

Exploring Robot Programming with Python

DevFest Brunei 2023

This document can be downloaded at <https://ailab.space/events/devfest2023>.

Contents

1	Introduction	2
2	Remote access the robot	2
3	Programming environment for Raspberry Pi.....	3
4	Inputs and outputs of Raspberry Pi	4
5	Robot actuators	5
5.1	Motor teleoperation and pan-tilt camera	5
5.2	Buzzer.....	9
6	Robot Sensors	10
6.1	Ultrasonic (Ping) sensor for range measurement	10
6.2	Tracker (TR) / Line Sensors for infrared line tracking	12

1 Introduction

In this workshop, you will learn to program a mobile robot to perform specific tasks both manually controlled (teleoperate) and autonomously. You will learn the following skills:

- How to **program a robot remotely**, i.e., without tethered to the robot?
- How to **program a Raspberry Pi** single-board computer?
- How to program the **control of the actuators**?
- How to program the **data retrieval from the sensors**?

You will use this skillset to control a mobile robot in completing a challenge.

This workshop will use **Python** as the programming language.

Each participant will be provided with an **AlphaBot2**. There are 10 Alphabot2. If there are more than 10 participants, a few participants will work together with one robot.

AlphaBot2 is equipped with Raspberry Pi 3. The source of power of AlphaBot2 would be either battery or using power adapter. When developing the programs, you will use the robot with power adapter.

2 Remote access the robot

All AlphaBot2 are connected to "robolab2" network. Please connect to the same **network** as the robots.

```
SSID: robolab2
Password: 12345678
```

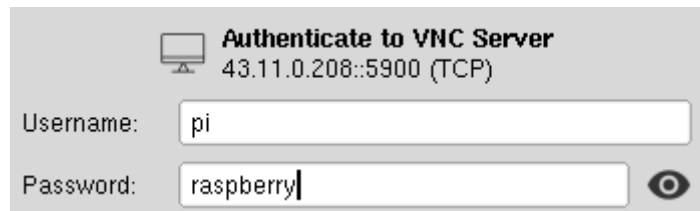
Each Alphabot2's **IP address** is 43.11.0.2XX where 'XX' refers to the Alphabot2's label "ROBOLAB XX". For example:

```
Robot label: ROBOLAB 08
IP Address: 43.11.0.208
```

With the VNC Viewer installed in your system, enter the IP address of the robot into this bar. Please make sure you are on the same network.

