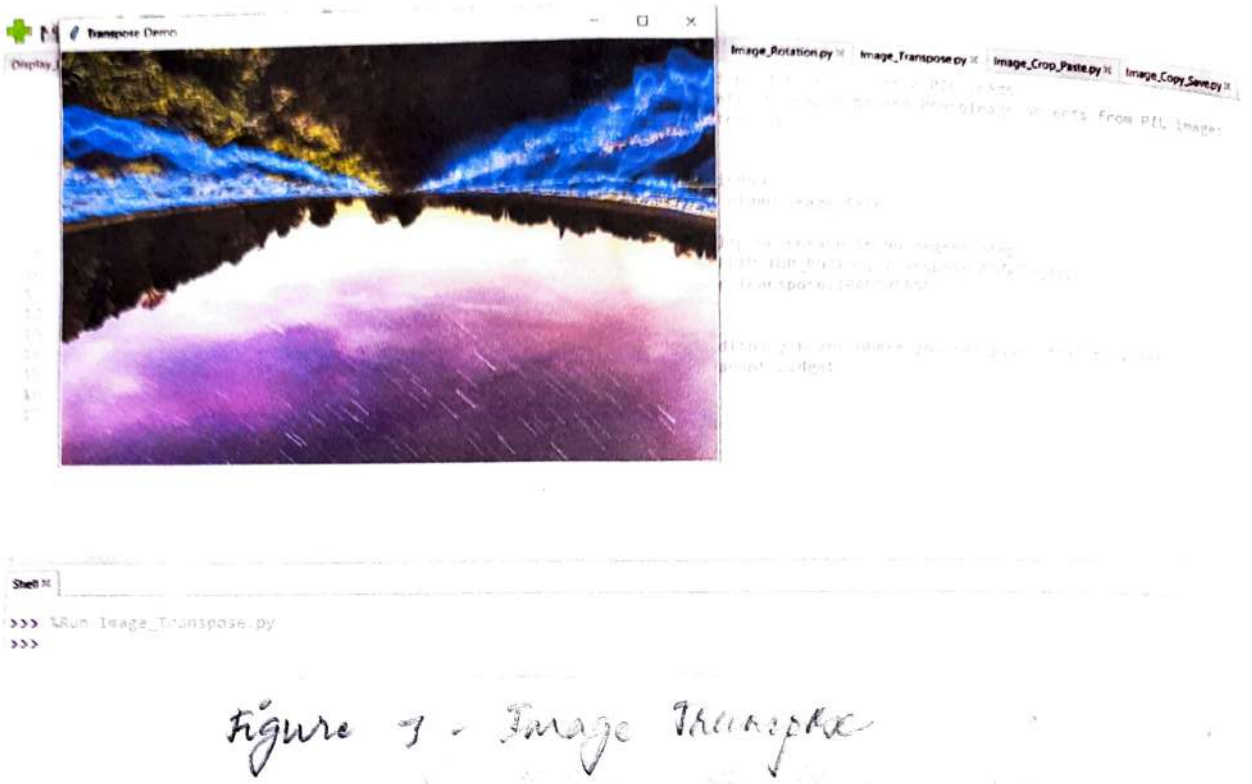


Figure 7 - Image Transpose

(9) Image Crop and Paste :

→ ALGORITHM -

① Import Image from PIL, provides a class with the same name which is used to represent a PIL image.

② Open and identify the given image file. Return a rectangular region from the image.

③ Transpose image (flip or rotate in 90-degree steps).

PARAMETERS : method - One of

Transpose.FLIP_LEFT_RIGHT
Transpose.FLIP_TOP_BOTTOM
Transpose.ROTATE_90
Transpose.ROTATE_180
Transpose.ROTATE_270
Transpose.TRANSPOSE
Transpose.TRANSVERSE

④ Paste another image into the image and diplay the image.

→ PYTHON CODE -

```
from PIL import Image
im = Image.open('Sample_Image_Web.tiff')
face_box = (100, 100, 300, 300)
face = im.crop(face_box)

rotated_face = face.transpose(Image.Transpose.ROTATE_180)
im.paste(rotated_face, face_box)
im.show()
```

(10) Image Copy and Save :

→ ALGORITHM -

① Import Image from PIL, provides a class with the same name which is used to represent a PIL image.

② Opens and identifies the given image file.

③ Copies the image.

④ Saves the image under the given filename.

PTO

Figure 8. Image Crop and Paste

→ PYTHON CODE -

```
from PIL import Image
im = Image.open('Sample_Image_web.tiff')
im_temp = im.copy()
im_temp.save("Image_Save.tiff")
```

(i) Assignment : To implement Gaussian blur, emboss, and image rotation while having their sliders in the left, right and down positions respectively with respect to the image.

→ ALGORITHM -

① From PIL import Image, provides a class with the same name which is used to represent a PIL image.

② From PIL import ImageTk, contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL images.

③ From PIL import ImageFilter, contains definitions for a pre-defined set of filters, which can be used with the Image.filter() method.

④ Import tkinter as tk, standard Python interfaces to the Tcl/Tk GUI toolkit.

⑤ Module blur_image :-
(i) Blurs the image with a sequence of extended box filters.
(ii) Filters the image using the given filter.
(iii) A Tkinter-compatible photo image.

