ROBOLAB
School of Digital Science
Universiti Brunei Darussalam

A Taste of Robot Programming

UBD STEAM Festival 2025

This document can be downloaded at https://ailab.space/events/ubdrws2025.

Contents

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**?
- How to **program a Raspberry Pi** single-board computer?
- How to program the **control of the actuators**?
- How to program **reading data from the sensors**?
- How to **remotely view the image of the camera** on the robot?

We will skip in-depth details to keep the time short for the workshop. You can read the details given on this handout and research further any missing details after the workshop. You are encouraged to do so!

You will use this skillset to program a mobile robot, in collaboration with other robots, in competing in a game.

This workshop will use **Python** as the programming language. Note Python is **case sensitive**.

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 on one robot.

AlphaBot2

AlphaBot2 is equipped with **Raspberry Pi 3** (or **Raspberry Pi Zero W**), which serves as the **brain** (computer, controller) of the robot. Raspberry Pi is actually a **mini computer**. The source of power of AlphaBot2 would be either **battery** or using **power adapter**. When developing the programs, you will use the robot with the power adapter most of the time. You will need to untether, i.e. not connected to any wire, your robot when you want it to move around. In this case, you will use batteries to power your robot.

Raspberry Pi 3

Raspberry Pi Zero W

2 Remote access the robot

All AlphaBot2 are connected to "**robolab2**" network. You can program the robot remotely from your computer. To remote access your robot's software environment, your computer needs to be **connected to the same network**. Please connect your computer to the same network as the robot.

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

Each Alphabot2's **IP address** (network address) follows the format of **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
```

You will use **Putty** to run programs in the robot and **WinSCP** to access the files on the robot. All the programs for this workshop will be stored in the **Desktop > workspace** folder.

3 Programming environment for Raspberry Pi

To write programs, you can use a **Text Editor** or an **Integrated Development Environment** (**IDE**). IDE is a software designed to write programs and to convert (compile or interpret) the programs for the computer to execute (run).

In this workshop, you will use **NotePad++** to write and edit the robot's program. NotePad++ is a lightweight text editor with syntax highlight (coloring the program codes).

Before you get into writing the programs for the robot, you will program a simple Python **Hello world** program. Use WinSCP to create a file '**hello_world.py**'. And run the program in Putty.

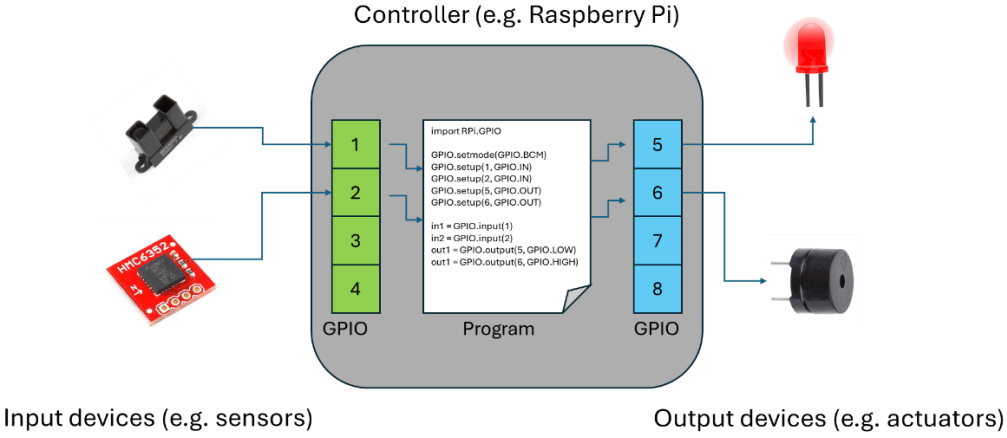
```
pi@alphabot208: ~ $ cd ~/Desktop/workspace
```

```
pi@alphabot208: ~/Desktop/workspace $ python3 hello_world.py
```

4 Inputs and outputs of Raspberry Pi

The **Raspberry Pi** (**RPi**) is the brain of the Alphabot2.

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 are broadly distinguished as **input** or **output** devices.



The idea of **output** is to send an electronic signal to a GPIO pin of the RPi from the RPi's processor running its program. An output device connected to the GPIO pin will do something, i.e. take an action. For example, a light bulb (LED) may turn ON/OFF, a buzzer may buzz or a motor may turn or change speed. The light bulb (LED), buzzer and motor are **output devices**. They are also called **actuators** since they cause an action in the environment.

The idea of **input** is to read the electronic signal at a GPIO pin of the RPi into the RPi's processor by its program. An **input device** connected to the GPIO pin will provide an electronic signal corresponding to what it is detecting or sensing. For example, a touch switch may signal if it is being touched, a temperature sensor may provide signal corresponding to the temperature in its environment or a light sensor may signal if the ambient light is bright or dim. The touch switch, temperature and light sensors sense their environment and are called **sensors**.

You will next learn to program the RPi to give output (takes actions in its environment) and read input (senses its environment).

Let's get started with **robot programming**!

5 Robot actuators

You want the robot to do something, i.e. take actions. The goal of this section is to program the robot to control its **locomotion** (maneuver) and **pan-tilt** the camera for live streaming through **teleoperation** (remotely). The actuators used in this section will be the **motors**, **servos** and **buzzer**.

5.1 Buzzer

A **buzzer** is an actuator device that can be used to give audible signal. 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**.

In Python (and other programming languages), we make use of pre-built functions in **libraries** or **packages** as much as possible. For RPi, 'RPi.GPIO' is the library that has pre-built functions to program the GPIO of the RPi. To use the libraries, you first import them at the beginning of the program codes.

```
# Import the libraries
import RPi.GPIO as GPIO
import time
```

The **'#'** is a way to write comments in Python. **Comments** are your own sentences for your own references. They will not be run by the Python interpreter.

The **'time'** library will be used to program time related functions.

To know about the functions in a library or package, you can refer to its documentation usually provided on a website.

The buzzer on Alphabot2 is connected to the GPIO "pin 4" of the RPi. So, you will specify this information in the 'RPi.GPIO' functions.

```
BUZ = 4                         # Define buzzer's GPIO pin

GPIO.setmode(GPIO.BCM)          # Use BCM mode for GPIO numbering
GPIO.setwarnings(False)         # Disable warning on GPIO
GPIO.setup(BUZ, GPIO.OUT)       # Set GPIO pin BUZ as output pin
```

The last instruction in the above code snippet could be rewritten **'GPIO.setup(4, GPIO.OUT)'**. It sets channel 4 as an output pin.

However, it is good practice to use a container or **variable** so that the value can be reused in other instructions, and you just need to edit the variable (one instruction) if the value changes. In this case, you use the variable 'BUZ' to store the channel for the buzzer. Note Python is **case sensitive**, e.g. BUZ is different from Buz or buz.

RPi's GPIO has two modes of **pin-numbering scheme**, BOARD and BCM. The program uses **BCM mode** in which each GPIO is specified by its channel number (instead of physical pin number). Warning is disabled for GPIO since we don't want to deal with any GPIO issues.

Next, write two functions: one to turn ON the buzzer, one to turn OFF the buzzer. You simply use the function from the 'RPi.GPIO' library to set the channel as HIGH (to turn ON) or LOW (to turn OFF). In Python, you create a reusable **function** using the **'def'** keyword; **'def beep_on():'** defines the function 'beep_on()'.

```
# Function to turn ON the buzzer (beep)
def beep_on():
     GPIO.output(BUZ, GPIO.HIGH) # Turn ON (HIGH) buzzer

# Function to turn OFF the buzzer (beep)
def beep_off():
     GPIO.output(BUZ, GPIO.LOW) # Turn OFF (LOW) buzzer
```

Next, in a **'try'** block, call the **'beep_on()'** function for 3 seconds by adding a time delay using the **'time.sleep()'** function before calling the **'beep_off()'** function to switch off the buzzer. The first line is the Python way of starting your **main program**. The **'try'** block is a way to capture errors during the running of the codes within the block.

```
if __name__ == '__main__':       # Main program
    try:
        beep_on()                # Beep ON
        time.sleep(3)            # Wait 3 seconds
        beep_off()               # Beep OFF
```

The 'try' block is followed by the **'except'** block to do something when there is an error. In this case, you want to capture keyboard interrupt (Ctrl+c) and end the program when Ctrl+c is pressed. When that happens, clean up the GPIO before exiting the program.

```
    except KeyboardInterrupt:    # Catch keyboard interrupt (Ctrl+c)
        GPIO.cleanup()           # Clean up GPIO
```

Save and run the **buzzer.py** and hear the beeping sound!

```
pi@alphabot208: ~/Desktop/workspace $ python3 buzzer.py
```

5.2 Teleoperate robot locomotion

Locomotion refers to the motion of moving from one place to another. 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

You will use **W, A, S, D,** and **space bar** keys for teleoperating the driving of the robot.

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

Teleoperation or **remote** operation refers to operating the robot at a distance. In this case, you will teleoperate the robot from our laptop while the robot can be (without physical connection) anywhere else (as long as connected to the network).

Create a new (empty) file and save it as **teleop.py**.

First, import the necessary packages below.

```
# Import libraries
import curses
from AlphaBot2 import AlphaBot2
```

Note:
'curses' is a library that provides screen-painting and keyboard-handling for text-based terminals. 'curses' basically allows programmers to create text-based user interfaces (TUIs). **'AlphaBot2'** library is used to control the drivetrain motors.

The 'AlphaBot2' library is used to create an 'AlphaBot2' object for our program. This object (**'Ab'**) will be used to call functions available in the library throughout our program, e.g. to control the motors.

```
# Create AlphaBot2 object
Ab = AlphaBot2()
```

Create a 'curses' object and initialize the settings to handle the keyboard. The 'curses' object (**'screen'**) will be used to call functions available in the library, e.g. to read the keyboard or to format the screen display. In this case, you only use it to read the keyboard. Details of the functions can be found from the documentation of 'curses' package.

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

In a 'try' block, use a **'while'** block to map **W, A, S, D** keys and **space** bar to the correct actions. A **'while True'** block is a continuous (endless) loop. The block (loop) reads the character entered at the keyboard and checks which character it is. It then calls the corresponding function of the 'AlphaBot2' object ('Ab') according to the key (character) being pressed. No action will be taken if there is no match. In programming, the **'if-elif-else'** serves the purpose of checking different conditions in the program. **'elif'** means **'else if'** in Python.

You can adjust the speed in the parentheses if your robot is too fast (strong) or slow (weak).

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

         # Drivetrain motors teleoperation - WASD keys
         if char == ord('w'):      # If 'w' is pressed
             Ab.forward(20)        # Move forward at speed 20

         elif char == ord('a'):    # If 'a' is pressed
             Ab.left(20)           # Turn left at speed 20

         elif char == ord('s'):    # If 's' is pressed
             Ab.backward(20)       # Move backward at speed 20

         elif char == ord('d'):    # If 'd' is pressed
             Ab.right(20)          # Turn right at speed 20

         elif char == ord(' '):    # If space bar is pressed
              Ab.stop()            # Stop
```

At the end, use the **'finally'** block to close objects and clean up resources. The 'finally' block goes along with the 'try' block. It differs from the 'exception' block in that it will be executed (run)

when the program exit the try block with or without exception. You could use the 'exception KeyboardInterrupt' instead.

```
# Clean up Curses
finally:
    curses.nocbreak(); screen.keypad(0); curses.echo(0)
    curses.endwin()
```

Save and run the **teleop.py** and try to control the robot with your keyboard.

```
pi@alphabot208: ~/Desktop/workspace $ python3 teleop.py
```

Use **'Ctrl+c'** (press Ctrl and C keys simultaneously) to exit the program.

6 Robot Sensors

Taking actions within an environment without knowing what's the environment is very blind folded. In this section, you will learn to program the **camera**, **ultrasonic**, **infrared** and **floor/line** sensors to obtain information about the robot's environment.

6.1 Camera streaming

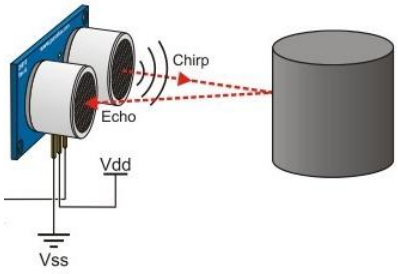
Camera is an important sensor for the robot to see its environment. In teleoperation, the robot can show us the situation at the remote places, which are not safely reachable by humans. In this case, we require **camera streaming** from the location of the robot to our computer screen. The camera streaming can also be sent over the network and displayed on a webpage.

We will not go through the codes as it would require more time than we have for this workshop. However, you can view the streaming of the camera from your instructor's robot.

Open a **browser** and enter the IP and port of your instructor's robot camera (http://43.11.0.220:1064). You should see the robot's camera view. You can view this from any computer including your mobile phone, as long as they are connected to the same network (same wifi).

6.2 Ultrasonic (Ping) sensor for range (distance) measurement

The Alphabot2 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).



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

Import the necessary packages. Use **'RPi.GPIO'** again to program the ping sensor.

```
# Import libraries
import RPi.GPIO as GPIO
import time
```

Set the GPIO pins used for the ping sensor, which are TRIG=22 and ECHO=27. GPIO.BCM is the pin-numbering system chosen for this program and the warnings for GPIO is disabled. Since the ping sensor has both input and output pins, you 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. ECHO is the one that receives the echo signal, the GPIO is set to GPIO.IN.

```
TRIG = 22                   # GPIO pin of trigger
ECHO = 27                   # GPIO pin of echo

# Configure GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
# Initialize TRIG and ECHO pins
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.

```
# Function to compute distance
def dist():
     GPIO.output(TRIG, GPIO.HIGH)      # Send trigger pulse
     time.sleep(0.000015)              # Wait a bit
     GPIO.output(TRIG, GPIO.LOW)       # Turn off trigger pulse

     while not GPIO.input(ECHO):       # Waiting for ECHO to go HIGH
          pass

     t1 = time.time()                  # Get current time
```

```
    while GPIO.input(ECHO):          # Waiting for ECHO to go LOW
        pass

    t2 = time.time()                 # Get current time
    return (t2-t1) *34000/2          # Compute distance and return
                                     # the value
```

Lastly, print the distance detected from the function created above. Clean up the GPIO when the program is terminated with Ctrl+c.

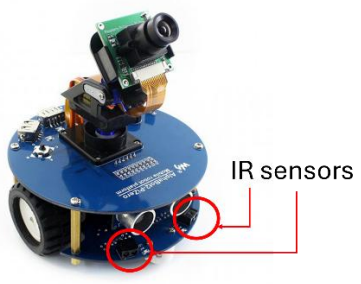
```
if __name__ == '__main__':
    try:
        while True:
            print("Distance: %0.2f cm" % dist())
            time.sleep(1)
    except KeyboardInterrupt:
        GPIO.cleanup()
```

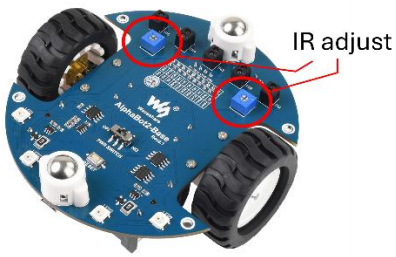
Save and run the **ping.py** and observe the readings printed on the Terminal as you move an object in front of the robot.

```
pi@alphabot208: ~/Desktop/workspace $ python3 ping.py
```

6.3 Infrared (IR) sensors for obstacle detection

The Alphabot2 has two **infrared (IR) sensors** for detecting obstacles. There are two variable resistors (think of these as adjustable knobs) to adjust the sensitivity of the sensors (think of this as how far it can detect).





The IR sensors on the AlphaBot2 have been connected in an electronic circuit with LED lights. Each sensor has an LED light that will light up if the sensor receives a reflection from an object in front. This is done within the electronic circuit and is not programmable. Unlike ping sensors, IR sensors only tell us that an obstacle is within their proximity (detection range); IR sensors do not usually provide information of distance.

Now, you can program the IR sensors to help robots to detect an obstacle. Create a new file and save it as **ir.py**.

Since IR is connected using GPIO pins, 'RPi.GPIO' library is required. Import these packages:

```
# Import libraries
import RPi.GPIO as GPIO
import time
```

Next, set the pins for IR sensors. 'DR' is the right IR sensor and 'DL' is the left IR sensor.

```
DR = 19          # Right IR sensor's GPIO pin
DL = 16          # Left IR sensor's GPIO pin
```

Then, set the rest of the GPIO configurations. Use BCM as the GPIO pin numbering system and turn off the warnings. Set the IR sensors as input pins with pull up (pull up makes the input HIGH (1) when there is no reflection).

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(DL,GPIO.IN,GPIO.PUD_UP)     # Set DL pin as input
GPIO.setup(DR,GPIO.IN,GPIO.PUD_UP)     # Set DR pin as input
```

Now in the main method and in a 'try' block with a 'while True' loop, check the status of the IR sensors by retrieving them from GPIO.input:

- If the left or right IR sensor detects an obstacle, the status will return as '0'
- Else, when either sensor does not detect any obstacle, then the status will return as '1'

You will program the robot to detect obstacles according to these rules:

- When obstacles are detected at both sides, the robot will print "obstacle on the left and right side".
- When there is an obstacle on the left side, the robot will print "obstacle on the left side" to indicate an obstacle detected on its left side.
- When there is an obstacle on the right side, the robot will print "obstacle on the right side".
- Else, when there is no obstacle detected, the robot will print "no obstacle".

A **'print()'** instruction display a string of text on the Terminal screen. In this case, you are only seeing the **actions on the screen**. To actually move the robot accordingly, you will need to create an

AlphaBot2 object and use it to **move the robot** in place of where the 'print()' instructions are.

```
if __name__ == "__main__":
      try:
            while True:
                  DR_status = GPIO.input(DR)  # Read right sensor
                  DL_status = GPIO.input(DL)  # Read left sensor

                  if ((DL_status == 0) and (DR_status == 0)):
                        #when DL and DR status are 0
                        #this indicate that an obstacle is detected
                        print("obstacle on the left and right side")
                        #add robot moving instruction if required

                  elif ((DL_status == 0) and (DR_status != 0)):
                        #when DL is 0 and DR is not 0
                        #obstacle is detected on its left side
                        print("obstacle on the left side")
                        #add robot moving instruction if required

                  elif ((DL_status != 0) and (DR_status == 0)):
                        #when DL is not 0 and DR is 0
                        #obstacle is detected on its right side
                        print("obstacle on the right side")
                        #add robot moving instruction if required

                  else:
                        #when DL and DR is not 0
                        #this indicate that no obstacle is detected
                        print("no obstacle")
                        #add robot moving instruction if required
```

Finish the program with capturing keyboard "Ctrl+c" to stop the program and cleaning up the GPIO.

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

Save and run the **ir.py** and test the program by moving an obstacle in front of the robot.

```
pi@alphabot208: ~/Desktop/workspace $ python3 ir.py
```

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

The Alphabot2 has one **Tracker (TR) / Line Sensor** for infrared (IR) line tracking. These sensors are sensors that detect the

reflectiveness (due to color and texture) of the **floor**. The Line Sensor has five IR sensors providing signals about the surface below them (the floor). The values returned from the sensors are a measure of the reflectance in abstract units, with higher values corresponding to lower reflectance (e.g. a black surface or a void). So, if a white line on black floor (or a black line on white floor) is below the sensors, by knowing the reading of each sensor, we can detect the position of the line (whether it is to the right, left, center or not seen at all).

Instead of writing the codes ourselves to detect the line, we use the 'TRSensors' package. The package has the necessary codes and functions to detect a line under the robot. This is useful in line-following robots.

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

Import the 'TRSensors' and 'time' packages. Then create a TR Sensor object.

```
# Import libraries
from TRSensors import TRSensor
import time

TR = TRSensor() #initialize Line Sensor object
TR_threshold = [300, 300, 300, 300, 300]
# 5 values because there are 5 floor sensors, these values separate white and black, black has lower value (low reflectance) and white has higher value (high reflectance)
```

With the 'while True' loop, the line sensors values can be obtained using the 'AnalogRead()' function of the 'TRSensors' package.

- The terminal window will display a 'list', [x, x, x, x, x], where each value inside the list corresponds to each of the line sensors.
- If a sensor is above a black surface, the value should be low (low reflectance).
- If a sensor is above a white surface, the value should be high (high reflectance).

```
while True:
      sensors = TR.AnalogRead()        # Read TR sensor values
      left_sensor = 'Black'
      right_sensor = 'Black'
      print(sensors)
      if (sensors[0] > TR_threshold[0]):    # Compare value
            left_sensor = 'White'
      print('Left most sensor: ', sensors[0], left_sensor)

      if (sensors[4] > TR_threshold[4]):    # Compare value
            right_sensor = 'White'
      print('Right most sensor: ', sensors[4], right_sensor)
      time.sleep(0.25)
```

Save and run the **line_detect.py** and test the program by completing the calibration process and testing the robot on a white line over black surface.

```
pi@alphabot208: ~/Desktop/workspace $ python3 line_detect.py
```

- The End -