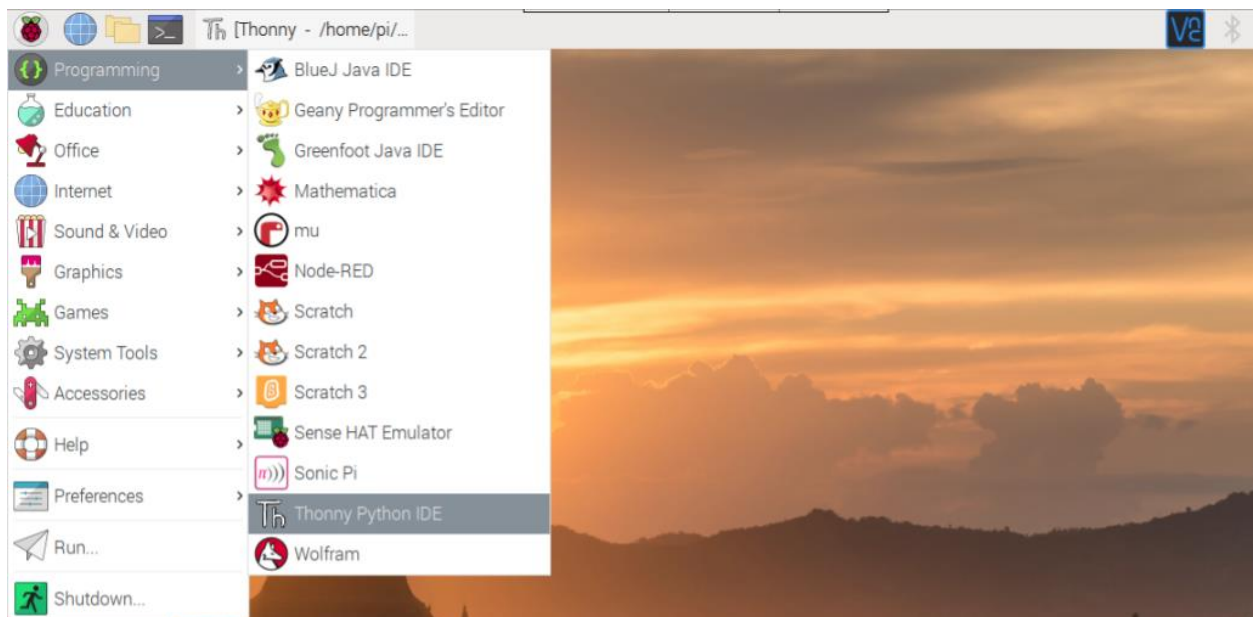


VNC Viewer is a tool used to control the AlphaBot2 by remotely accessing it from your machine (computer). It shows the display of your Raspberry Pi just as it is connected as the display (e.g., monitor).



3 Programming environment for Raspberry Pi

The **IDE** (Integrated Development Environment) you will use in this workshop is **Thonny**, which is an IDE for Python designed for beginners.

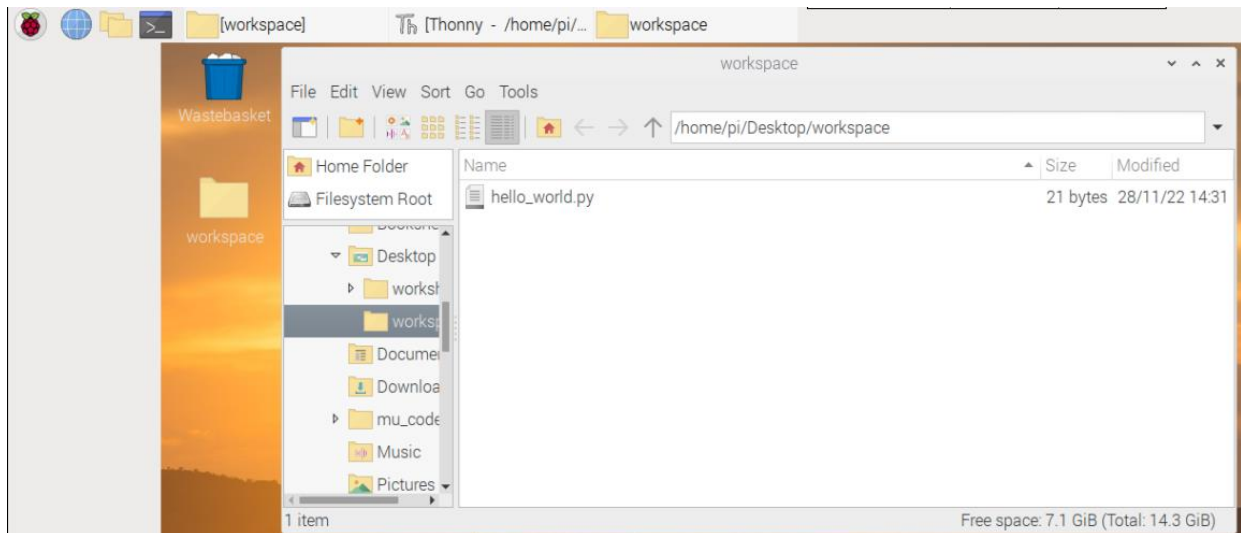


Before we get into programming the script of the robot, you will program a simple **Hello world script**.

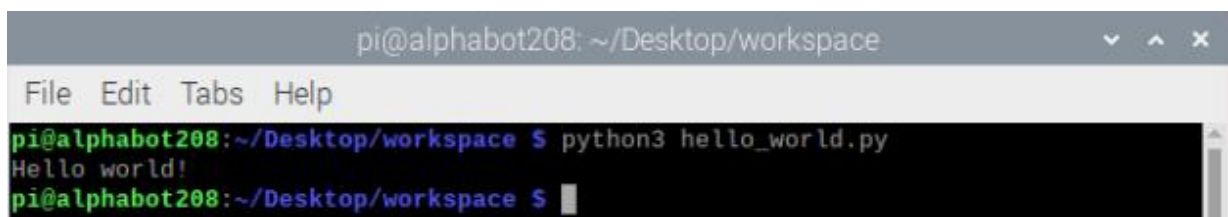


Create a new file, write the above Python command, and save the script as **hello_world.py** in the provided **workspace** folder.