⑥ Module rotate_image :-
(i) Return a rotated copy of the image
(ii) A Tkinter-compatible photo image

⑦ module emboss_image :-
(i) Apply emboss filter on the image
(ii) Filter the image using the given image
(iii) A Tkinter-compatible photo image

⑧ Construct a toplevel Tk widget and refer to the name provided to the window. Open and identify the given image file.

figure 9- Image copy and Save

(9) A Tkinter - compatible photo image and widget that is used to implement display boxes where you can place text or images.

(10) Organize widgets in a table-like structure in the parent widget :-
(i) column - The column to put widget in ; default 0 (leftmost column)
(ii) row - The row to put widget in ; default the first row that is still empty.
(iii) padx - How many pixels to pad widget, horizontally and vertically, outside v's borders
(iv) sticky - what to do if the cell is larger than widget. By default, with sticky = '', widget is centered in its cell.

(11) Provide a graphical slider object that allows you to select values from a specific scale. Loop forever, waiting for events from the user, until the user exits the program.

→ PYTHON CODE -

```
from PIL import Image
from PIL import ImageTk
from PIL import ImageFilter
import tkinter as tk

def blur_image (blur_radius):
    print ('Gaussian Blur Radius: ', blur_radius)
    custom_filter = ImageFilter.GaussianBlur (radius = float
(blur_radius))
    img = im.filter (custom_filter)
    photo = ImageTk.PhotoImage (img)
    l ['image'] = photo
    l.photo = photo
```

PTO


```
57 root.mainloop() # Loop forever, waiting for events from the user, until the user exits the program
58
```

Shell

```
>>> %Run 19CCE381_Expt_6_Assignment.py
  Gaussian Blur Radius:  1
  Gaussian Blur Radius:  2
```

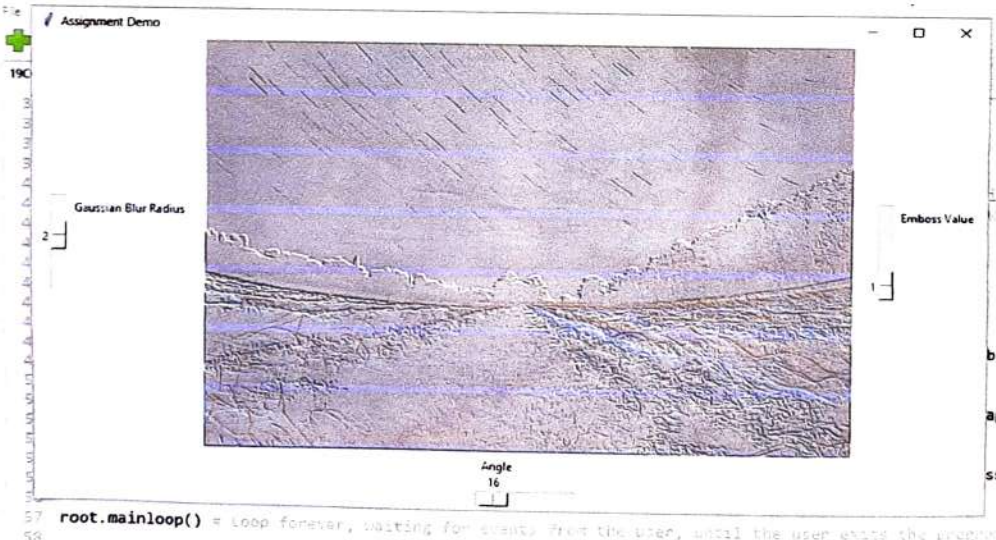
Figure 10 - Assignment : Gaussian Blur Radius

```
def rotate_image(angle):
    print('Angle: ', angle)
    img = im.rotate(float(angle))
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

def emboss_image(emboss_value):
    print('Emboss Value: ', emboss_value)
    custom_filter = ImageFilter.EMBOSS()
    img = im.filter(custom_filter)
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk()
root.title('Assignment Demo')
im = Image.open('Sample_Image_Web.tiff')
photo = ImageTk.PhotoImage(im)

l = tk.Label(root, image = photo)
l.grid(column=50, row=0, padx=5, sticky='n')
l.photo = photo

image_blur = (tk.Scale(root, label="Gaussian Blur Radius",
from_=0, to=5, resolution=1, command=blur_image,
orient=tk.VERTICAL))
image_blur.grid(column=0, row=0, padx=5, sticky='W')

rotate_image = (tk.Scale(root, label="Angle", from_=0, to=360,
resolution=1, command=rotate_image, orient=tk.HORIZONTAL))
rotate_image.grid(column=50, row=10, padx=5, sticky='s')
```


```
root.mainloop()
```

```
Shell
>>>
Gaussian Blur Radius:  1
Gaussian Blur Radius:  2
Angle:  5
Angle:  11
Angle:  16
```

Figure 11 - Assignment : Angle




```
root.mainloop() # Loop forever, waiting for events from the user, until the user exits the program
```

```
Shell
Gaussian Blur Radius:  1
Gaussian Blur Radius:  2
Angle:  5
Angle:  11
Angle:  16
Emboss Value:  1
Emboss Value:  0
Emboss Value:  1
```

Figure 12 - Assignment : Emboss Value

```
image_emboss = tk.Scale(root, label="Emboss Value", from_=0,
to=1, resolution=1, command=emboss_image, orient=tk.VERTICAL))
image_emboss.grid(column=60, row=0, padx=5, sticky='e')

root.mainloop()
```

* **RESULT:**

Thus, introduced basic image processing techniques using Raspberry Pi. All the simulation results were verified successfully.

