Exploring Robot Programming with Python

DevFest Brunei 2022

This document can be downloaded at https://ailab.space/events/devfest2022.

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**, 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**?
- How to **remotely view the image of the camera** on the robot?
- How to perform basic computer vision task of **color detection**?

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.

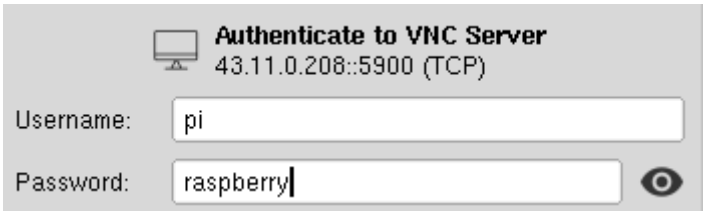
```
SSID: robolab2
Password: 12345678
```

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
```

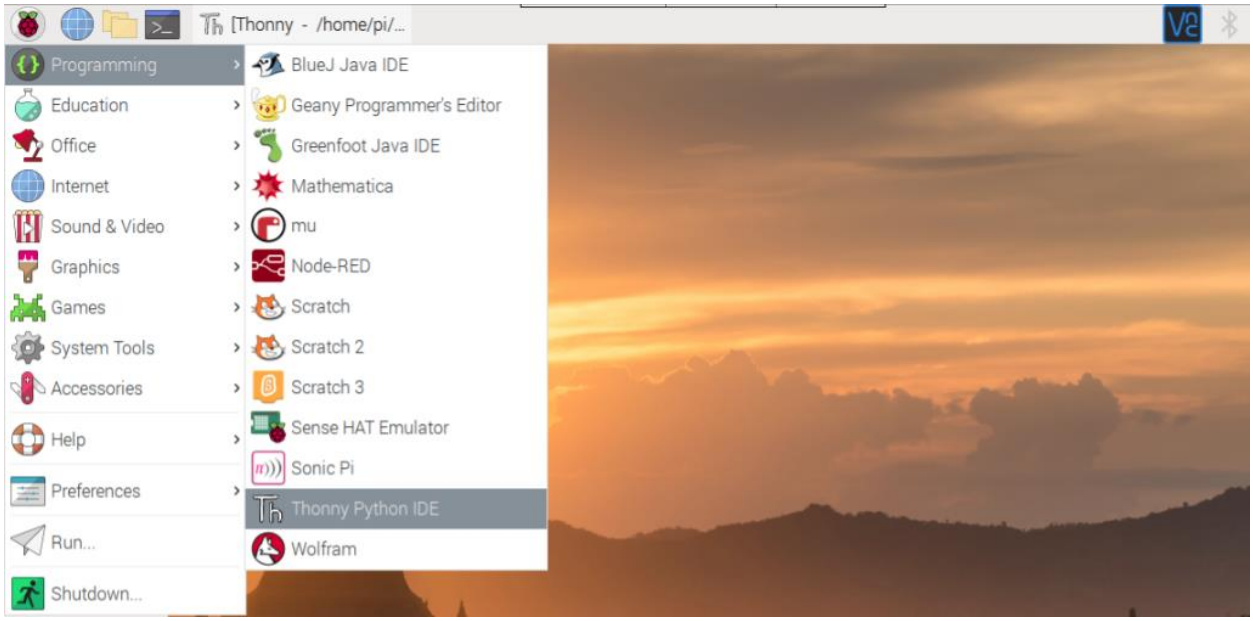
With the VNC Viewer installed in your system, enter the IP address of the robot into this bar. Please make sure you are on the same network.

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).

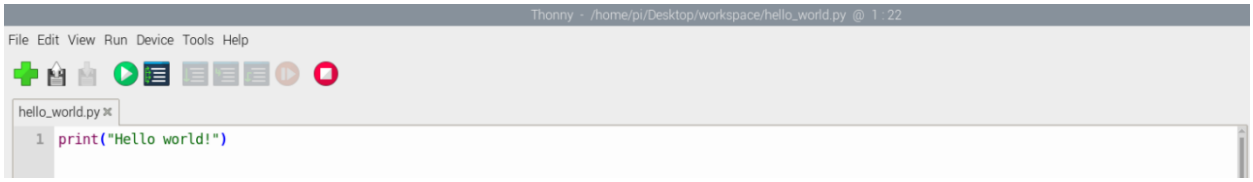


3 Programming environment for Raspberry Pi

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

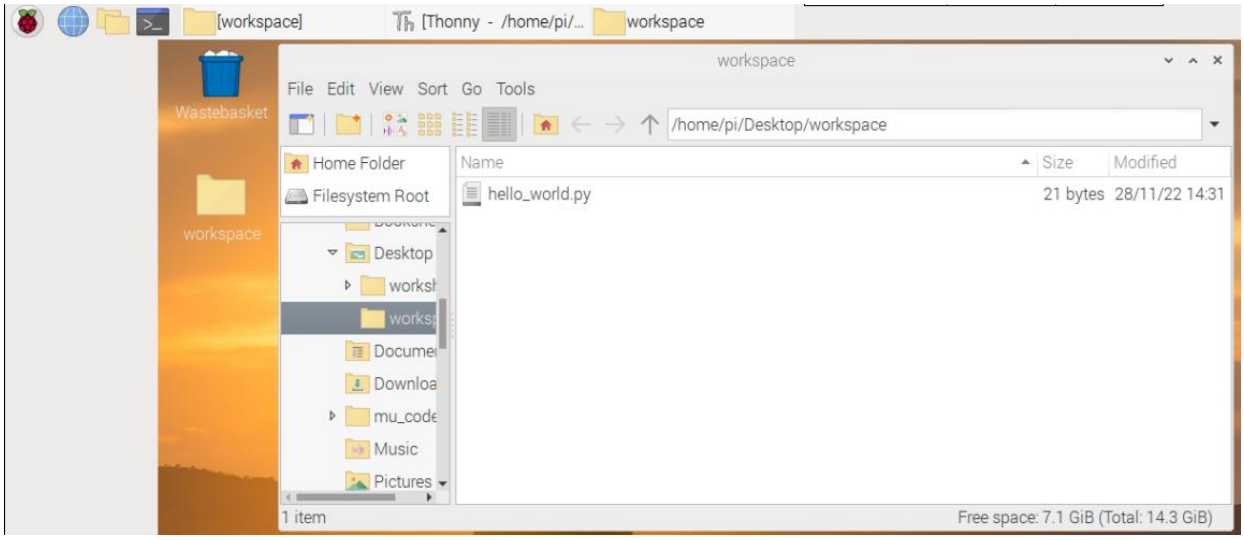


Before we get into programming the script of the robot, you will program a simple **Hello world script.**

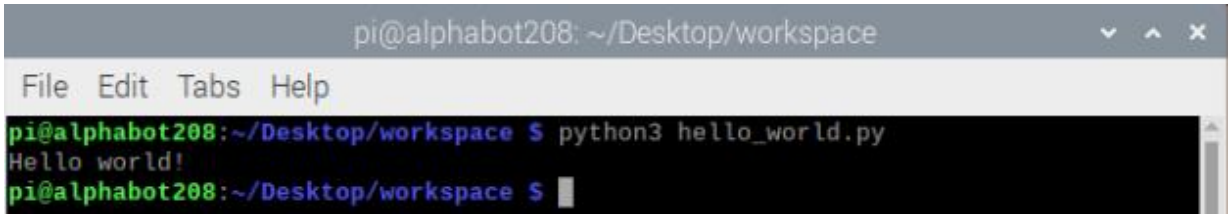


Create a new file, write the above Python command, and save the script as **hello_world.py** in the provided **workspace** folder.

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.



All your robot scripts will be saved and run from this folder. Now, you will be ready to program the Alphabot2!

4 Inputs and outputs of Raspberry Pi

The **Raspberry Pi** 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 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.

5 Robot actuators

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 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 curse 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(20)

         elif char == ord('a'):
             Ab.left(10)

         elif char == ord('s'):
             Ab.backward(10)

         elif char == ord('d'):
             Ab.right(10)

         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** 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 curse 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** 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** and hear the beeping sound.

6 Robot Sensors

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

6.1 Camera streaming

One of the examples of utilizing sensors is **camera streaming**, where we use cameras to perceive our environment. To do that, we can use **OpenCV** to display from the camera. The video is simply a (streaming of) sequence of images at the frequency (speed) that we perceive as continuous motion. Once executed, the program will launch a window to show the image received from the camera.

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

Import these necessary packages for this section to get started:

```
import os
import cv2
import gc
from multiprocessing import Process, Manager
```

OpenCV (cv2) is a python library that allows you to perform image processing computer vision tasks. Python's **garbage collector** (gc) detects objects with reference cycles and cleans up the memory after stack operation. A shared buffer stack is a temporary location created within a computer's memory for storing and retrieving data from the stack.

We will use stack in our program to store and stream the images of the environment received from the camera:

```
# Write data to the shared buffer stack:

def write(stack, cam, top: int) -> None:

    """
         :param cam: camera parameters
         :param stack: Manager.list object
         :param top: buffer stack capacity
    :return: None
    """

    print('Process to write: %s' % os.getpid())
    cap = cv2.VideoCapture(cam)
    while True:
        _, img = cap.read()
        if _:
            stack.append(img)

   # Clear the buffer stack every time it reaches a certain capacity
   # Use the gc library to manually clean up memory garbage to prevent
memory overflow

            if len(stack) >= top:
                del stack[:]
                gc.collect()
```

Then write a function to read data in the buffer stack. Also setting a condition to cancel the stack by pressing 'q' key to quit the process:

```
# Read data in the buffer stack:

def read(stack) -> None:

    """
        :param stack: Manager.list object
        :return: None
    """

    print('Process to read: %s' % os.getpid())
    while True:
```

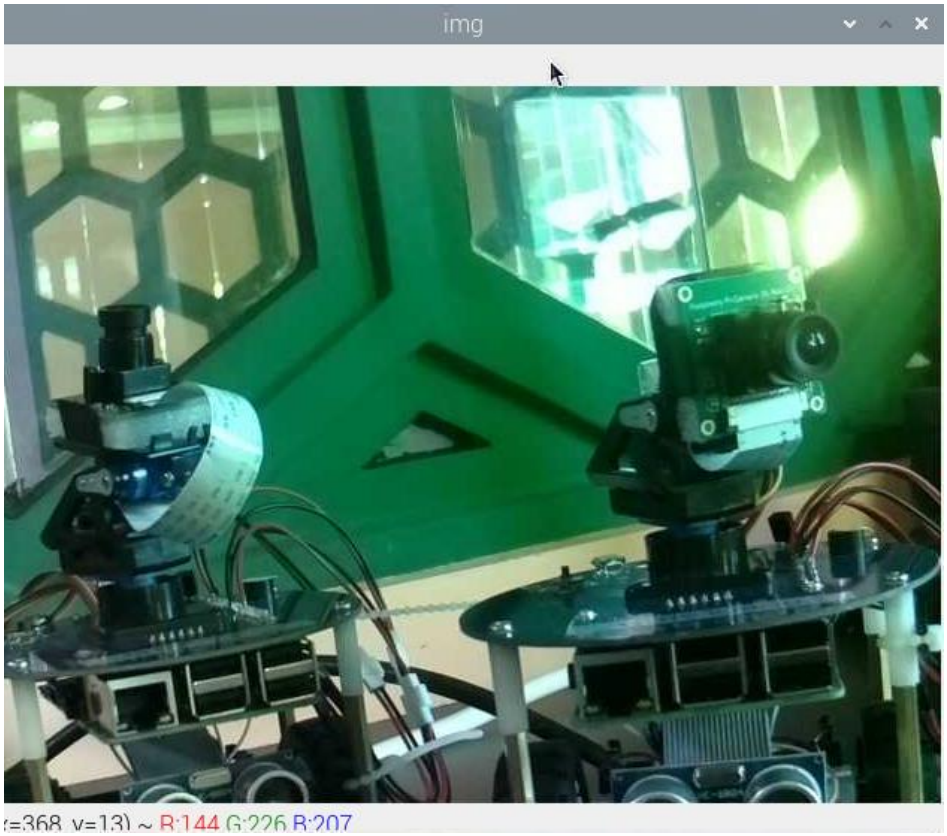
```
        if len(stack) != 0:
            value = stack.pop()
            cv2.imshow("img", value)
            key = cv2.waitKey(1) & 0xFF
            if key == ord('q'):
                break
```

The cv2.imshow() function displays the image on an image window.

Lastly, write the main function to create the buffer stack to stream the camera:

```
if __name__ == '__main__':

# The parent process creates a buffer stack and passes it to each child process.

# The buffer stack is used to share data/messages between processes.

    q = Manager().list()
    pw = Process(target=write, args=(q, 0, 100))
    pr = Process(target=read, args=(q,))

    # Start the child process pw, write:
    pw.start()

    # Start the child process pr, read:
    pr.start()

    # Wait for pr to end:
    pr.join()

# pw Process is an infinite loop, cannot wait for its end, can only be forced to terminate:
    pw.terminate()
```

Run the **camera.py**. You should see the live view of the Alphabot's camera. Below is the example of the output of this program:



We have used the **multiprocessing** capability of Python to run the camera streaming as a separate process in the OS. This will prevent the camera streaming process from blocking the main program and prevent the other processes (reading sensors, controlling actuators) from running. This is an important concept in programming real-time systems such as robots.

6.2 Color detection

Detecting color is another use case of using robot sensors. This is a common task in computer vision. It is useful in multiple fields like industries, e.g., sorting items according to colors, finding markers etc.

Create a new file and save it as **color_detection.py**. As usual, import the packages required:

```
import numpy as np
import time
import cv2
```

Each color will have different ranges. Thus, we need to set the boundaries of each color. Today, we will use red, green and blue. Then we define the color ranges and store them in the boundaries list:

```
# Initialise camera
```

```
camera = cv2.VideoCapture(0)
time.sleep(2.0)

# Define the list of boundaries for color mask

boundaries = [
    ([0, 0, 100], [50, 56, 255]), # red
    ([70, 120, 0], [100, 170, 40]), # green
    ([180, 50, 0], [200, 115, 40]), # blue
]
```

In a while loop with True as the condition, run the display with the color and print which color is detected, and set a rule to break the loop and stop the program by pressing 'Esc' key:

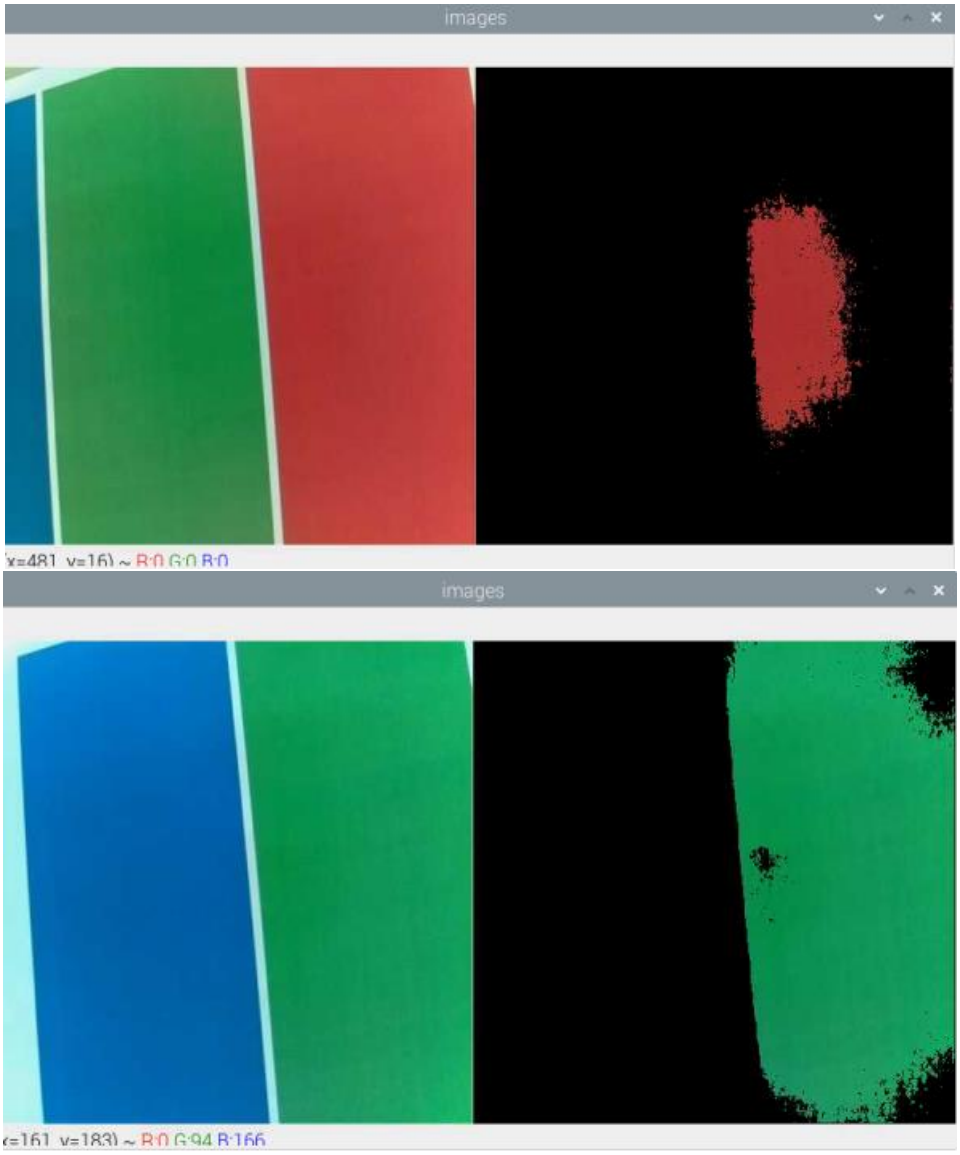
```
while True:
    okay, image = camera.read()
    image = cv2.resize(image, (400, 400))
    if not okay:
        continue

    # Create NumPy arrays from the boundaries
    lower1 = np.array(boundaries[0][0], dtype = "uint8")
    upper1 = np.array(boundaries[0][1], dtype = "uint8")

    lower2 = np.array(boundaries[1][0], dtype = "uint8")
    upper2 = np.array(boundaries[1][1], dtype = "uint8")

    lower3 = np.array(boundaries[2][0], dtype = "uint8")
    upper3 = np.array(boundaries[2][1], dtype = "uint8")

    mask1 = cv2.inRange(image, lower1, upper1)
    mask2 = cv2.inRange(image, lower2, upper2)
    mask3 = cv2.inRange(image, lower3, upper3)

    red_output = cv2.bitwise_and(image, image, mask = mask1)
    green_output = cv2.bitwise_and(image, image, mask = mask2)
    blue_output = cv2.bitwise_and(image, image, mask = mask3)

    if mask1.any():
        print("mask1: red!")
    if mask2.any():
        print("mask2: blue!")
    if mask3.any():
        print("mask1: green!")
    else:
        print("NONE")

    # Show the images
    cv2.imshow("images", np.hstack([image, green_output]))
```

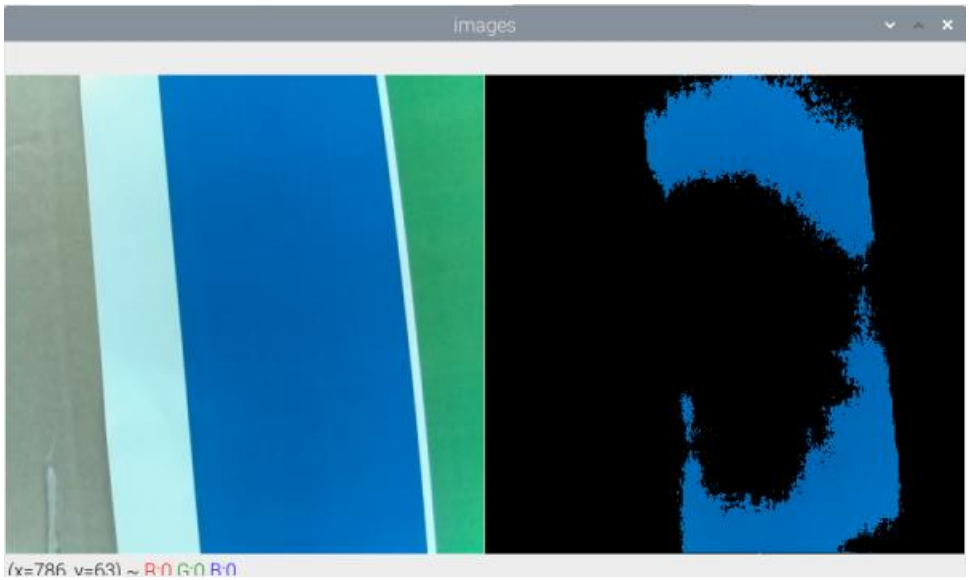
```
    # Allow correct exit point with a designated key,
    # ESC key in this case.
    # ESC key is detected at the application windows

    k = cv2.waitKey(30) & 0xff
    if k == 27: # press 'ESC' to quit
        break
```

Lastly, close the camera with camera.release() and destroy all display windows with cv2.destroyAllWindows() before shutting down the code.

```
# Clean up, release camera object
camera.release()
cv2.destroyAllWindows()
```

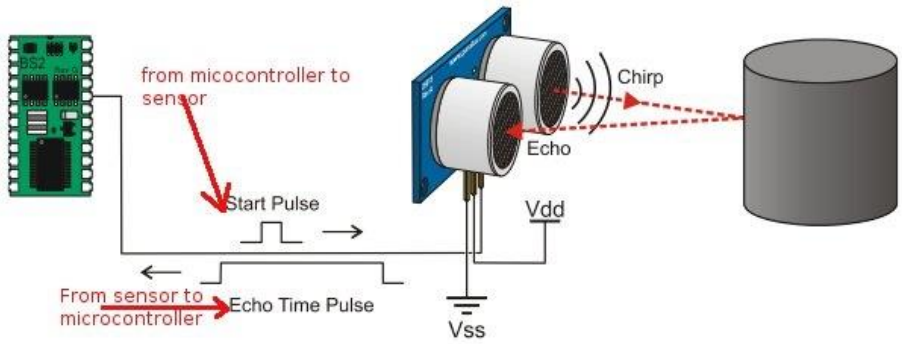
Run the **color_detection.py**. Below are the examples of the window output once the program executed, according to the respective color output:





6.3 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** and observe the readings printed on the Terminal as you move an obstacle in front of the robot.

6.4 Infrared (IR) sensors for obstacle detection

The Alphabot2-Pi has two **infrared (IR) sensors** for detecting obstacles. These IR sensors are electronic devices and 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 its range; IR sensors do not usually provide the information of distance.

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

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

```
import RPi.GPIO as GPIO
import time
from AlphaBot2 import AlphaBot2

#Ab = AlphaBot2() #Initialize AlphaBot2
```

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
DL = 16 # Left IR sensor
```

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 IR sensors as input pins and pull up.

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(DL,GPIO.IN,GPIO.PUD_UP) # Left IR sensor
GPIO.setup(DR,GPIO.IN,GPIO.PUD_UP) # Right IR sensor
```

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

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

We 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”. The robot should “stop” because there is an obstacle in the way.
- 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”.

```
if __name__ == "__main__":
     try:
          while True:
               DR_status = GPIO.input(DR)
               DL_status = GPIO.input(DL)

               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")

               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")

               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")

               else:
                    #when DL and DR is not 0
                    #this indicate that no obstacle is detected
                    print("no obstacle")
```

Finish the program with setting “Ctrl+c” to stop the program and cleaning up the GPIO:

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

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

- The End –