Transactions on Computer Systems and Networks

Jivan S. Parab · Madhusudan Ganuji Lanjewar · Marlon Darius Sequeira · Gourish Naik · Arman Yusuf Shaikh

Python Programming Recipes for IoT Applications

Springer

Transactions on Computer Systems and Networks

Series Editor

Amlan Chakrabarti, Director and Professor, A. K. Choudhury School of Information Technology, Kolkata, West Bengal, India

Jivan S. Parab · Madhusudan Ganuji Lanjewar ·
Marlon Darius Sequeira · Gourish Naik ·
Arman Yusuf Shaikh

Python Programming Recipes for IoT Applications

Springer

Jivan S. Parab
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

Madhusudan Ganuji Lanjewar
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

Marlon Darius Sequeira
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

Gourish Naik
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

Arman Yusuf Shaikh
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

ISSN 2730-7484 ISSN 2730-7492 (electronic)
Transactions on Computer Systems and Networks
ISBN 978-981-19-9465-4 ISBN 978-981-19-9466-1 (eBook)
https://doi.org/10.1007/978-981-19-9466-1

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Contents

Chapter 3
Simple Applications with Raspberry Pi

Abstract All over the world, people use the Raspberry Pi to learn the coding skills, build simple projects, implement simple clusters and a little bit of Edge computing. Raspberry Pi is considered as a low-cost pocket-sized minicomputer used in various applications. In this chapter, we have focused on hands on implementation of few simple applications such as, Interfacing LEDs, Organic LED (OLED) display, Camera Interfacing to capture video/pictures, different types of motor control and lastly implementation of Bluetooth with mobile .To implement these examples, we have used Thonny Python IDE which is already integrated with Raspbian OS.

Keywords LED · PWM · Camera interfacing · Motor control · Bluetooth

3.1 Blinking of LED

To interface the LEDs with the Raspberry Pi through General-Purpose Input/Output (GPIO) pins, we need to understand how to access the Raspberry Pi GPIO.

Raspberry Pi GPIO Access:

GPIO pins of Raspberry Pi can be used to interface with the general-purpose input/output (I/O) devices. Raspberry Pi 3 B+ has on-board 26 programmable GPIO pins to interface and control many I/O devices. It can connect to the internet using on-board WiFi or WiFi USB adapter and also some pins are multiplexed as I2C, SPI, UART, etc. There are few following options available to specify GPIO pins of Raspberry Pi as shown in Fig. 3.1.

i. Physical (BOARD): Pin number from 1 to 40 corresponds to the physical location on the header.
ii. Broadcom (BCM): It is generally called “GPIO” (GPIO1–GPIO26) or RPi.GPIO
iii. WiringPi—Fig. 3.1b shows the pin numbering in Wiring Pi

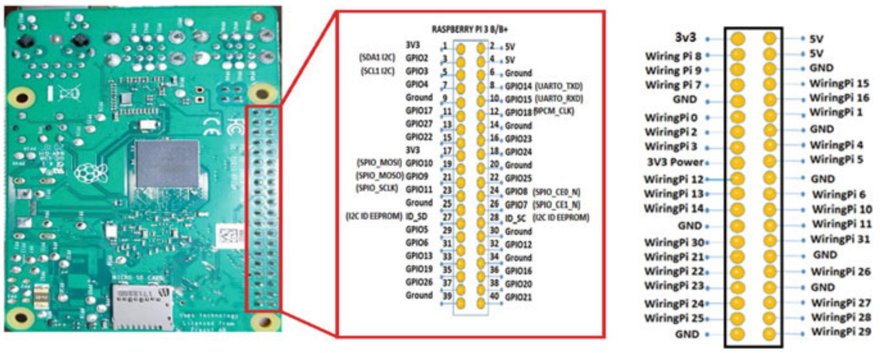


Fig. 3.1 GPIO pins of physical (BOARD) and BCM. **a** Actual backside physical pin assignments and **b** WiringPi

Raspberry Pi 3 Model B+ GPIO Pins Numbering

For accessing the GPIO pin for I/O interface, Raspberry Pi has different ways of defining pin assignments from which normally two types are used, namely Physical and BCM. In GPIO Numbering (BCM), pin number refers to the number on Broadcom SoC (System on Chip), while, in Physical Numbering (BOARD), pin number refers to the pin of 40-pin P1 header on Raspberry Pi Board. The above physical numbering is simple as we can count pin number on P1 header and assign it as GPIO. To access GPIO through Python programming, first, take a simple example of how to blink the LED through Raspberry GPIO.

Installation of RPi.GPIO Python Library

The configuration of the I/O pins for read and write can be done using the 'RPi.GPIO' Python library. There are two simple methods to install the GPIO library.

Method 1: Installation of GPIO from Repository

Step 1: Open the Raspberry Pi console and update the available package versions by using the following command:

```
  sudo apt-get update
```

If the package exists, then there is no need to install; otherwise, it can be installed using 'apt-get'.

Step 2: Install the RPi.GPIO package by using the following command:

```
sudo apt-get install rpi.gpio
```

If it is already installed, it will be upgraded to newer version.

Method 2: Manual Installation

The package can be downloaded from http://pypi.python.org/pypi/RPi.GPIO

Step 1: Download the library by using the following command:

```
wget    https://pypi.python.org/packages/source/R/RPi.GPIO/RPi.
GPIO-.5.11.tar.gz
```

Step 2: The downloaded file is in '.tar' format, so extract the archive to a new folder by using the following command:

```
tar -xvf RPi.GPIO-0.5.11.tar.gz
```

Step 3: Navigate to the new directory.

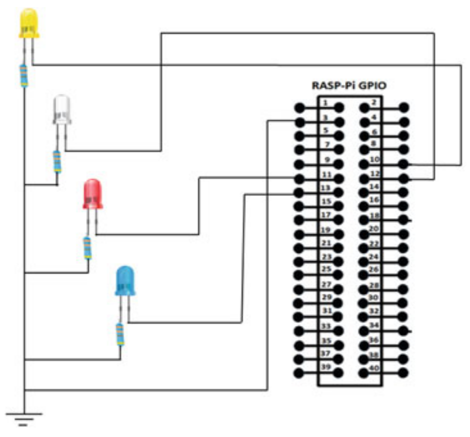
```
cd RPi.GPIO-0.5.11
```

Step 4: RPi.GPIO is installed using the following command:

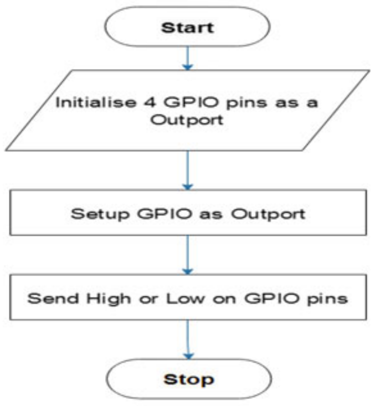
```
sudo python setup.py installs
```

Python Program to Blink LEDs

In this example, the four LEDs are interfaced to GPIOs of Raspberry Pi. Here, one requires four LEDs, four 330 Ω resistors, and a Raspberry Pi board. Every LED has two leads—one cathode (shorter lead) and one anode (longer lead). Choose the cathode and use a 330-Ω resistor to ground it (Pin 6). The other end goes to pins 10, 11, 12, and 13, respectively. Do the connection as shown in Fig. 3.2a and the flow diagram of the entire implementation is given in Fig. 3.2b. Type the following code in Thonny IDE and execute the same.



a)


Start
Initialise 4 GPIO pins as a Outport
Setup GPIO as Outport
Send High or Low on GPIO pins
Stop

b)

Fig. 3.2 **a** Interfacing of four LEDs with GPIO of Raspberry Pi and **b** flowchart for LED blinking

```
# To blink 4 -LED

import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BOARD)
pin_1=10
pin_2=11
pin_3=12
pin_4=13
GPIO.setup(pin_1,GPIO.OUT)
GPIO.setup(pin_2,GPIO.OUT)
GPIO.setup(pin_3,GPIO.OUT)
GPIO.setup(pin_4,GPIO.OUT)
GPIO.output(pin_1, HIGH)
sleep(2)
GPIO.output(pin_2, LOW)
sleep(2)
GPIO.output(pin_3, HIGH)
sleep(2)
GPIO.output(pin_4, LOW)
GPIO.cleanup()
```

Explanations of the Above Program

import RPi.GPIO as GPIO: Import RPi.GPIO package which has class to control GPIO to use Raspberry Pi GPIO pins in Python.

from time import sleep: Importing time module, when the statement "sleep (t)" is executed then the next line of code will be executed after t seconds. Example sleep (2) means the next statement will be executed after 2 s.

GPIO.setmode(GPIO.BOARD): This function is used to define pin numbering system, i.e. GPIO numbering or Physical numbering. You can also use GPIO.setmode (GPIO.BCM).

GPIO.setup(pin1,GPIO.OUT): This command is used to set the GPIO pin as the output pin.

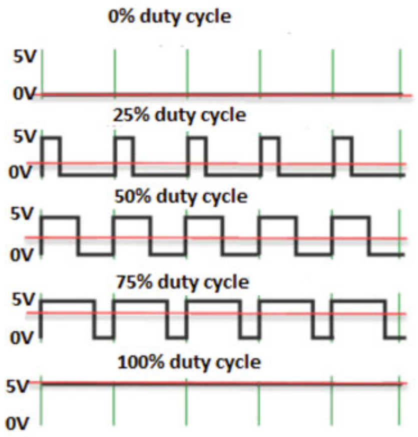
GPIO.output(pin1, HIGH): This command is used set GPIO pin to High.

sleep (2): It will act as a delay of 2 s.

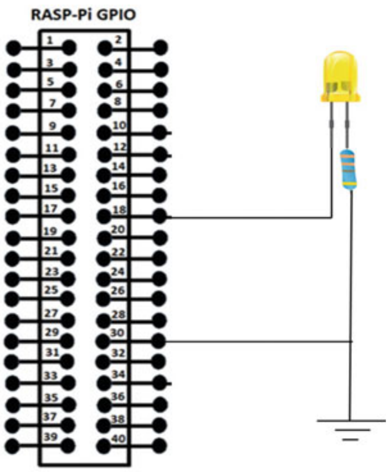
Python Program to Control LEDs using Pulse Width Modulation (PWM)

A PWM is a method for generating an analog signal using a digital source (Christ and Wernli 2014). A PWM signal consists of two main components, duty cycle and frequency. The duty cycle describes the amount of time the signal is in a high (ON) state as a percentage of the total time it takes to complete one cycle. The frequency determines how fast the PWM completes a cycle, i.e. 100 Hz would be 100 cycles per second), and therefore how fast it switches between high and low states. PWM is used for controlling the amplitude of digital signals in order to control devices. A powerful benefit of PWM is that power loss is very minimal. Compared to regulating power levels using an analog potentiometer to limit the power output by essentially choking the electrical pathway, thereby resulting in power loss as heat, PWM actually turns OFF the power output rather than limiting it. Figure 3.3a shows the PWM signal with 0–100% duty cycle and Fig. 3.3b shows the circuit diagram for understanding the concept of PWM and Fig. 3.3c shows the flowchart. Type the following code in Thonny IDE and execute the same.