R/L

EXPERIMENT NUMBER : 7

EXPERIMENT NAME : PI CAMERA INTERFACING WITH RASPBERRY PI

DATE : 28/11/2022, MONDAY

8/10

★ AIM :

To interface pi camera with Raspberry Pi and implement image processing techniques.

★ INTEGRATED DEVELOPMENT ENVIRONMENT (IDE) :

Name - Thonny 4.0.1

Publisher - Aivar Annamaa

Support link - https://thonny.org

★ IMPORT NECESSARY LIBRARIES :

① Import picamera, provides a pure Python interface to the Raspberry Pi camera module.

② From picamera import Color, the color class is a tuple which represents a color as red, green, and blue components.

③ Import time, provides various time-related functions.

④ From http import server, defines classes for implementing HTTP servers.

⑤ Import io, provides Python's main facilities for dealing with various types of I/O.

⑥ Import logging, a means of tracking events that happen when some software runs.

⑦ Import socketserver, simplifies the task of writing network servers.

⑧ From threading import Condition, constructs higher-level threading interfaces on top of the lower level _thread module.

⑨ Import tkinter as tk, standard Python interface to the Tcl/Tk GUI toolkit.

PTO

(iii) From PIL import -
(i) Image, provides a class with the same name which is used to represent a PIL image.
(ii) ImageTk, contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL objects.
(iii) ImageFilter, contains definitions for a pre-defined set of filters, which can be used with the Image.filter() method.

(i) Test Raspberry Pi camera :

→ ALGORITHM -

1. Display the preview overlay.
2. Suspend execution for a given number of seconds.
3. Hide the preview overlay.
4. Finalize the state of the camera.

→ PYTHON CODE -

```
import picamera
import time
picam = picamera.PiCamera()

picam.start_preview()
time.sleep(30)
picam.stop_preview()
picam.close()
```

(ii) Capture Image :

→ ALGORITHM -

1. Retrieve or set the current rotation of the camera's image.
2. Retrieve or set the opacity of the renderer.
3. Suspend execution for the given number of seconds.
4. Capture an image from the camera, storing it in output.
5. Hide the preview overlay.
6. Finalize the state of the camera.

Figure 1 - Capture Image

→ PYTHON CODE -

```
import picamera
import time
picam = picamera.PiCamera()

picam.rotation = 180
picam.start_preview(alpha=200)
time.sleep(5)

picam.capture("Rotated_Image.png")
picam.stop_preview()
picam.close()
```

(iii) Display Text:

→ ALGORITHM -

① Display the preview overlay.
② Retrieve or set a text annotation for all output.
③ Control the size of the annotation text.
④ Control what background is drawn behind the annotation.
⑤ Suspend execution for the given number of seconds.
⑥ Capture an image from the camera, storing it in output.
⑦ Hide the preview overlay and finalize the state of the camera.

→ PYTHON CODE -

```
import picamera
from picamera import Color
import time
picam = picamera.PiCamera()
picam.start_preview()

picam.annotate_text = "Hello From Raspberry Pi!"
picam.annotate_text_size = 60
picam.annotate_background = Color('blue')
picam.annotate_foreground = Color('yellow')
```

Figure 2 - Display Test

```
time.sleep(5)
picam.capture('Text_Image.png')
picam.stop_preview()
picam.close()
```

(iv) Brightness control:

→ ALGORITHM -

① Display the preview overlay.
② Retrieves or sets a text annotation for all output.
③ Retrieve or set the contrast setting of the camera.
④ Suspend execution for the given number of seconds.
⑤ Hide the preview overlay and finalize the state of the camera.

→ PYTHON CODE -

```
import picamera
from picamera import Color
import time
picam = picamera.PiCamera()
picam.start_preview()

for i in range(-100, 100):
    picam.annotate_text = "Contrast: %s" % i
    picam.contrast = i
    time.sleep(01)

picam.stop_preview()
picam.close()
```

(iv) Image Effects:

→ ALGORITHM -

① Retrieve or set the opacity of the renderer.
② Retrieve or set the exposure mode of the camera.
③ Retrieve or set a text annotation for all output.

Figure 3 - Brightness control

④ Suspend execution for the given number of second(s).
⑤ Hide the preview overlay and finalize the state of the camera.

→ PYTHON CODE -

```
import picamera
from picamera import Color
import time
picam = picamera.PiCamera()
picam.start_preview(alpha = 200)

for effect in picam.EXPOSURE_MODES: # exposure_mode, awb_mode,
    picam.exposure_mode = effect      image_effect, ....
    picam.annotate_text = "Effect: %s" % effect
    time.sleep(5)

picam.stop_preview()
picam.close()
```

(iv) Web streaming :

→ ALGORITHM -

① Write the HTML code for web streaming.
② Define classes for the following functionalities -
(i) StreamingOutput :- initialization & write
(ii) StreamingHandler :- GET for index.html and stream.mjpg paths
(iii) StreamingServer (socketserver.ThreadingMixIn, server.HTTPServer)
③ Retrieve or set the framerate at which video-port based image captures, video recordings, and previews will run.
④ Retrieve or set the current rotation of the camera's image.
⑤ Start recording video from the camera, storing it in output.
⑥ Stop recording video from the camera.