You can run the script by pressing the green play button.



Alternatively, you can run the script by going to workspace folder in the file explorer, and on the menu bar, click **"Tools"** and **"Open Current Folder in Terminal"** or simply pressing **F4** in the workspace file explorer. This will open a command line interface where you can run your hello world script (`python3 hello_world.py`).



All your robot scripts will be saved and run from this folder. Now, you will be ready to program the Alphanot2!

4 Inputs and outputs of Raspberry Pi

The **Raspberry Pi** is the brain of the Alphanot2.

Raspberry Pi uses Input/Output pins which are called **General Purpose Input Output (GPIO)**. These pins allow us to programmatically interact with electronic devices, which broadly distinguished as input or output devices

The idea of **output** is to send signal to a GPIO (pin) of the RPi from within the RPi by its program. The GPIO will have an electrical voltage of +3.3V when a High/Logic 1 is sent to it. When a Low/Logic 0 is sent to the GPIO, there will be no voltage at the GPIO. Based on the voltage at the GPIO, an output device (e.g. a light) connected to the GPIO will be turned ON or OFF.

The idea of **input** is to receive signal at the GPIO from an input device (e.g. a sensor, a switch). The signal is in the form of electrical voltage. Raspberry Pi is a 3.3V device. To give it a signal of HIGH (logic 1) at its GPIO pin, an electronic device (or circuit) needs to provide +3.3V. Providing higher voltage (e.g. +5V) may damage the GPIO circuit of the Raspberry Pi. Many components or

devices designed for microcontroller (MCU) are 5V devices, i.e. their HIGH (logic 1) is at +5V. It is important to modify the signal from these components or devices with additional components to scale down the voltage to +3.3V.

Below is the pin layout of the GPIO header on the Raspberry Pi.

Raspberry Pi B+, 2, 3 & Zero				Legend
3V3	1	2	5V	Physical Pin Number
GPIO 2	3	4	5V	Power +
GPIO 3	5	6	GND	Ground
GPIO 4	7	8	GPIO 14	UART
GND	9	10	GPIO 15	I2C
GPIO 17	11	12	GPIO 18	SPI
GPIO 27	13	14	GND	GPIO
GPIO 22	15	16	GPIO 23	Do Not Connect
3V3	17	18	GPIO 24	
GPIO 10	19	20	GND	
GPIO 9	21	22	GPIO 25	
GPIO 11	23	24	GPIO 8	
GND	25	26	GPIO 7	
DNC	27	28	DNC	
GPIO 5	29	30	GND	
GPIO 6	31	32	GPIO 12	
GPIO 13	33	34	GND	
GPIO 19	35	36	GPIO 16	
GPIO 26	37	38	GPIO 20	
GND	39	40	GPIO 21	

5 Robot actuators

The goal of this section is to program the robot to control its **locomotion** (maneuver) and **pan-tilt** the arm/stick actuator through **teleoperation** (remotely). The actuators used in this section will be the **motors**, **servos** and **buzzer**.

5.1 Motor teleoperation and pan-tilt camera

The Alphabot2 uses two **geared DC motors** to drive itself and maneuver. This set of motors is usually referred to as the **drivetrain** motors.



N20 micro gear motor



N20 with encoder

Besides the two geared motors, the Alphabot2 uses two **servo motors** to pan and tilt its camera.



SG90 micro servo motor (standard)

We will use **W, A, S, D,** and **space bar** keys for teleoperating the driving of the robot. The **arrow keys** will be used to pan and tilt the camera and X key will be used to reset the pan and tilt position.

```
W = Robot moving forward
A = Robot turning left
S = Robot moving backward
D = Robot turning right
[space bar] = Robot stop moving

↑ = Tilting camera up
← = Panning camera left
↓ = Tilting camera down
→ = Panning camera right
X = to reset pan and tilt position
```

We will first start with motor teleoperation. Let's get started with programming. Create a new (empty) file and save it as **teleop.py**.

To achieve our goal for this section, first, we need to import these necessary packages below:

```
import curses
import time
from AlphaBot2 import AlphaBot2
```

Note:

'curses' is a library that supplies keyboard-handling. 'time' is a python module to handle time-related tasks. AlphaBot2 library is used to control the drivetrain motors.

Then, initialize this AlphaBot2 object to use the motors:

```
# Alphabot motor object
Ab = AlphaBot2()
```

We will also need to initialize curses settings which handles the keyboard:

```
# Curses keyboard input settings
screen = curses.initscr()
curses.noecho()
curses.cbreak()
screen.keypad(True)
```

Then, in a 'try' block, using a while loop to do set W, A, S, D keys and space bar to map with the actions:

```
try:
    while True:
        char = screen.getch()

        # Drivetrain motors teleoperation - WASD keys
        elif char == ord('w'):
            Ab.forward(40)

        elif char == ord('a'):
            Ab.left(40)

        elif char == ord('s'):
            Ab.backward(40)

        elif char == ord('d'):
            Ab.right(40)

        elif char == ord(' '): # space bar
            Ab.stop()
```

Finally, a 'finally' block to close objects and clean up resources.

```
finally:
    curses.nocbreak(); screen.keypad(0); curses.echo(0)
    curses.endwin()
```

Run the **teleop.py** (**sudo python3 teleop.py**) and try to control the robot with your keyboard

Next, to control pan-tilt, create a new (empty) file and save it as **pan_tilt.py**. Then, import these necessary packages below:

```
import curses
```

```
import time
from PCA9685 import PCA9685
```

Note:

'curses' is a library that supplies keyboard-handling. 'time' is a python module to handle time-related tasks. 'PCA9685' is a library to control the Pulse-width modulation (PWM) of the servos.

Then, initialize PCA9685 object to use the servos:

```
# Pan and tilt object
pwm = PCA9685(0x40)
pwm.setPWMFreq(50)
```

Afterwards, set these configurations for pan-tilt settings:

```
# Tilt settings (up and down)
tilt_channel = 1      # Tilt motor is on channel 1 of PCA9685 mod.
tilt_pulse = 1200    # Sets the initial pulse width
tilt_pulse_step = 30 # Sets the initial step size

# Pan settings (left and right)
pan_channel = 0      # Pan motor is on channel 0 of PCA9685 module
pan_pulse = 1700     # Sets the initial pulse width
pan_pulse_step = 30  # Sets the initial step size
```

We will also need to initialize curses settings which handles the keyboard:

```
# Curses keyboard input settings
screen = curses.initscr()
curses.noecho()
curses.cbreak()
screen.keypad(True)
```

Then, in a 'try' block, using a while loop to do pan-tilt using arrow and X keys:

```
try:
    while True:
        char = screen.getch()

        # Pan and tilt teleoperation - arrow keys
        if char == curses.KEY_UP:
            pwm.setServoPulse(tilt_channel, tilt_pulse -
                               tilt_pulse_step)
            tilt_pulse -= tilt_pulse_step

        elif char == curses.KEY_DOWN:
            pwm.setServoPulse(tilt_channel, tilt_pulse +
                               tilt_pulse_step)
```



```

        tilt_pulse += tilt_pulse_step

    elif char == curses.KEY_RIGHT:
        pwm.setServoPulse(pan_channel, pan_pulse -
            pan_pulse_step)
        pan_pulse -= pan_pulse_step

    elif char == curses.KEY_LEFT:
        pwm.setServoPulse(pan_channel, pan_pulse +
            pan_pulse_step)
        pan_pulse += pan_pulse_step

    elif char == ord('x'): # X to reset
        pan_pulse = 1700
        pwm.setServoPulse(pan_channel, pan_pulse)
        tilt_pulse = 1200
        pwm.setServoPulse(tilt_channel, tilt_pulse)

```

Finally, a 'finally' block to close objects and clean up resources.

```

finally:
    curses.nocbreak(); screen.keypad(0); curses.echo(0)
    curses.endwin()

```

Run the **pan_tilt.py** (*python3 pan_tilt.py*) and try to control the direction of the robot's camera with your keyboard.

5.2 Buzzer

A **buzzer** is another actuator component that can be used to give audible signal in any application. We can program to switch on the buzzer using **RPi.GPIO** python library.



A buzzer

Create a new file and save it as **buzzer.py**.

First, import RPi.GPIO library and set the buzzer channel to 4 to access the GPIO pin:

```

import RPi.GPIO as GPIO
Import time
BUZ = 4

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(BUZ, GPIO.OUT)

```

GPIO has two modes of pin-numbering scheme, BOARD and BCM. We will use **BCM mode** in which we specify each GPIO by its channel number (instead of physical pin number). Warning channel for GPIO is disable. Since Buzzer is an output device, we will set it as GPIO.out.

Below is the function to switch on the buzzer:

```
def beep_on():
    GPIO.output(BUZ, GPIO.HIGH) # beep on

def beep_off():
    GPIO.output(BUZ, GPIO.LOW) # beep off
```

Now, in a 'try' block, we call out the beep_on function for 3 seconds using time.sleep() function for delay then call out the beep_off function to switch off:

```
if __name__ == '__main__':
    try:
        beep_on()
        time.sleep(3)
        beep_off()
```

We will also need to create GPIO clean up when shutting down the program with Ctrl+c:

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

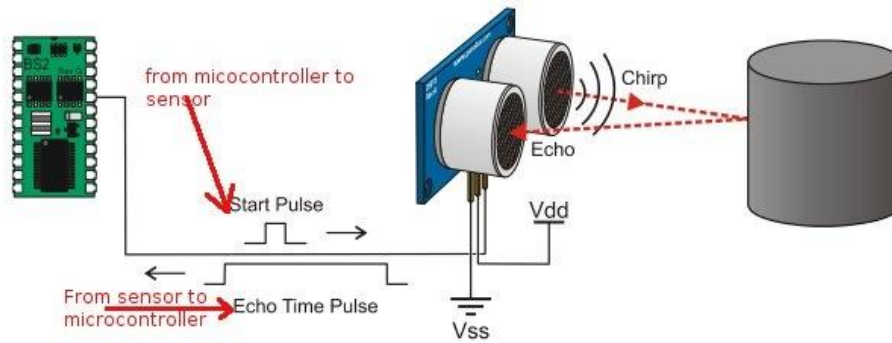
Run the **buzzer.py** (*python3 buzzer.py*) and hear the beeping sound.

6 Robot Sensors

In this section, you will learn to program **ultrasonic** and **infrared** sensors to obtain information about the robot's environment.

6.1 Ultrasonic (Ping) sensor for range measurement

The Alphabot2-Pi has an **HC-SR04 Ultrasonic sensor**. This sensor is often called a **Ping sensor**. It works by emitting ultrasonic (sound) signal (kind of send a ping signal) and detect the echo from an object in front. By computing the time to receive the echo, we can determine the distance (range) of the object in front (if any).



Now, it is time to program the ping sensor to detect the distance of an object.

Create a new file and save it as **ping.py**.

First, import the necessary packages below. We will use RPi.GPIO again to program the ping sensor.

```
import RPi.GPIO as GPIO
import time
```

Next, set the GPIO pins used for the ping sensor, which are TRIG=22 and ECHO=27. GPIO.BCM is the pin-numbering system for chosen for this program and the warnings for GPIO is disabled. Since the ping sensor has both input and output pins, we need to set up the pins according to the type. TRIG pin is that emitting the signal; hence, it is set up as GPIO.OUT with the initial signal set as GPIO.LOW, which means no voltage is set. Lastly, as ECHO is the one that receives the echo signal, the GPIO is set to GPIO.IN.

```
TRIG = 22
ECHO = 27

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(TRIG, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(ECHO, GPIO.IN)
```

The function below is used to determine the distance between the ping sensor and the object detected:

```
def dist():

    GPIO.output(TRIG, GPIO.HIGH)
    time.sleep(0.000015)
    GPIO.output(TRIG, GPIO.LOW)

    while not GPIO.input(ECHO):
        pass

    t1 = time.time()
```

```

while GPIO.input(ECHO):
    pass

t2 = time.time()
return (t2-t1) *34000/2

```

Lastly, print the distance detected from the function created above. We also set to clean up the GPIO when the program is terminated with Ctrl+c:

```

if __name__ == '__main__':
    try:
        while True:
            print("Distance: %0.2f cm" % dist())
            time.sleep(1)
    except KeyboardInterrupt:
        GPIO.cleanup()

```

Run the **ping.py** (*python3 ping.py*) and observe the readings printed on the Terminal as you move an obstacle in front of the robot.

6.2 Tracker (TR) / Line Sensors for infrared line tracking

The Alphabot2-Pi has one **Tracker (TR) / Line Sensors** for infrared line tracking. The Line Sensor features five analog outputs, providing data influenced by both distance and the color of the detected object. Objects with higher infrared reflectance (black) generate larger output values, while those with lower infrared reflectance (white) produce smaller values. As the sensor approaches a white line, the values decrease. The analog output offers more precision compared to modules with only logical outputs (HIGH/LOW). This is done within the electronic circuit and is not programmable. Unlike ping sensors, Line sensors only indicate if the robot is within the white surface, i.e., the lane.



Tracker / Line Sensor, robot infrared line tracking

Now, using the provided white paper, we can program the Line sensors to assist the robot in detecting infrared line reflectance. Create a new file and save it as **Line_detect.py**.

Since Line Sensors are connected using GPIO pins, we will use RPi.GPIO. Import these packages and AlphaBot2 object. We also need to initialize the Line Sensor object:

```
import RPi.GPIO as GPIO
From TRSensors import TRSensors
Import time

TR = TRSensor() #initialize Line Sensor object
```

Then, set the rest of the GPIO configurations. We will use BCM as the GPIO pin numbering system. We will also be turning off the warnings if the pins are used. We will set the Line sensors as input pins.

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
```

Before using the Line sensor, we must provide them with a calibration value for the array of sensors. Given the inherent inaccuracies in surface measurements within the environment, this calibration ensures reliable and precise sensor performance. We measure the reflectance of both the white and black surfaces. As black paper is **not provided**, we'll utilize the **table surface** as a reference for the black surface.

- For consistent calibration values, we'll use a 'for' loop to obtain four readings from the Line sensor, to compute statistics, i.e., the average of the infrared reflectance.
- We will code an input prompt to ensure the robot is on both white and black surfaces, i.e., the table surfaces
- Place the robot on white surface, press Enter to initiate the calibration process, executed by the **TR.calibrate()** function. And we'll repeat this process on a black surface.
- This calibration aims to establish baseline values for the line tracking sensors in response to the reflectance differences between white and black surfaces.

```
for i in range(0,4):
print("Place the robot on white surface. Please ensure all line
tracking sensors are above the white surface.")
    input("Press Enter to continue...")
    TR.calibrate()
    print("Place the robot on black surface. Please ensure all line
tracking sensors are above the black surface.")
    input("Press Enter to continue...")
    TR.calibrate()
```

Next, we will analyze the calibrated values for our white and black reflectance surfaces. Press the Enter key when you are done.

```
print("Calibrated Min: ", TR.calibratedMin)
print("Calibrated Max: ", TR.calibratedMax)
input("Press Enter to continue...")
```

With the while loop with 'True' as condition, we retrieve the Line sensors status from the GPIO.input and the following observations must satisfy:

- The terminal will display a 'list', [x, x, x, x, x], where each value inside the list corresponds to each of the Line sensors.
- If the Line sensors are placed at a white surface, the reflectance should be near 0
- If the Line sensors are placed on a black surface I.e. table surface in our case, the reflectance value should be near 1000.

```
while True:
    position,Sensors = TR.readLine()
    print(position,Sensors)
    time.sleep(0.25)
```

Run the **Line_detect.py** (**python3 Line_detect.py**) and test the program by analyzing the change in values resulting when you place the robot on a different surface.

- The End -