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 server.

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.

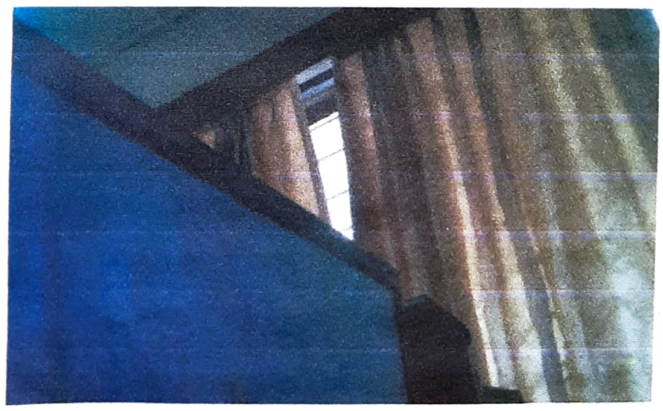

Figure 1 - Capture Image

→ 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()
```

(v) 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(2)

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.

→ PTO

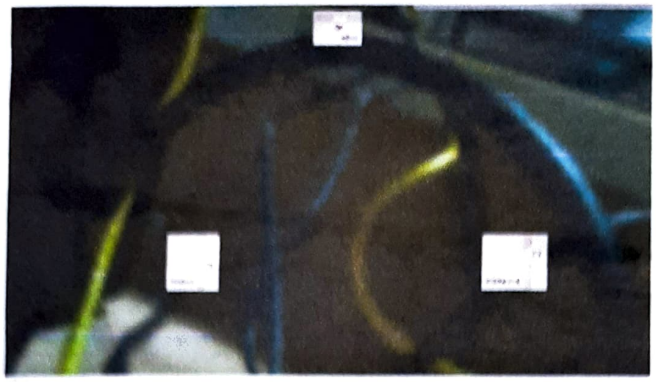

Figure 4 - Blur Image

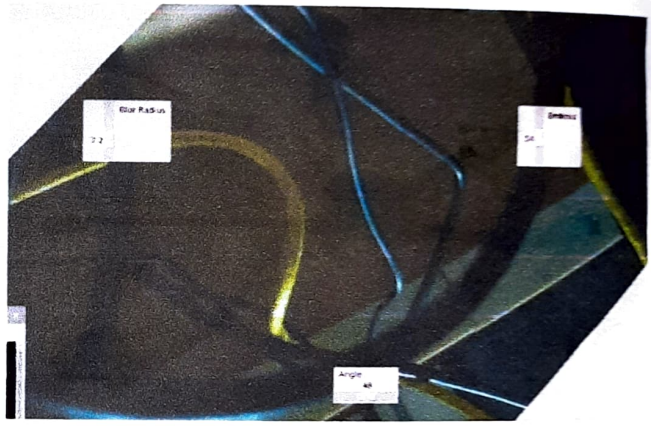

Figure 5 - Rotate Image

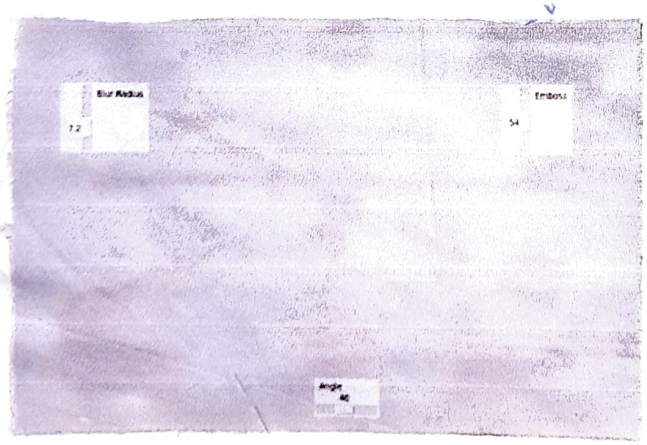

Figure 6 - Emboss Image

→ 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.