PTO

→ PYTHON CODE -

```
from http import server
import io
import logging
import picamera
import socketserver
from threading import Condition

PAGE = """\
<html>
<head>
<title> Raspberry Pi - Surveillance Camera </title>
</head>
<body>
<center> <h1> Raspberry Pi - Surveillance Camera </h1> </center>
<center> <img src = "stream.mjpg" width = "640" height = "480"> </center>
</body>
</html>
"""


class StreamingOutput(object):
    def __init__(self):
        self.frame = None
        self.buffer = io.BytesIO()
        self.condition = Condition()

    def write(self, buf):
        if buf.startswith(b'\xff\xd8'):
            # New frame, copy the existing buffer's content and
notify all clients its available
            self.buffer.truncate()
```

```
        with self.condition:
            self.frame = self.buffer.getvalue()
            self.condition.notify_all()
        self.buffer.seek(0)
    return self.buffer.write(buf)

class StreamingHandler(server.BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/':
            self.send_response(301)
            self.send_header('Location', '/index.html')
            self.end_headers()
        elif self.path == '/index.html':
            content = PAGE.encode('utf-8')
            self.send_response(200)
            self.send_header('Content-Type', 'text/html')
            self.send_header('Content-Length', len(content))
            self.end_headers()
            self.wfile.write(content)
        elif self.path == '/stream.mjpg':
            self.send_response(200)
            self.send_header('Age', 0)
            self.send_header('Cache-Control', 'no-cache, private')
            self.send_header('Pragma', 'no-cache')
            self.send_header('Content-Type', 'multipart/x-mixed-
replace; boundary=FRAME')
            self.end_headers()
            try:
                while True:
                    with output.condition:
                        output.condition.wait()
                        frame = output.frame
```

```
                self.wfile.write(b'--FRAME\r\n')
                self.send_header('Content-Type', 'image/jpeg')
                self.send_header('Content-Length', len(frame))
                self.end_headers()
                self.wfile.write(frame)
                self.wfile.write(b'\r\n')
            except Exception as e:
                logging.warning(
                    'Removed streaming client %s: %s',
                    self.client_address, str(e))
        else:
            self.send_error(404)
            self.end_headers()

class StreamingServer(socketserver.ThreadingMixIn, server.HTTPServer):
    allow_reuse_address = True
    daemon_threads = True

with picamera.PiCamera(resolution='640x480', framerate=24) as camera:
    output = StreamingOutput()
    #Uncomment the next line to change your Pi's camera rotation (in degrees)
    camera.rotation = 180
    camera.start_recording(output, format='mjpeg')
    try:
        address = ('', 8000)
        server = StreamingServer(address, StreamingHandler)
        server.serve_forever()
    finally:
        camera.stop_recording()
```

(vii) Assignment:
To capture photo using Raspberry Pi and modify angle, blur and emboss the captured image using sliders.

→ ALGORITHM:-

① Module blur_image (blur_radius):
(i) Blurs the image with a sequence of extended box filters
(ii) Filters the image using the given filter
(iii) A Tkinter-compatible photo image

② Module rotate_image (angle):
(i) Returns a copy rotated of the image
(ii) A Tkinter-compatible photo image

③ Module emboss_image (emboss_value):
(i) Apply emboss filter on to the image
(ii) Filters the image using the given filter
(iii) A Tkinter-compatible photo image

④ Retrieve or set the current rotation of the camera's image and the opacity of the renderer. Suspend execution for the given number of seconds.

⑤ Capture an image from the camera, storing it in output. Hide the preview overlay and finalize the state of the camera.

⑥ Construct a toplevel Tk widget and refer to the name provided to the window. Open and identify the given image file.

⑦ Create a Tkinter-compatible photo image and widget that is used to implement display boxes where you can place text or images.

⑧ Organize widgets in a table-like structure in the parent widget and provide a graphical slider object that allows you to select values from a specific scale.

⑨ Loop forever, waiting for events from the user, until the user exits the program.