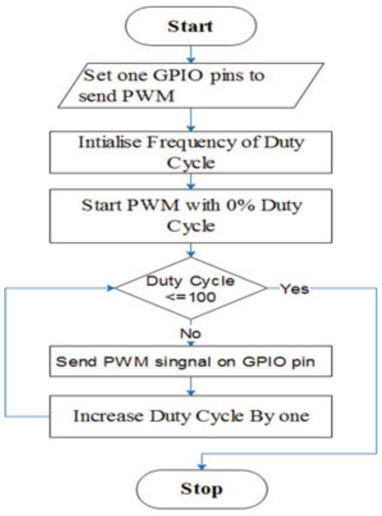
```
import RPi GPIO as GPIO
from time import sleep
GPIO.Setwarning(False)
GPIO.setmode(GPIO.BOARD)
pin=18                                      #  PWM output is on pin 18
GPIO.setup(pin,GPIO.OUT)
frequency=200                               # Set duty Cycle
pwm1=GPIO.PWM(pin, frequency)
pwm1.start(0)                                # 0% Duty Cycle PWM
for i in range (0, 100):
       pwm1.ChangeDutyCycle(i)
       sleep(0.02)
for i in range (100, 0, -1):
       pwm1.ChangeDutyCycle(i)
       sleep(0.02)
pwm1.stop()
GPIO.Cleanup()
```



a)



b)


Start
Set one GPIO pins to send PWM
Intialise Frequency of Duty Cycle
Start PWM with 0% Duty Cycle
Duty Cycle <=100
Yes
No
Send PWM singnal on GPIO pin
Increase Duty Cycle By one
Stop

c)

Fig. 3.3 **a** PWM signals with 0–100% duty cycle, **b** circuit for LED interfacing to Raspberry Pi, and **c** flowchart of the Python program

3.2 OLED Display Interface

To interact with the outside world and make the display more attractive as well as readable, OLED is interfaced to Raspberry Pi Board. It supports I2C communication through pins 3 (SDA) and 5 (SCL). Raspberry Pi pin numbers 1 (3.3 V) and 9 (GND) are used to power up the OLED, as shown in Fig. 3.4. The hardware connection details are shown in Fig. 3.5.

Figure 3.4 shows the interfacing circuit diagram of SSD1306 I2C OLED display with the Raspberry Pi. Power supply 3.3 V (pin 1) is connected to VCC pin of the OLED display. The SDA of the Raspberry Pi is connected to the OLED display's SDA pin. The SCL is connected with the OLED display's SCL pin. The GND of the Raspberry Pi is connected with the GND pin of the OLED display. The following steps are required to display message on OLED display.

Step 1: Connect power supply to the Raspberry Pi.

Step 2: I2C set up on Raspberry Pi:

Check if the I2C bus is activated on Raspberry Pi**.** Click on Raspberry Pi icon(left side corner) → preferences → Raspberry Pi configuration, as shown in Fig. 3.6a. After clicking on Raspberry Pi configuration, window will open as shown in Fig. 3.6b. The configuration window has the System, Display, Interfaces, Performance, and Localization tabs. Click on the interfaces tab and enabled the I2C (Fig. 3.6b).

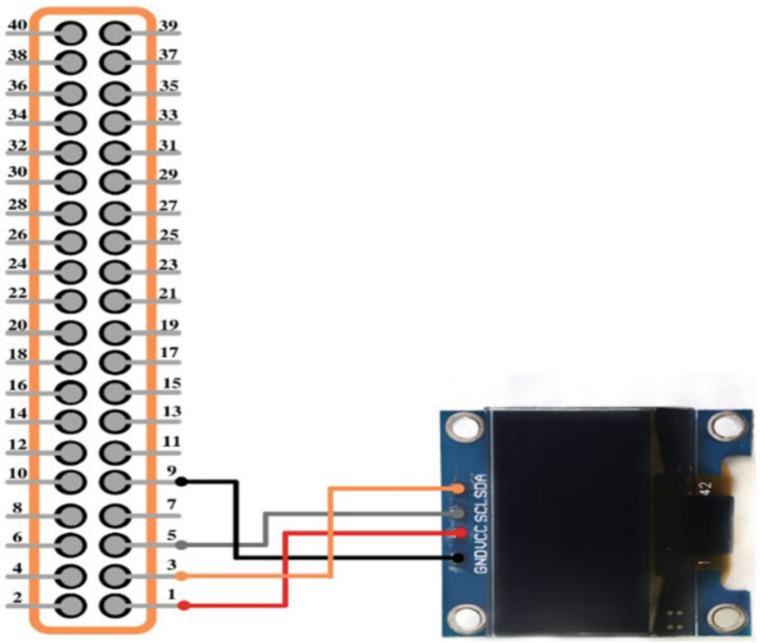


Fig. 3.4 OLED interfacing with the Raspberry Pi

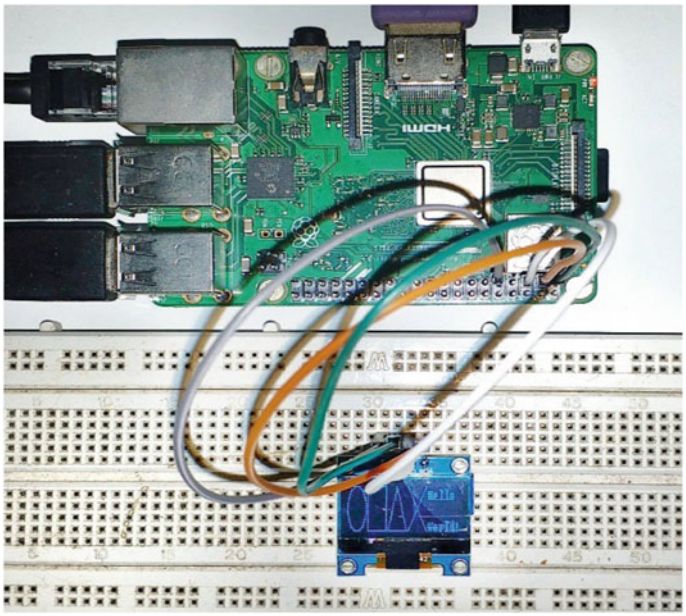
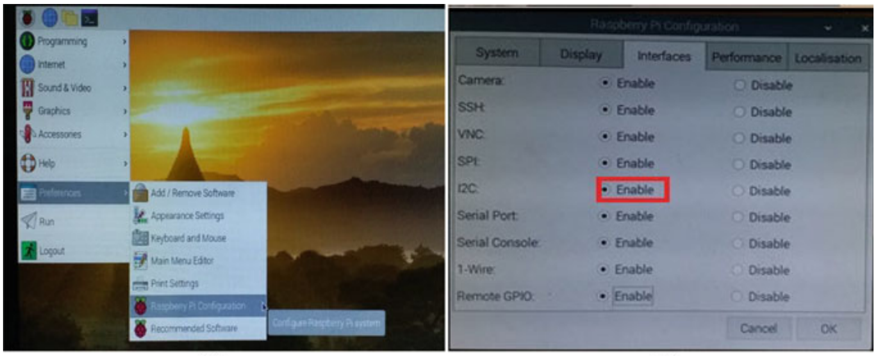


Fig. 3.5 Hardware connections for OLED



Camera: [✓ Enable] Disable
SSH: [✓ Enable] Disable
VNC: [✓ Enable] Disable
SPI: [✓ Enable] Disable
I2C: [✓ Enable] Disable
Serial Port: [✓ Enable] Disable
Serial Console: [✓ Enable] Disable
1-Wire: [✓ Enable] Disable
Remote GPIO: [✓ Enable] Disable

Fig. 3.6 I2C set up on Raspberry Pi

Step 3: The I2C activation on Raspberry Pi can also be enabled using the command terminal. Open the command terminal and enter the following command:

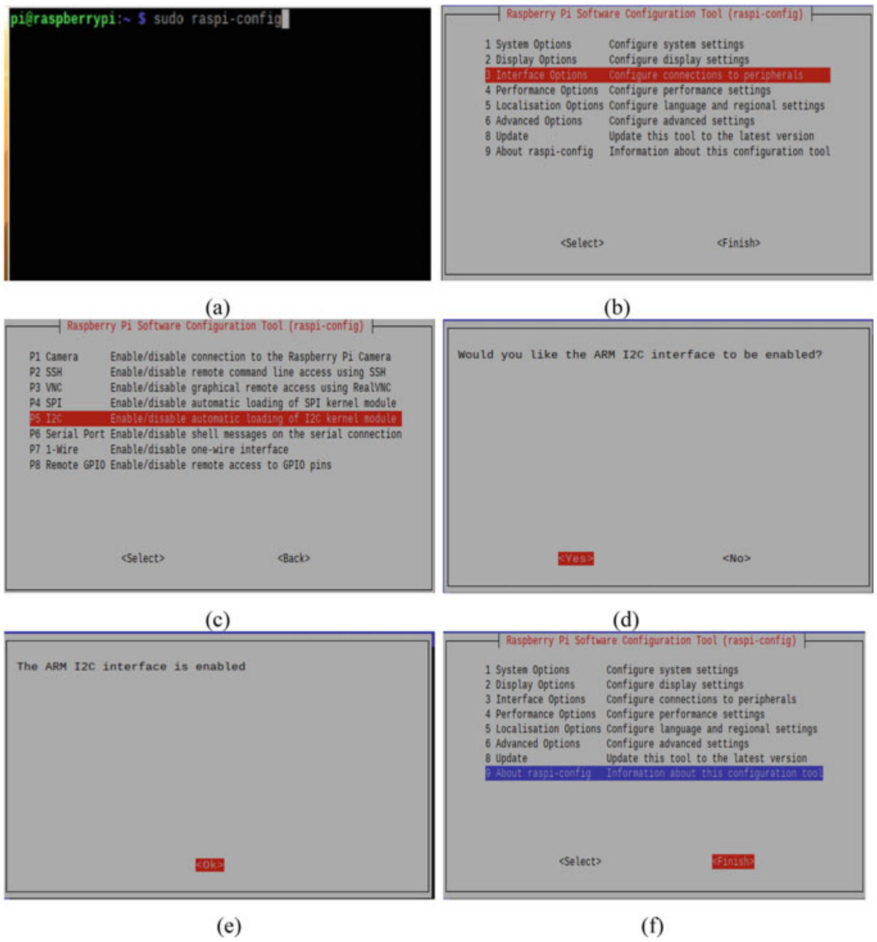
```
sudo raspi-config
```

This command will open the Raspberry Pi Software Configuration Tool. Use arrow keys to scroll down and select Interface options as shown in Fig. 3.7a and press enter → the pop-up window will open as shown in Fig. 3.7b → then select the interface options → I2C option as shown in Fig. 3.7c. After this, message will be displayed as shown in Fig. 3.7d, then click on "yes" to enable I2C → Click "Ok" followed by clicking on option "Finish" as shown in Figs. 3.7e and 3.6f, respectively.

After enabling the I2C, the next step is to install the libraries required to access OLED module.

Step 1: Update and upgrade Raspberry Pi with help of the following command:

```
sudo apt update
sudo apt upgrade
```



```
pi@raspberrypi:~ $ sudo raspi-config
```

```
Raspberry Pi Software Configuration Tool (raspi-config)

1 System Options       Configure system settings
2 Display Options      Configure display settings
3 Interface Options    Configure connections to peripherals
4 Performance Options  Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

<Select>                <Finish>
```

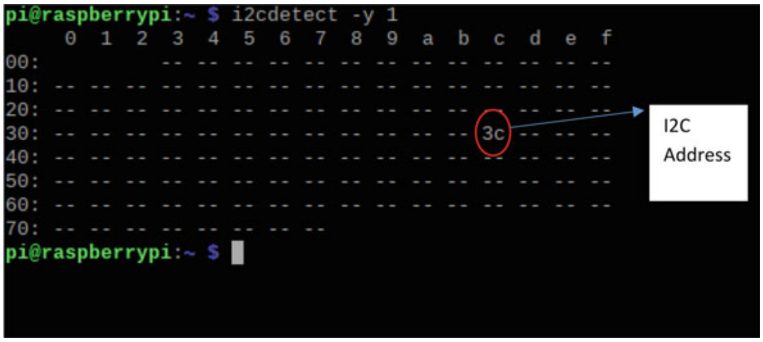
```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera      Enable/disable connection to the Raspberry Pi Camera
P2 SSH         Enable/disable remote command line access using SSH
P3 VNC         Enable/disable graphical remote access using RealVNC
P4 SPI         Enable/disable automatic loading of SPI kernel module
P5 I2C         Enable/disable automatic loading of I2C kernel module
P6 Serial Port Enable/disable shell messages on the serial connection
P7 1-Wire      Enable/disable one-wire interface
P8 Remote GPIO Enable/disable remote access to GPIO pins

<Select>                <Back>
```

```
Would you like the ARM I2C interface to be enabled?

<Yes>                   <No>
```

```
The ARM I2C interface is enabled

<Ok>
```

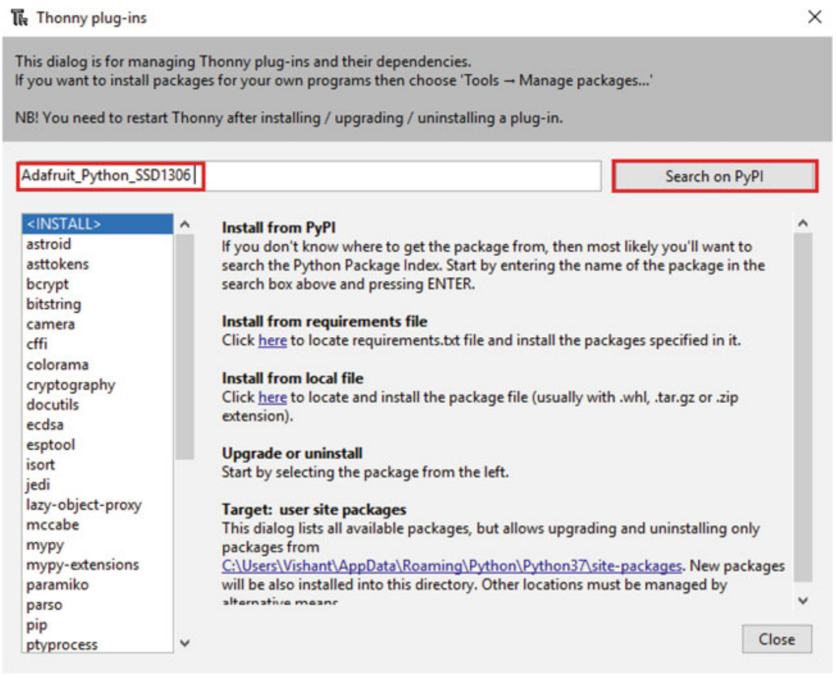
```
Raspberry Pi Software Configuration Tool (raspi-config)

1 System Options       Configure system settings
2 Display Options      Configure display settings
3 Interface Options    Configure connections to peripherals
4 Performance Options  Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

<Select>                <Finish>
```

Fig. 3.7 Enabling I2C using command Prompt



```
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```

Fig. 3.8 Identifying the I2C address



Thonny plug-ins

This dialog is for managing Thonny plug-ins and their dependencies.
If you want to install packages for your own programs then choose 'Tools → Manage packages...'

NB! You need to restart Thonny after installing / upgrading / uninstalling a plug-in.

Adafruit_Python_SSD1306

Install from PyPI
If you don't know where to get the package from, then most likely you'll want to search the Python Package Index. Start by entering the name of the package in the search box above and pressing ENTER.

Install from requirements file
Click here to locate requirements.txt file and install the packages specified in it.

Install from local file
Click here to locate and install the package file (usually with .whl, .tar.gz or .zip extension).

Upgrade or uninstall
Start by selecting the package from the left.

Target: user site packages
This dialog lists all available packages, but allows upgrading and uninstalling only packages from C:\Users\Vishant\AppData\Roaming\Python\Python37\site-packages. New packages will be also installed into this directory. Other locations must be managed by alternative means

Fig. 3.9 Installation of Adafruit_Python_SSD1306 library in Thonny IDE

Step 2: Check the I2C hex address using the following command (Fig. 3.8):

```
i2cdetect -y 1
```

Step 3: Connect to the internet for installing Adafruit_Python_SSD1306 library in Thonny IDE.

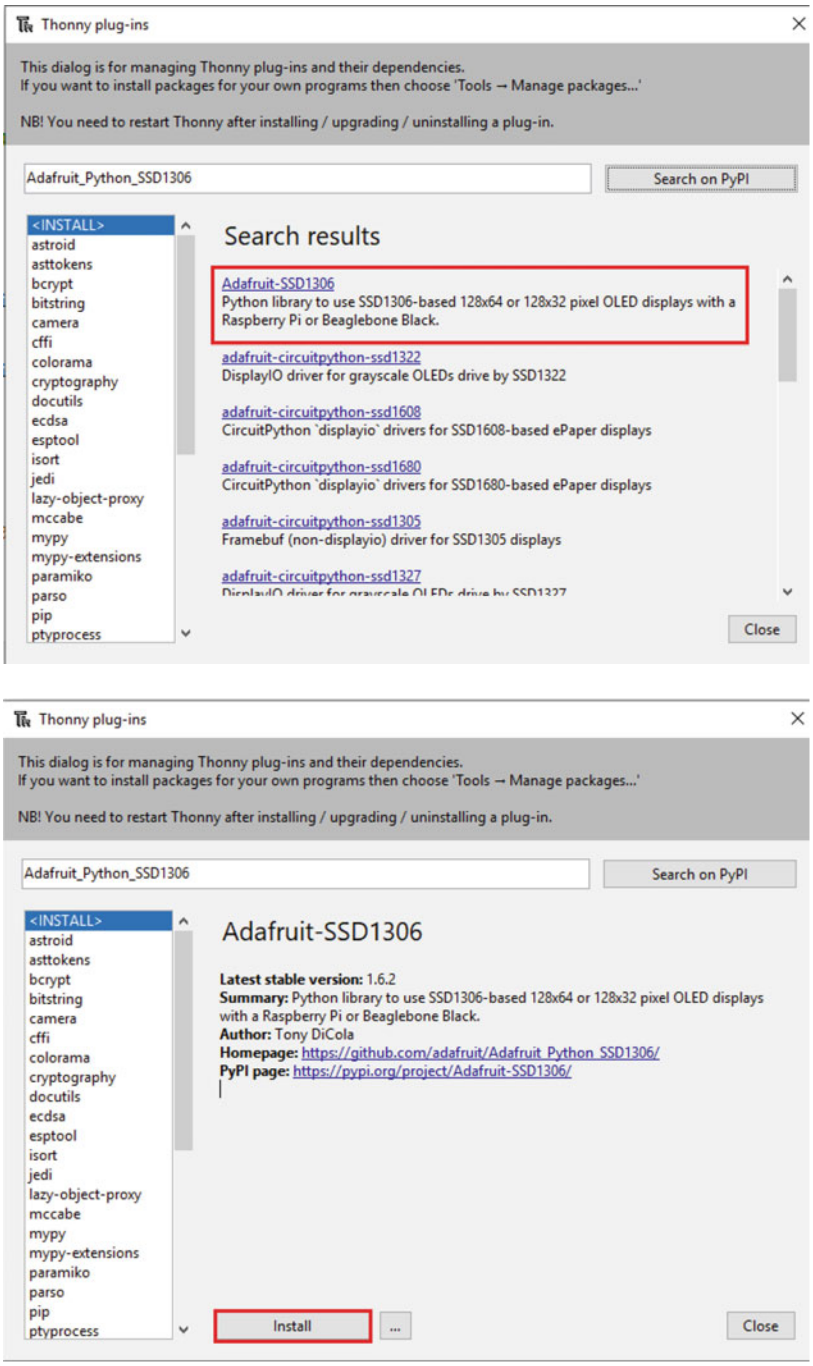
Open Thonny IDE → select Tools → Manage Plug ins → enter Adafruit_Python_SSD1306 (Fig. 3.9) and then click on "*Search on PyPi*" (Fig. 3.9). The search result will be displayed and then select the library (Fig. 3.10a). Click on Install (Fig. 3.10b) to install library.

Step 4: Testing sample program on OLED

Visit https://github.com/adafruit/Adafruit_Python_SSD1306 link and click on example folder and then open shapes.py. Copy the shapes.py program and paste it in Thonny IDE followed by saving the program.

The entire flow diagram for displaying text, rectangle, etc., on OLED display is shown below in Fig. 3.11

```
# Code to display on OLED

import time
import Adafruit_GPIO.SPI as SPI
import Adafruit_SSD1306
from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont

# Pi pin configuration:
RST = 24
# These lines are used with SPI:
DC = 23
SPI_PORT = 0
SPI_DEVICE = 0

# display with I2C:
disp1 = Adafruit_SSD1306.SSD1306_128_32(rst=RST)

# library initialization.
disp1.begin()

# display clear  function.
disp1.clear()
disp1.display()
```



(a)

Thonny plug-ins

This dialog is for managing Thonny plug-ins and their dependencies.
If you want to install packages for your own programs then choose 'Tools → Manage packages...'

NB! You need to restart Thonny after installing / upgrading / uninstalling a plug-in.

Adafruit_Python_SSD1306

Search results

Adafruit-SSD1306
Python library to use SSD1306-based 128x64 or 128x32 pixel OLED displays with a Raspberry Pi or Beaglebone Black.

adafruit-circuitpython-ssd1322
DisplayIO driver for grayscale OLEDs drive by SSD1322

adafruit-circuitpython-ssd1608
CircuitPython `displayio` drivers for SSD1608-based ePaper displays

adafruit-circuitpython-ssd1680
CircuitPython `displayio` drivers for SSD1680-based ePaper displays

adafruit-circuitpython-ssd1305
Framebuf (non-displayio) driver for SSD1305 displays

adafruit-circuitpython-ssd1327
DisplayIO driver for grayscale OLEDs drive by SSD1327

(b)

Thonny plug-ins

This dialog is for managing Thonny plug-ins and their dependencies.
If you want to install packages for your own programs then choose 'Tools → Manage packages...'

NB! You need to restart Thonny after installing / upgrading / uninstalling a plug-in.

Adafruit_Python_SSD1306

Adafruit-SSD1306

Latest stable version: 1.6.2
Summary: Python library to use SSD1306-based 128x64 or 128x32 pixel OLED displays with a Raspberry Pi or Beaglebone Black.
Author: Tony DiCola
Homepage: https://github.com/adafruit/Adafruit_Python_SSD1306/
PyPI page: https://pypi.org/project/Adafruit-SSD1306/

Fig. 3.10 a Selection of Adafruit_Python_SSD1306 library and **b** Installation of Adafruit_Python_SSD1306 library

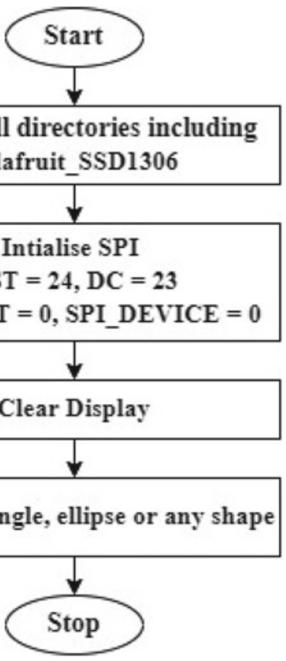

Start
Import all directories including Adafruit_SSD1306
Intialise SPI
RST = 24, DC = 23
SPI_PORT = 0, SPI_DEVICE = 0
Clear Display
Draw rectangle, ellipse or any shape
Stop

Fig. 3.11 Flowchart for displaying text, rectangle, etc., on OLED display

```
# Create blank image for drawing.
# Make sure to create image with mode '1' for 1-bit color.
width = disp1.width
height = disp1.height
image = Image.new('1', (width, height))

# draw on image with the help of drawing object.
draw = ImageDraw.Draw(image)

# clear the image with  black filled box.
draw.rectangle((0,0,width,height), outline=0, fill=0)

# Draw some shapes.
# Define  constants to allow resizing of shapes with ease.
padding = 2
shape_width = 20
top = padding
bottom = height-padding
# left to right movement keeping track of present x position for drawing shapes.
x = padding
```

```
# ellipse is drawn.
draw.ellipse((x, top , x+shape_width, bottom), outline=255, fill=0)
x += shape_width+padding
# rectangle is drawn.
draw.rectangle((x, top, x+shape_width, bottom), outline=255, fill=0)
x += shape_width+padding
# triangle is drawn.
draw.polygon([(x, bottom), (x+shape_width/2, top), (x+shape_width,
bottom)], outline=255, fill=0)
x += shape_width+padding
# Draw an X.
draw.line((x, bottom, x+shape_width, top), fill=255)
draw.line((x, top, x+shape_width, bottom), fill=255)
x += shape_width+padding


# default font is selected.
font = ImageFont.load_default()

#Write text on line.
draw.text((x, top),    'Hello',  font=font, fill=255)
draw.text((x, top+20), 'Friends!', font=font, fill=255)

# Display image.
disp1.image(image)
disp1.display()
```

3.3 Camera Interfacing

In this section, we will learn how to interface/connect the Raspberry Pi Camera Module to take pictures, record videos, and apply image effects. All current models of Raspberry Pi have a Camera Serial Interface (CSI) port (3.12a) for connecting the Camera Module as shown in following Fig. 3.12b.

Raspberry Pi Camera Module

There are two versions of the Camera Module:

- The standard version (Fig. 3.12b): Designed to take pictures in normal light.
- The NoIR version: Provided with Infrared filter so that one can use it along with an infrared light source to take pictures in the dark.

Installation Step of Picamera() Library is Given Below:

Step: To install picamera using ’apt’, open console and enter the following commands:

```
sudo apt-get update
sudo apt-get install python-picamera # for python2
sudo apt-get install python3-picamera # for python3
```

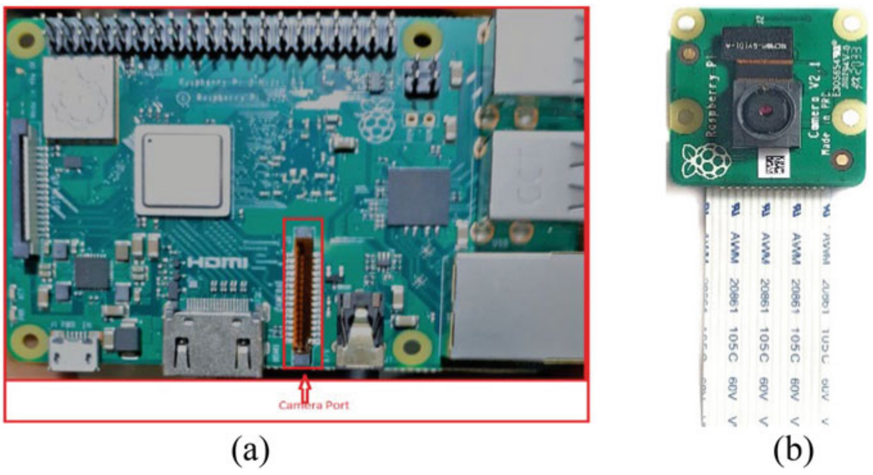


Fig. 3.12 **a** Raspberry Pi camera port and **b** camera module

One can also try the alternate method of picamera installation using Python's pip tool. The steps are given below:

Step 1: Enter the following commands to install picamera library by using Python's pip tool:

```
sudo pip install picamera
```

Step 2: Enter the following command to use the classes in the picamera array module.

```
sudo pip install "picamera[array]"
```

Step 3: Enter the following command to upgrade your installation when new releases are made:

```
sudo pip install -U picamera
```

Method to connect the Camera Module to Raspberry Pi is given below.

Step 1: Open the Camera Port on the Raspberry Pi: First ensure that Raspberry Pi is turned off and locate the Camera Module port on the Raspberry Pi 3 B+, 2 and 3, the camera port is between the LAN port and the HDMI port as shown in Fig. 3.12a. To open the port, use two fingers and lift the ends up slightly, as shown in Fig. 3.13a.

Step 2: Insert the Camera Cable: The cable has to be inserted with the right orientation with the blue side facing the Ethernet port, and the silver side is facing the HDMI port. Gently pull the clip of CSI port and insert the ribbon cable of camera module as shown in Fig. 3.13b.

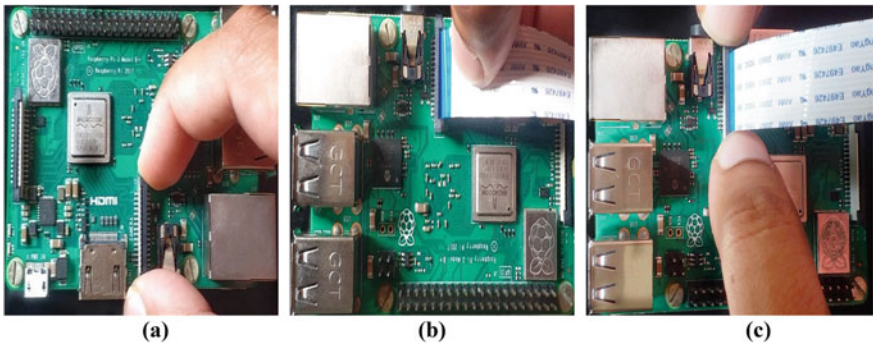


Fig. 3.13 Camera module installation

Step 3: Close the Camera Port: To close the port, push the top of the plastic clip back into the place as shown in Fig. 3.13c.

Step 4: To enable Camera interface on Raspberry Pi: Power up the Raspberry Pi → go to the main menu → open the Raspberry Pi Configuration tool (Fig. 3.14a) → Select the Interfaces tab and ensure that the camera is enabled (Fig. 3.14b).

Step 5: Testing Camera Module Via the Command Line: Open a terminal window and enter the following command to take a still picture and save it on the Desktop (Fig. 3.14c):

```
raspistill -o Desktop/image.jpg
```

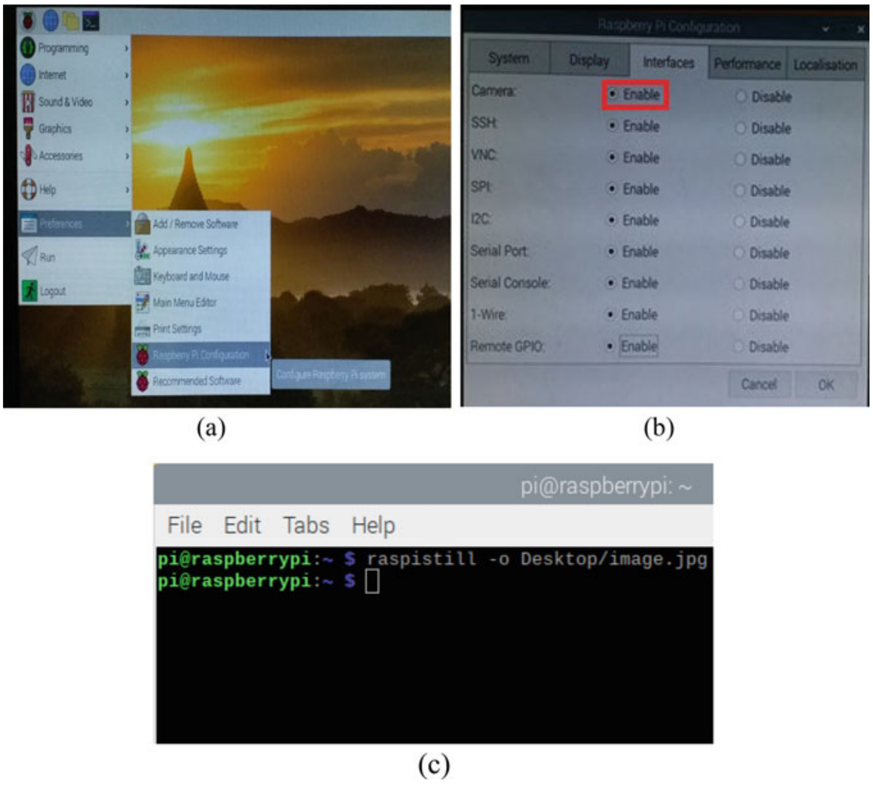
When the above command runs, you can see the camera preview opens up for 5 seconds before a still picture is taken. Look for the picture file icon on the Desktop, and double-click to open the picture.

The entire flow diagram for capturing the images using Picamera is shown below in Fig. 3.15.

Python Code to Capture Image Using Camera Module

The Python "picamera" library allows to control Camera Module.

Step 1: Open a Thonny Python IDE → Create a new file and save it as test_camera.py or any other suitable name as per your choice but make sure to avoid naming it as "picamera.py" and enter the code given below.



Camera: [✓ Enable] Disable
SSH: [✓ Enable] Disable
VNC: [✓ Enable] Disable
SPI: [✓ Enable] Disable
I2C: [✓ Enable] Disable
Serial Port: [✓ Enable] Disable
Serial Console: [✓ Enable] Disable
1-Wire: [✓ Enable] Disable
Remote GPIO: [✓ Enable] Disable

```
pi@raspberrypi:~ $ raspistill -o Desktop/image.jpg
pi@raspberrypi:~ $ 
```

Fig. 3.14 **a,b** Camera enabling process on Raspberry Pi. **c** Terminal window to enter the command

```
#Python program for taking image by using camera module

from picamera import PiCamera
from time import sleep
filepath="/home/pi/Desktop/img%s.jpg"
camera = PiCamera()
camera.resolution=(1280,720)

for i in range (0,5):
        sleep(5)
        camera.capture(filepath %i)
print("image taken")
```

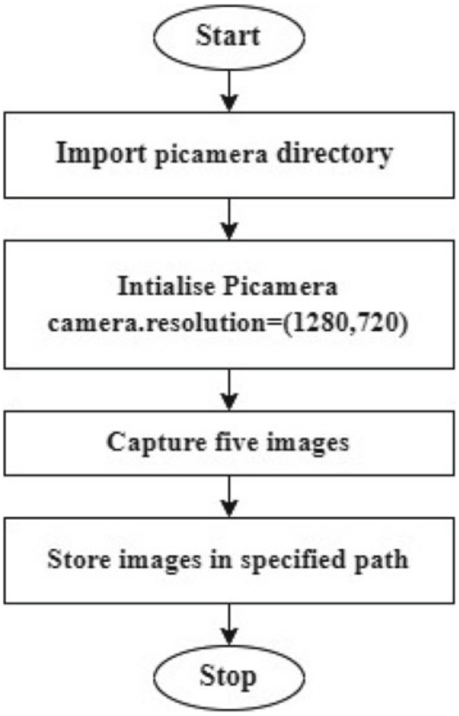

Start
Import picamera directory
Intialise Picamera
camera.resolution=(1280,720)
Capture five images
Store images in specified path
Stop

Fig. 3.15 Flowchart for capturing the images using Picamera

Step 2: Save and run the above program. The camera will take one picture every 5 s. Once the fifth image is taken, check the desktop or filepath to find the captured images (Fig. 3.16).

#Python Program to Add Text on the Captured Image

Sometimes, it is required to put some text on the captured image. An example of this can be putting a time stamp on it. Here, we have written the code which captures the image and puts the desired text on it. The flow diagram for displaying text on camera captured image is shown in below Fig. 3.17a.

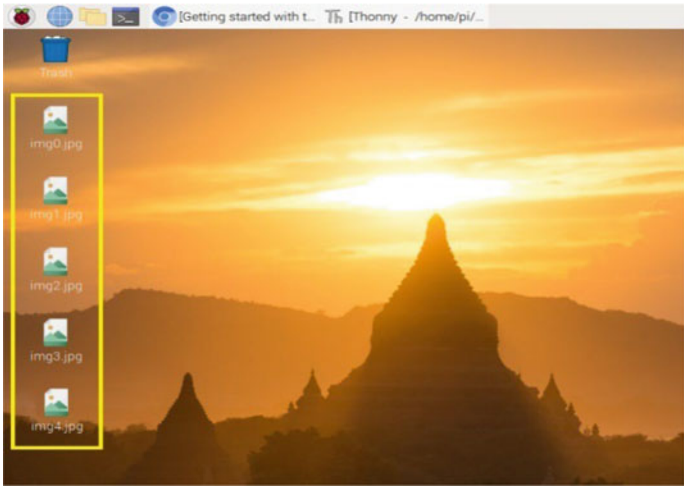


Fig. 3.16 Five pictures taken by picamera module

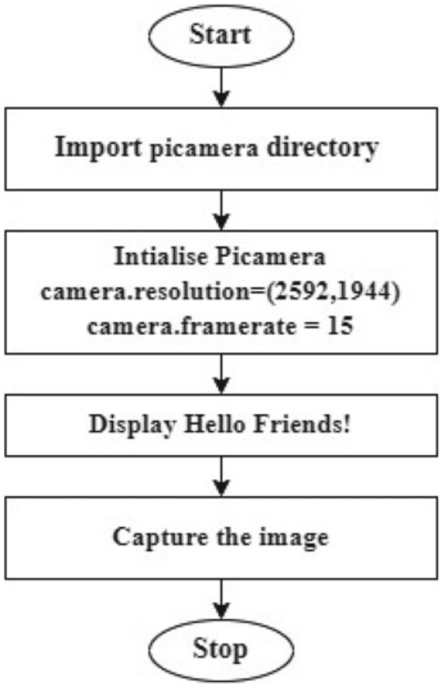
```
#In this program we will add text "Hello Friends!" on captured image.

from picamera import PiCamera
from time import sleep
filepath="/home/pi/Desktop/image.jpg"
camera = PiCamera()
camera.resolution = (2592, 1944)
camera.framerate = 15
camera.start_preview()
camera.annotate_text = "Hello Friends!"
camera.annotate_text_size = 100
sleep(5)
camera.capture(filepath)
camera.stop_preview()
```

After executing the above code in Thonny IDE, the text is imposed on image as shown in Fig. 3.17b.

#Python Program to Record Video by Using Camera Module

We have already learned how to take images by using camera module. Now, we will see how to record the video by using the same camera module and executing the below code in Thonny IDE.

(a)



(b)

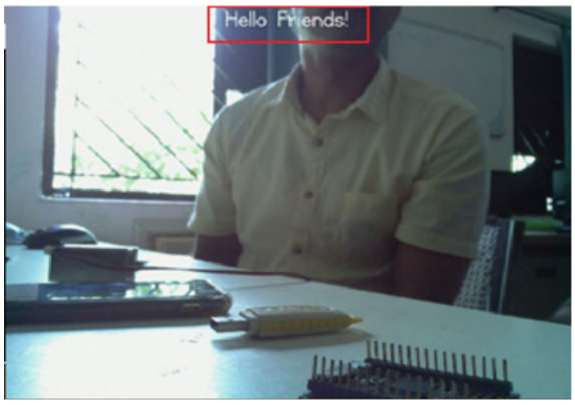


Fig. 3.17 **a** flowchart for displaying text on camera-captured image. **b** Text on picamera captured image

```
from picamera import PiCamera
from time import sleep
filepath="/home/pi/Desktop/video1.h264"
camera = PiCamera()
camera.start_preview()                    # live video preview on screen
camera.start_recording(filepath)     # Camera will start video recording
sleep(5)
camera.stop_recording()
camera.stop_preview()
```

3.4 Motor Control (DC Motor, Stepper Motor, and Servo Motor)

In this section, authors have explored various types of motor interfacing such as DC Motor, Stepper Motor, and Servo Motor with Raspberry Pi.

DC (Direct Current) Motor Control

The working principle of DC motor is that, when a magnetic field and an electric field interact, a mechanical force is produced. This is known as motoring action. There are two types of DC motors, standard and Brushless DC motors.

The DC Motor Speed Control Using the PWM Technique

The speed of DC motor is directly proportional to the supply voltage; if voltage is reduced from 12 to 6 V, then the speed becomes half that of the original speed of DC motor. But in practice, for changing the speed of a DC motor we cannot go on changing the supply voltage all the time. The voltage provided to the DC motor must be adjusted to control the speed at different torque levels (Weber 1965). The speed of the DC motor can be controlled by using PWM technique by varying the duty cycle of applied signals.

As we have already seen how to use PWM in Sect. 3.1 (control LEDs using PWM). Here, we will directly use the PWM to control the speed of DC motor. GPIO pin can source a maximum of 15 mA and the sum of currents from all 26 GPIO Pins should not exceed 50 mA. Raspberry Pi has a provision of +5 V and +3.3 V power output pins on the board for connecting other modules and sensors. This power rail is also giving power to the processor. Drawing high current from this power rail affects the processor. One can draw 100 mA safely from the +3.3 V rail. To avoid this loading effect, a separate power source is used for DC motor. The motor Driver IC L293D module is used to drive the motors. L293D is a powerful IC that can control direction and speed of two DC motors running with supply voltage ranging from 4.5 to 36 V.

Motor Driver—L293D Driver Module

L293D is a medium-power H-bridge motor driver for driving DC Motors. It consists of two H-bridge circuits for controlling each motor. H-bridge is used to change the polarity of the output, so that DC motors can be controlled in both directions. It can drive motors up to 12 V with a total DC current of up to 600 mA. The pin diagram of L293D is shown in Fig. 3.18 and pin description is as follows:

- Enable 1 and Enable 2 are the enable pins. Motors will only move if these pins are High.
- Vs is the supply voltage.
- Vss is the Logic supply voltage.
- Input1, Input2, Input3, and Input4 are the input pins.
- Output1, Output2, Output3, and Output4 are the output pins. The motors will be connected to these pins.
- GND pins are for device ground and heat sink.

Truth Table

The truth Table 3.1 provides the logical conditions to rotate motor in either clockwise or anti-clockwise direction. Table 3.2 shows the pin connections of motor with Raspberry Pi having L293 motor driver,

Figure 3.19a and b show the circuit diagram and photo of motor interfaced to Raspberry Pi, respectively. Figure 3.19c gives the flowchart for controlling DC motor. Type the below code in Thonny IDE and execute the same.

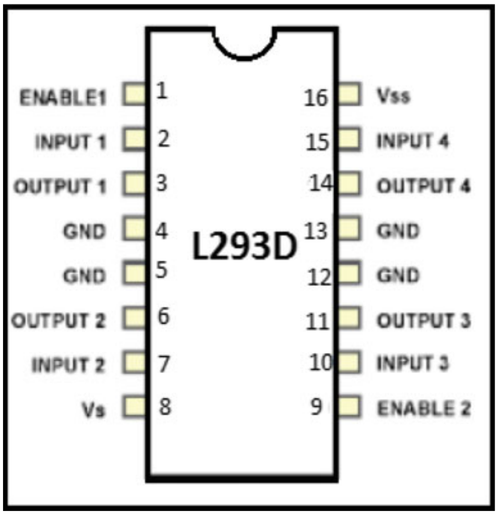


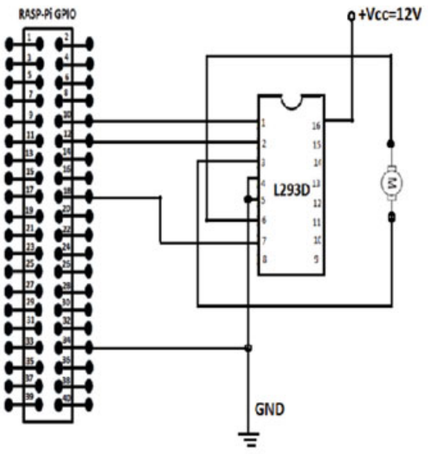
Fig. 3.18 Pin diagram of L293D

Table 3.1 Truth table for motor control

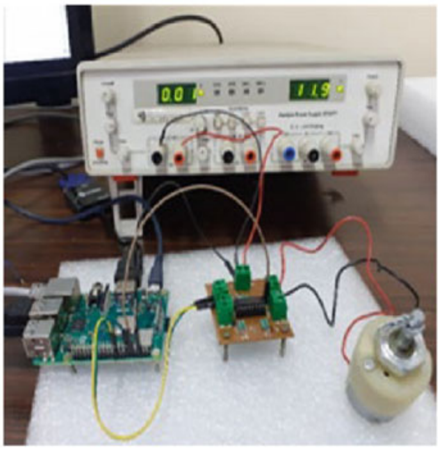
Enable	Input-1	Input-2	Output
High	High	Low	Turn anti-clockwise
High	Low	High	Turn clockwise
High	High	Low	Stop
High	Low	High	Stop
Low	X	X	X

Table 3.2 Connection of L293D motor driver module with Raspberry Pi

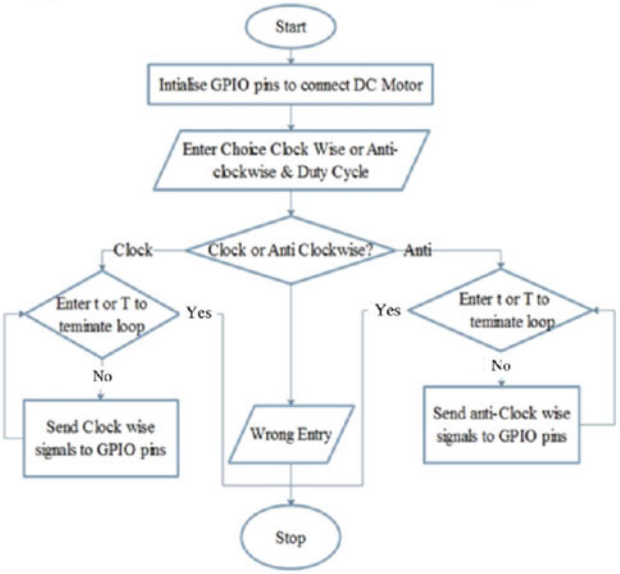
L293D motor driver module	Raspberry Pi
Input 1	Physical Pin 12 (GPIO18)
Input 2	Physical Pin 18 (GPIO24)
GND	GND
Enable	Physical Pin 10 (GPIO 15)



(a)



(b)


Start
Intialise GPIO pins to connect DC Motor
Enter Choice Clock Wise or Anti-clockwise & Duty Cycle
Clock or Anti Clockwise?
Clock
Anti
Enter t or T to teminate loop
Yes
No
Send Clock wise signals to GPIO pins
Wrong Entry
Enter t or T to teminate loop
Yes
No
Send anti-Clock wise signals to GPIO pins
Stop

(c)

Fig. 3.19 **a** Circuit diagram for DC motor interfacing with Raspberry Pi, **b** hardware connection, and **c** flowchart

```
# Program to control DC motor

import RPi.GPIO as GPIO

from time import sleep

GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)

Enable1=10

input1=12

input2=18

GPIO.setup(Enable1,GPIO.OUT)

GPIO.setup(input1,GPIO.OUT)

GPIO.setup(input2,GPIO.OUT)

pwm=GPIO.PWM(Enable1,100)

pwm.start(0)

Rotation=input(" Enter c/C for clockwise & a/A for Anticlockwise:  ")
duty_cycle=int(input("Enter Duty Cycle from 1 to 100:   "))

pwm.start(duty_cycle)

delay=0.1

print("Press Ctrl+c for termination")

if Rotation=='c' or Rotation=='C':

   print ("Clockwise MOTION")

   try:

     while True :

        GPIO.output(input1,False)    # pin 2 IN1 of Driver

        GPIO.output(input2,True)   # pin 7 IN1 of Driver

        GPIO.output(Enable1,True)    # pin 1 IN1 of Driver

        sleep(delay)

    exceptKeyboardInterrupt:

        pass
     elif Rotation=='a' or Rotation=='A':

         print (" AntiClockwise MOTION")

         try:

             while True:

                 GPIO.output(input1,True)

                 GPIO.output(input2,False)

                 GPIO.output(Enable1,True)

                 sleep(delay)

             exceptKeyboardInterrupt:

                 pass


     else:
          print("Wrong Entry")


      pwm.stop()

      pwm.stop()                     #stop the Pulse

      GPIO.cleanup()

      print("Finished")       #cleanup all of the GPIO channels.
```

Controlling Servo Motor using PWM

A servomotor is a rotary or linear actuator that enables precise control of angular or linear position, acceleration, and velocity. The main application of DC servo motors is in remote-controlled devices, robotics, and even in industrial applications (Ferrari and Ferrari 2007). Servo motors are different from ordinary motors. Depending on the specification of the servo motor, they can rotate from 0-degree to 180-degree by varying the duty cycle of the PWM signal. Servo Arm-Head moving speed can also be controlled by varying the Duty Cycle using PWM. Servo motor 90-degree position is generally referred to as “neutral” position, because it can rotate equally in either direction from that point. In this implementation, Tower pro servo motor SG90 is used. Servos can push heavy loads but they cannot lift heavy loads. PWM duration given for tower pro servo motor is 20 ms (frequency 50 Hz). PWM signal having a duration of 20 ms and signal duty cycle in between 0 to 2 ms must be generated to rotate the servo motor. Table 3.3 tabulates the duty cycle with respect to the applied current.

Table 3.3 Duty cycle with respect to the applied current

Position (degrees)	Duty cycle (ms)	Duty cycle (%)
90	1.5	7.5
180	2	12.5
0	1	2.5

PWM signal will set the angle of the servo. This can differ from servo to servo, as normally it is from 2.5 to 12.5%. To calculate the duty cycle for the desired angle, divide it by 18, then add the lowest available value, and, in this case, it is 2.5.

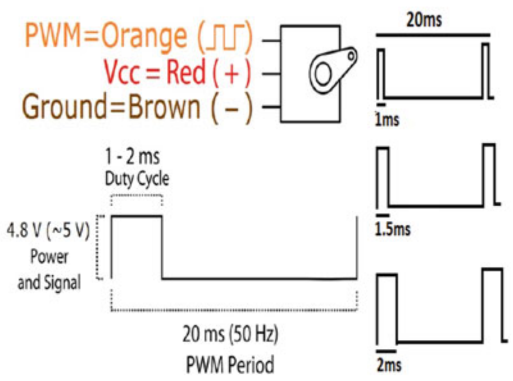
The formula to calculate the duty cycle is as follows:

Duty_cycle = (set_angle/18) + 2.5

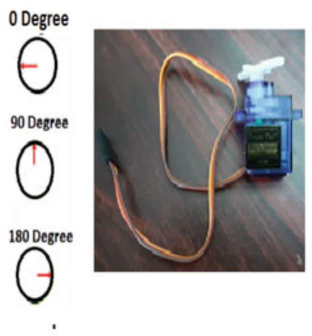
So, for 90 degrees, 7.5% duty cycle and, for 180 degrees, 12.5%.

One can easily rotate the arm at a fixed angle by just varying the duty cycle. Figure 3.20a–c shows the Tower Pro SG90 9G servo Motor interfaced to Raspberry Pi with duty cycle. Figure 3.20d shows the flowchart for controlling servo motor. Type the below code in Thonny IDE and execute the same.

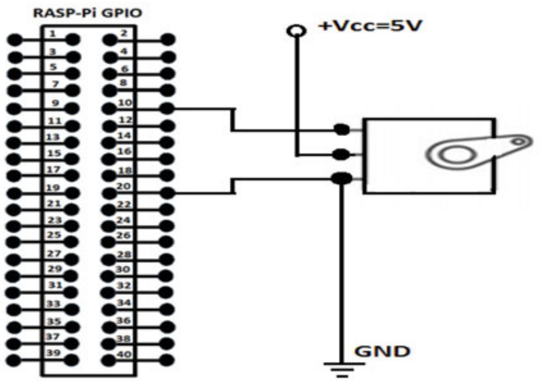
```
# program to control Servo motor

import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
control_pin=10
GPIO.setup(control_pin,GPIO.OUT)  # connect to enable pin-1 of Driver IC
pwm=GPIO.PWM(control_pin,50)     # to setup the pwm commands type 50Hz freqency
set_angle=int(input(" Enter Servo Motor Rotation Angle:  "))
pwm.start(0)
delay=1
duty_cycle=(set_angle/18)+2.5
GPIO.output(control_pin,True)
pwm.ChangeDutyCycle(duty_cycle)
#GPIO.output(control_pin,True)
sleep(delay)
GPIO.output(control_pin,False)
pwm.ChangeDutyCycle(0)
pwm.stop()
GPIO.cleanup()
```



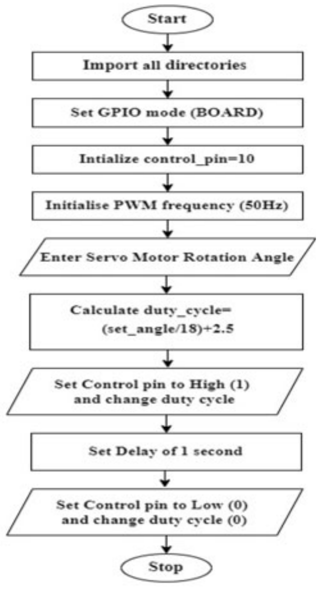
(a)



(b)



(c)


Start
Import all directories
Set GPIO mode (BOARD)
Intialize control_pin=10
Initialise PWM frequency (50Hz)
Enter Servo Motor Rotation Angle
Calculate duty_cycle= (set_angle/18)+2.5
Set Control pin to High (1) and change duty cycle
Set Delay of 1 second
Set Control pin to Low (0) and change duty cycle (0)
Stop

(d)

Fig. 3.20 **a** Tower Pro SG90 9G servo Motor Duty Cycle and PWM Period, **b** duty cycle with respect to the angle, **c** circuit diagram of servo motor interfacing with Raspberry Pi, and **d** flowchart for controlling servo motor

Stepper Motor Control

Stepper Motor

A stepper motor is a device that translates a DC voltage pulse train into a mechanical rotation of its shaft in a proportionate manner (Fraser 1994). Stepper motor is made up of mainly two parts, a stator and a rotor. Stator is of coil winding and rotor is mostly permanent magnet or ferromagnetic material. Stepper motors are generally used for position control. One of the best things about these motors is that they can be positioned accurately and one “step” at a time. They are a special type of brushless motor that divide a full rotation into a number of equal “steps”. They are usually found in desktop printers, 3D printers, CNC milling machines, and anything else that requires precise positioning control. The speed of rotation depends upon the rate at which the control signals (Duty Cycle) are applied. A driver IC ULN2003 is used to drive the stepper motor as GPIOs of Raspberry Pi are not able to provide sufficient drive current.

In the stepper motor, continuous and limited angle rotation is obtained by providing sequential steps. Mostly, there are two step sequences, i.e. full step and half step used to rotate stepper motor as shown in Fig. 3.21. In the full-step mode, at a time two coils are excited, while, in the half-step sequence, motor moves half of its basic step angle. Tables 3.4 and 3.5 tabulate the full-step mode and half-step mode coil energizing sequence, respectively.

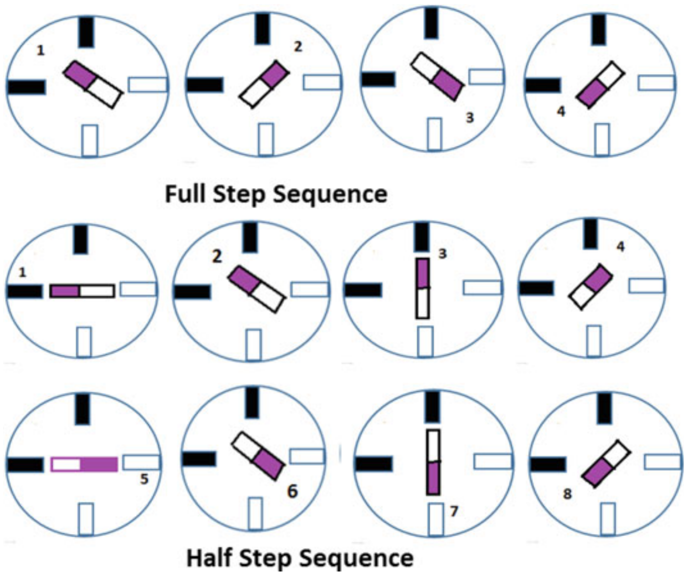


Fig. 3.21 Full-step and half-step sequence of stepper motor

Table 3.4 Full-step mode

Full-step mode				
Step	A	B	C	D
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

Table 3.5 Half-step mode

Half-step mode				
Step	A	B	C	D
1	1	0	0	1
2	1	0	0	0
3	1	1	0	0
4	0	1	0	0
5	0	1	1	0
6	0	0	1	0
7	0	0	1	1
8	0	0	0	1

In this implementation, 28BYJ-48 motor is used, which runs in full-step mode, and each step corresponds to a rotation of 11.25°. That means, there are 32 steps per revolution (360°/11.25° = 32). The gear ratios are 32/9, 22/11, 26/9, and 31/10.

Multiplying all gear ratios:

32/9 * 22/11 * 26/9 * 31/10 = 63.68395 ~ 64.

This gives 64:1 gear ratio. The motor has a 1/64 reduction gear set.

Total Full Steps = 32*64 = 2038 steps and Half steps = 4076.

Figure 3.22a shows the internal gears and Fig. 3.22b shows the pin number and wire colors of the stepper motor.

The motor 28BYJ-48 has a four unipolar coils and each coil is rated at +12 V; hence, it is relatively easy to control with Raspberry Pi. This motor has a stride angle of 5.625°/64, which means the motor will have to make 64 steps to complete one rotation and, for every step, it will cover 5.625°. The power consumption of the motor is around 240 mA. The current supplied by GPIO of Raspberry Pi is not sufficient to drive the motor hence the ULN 2003 motor driver IC is used.

ULN 2003

ULN2003 is a driver IC consisting of a Darlington array and capability of handling seven different inputs/outputs simultaneously. It operates in the range of 500 mA–600 mA current. Figure 3.25 shows the pin diagram of the ULN2003 (Fig. 3.23).

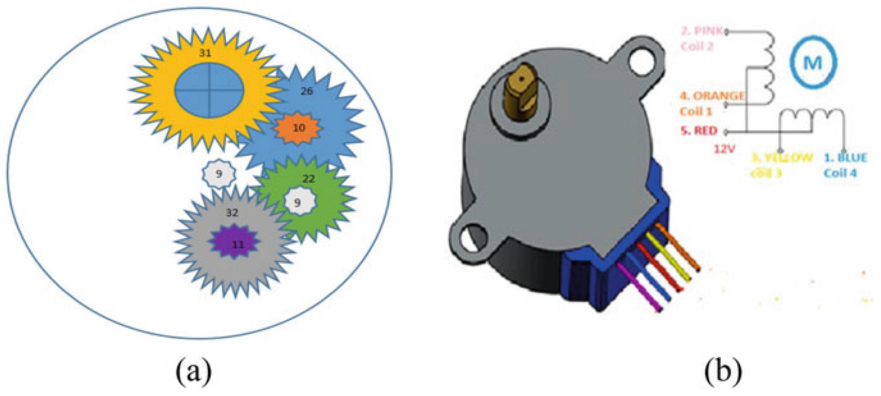


Fig. 3.22 **a** internal gears of the stepper motor and **b** pin number and wire colors of the stepper motor

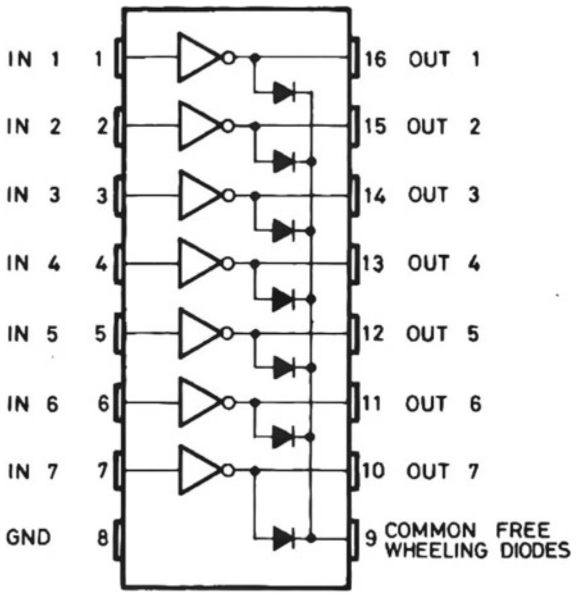


Fig. 3.23 Pin diagram of the ULN2003

Table 3.6 tabulates the stepwise sequence of the rotate motor in the clockwise direction. Figure 3.24 shows (a) circuit diagram, (b) photo of hardware connection, and (c) flowchart to control stepper motor.

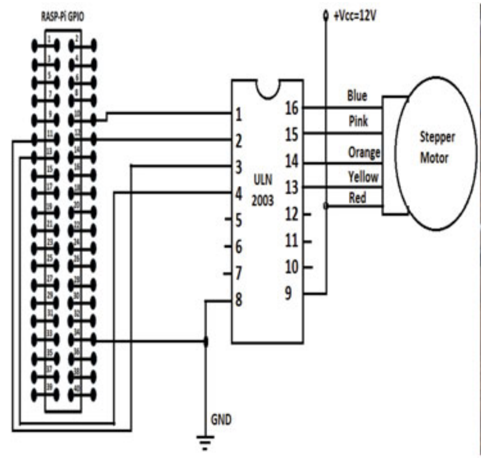
Type the below code in Thonny IDE and execute the same to control stepper motor.

Table 3.6 The stepwise sequence of the rotate motor in the clockwise direction

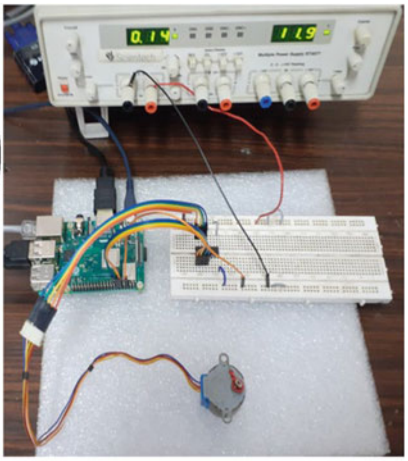
Motor wire color	Sequence to rotate in clockwise direction							
	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8
Orange	0	0	1	1	1	1	1	0
Yellow	1	0	0	0	1	1	1	1
Pink	1	1	1	0	0	0	1	1
Blue	1	1	1	1	1	0	0	0
Red	1	1	1	1	1	1	1	1

```
# Program for Stepper motor control
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

pins=[10,11,12,13]
# Full step Clock wise sequence reverse can be used for anticlockwise
full_step=[[0,1,1,0],
           [1,1,0,0],
           [1,0,0,1],
           [0,0,1,1]]

# half step Clock wise sequence reverse can be used for anticlockwise
half_step=[[0,1,1,0],
           [1,1,1,0],
           [1,1,0,0],
           [1,1,0,1],
           [1,0,0,1],
           [1,0,1,1],
           [0,0,1,1],
           [0,1,1,1]]
```

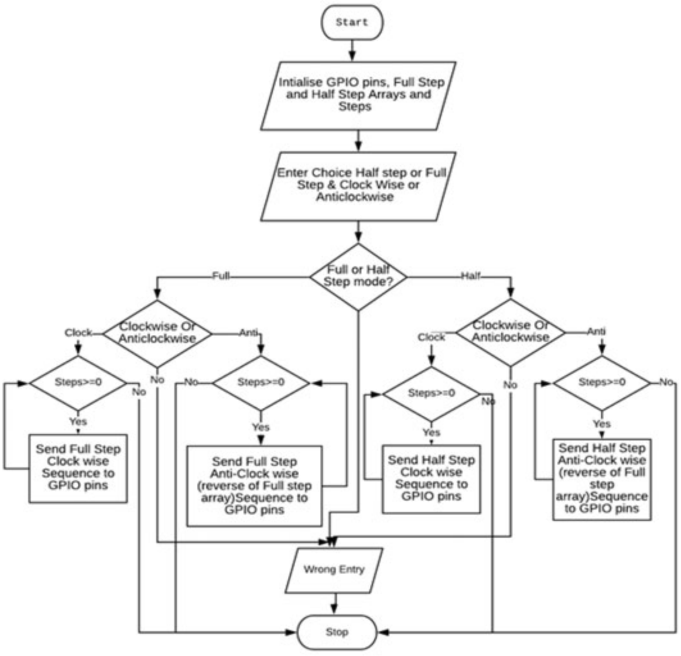
```
# To reset all pins
for pin in pins:
    GPIO.setup(pin,GPIO.OUT)
    GPIO.output(pin,0)

seq=input("Enter f/F for full step & h/H for half Step: ")

rotation=input("Enter c/C clockwise & a/A for Anticlock wise: ")
steps=512
delay=0.01
if seq=='f' or seq=='F':
    if rotation=='c' or rotation == 'C':
        for i in range(0,steps):
            for fullstep in range(0,4):
                for pin in range(0,4):
                    GPIO.output(pins[pin],full_step[fullstep][pin])
                    sleep(delay)


    elif rotation=='a' or rotation == 'A':
        for i in range(0,steps):
            for fullstep in range(3,0,-1):
                for pin in range(0,4):
                    GPIO.output(pins[pin],full_step[fullstep][pin])
                    sleep(delay)
    else:
        print("Wrong Entry")


elif seq=='h' or seq=='H':
    if rotation=='c' or rotation == 'C':
        fori in range(0,steps):
            for halfstep in range(0,8):

                for pin in range(0,4):
                    GPIO.output(pins[pin],half_step[halfstep][pin])
                    sleep(delay)
    elif rotation=='a' or rotation == 'A':
        for i in range(0,steps):
            for halfstep in range(7,0,-1):
                for pin in range(0,4):
                    GPIO.output(pins[pin],half_step[halfstep][pin])
                    sleep(delay)
    else:
        print("Wrong Entry")
else:
    print("Wrong Entry")

print("finished")
```



(a)



(b)


Start
Intialise GPIO pins, Full Step and Half Step Arrays and Steps
Enter Choice Half step or Full Step & Clock Wise or Anticlockwise
Full or Half Step mode?
Full
Half
Clockwise Or Anticlockwise
Clock
Anti
Steps>=0
Yes
No
Send Full Step Clock wise Sequence to GPIO pins
Send Full Step Anti-Clock wise (reverse of Full step array)Sequence to GPIO pins
Send Half Step Clock wise Sequence to GPIO pins
Send Half Step Anti-Clock wise (reverse of Full step array)Sequence to GPIO pins
Wrong Entry
Stop

(c)

Fig. 3.24 **a** circuit diagram of the stepper motor connected with Raspberry Pi, **b** hardware connection of motor and Raspberry Pi, and **c** flowchart of the program

3.5 Raspberry Pi and Mobile Interface Through Bluetooth

Bluetooth is a wireless alternative to many of the wired communication that we use to transport voice and data (Ramandeep Kaur2, Manpreet Kaur3, J. K. 2017). Raspberry Pi 3 B+ has BCM43438 integrated chip which includes 2.4 GHz WLAN, Bluetooth, and FM receiver. The main purpose of using Bluetooth is to free up the on-board GPIO ports. Here, we have established Bluetooth communication between Raspberry Pi and smartphone to control devices.

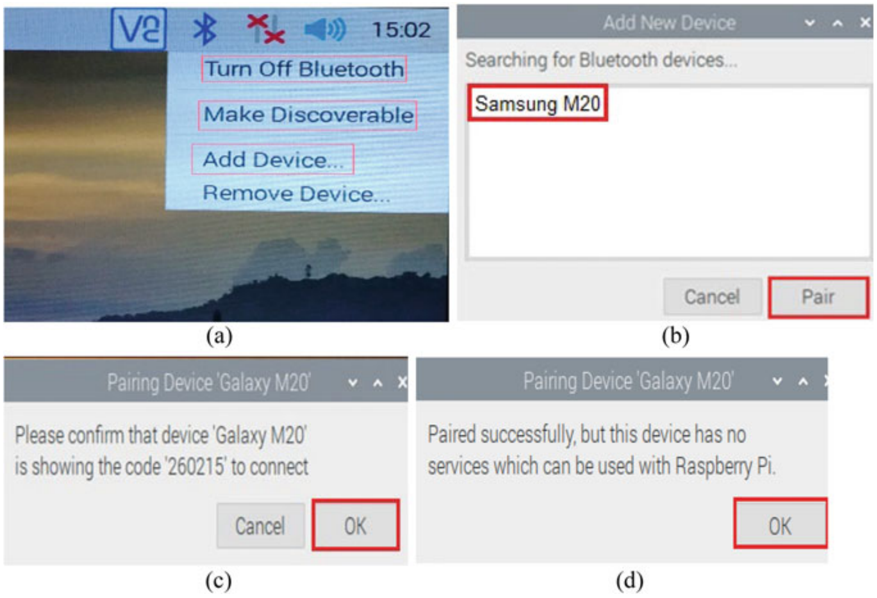
Configuration of On-board Bluetooth of Raspberry Pi 3B+:

Raspberry Pi has an on-board Bluetooth which can be used for sending/receiving files to/from smartphone. Pairing a Bluetooth device on Raspberry Pi is same as that of a smartphone or Laptop.

Step 1: Turn-ON Bluetooth → make discoverable (Fig. 3.25a).

Step 2: Turn on Bluetooth of smartphone. Simultaneously, Select Add Device on Raspberry Pi (Fig. 3.25a). After selecting Add device, we can see mobile Bluetooth device, e.g "Samsung M20" → Select device and then click on pair as shown in Fig. 3.25b.

Step 3: Following the above step prompts for confirming the pairing code sent on the smartphone as shown in Fig. 3.25c. After the device accepts the connection by using the pair option, Raspberry Pi and the Bluetooth device will be paired and connection



Please confirm that device 'Galaxy M20' is showing the code '260215' to connect

Paired successfully, but this device has no services which can be used with Raspberry Pi.

Fig. 3.25 Bluetooth connection setting between mobile and Raspberry Pi

is established as shown in Fig. 3.25d. Now the Raspberry Pi is ready to communicate via Bluetooth.

Controlling Device by Using Smartphone Through Bluetooth (Blue Dot app)

Step 1: From the Google Play store, download the Blue Dot app on smartphone for controlling devices.

Step 2: Open a terminal and enter the following command to install dbus and bluedot packages for Python 3.

```
sudo apt install python3-dbus
sudo pip3 install bluedot
```

optionally following commands shall be used for Python 2:

```
sudo apt install python-dbus
sudo pip install bluedot
```

Step 3: Upgrade to the latest version of bluedot using the following command:

```
sudo pip3 install bluedot—upgrade
```

Figure 3.26 shows the circuit diagram of the Raspberry Pi interfaced with Relay for controlling the appliances.

#Python Program to Control the Appliances Using Bluetooth

```
import os
from bluedot import BlueDot

import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)

pin=11                                      # Physical Pin-11=GPIO 17
 GPIO.setup(pin,GPIO.OUT)

bd=BlueDot()

while True:
      bd.wait_for_press()
     GPIO.output(pin, HIGH)                # to turn on the appliance
      bd.wait_for_release()
      GPIO.output(pin, LOW)               # to turn off the appliance
```

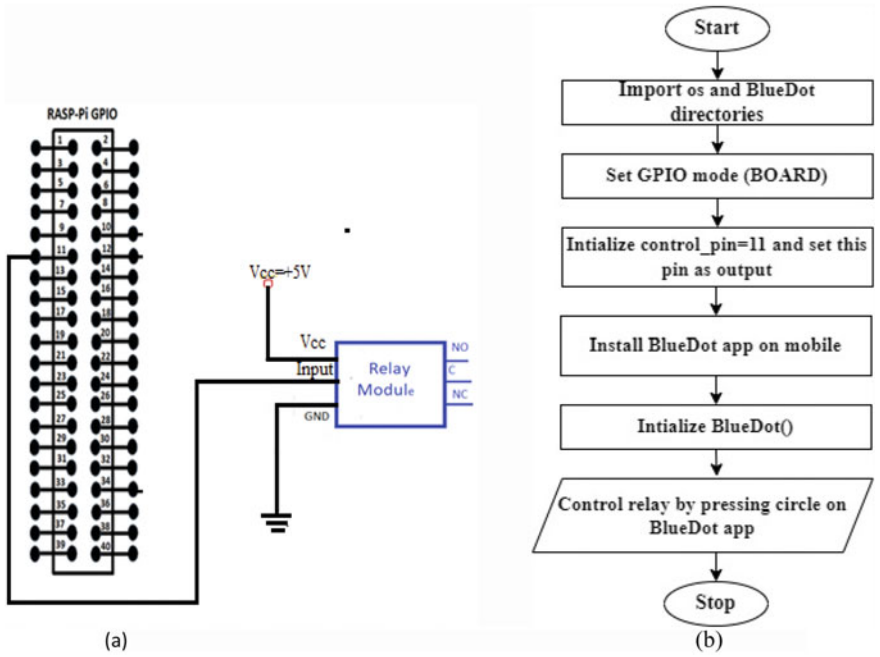

RASP-Pi GPIO
Vcc=+5V
Vcc
Input
GND
Relay Module
NO
C
NC
(a)
Start
Import os and BlueDot directories
Set GPIO mode (BOARD)
Intialize control_pin=11 and set this pin as output
Install BlueDot app on mobile
Intialize BlueDot()
Control relay by pressing circle on BlueDot app
Stop
(b)

Fig. 3.26 **a** Circuit diagram of Raspberry Pi interfaced with Relay and **b** flowchart for controlling relay using BlueDot app

After executing the above program in Thonny IDE, open the Blue Dot app on your smartphone as shown in Fig. 3.27a. After connecting smartphone with Rasberry Pi, it will show a blue circle as shown in Fig. 3.27b. When the blue circle is pressed, the appliance connected to GPIO 17 (Physical Pin = 11) through relay will turn on.

Conclusion:

Raspberry Pi is an SBC installed with Raspbian OS. Various peripherals such as keyboard, mouse, and display are connected to the Raspberry Pi, which makes this system act as a mini personal computer. Raspberry Pi is popularly used for real-time Image Video Processing, IoT-based applications, and Robotics applications. In this chapter, authors have implemented Python code for controlling LEDs by using simple delay and PWM. Detailed steps involved in accessing the GPIO along with a connection diagram and execution are given. A simple I2C-based Adafruit SSD 1306 OLED is interfaced with Pi board and is also covered in a simple manner. For image and video processing, camera interfacing plays an important role. Here, authors have given the interfacing of CSI camera to Pi board in a simple manner. At the end, motor control and IoT-based home appliance control using mobile phone is covered in detail.

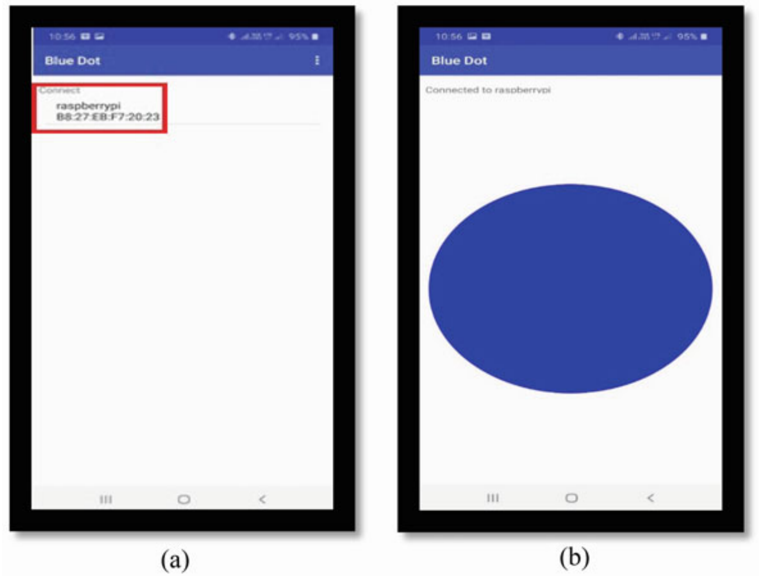
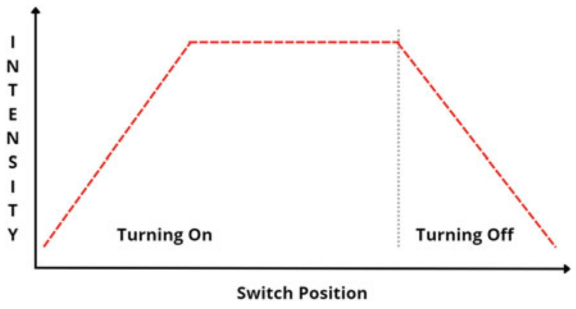


Fig. 3.27 Controlling Relay using Bluedot app

Exercise:

(1) Connect 5 LEDs to Raspberry Pi. Configure the Raspberry Pi in BCM mode and write a program to blink LEDs as shown in the below pattern with an interval of 1 s between each pattern in a continuous cycle.
(2) Interface a switch and a led with Raspberry Pi. When the switch is turned ON the LED intensity should increase from Low to High and remain High. When the switch is turned OFF the intensity of LED should reduce gradually from High to Low and turn OFF(Hint: Refer to the below figure).
(3) Blink Eeight LEDs with different duty cycles using Raspberry Pi.



(4) Display string "Hello World", integer and floating-point number on OLED display.

(5) Interface a couple of stepper motors/DC with Raspberry Pi. Program it to run in full-step mode in a clockwise direction for 5 s followed by 10 s in anti-clockwise direction.
(6) Write Python code to control two servo motors.

References

Christ RD, Wernli RL (2014) Power and telemetry. In: Christ RD, Wernli RL (eds) The ROV manual (Second Edition), Butterworth-Heinemann, pp 141–161. https://doi.org/10.1016/B978-0-08-098288-5.00007-5

Ferrari M, Ferrari G (2007) Controlling motors. Building robots with LEGO mindstorms NXT, syngress, 2007, pp 41–59. https://doi.org/10.1016/B978-159749152-5/50008-5

Fraser CJ (1994) 2 - Electrical and electronics principles. In: Smith EH (ed) Mechanical engineer's reference book (Twelfth Edition), Butterworth-Heinemann, p 2-1-2-57. https://doi.org/10.1016/B978-0-7506-1195-4.50006-3

Kaur R, Kaur M, Kaur J (2017) Bluetooth technology. Int J Eng Comput Sci 5(3).http://www.ijecs.in/index.php/ijecs/article/view/702

Weber HF (1965) Pulse-width modulation DC motor control. IEEE Trans Ind Electron Control Instrum IECI 12(1):24–28. https://doi.org/10.1109/TIECI.1965.229545