P.T.O

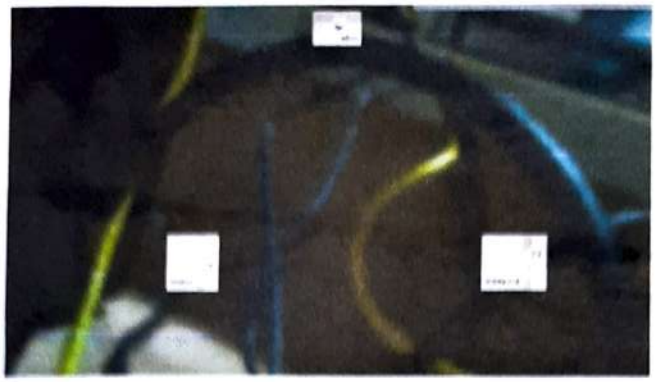


Figure 4. Blur Image

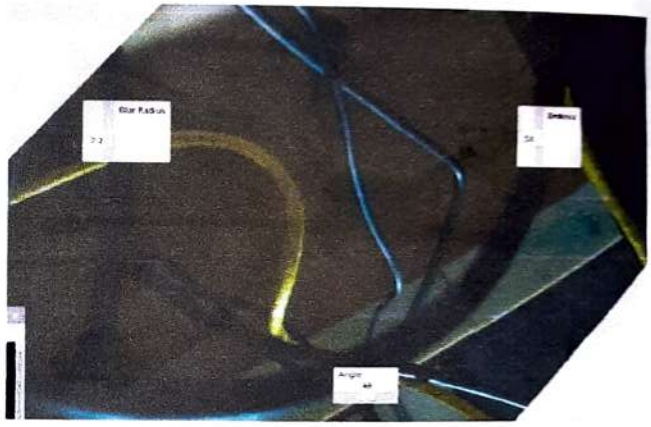


Figure 5. Rotate Image

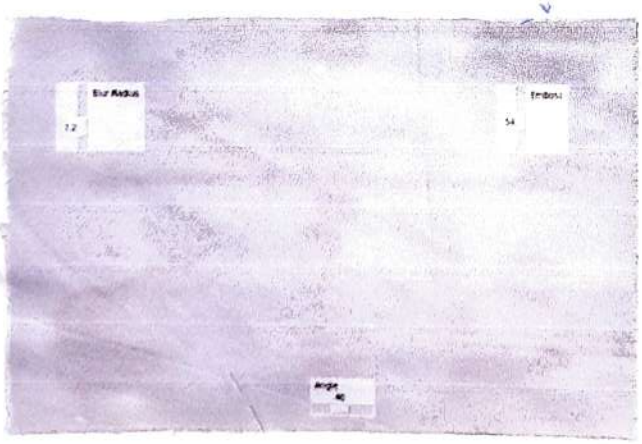


Figure 6 - Emboss Image

→ PYTHON CODE:

```
import picamera
from picamera import color
import time

from PIL import Image
from PIL import ImageTk
from PIL import ImageFilter
import tkinter as tk

def blur_image(blur_radius):
    print('Gaussian Blur Radius: ', blur_radius)
    custom_filter = ImageFilter.GaussianBlur(radius=float
(blur_radius))
    img = im.filter(custom_filter)
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

def rotate_image(angle):
    print('Angle: ', angle)
    img = im.rotate(float(angle))
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

def emboss_image(emboss_value):
    print('Emboss Value: ', emboss_value)
    custom_filter = ImageFilter.EMBOSS()
    img = im.filter(custom_filter)
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo
```

```
picam = picamera.PiCamera()
picam.rotation = 180
picam.start_preview(alpha=200)
time.sleep(5)

picam.capture("Rotated_Image.png", resize=(300, 450))
picam.stop_preview()
picam.close()

root = tk.Tk()
root.title('Assignment Demo')
im = Image.open('Rotated_Image_Assignment.tiff')
photo = ImageTk.PhotoImage(im)

l = tk.Label(root, image=photo)
l.grid(column=50, row=0, padx=5, sticky='n')
l.photo = photo

image_blur = (tk.Scale(root, label="Gaussian Blur Radius",
from_=0, to=5, resolution=1, command=blur_image,
orient=tk.VERTICAL))
image_blur.grid(column=0, row=0, padx=5, sticky='w')

rotate_image = (tk.Scale(root, label="Angle", from_=0, to=360,
resolution=1, command=rotate_image, orient=tk.HORIZONTAL))
rotate_image.grid(column=0, row=10, padx=5, sticky='s')

image_emboss = (tk.Scale(root, label="Emboss Value", from_=0,
to=1, resolution=1, command=emboss_image, orient=tk.VERTICAL))
image_emboss.grid(column=60, row=0, padx=5, sticky='e')

root.mainloop()
```

* **RESULT:**

Thus, interfaced Pi camera with Raspberry Pi and implemented various image processing techniques. All the simulation results were verified successfully.

10/10
15/10

EXPERIMENT NUMBER : 8

EXPERIMENT NAME : GPIO AND ASSOCIATED PERIPHERALS INTERFACING

DATE : 10/12/2022, SATURDAY

9/10

* AIM :

To interface General - Purpose Input - Output (GPIO) and associated peripherals.

* INTEGRATED DEVELOPMENT ENVIRONMENT (IDE) :

Name - Thonny 4.0.1

Publisher - Aivar Annamaa

Support link - https://thonny.org

* IMPORT NECESSARY LIBRARIES :

① Import picamera, provides a pure Python interface to the Raspberry Pi camera module.

② Import RPi.GPIO as GPIO, module to control the GPIO on a Raspberry Pi.

③ Import time, provides various time-related functions.

④ Disable warnings

(a) LED Blinking Using Raspberry Pi

→ ALGORITHM -

① Select a pin to be used for connecting LED.

② Physical pin number numbering scheme is followed; for GPIO numbering, choose BCM.

③ Selected pin is configured as output.

④ Turn the LED ON; sleep for 1 second.

⑤ Turn the LED OFF; sleep for 1 second.

⑥ Clear GPIO pins.

PTO

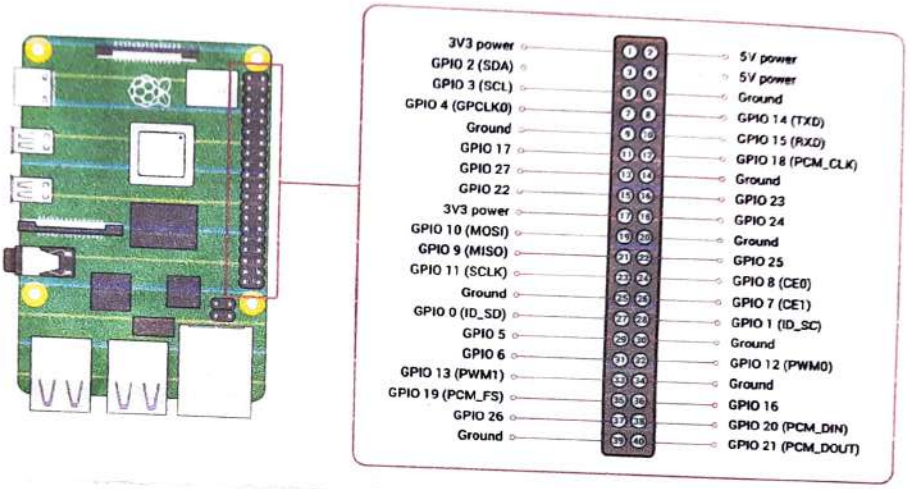


Figure 1 - Raspberry Pi Pin Diagram

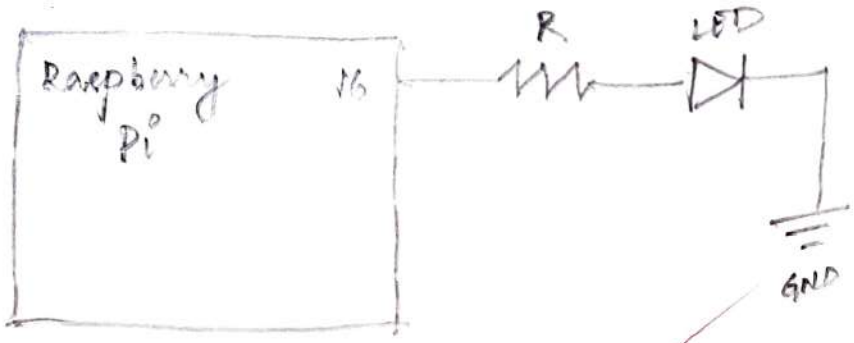


Figure 2 - LED Blinking Using Raspberry Pi

→ PYTHON CODE -

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)

# ledpin = 36
ledpin = 16

# GPIO.setmode(GPIO.BOARD)
GPIO.setmode(GPIO.BCM)

GPIO.setup(ledpin, GPIO.OUT)

while True:
        GPIO.output(ledpin, True)
        time.sleep(1)
        GPIO.output(ledpin, False)
        time.sleep(1)

GPIO.cleanup()
```

(b) **LED Control Using Switch:**

→ ALGORITHM -

① Physical pin number numbering scheme is followed.
② Selected pin is configured as output.
③ Enable Pull.
④ Read status of pin/port and assign to variable switch.
- Turn the LED ON.
- Turn the LED OFF.
⑤ Clear GPIO pins.

P.T.O.

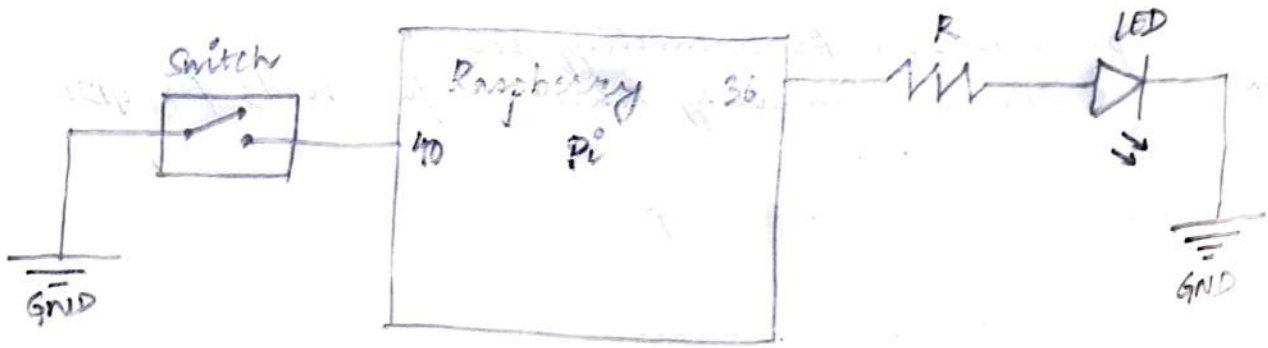


Figure 3 - LED control using Switch

→ PYTHON CODE -

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)

ledpin = 36
switch = 40

GPIO.setmode(GPIO.BOARD)
GPIO.setup(ledpin, GPIO.OUT)
GPIO.setup(switch, GPIO.IN, pull_up_down = GPIO.PUD_UP)

while True:
    status = GPIO.input(switch)
    if (status):
        GPIO.output(ledpin, True)
        print("LED ON: Button Pressed!")
    else:
        GPIO.output(ledpin, False)
        print("Button NOT Pressed!")

GPIO.cleanup()
```

(C) LED Fade In and Fade Out using PWM:

→ ALGORITHM -

① Physical pin number numbering scheme is followed and selected pin is configured as output.

② Select pin number and frequency in Hertz. Start PWM signal generation with 0% duty cycle.

③ Increase & Decrease duty cycle from 0 to 100 and 100 to 0 consecutively.
(i) Change duty cycle.
(ii) Suspend execution for a given number of seconds.

④ For exception handling, turn PWM off and reset GPIO pins.

→ PYTHON CODE -

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)

ledpin = 32
GPIO.setmode(GPIO.BOARD)
GPIO.setup(ledpin, GPIO.OUT)

dimmer = GPIO.PWM(ledpin, 50)
dimmer.start(0)
try:
    while(True):

        for i in range(0,100,5):
            dimmer.ChangeDutyCycle(i)
            time.sleep(0.2)

        for i in range(100,0,-5)
            dimmer.ChangeDutyCycle(i)
            time.sleep(0.2)

except KeyboardInterrupt:
        dimmer.stop()
        GPIO.cleanup()
```

(9) Direction Control of Servo Motor using PWM:

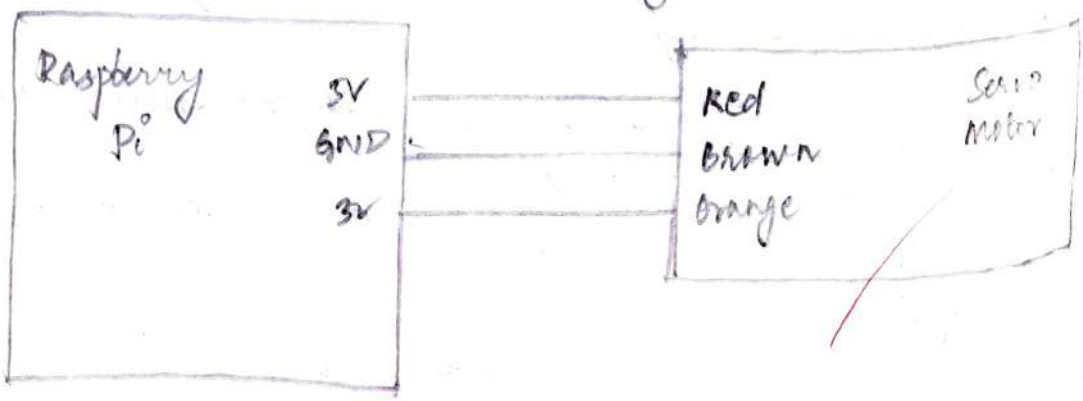
→ ALGORITHM -

① Pin to be connected to motor: GPIO 18. The selected pin is configured as output. Select pin number and frequency in Hertz.

② Start PWM signal generation with 0% duty cycle. Increase duty cycle from 0 to 100 and change duty cycle. Suspend execution for a given number of seconds.

③ For exceptions handling, turn PWM off and reset GPIO pins.

Figure 4 - Direction control of Servo motor Using PWM



→ PYTHON CODE -

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)

motorpin = 12
GPIO.setup(motorpin, GPIO.OUT)
servo = GPIO.PWM(motorpin, 50)
servo.start(0)

try:
  while(TRUE):
    for i in range(0, 100, 5):
      servo.ChangeDutyCycle(i)
      time.sleep(0.2)

except KeyboardInterrupt:
  print("Motor stopped!")
  servo.stop()
  GPIO.cleanup()
```

(e) Assignment:

Capture image when button is pressed.
Save the image as well.

→ ALGORITHM -

① Physical pin number numbering scheme is followed and pull is enabled.

② Read status of pin/port and assign to variable switch.

③ Retrieves or sets the current rotation of the camera's image and set the opacity of the renderer.

④ suspends execution for the given number of seconds. Capture an image from the camera, storing it in output.

⑤ Hide the preview overlay and finalize the state of the camera. Clear GPIO pins.

→ PYTHON CODE -

```
import picamera
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)

switch = 40
GPIO.setmode(GPIO.BOARD)
GPIO.setup(switch, GPIO.IN, pull_up_down = GPIO.PUD_UP)

while True:
    status = GPIO.input(switch)
    if (status):

        picam = picamera.PiCamera()
        picam.rotation = 180
        picam.start_preview(alpha = 200)
        time.sleep(5)

        picam.capture("Rotated_Image.png")
        picam.stop_preview()
        picam.close()

GPIO.cleanup()
```

* RESULT:

Thus GPIO and other peripherals interfacing demonstrated. All the simulation results were verified successfully.

Figure 5 - Assignment output

EXPERIMENT NUMBER - 9

EXPERIMENT NAME - LCD INTERFACING USING RASPBERRY PI

DATE - 22/12/2022, THURSDAY

---

* AIM:

To interface LCD module with Raspberry Pi.

* INTEGRATED DEVELOPMENT ENVIRONMENT: (IDE)

Name - Thonny 4.0.1

Publisher - Aivar Annamaa

Support link - https://thonny.org

* IMPORT NECESSARY LIBRARIES:

(sudo pip install adafruit-circuitpython-charlcd)

① Import time, provides various time-related functions.

② Import board, implements a general-purpose board structure which has the functionality needed for a range of purposes.

③ Import digitalio, contains classes to provide access to basic digital IO.

④ Import adafruit_character_lcd.character_lcd as characterlcd, module for interfacing with monochromatic character LCDs.

(a) Hello World Display as LCD (GPIO):

→ Algorithm -

① Metro NO/MA Pin config.

② Initialise the LCD class.

③ Turn backlight ON.

④ Print a two line message.

⑤ Wait for some time.

⑥ Clear the LCD.

→ PYTHON CODE -

```
import time
import board
import digitalio
import adafruit_character_lcd.character_lcd as characterlcd

# Modify this if you have a different sized character LCD :-
lcd_columns = 16 ; lcd_rows = 2

lcd_rs = digitalio.DigitalInOut(board.D7)
lcd_en = digitalio.DigitalInOut(board.D8)
lcd_d4 = digitalio.DigitalInOut(board.D9)
lcd_d5 = digitalio.DigitalInOut(board.D10)
lcd_d6 = digitalio.DigitalInOut(board.D11)
lcd_d7 = digitalio.DigitalInOut(board.D12)
lcd_backlight = digitalio.DigitalInOut(board.D13)

lcd = characterlcd.character_LCD_Mono(lcd_rs, lcd_en, lcd_d4, lcd_d5,
lcd_d6, lcd_d7, lcd_columns, lcd_rows, lcd_backlight)
lcd.backlight = True
lcd.message = "Hello In CircuitPython"
time.sleep(5)
lcd.clear()
```

(b) Display Scrolling Text on LCD :

→ ALGORITHM -

① Metro M0/M4 Pin Config.
② Initialize the LCD class.
③ Turn backlight ON.
④ Print a two line message.
⑤ Wait for some time.
⑥ Clear the LCD.
⑦ Turn backlight off.

→ FUNCTIONALITIES -

① Print message right to left.
② Print message left to right.
③ True if cursor is visible.
④ True to blink the cursor.
⑤ Create message to scroll.
- Scroll message to left
- Move displayed text left one column.

→ PYTHON CODE -

```
import time
import board
import digitalio
import adafruit_character_lcd.character_lcd as characterlcd

# Modify this if you have a different sized character LCD:-
lcd_columns = 16; lcd_rows = 2

lcd_rs = digitalio.DigitalInOut(board.D7)
lcd_en = digitalio.DigitalInOut(board.D8)
lcd_d4 = digitalio.DigitalInOut(board.D9)
lcd_d5 = digitalio.DigitalInOut(board.D10)
lcd_d6 = digitalio.DigitalInOut(board.D11)
lcd_d7 = digitalio.DigitalInOut(board.D12)
lcd_backlight = digitalio.DigitalInOut(board.D13)

lcd = characterlcd.character_LCD_Mono(lcd_rs, lcd_en, lcd_d4, lcd_d5,
lcd_d6, lcd_d7, lcd_columns, lcd_rows, lcd_backlight)
lcd.backlight = True
lcd.message = "Hello\n CircuitPython"
time.sleep(5)
lcd.clear()

lcd.text_direction = lcd.RIGHT_TO_LEFT
lcd.message = "Hello\n CircuitPython"
time.sleep(5)
lcd.clear()

lcd.text_direction = lcd.LEFT_TO_RIGHT
lcd.message = "Hello\n CircuitPython"
time.sleep(5)
lcd.clear()
```

```
lcd.cursor = True
lcd.message = "Cursor!"
time.sleep(5)
lcd.clear()

lcd.blink = True
lcd.message = "Blinky Cursor!"
time.sleep(5)
lcd.blink = False
lcd.clear()

scroll_msg = "<-- Scroll"
lcd.message = scroll_msg

for i in range(len(scroll_msg)):
    time.sleep(0.5)
    lcd.move_left()
lcd.clear()

lcd.message = "Going to sleep\n CU ya later!"
time.sleep(3)
lcd.backlight = False
time.sleep(2)
```

(1) Assignment - There is a digital board installed in a smart restaurant. Read in numbers of today's special food items as input through keyboard. Scroll through the read foods one by one on LCD for customer's view.

→ ALGORITHM -

① Metro M0/M4 Pin config.
② Initialise the LCD class.
③ Turn backlight ON.
④ Print a message.
⑤ Wait for some time.
⑥ Scroll message to the left.
- wait for half a second.
- move displayed text left one column.

→ Python Code -

```
import time
import board
import digitalio
import adafruit_character_lcd.character_lcd as characterlcd

# Modify this if you have a different sized character LCD :-
lcd_columns = 16 ; lcd_rows = 2

lcd_rs = digitalio.DigitalInOut(board.D7)
lcd_en = digitalio.DigitalInOut(board.D8)
lcd_d4 = digitalio.DigitalInOut(board.D9)
lcd_d5 = digitalio.DigitalInOut(board.D10)
lcd_d6 = digitalio.DigitalInOut(board.D11)
lcd_d7 = digitalio.DigitalInOut(board.D12)
lcd_backlight = digitalio.DigitalInOut(board.D13)

lcd = characterlcd.character_LCD_Mono(lcd_rs, lcd_en, lcd_d4,
lcd_d5, lcd_d6, lcd_d7, lcd_columns, lcd_rows, lcd_backlight)
lcd.backlight = True
lcd.message = "Welcome!"
time.sleep(5)
lcd.clear()

n = int(input("Enter n value : "))
food_items = []

for i in range(n):

    item = str(input("Enter food item : "))
    food_items.append(item)
    time.sleep(1)
    print(food_items[i])
```

```
    # Scroll message to the left :-
    for j in range (len (food. items [i])) :
        time. sleep (0.5)
        lcd. move_ left ()
    lcd. clear ()

time. sleep (3)
lcd. backlight = False
time. sleep (2)
```

* **RESULT :**

Thus, interfaced LCD module with Raspberry Pi. All the simulation results were verified successfully.

Circuit diagram: