HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Hochschule Ravensburg-Weingarten

University of Applied Sciences

Task 2: Robot Testing and Control

Sr No	Student Names	Matriculation Numbers
1	Arjun Ramwshbhai Beladiya	28744163

Guided by:
Prof. Markus Pfeil

Task A) Using Python, query the pushbutton and use it to switch LED on and off to test the functionality of the Button.

Answer:

```
import RPi.GPIO as GPIO
import time

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

LED = 7
Taster = 5

GPIO.setup(LED, GPIO.OUT)
GPIO.setup(Taster, GPIO.IN, pull_up_down = GPIO.PUD_UP)

# Pressing the pushbutton (Taster) LED is off
while (1):
        if GPIO.input(Taster) == GPIO.HIGH:
            GPIO.output(LED, GPIO.HIGH)
        if GPIO.input(Taster) == GPIO.LOW:
            GPIO.output(LED, GPIO.LOW)
```

Output:

The LED switches off when the button is pressed and turns back on when the button is released. This cycle continues, with the system pausing for one second before rechecking the button state

Task b) Using the pushbutton, cause the robot to rotate on the spot for 10 seconds, using the left wheel and then another 10 seconds using the right wheel.

Answer:

```
import RPi.GPIO as GPIO
import time

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

LED = 7
push_button = 5

# Motor-1 pins
Motor1_PWM = 18
Motor1_IN1 = 17
Motor1_IN2 = 22

# Motor-2 pins
Motor2_PWM = 19
Motor2_IN1 = 24
Motor2_IN2 = 4

# GPIO setup
GPIO.setup(Motor1_PWM, GPIO.OUT)
GPIO.setup(Motor1_IN1, GPIO.OUT)
GPIO.setup(Motor1_IN2, GPIO.OUT)

GPIO.setup(Motor2_PWM, GPIO.OUT)
GPIO.setup(Motor2_IN1, GPIO.OUT)
GPIO.setup(Motor2_IN2, GPIO.OUT)

GPIO.setup(LED, GPIO.OUT)
GPIO.setup(push_button, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.output(LED, GPIO.LOW)

def left():
    GPIO.output(Motor1_PWM, GPIO.HIGH)
    GPIO.output(Motor1_IN1, GPIO.HIGH)
    GPIO.output(Motor1_IN2, GPIO.LOW)
    GPIO.output(Motor2_PWM, GPIO.HIGH)
    GPIO.output(Motor2_IN1, GPIO.LOW)
    GPIO.output(Motor2_IN2, GPIO.HIGH)

def right():
    GPIO.output(Motor1_PWM, GPIO.HIGH)
    GPIO.output(Motor1_IN1, GPIO.LOW)
    GPIO.output(Motor1_IN2, GPIO.HIGH)
    GPIO.output(Motor2_PWM, GPIO.HIGH)
    GPIO.output(Motor2_IN1, GPIO.HIGH)
    GPIO.output(Motor2_IN2, GPIO.LOW)

while True:
    if GPIO.input(push_button) == GPIO.LOW:
        GPIO.output(LED, GPIO.HIGH)
        left()
        print("Left")
        time.sleep(10)
        right()
        time.sleep(10)

        GPIO.output(Motor1_IN1, GPIO.LOW)
        GPIO.output(Motor1_IN2, GPIO.LOW)
        GPIO.output(Motor2_IN1, GPIO.LOW)
        GPIO.output(Motor2_IN2, GPIO.LOW)
        GPIO.output(LED, GPIO.LOW)
    else:
        GPIO.output(LED, GPIO.LOW)
```

Observation:

When the push button is pressed, the LED lights up, signaling the start of the motion. The robot first spins in place clockwise for 10 seconds by activating the left motor, then rotates counter-clockwise for another 10 seconds by using the right motor. After completing both rotations, it stops all motor activity and turns off the LED.

Problem:

Due to tire slippage on the surface, the robot doesn't rotate exactly in place but instead slides slightly.

Task C) Using the pushbutton, cause the robot to drive ahead with increasing speed and then decelerate and stop. Then drive the robot in exactly the same manner backwards.

Answer:

```
import RPi.GPIO as GPIO
import time

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

LED = 21
push_button = 5

Motor1_PWM = 18
Motor1_IN1 = 17
Motor1_IN2 = 22

Motor2_PWM = 19
Motor2_IN1 = 24
Motor2_IN2 = 4

GPIO.setup(Motor1_PWM, GPIO.OUT)
GPIO.setup(Motor1_IN1, GPIO.OUT)
GPIO.setup(Motor1_IN2, GPIO.OUT)
pwm_motor1 = GPIO.PWM(Motor1_PWM, 100)
pwm_motor1.start(0)

GPIO.setup(Motor2_PWM, GPIO.OUT)
GPIO.setup(Motor2_IN1, GPIO.OUT)
GPIO.setup(Motor2_IN2, GPIO.OUT)
pwm_motor2 = GPIO.PWM(Motor2_PWM, 100)
pwm_motor2.start(0)

GPIO.setup(LED, GPIO.OUT)
GPIO.setup(push_button, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def Forward(speed):
    pwm_motor1.ChangeDutyCycle(speed)
    GPIO.output(Motor1_IN2, GPIO.LOW)
    GPIO.output(Motor1_IN1, GPIO.HIGH)

    pwm_motor2.ChangeDutyCycle(speed)
    GPIO.output(Motor2_IN2, GPIO.LOW)
    GPIO.output(Motor2_IN1, GPIO.HIGH)

def Backward(speed):
    pwm_motor1.ChangeDutyCycle(speed)
    GPIO.output(Motor1_IN2, GPIO.HIGH)
    GPIO.output(Motor1_IN1, GPIO.LOW)

    pwm_motor2.ChangeDutyCycle(speed)
    GPIO.output(Motor2_IN2, GPIO.HIGH)
    GPIO.output(Motor2_IN1, GPIO.LOW)

def stop():
    pwm_motor1.ChangeDutyCycle(0)
    pwm_motor2.ChangeDutyCycle(0)

while True:
    if GPIO.input(push_button) == GPIO.LOW:
        GPIO.output(LED, GPIO.HIGH)

        for speed in range(0, 101, 10):
            Forward(speed)
            print("Speed increasing:", speed)
            time.sleep(1)

        for speed in range(100, -1, -10):
            Forward(speed)
            print("Speed decreasing:", speed)
            time.sleep(1)

        stop()
        time.sleep(10)

        for speed in range(0, 101, 10):
            Backward(speed)
            print("Speed increasing backward:", speed)
            time.sleep(1)

        for speed in range(100, -1, -10):
            Backward(speed)
            print("Speed decreasing backward:", speed)
            time.sleep(1)

        stop()
        time.sleep(10)
        GPIO.output(LED, GPIO.LOW)
    else:
        GPIO.output(LED, GPIO.LOW)
```

Conclusion:

At first,the motors moves forward. Their speed gradually increases from 0 to 100 in increments of 10, then decreases back to 0 in the same manner. After a 10-second pause, the motors run in reverse, following the same speed-up and slow-down pattern. Once the backward motion is complete, the motors stop again for 10 seconds.

Task D) For the previous tasks evaluate the actions of the robot. Did they conform to your expecations? When driving back and forth, how far from the starting point did the robot end its movement?

Task A and B works well. but in task C, the robot does not follow the same path in reverse as it does when moving forward. Additionally, it does not move in a perfectly straight line as expected; the left motor runs faster than the right, causing the robot to turn to the right.

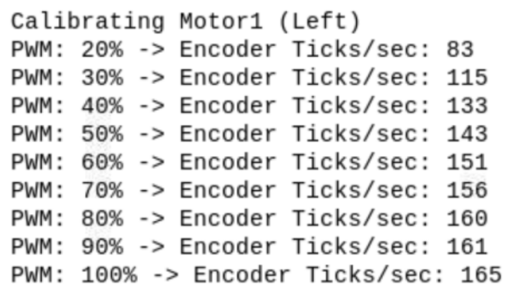
**Task E) Improve the accuracy of coming back to exactly the same point where you started use the rotary encoders on the wheels to measure how many revolutions each wheel travels. Implement two strategies:
1).Drive each motor for 5 complete wheel rotations, then reverse the motion
2).Drive each motor for 10 seconds while monitoring the rotary encoders and slowing the faster moving motor to match the speed of the slower moving motor continuously.**

Answer:

Getting PWM and Encoder Ticks/sec Data:

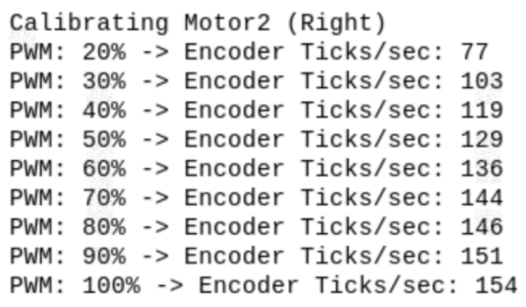
```
import RPi.GPIO as GPIO
import time

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

# Motor 1 (Left)
Motor1_PWM = 18
Motor1_IN1 = 17
Motor1_IN2 = 22
Encoder1_PIN = 16

# Motor 2 (Right)
Motor2_PWM = 19
Motor2_IN1 = 24
Motor2_IN2 = 4
Encoder2_PIN = 23

GPIO.setup([Motor1_IN1, Motor1_IN2, Motor1_PWM], GPIO.OUT)
GPIO.setup([Motor2_IN1, Motor2_IN2, Motor2_PWM], GPIO.OUT)
GPIO.setup([Encoder1_PIN, Encoder2_PIN], GPIO.IN)

pwm1 = GPIO.PWM(Motor1_PWM, 100)
pwm2 = GPIO.PWM(Motor2_PWM, 100)
pwm1.start(0)
pwm2.start(0)

def read_encoder(pin, duration=1):
    count = 0
    last_state = GPIO.input(pin)
    start = time.time()
    while time.time() - start < duration:
        current_state = GPIO.input(pin)
        if current_state != last_state:
            count += 1
        last_state = current_state
        time.sleep(0.001)
    return count

def run_motor_calibration(pwm_obj, in1, in2, encoder_pin, motor_name):
    GPIO.output(in1, GPIO.HIGH)
    GPIO.output(in2, GPIO.LOW)

    print(f"\nCalibrating {motor_name}")
    for speed in range(20, 101, 10):
        pwm_obj.ChangeDutyCycle(speed)
        time.sleep(0.5)  # motor stabilizes
        ticks = read_encoder(encoder_pin)
        print(f"PWM: {speed}% -> Encoder Ticks/sec: {ticks}")
    pwm_obj.ChangeDutyCycle(0)
    GPIO.output(in1, GPIO.LOW)
    GPIO.output(in2, GPIO.LOW)

try:
    run_motor_calibration(pwm1, Motor1_IN1, Motor1_IN2, Encoder1_PIN, "
       Motor1 (Left)")
    run_motor_calibration(pwm2, Motor2_IN1, Motor2_IN2, Encoder2_PIN, "
       Motor2 (Right)")

finally:
    pwm1.stop()
    pwm2.stop()
    GPIO.cleanup()
```

Output:



```
Calibrating Motor1 (Left)
PWM: 20% -> Encoder Ticks/sec: 83
PWM: 30% -> Encoder Ticks/sec: 115
PWM: 40% -> Encoder Ticks/sec: 133
PWM: 50% -> Encoder Ticks/sec: 143
PWM: 60% -> Encoder Ticks/sec: 151
PWM: 70% -> Encoder Ticks/sec: 156
PWM: 80% -> Encoder Ticks/sec: 160
PWM: 90% -> Encoder Ticks/sec: 161
PWM: 100% -> Encoder Ticks/sec: 165
```

Figure 1: Left Motor Data



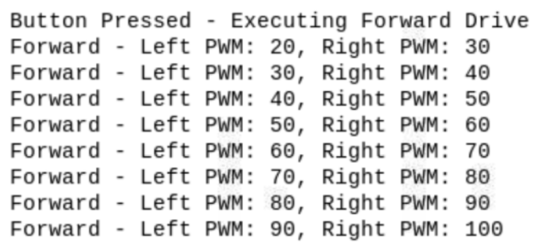
```
Calibrating Motor2 (Right)
PWM: 20% -> Encoder Ticks/sec: 77
PWM: 30% -> Encoder Ticks/sec: 103
PWM: 40% -> Encoder Ticks/sec: 119
PWM: 50% -> Encoder Ticks/sec: 129
PWM: 60% -> Encoder Ticks/sec: 136
PWM: 70% -> Encoder Ticks/sec: 144
PWM: 80% -> Encoder Ticks/sec: 146
PWM: 90% -> Encoder Ticks/sec: 151
PWM: 100% -> Encoder Ticks/sec: 154
```

Figure 2: Right Motor Data

Test 1: by using PWM and Encoder Ticks/sec Data:

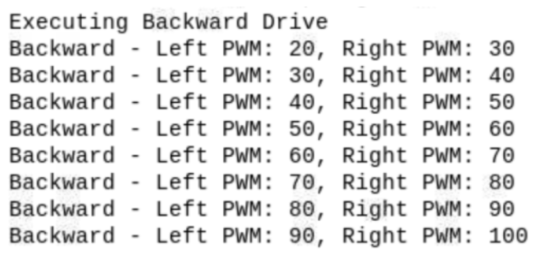
```
import RPi.GPIO as GPIO
import time

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

# Pin Assignments
LED = 21
push_button = 5

Motor1_PWM = 18
Motor1_IN1 = 17
Motor1_IN2 = 22

Motor2_PWM = 19
Motor2_IN1 = 24
Motor2_IN2 = 4

Encoder1_PIN1 = 16
Encoder2_PIN1 = 23

# Setup Encoder Pins
GPIO.setup(Encoder1_PIN1, GPIO.IN)
GPIO.setup(Encoder2_PIN1, GPIO.IN)

# Setup Motor1 Pins
GPIO.setup(Motor1_PWM, GPIO.OUT)
GPIO.setup(Motor1_IN1, GPIO.OUT)
GPIO.setup(Motor1_IN2, GPIO.OUT)
pwm_motor1 = GPIO.PWM(Motor1_PWM, 100)
pwm_motor1.start(0)

# Setup Motor2 Pins
GPIO.setup(Motor2_PWM, GPIO.OUT)
GPIO.setup(Motor2_IN1, GPIO.OUT)
GPIO.setup(Motor2_IN2, GPIO.OUT)
pwm_motor2 = GPIO.PWM(Motor2_PWM, 100)
pwm_motor2.start(0)

# Setup LED and Push Button
GPIO.setup(LED, GPIO.OUT)
GPIO.setup(push_button, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Calibrated PWM pairs to equalize speed
speed_profile = [
    (20, 30),  # (Motor1 PWM, Motor2 PWM)
    (30, 40),
    (40, 50),
    (50, 60),
    (60, 70),
    (70, 80),
    (80, 90),
    (90, 100)
]

def stop():
    pwm_motor1.ChangeDutyCycle(0)
    pwm_motor2.ChangeDutyCycle(0)
    GPIO.output(Motor1_IN1, GPIO.LOW)
    GPIO.output(Motor1_IN2, GPIO.LOW)
    GPIO.output(Motor2_IN1, GPIO.LOW)
    GPIO.output(Motor2_IN2, GPIO.LOW)

def Forward():
    for left_pwm, right_pwm in speed_profile:
        pwm_motor1.ChangeDutyCycle(left_pwm)
        GPIO.output(Motor1_IN1, GPIO.HIGH)
        GPIO.output(Motor1_IN2, GPIO.LOW)

        pwm_motor2.ChangeDutyCycle(right_pwm)
        GPIO.output(Motor2_IN1, GPIO.HIGH)
        GPIO.output(Motor2_IN2, GPIO.LOW)

        print(f"Forward - Left PWM: {left_pwm}, Right PWM: {right_pwm}"
            )
        time.sleep(1)

def Backward():
    for left_pwm, right_pwm in speed_profile:
        pwm_motor1.ChangeDutyCycle(left_pwm)
        GPIO.output(Motor1_IN1, GPIO.LOW)
        GPIO.output(Motor1_IN2, GPIO.HIGH)

        pwm_motor2.ChangeDutyCycle(right_pwm)
        GPIO.output(Motor2_IN1, GPIO.LOW)
        GPIO.output(Motor2_IN2, GPIO.HIGH)

        print(f"Backward - Left PWM: {left_pwm}, Right PWM: {right_pwm}
            ")
        time.sleep(1)

# Main loop
try:
    while True:
        if GPIO.input(push_button) == GPIO.LOW:
            GPIO.output(LED, GPIO.HIGH)
            print("Button Pressed - Executing Forward Drive")
            Forward()
            stop()
            time.sleep(2)
            print("Executing Backward Drive")
            Backward()
            stop()
            time.sleep(2)
            GPIO.output(LED, GPIO.LOW)
        else:
            GPIO.output(LED, GPIO.LOW)

except KeyboardInterrupt:
    print("Exiting Program")

finally:
    pwm_motor1.stop()
    pwm_motor2.stop()
    GPIO.cleanup()
```

output:



```
Button Pressed - Executing Forward Drive
Forward - Left PWM: 20, Right PWM: 30
Forward - Left PWM: 30, Right PWM: 40
Forward - Left PWM: 40, Right PWM: 50
Forward - Left PWM: 50, Right PWM: 60
Forward - Left PWM: 60, Right PWM: 70
Forward - Left PWM: 70, Right PWM: 80
Forward - Left PWM: 80, Right PWM: 90
Forward - Left PWM: 90, Right PWM: 100
```

Figure 3: Left Motor Data



```
Executing Backward Drive
Backward - Left PWM: 20, Right PWM: 30
Backward - Left PWM: 30, Right PWM: 40
Backward - Left PWM: 40, Right PWM: 50
Backward - Left PWM: 50, Right PWM: 60
Backward - Left PWM: 60, Right PWM: 70
Backward - Left PWM: 70, Right PWM: 80
Backward - Left PWM: 80, Right PWM: 90
Backward - Left PWM: 90, Right PWM: 100
```

Figure 4: Right Motor Data

Errors:

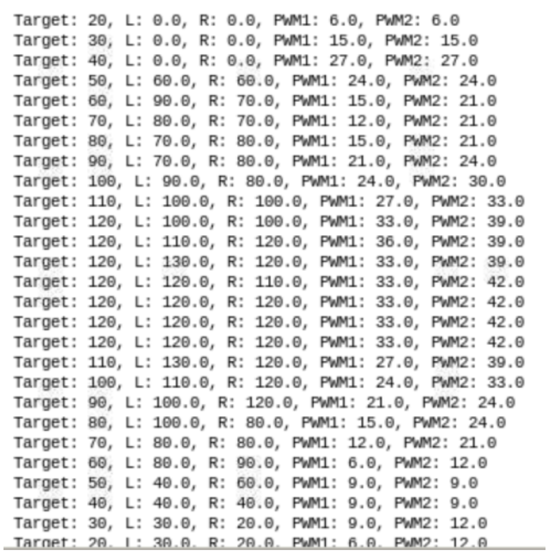
Using different duty cycles has improved the robot's performance compared to before; however, it still does not follow a perfectly straight path as expected.

Final Code: by using Encoder-based Feedback and PID based motor control

```
import RPi.GPIO as GPIO
import time

# GPIO pin setup
Motor1_PWM = 18
Motor1_IN1 = 17
Motor1_IN2 = 22
Encoder1_PIN = 16

Motor2_PWM = 19
Motor2_IN1 = 24
Motor2_IN2 = 4
Encoder2_PIN = 23

LED = 21
push_button = 5

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

# Motor setup
GPIO.setup([Motor1_PWM, Motor1_IN1, Motor1_IN2, Motor2_PWM, Motor2_IN1,
    Motor2_IN2], GPIO.OUT)
pwm_motor1 = GPIO.PWM(Motor1_PWM, 100)
pwm_motor2 = GPIO.PWM(Motor2_PWM, 100)
pwm_motor1.start(0)
pwm_motor2.start(0)

# Encoder setup
GPIO.setup([Encoder1_PIN, Encoder2_PIN], GPIO.IN)

# LED and Button
GPIO.setup(LED, GPIO.OUT)
GPIO.setup(push_button, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Encoder tick counters
encoder1_ticks = 0
encoder2_ticks = 0

# Encoder callbacks
def encoder1_callback(channel):
    global encoder1_ticks
    encoder1_ticks += 1

def encoder2_callback(channel):
    global encoder2_ticks
    encoder2_ticks += 1

GPIO.add_event_detect(Encoder1_PIN, GPIO.BOTH, callback=
   encoder1_callback)
GPIO.add_event_detect(Encoder2_PIN, GPIO.BOTH, callback=
   encoder2_callback)

# Direction control
```

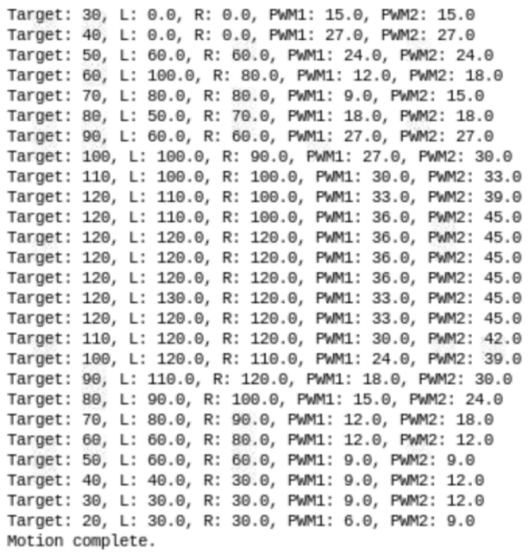
```
def set_motor_direction(forward=True):
    if forward:
        GPIO.output(Motor1_IN1, GPIO.HIGH)
        GPIO.output(Motor1_IN2, GPIO.LOW)
        GPIO.output(Motor2_IN1, GPIO.HIGH)
        GPIO.output(Motor2_IN2, GPIO.LOW)
    else:
        GPIO.output(Motor1_IN1, GPIO.LOW)
        GPIO.output(Motor1_IN2, GPIO.HIGH)
        GPIO.output(Motor2_IN1, GPIO.LOW)
        GPIO.output(Motor2_IN2, GPIO.HIGH)

# Stop both motors
def stop():
    pwm_motor1.ChangeDutyCycle(0)
    pwm_motor2.ChangeDutyCycle(0)

# Reset encoder ticks
def reset_ticks():
    global encoder1_ticks, encoder2_ticks
    encoder1_ticks = 0
    encoder2_ticks = 0

# Read encoder speeds
def read_encoder_speeds(duration=0.1):
    reset_ticks()
    time.sleep(duration)
    left_speed = encoder1_ticks / duration
    right_speed = encoder2_ticks / duration
    return left_speed, right_speed

# PID-based motor control
def drive_with_pid_profile(forward=True):
    set_motor_direction(forward)
    pwm1 = 0
    pwm2 = 0
    Kp = 0.3

    # Create speed profile: ramp up, hold, ramp down
    speed_profile = list(range(20, 121, 10)) + [120]*5 + list(range
       (120, 19, -10))  # in ticks/sec

    for target_speed in speed_profile:
        left_speed, right_speed = read_encoder_speeds()

        error1 = target_speed - left_speed
        error2 = target_speed - right_speed

        pwm1 += Kp * error1
        pwm2 += Kp * error2

        pwm1 = max(0, min(100, pwm1))
        pwm2 = max(0, min(100, pwm2))

        pwm_motor1.ChangeDutyCycle(pwm1)
        pwm_motor2.ChangeDutyCycle(pwm2)
        print(f"Target: {target_speed}, L: {left_speed:.1f}, R: {
            right_speed:.1f}, PWM1: {pwm1:.1f}, PWM2: {pwm2:.1f}")
        time.sleep(0.1)

# Main loop
try:
    while True:
        if GPIO.input(push_button) == GPIO.LOW:
            GPIO.output(LED, GPIO.HIGH)
            print("Button pressed: running motion sequence...")
            drive_with_pid_profile(forward=True)
            stop()
            time.sleep(1)
            drive_with_pid_profile(forward=False)
            stop()
            GPIO.output(LED, GPIO.LOW)
            print("Motion complete.")
        else:
            GPIO.output(LED, GPIO.LOW)

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

Output:



```
Target: 20, L: 0.0, R: 0.0, PWM1: 6.0, PWM2: 6.0
Target: 30, L: 0.0, R: 0.0, PWM1: 15.0, PWM2: 15.0
Target: 40, L: 0.0, R: 0.0, PWM1: 27.0, PWM2: 27.0
Target: 50, L: 60.0, R: 60.0, PWM1: 24.0, PWM2: 24.0
Target: 60, L: 90.0, R: 70.0, PWM1: 15.0, PWM2: 21.0
Target: 70, L: 80.0, R: 70.0, PWM1: 12.0, PWM2: 21.0
Target: 80, L: 70.0, R: 80.0, PWM1: 15.0, PWM2: 21.0
Target: 90, L: 70.0, R: 80.0, PWM1: 21.0, PWM2: 24.0
Target: 100, L: 90.0, R: 80.0, PWM1: 24.0, PWM2: 30.0
Target: 110, L: 100.0, R: 100.0, PWM1: 27.0, PWM2: 33.0
Target: 120, L: 100.0, R: 100.0, PWM1: 33.0, PWM2: 39.0
Target: 120, L: 110.0, R: 120.0, PWM1: 36.0, PWM2: 39.0
Target: 120, L: 130.0, R: 120.0, PWM1: 33.0, PWM2: 39.0
Target: 120, L: 120.0, R: 110.0, PWM1: 33.0, PWM2: 42.0
Target: 120, L: 120.0, R: 120.0, PWM1: 33.0, PWM2: 42.0
Target: 120, L: 120.0, R: 120.0, PWM1: 33.0, PWM2: 42.0
Target: 120, L: 120.0, R: 120.0, PWM1: 33.0, PWM2: 42.0
Target: 110, L: 130.0, R: 120.0, PWM1: 27.0, PWM2: 39.0
Target: 100, L: 110.0, R: 120.0, PWM1: 24.0, PWM2: 33.0
Target: 90, L: 100.0, R: 120.0, PWM1: 21.0, PWM2: 24.0
Target: 80, L: 100.0, R: 80.0, PWM1: 15.0, PWM2: 24.0
Target: 70, L: 80.0, R: 80.0, PWM1: 12.0, PWM2: 21.0
Target: 60, L: 80.0, R: 90.0, PWM1: 6.0, PWM2: 12.0
Target: 50, L: 40.0, R: 60.0, PWM1: 9.0, PWM2: 9.0
Target: 40, L: 40.0, R: 40.0, PWM1: 9.0, PWM2: 9.0
Target: 30, L: 30.0, R: 20.0, PWM1: 9.0, PWM2: 12.0
Target: 20, L: 30.0, R: 20.0, PWM1: 6.0, PWM2: 12.0
```

Figure 5: Forward Motion



```
Target: 30, L: 0.0, R: 0.0, PWM1: 15.0, PWM2: 15.0
Target: 40, L: 0.0, R: 0.0, PWM1: 27.0, PWM2: 27.0
Target: 50, L: 60.0, R: 60.0, PWM1: 24.0, PWM2: 24.0
Target: 60, L: 100.0, R: 80.0, PWM1: 12.0, PWM2: 18.0
Target: 70, L: 80.0, R: 80.0, PWM1: 9.0, PWM2: 15.0
Target: 80, L: 50.0, R: 70.0, PWM1: 18.0, PWM2: 18.0
Target: 90, L: 60.0, R: 60.0, PWM1: 27.0, PWM2: 27.0
Target: 100, L: 100.0, R: 90.0, PWM1: 27.0, PWM2: 30.0
Target: 110, L: 100.0, R: 100.0, PWM1: 30.0, PWM2: 33.0
Target: 120, L: 110.0, R: 100.0, PWM1: 33.0, PWM2: 39.0
Target: 120, L: 110.0, R: 100.0, PWM1: 36.0, PWM2: 45.0
Target: 120, L: 120.0, R: 120.0, PWM1: 36.0, PWM2: 45.0
Target: 120, L: 120.0, R: 120.0, PWM1: 36.0, PWM2: 45.0
Target: 120, L: 120.0, R: 120.0, PWM1: 36.0, PWM2: 45.0
Target: 120, L: 130.0, R: 120.0, PWM1: 33.0, PWM2: 45.0
Target: 120, L: 120.0, R: 120.0, PWM1: 33.0, PWM2: 45.0
Target: 110, L: 120.0, R: 120.0, PWM1: 30.0, PWM2: 42.0
Target: 100, L: 120.0, R: 110.0, PWM1: 24.0, PWM2: 39.0
Target: 90, L: 110.0, R: 120.0, PWM1: 18.0, PWM2: 30.0
Target: 80, L: 90.0, R: 100.0, PWM1: 15.0, PWM2: 24.0
Target: 70, L: 80.0, R: 90.0, PWM1: 12.0, PWM2: 18.0
Target: 60, L: 60.0, R: 80.0, PWM1: 12.0, PWM2: 12.0
Target: 50, L: 60.0, R: 60.0, PWM1: 9.0, PWM2: 9.0
Target: 40, L: 40.0, R: 30.0, PWM1: 9.0, PWM2: 12.0
Target: 30, L: 30.0, R: 30.0, PWM1: 9.0, PWM2: 12.0
Target: 20, L: 30.0, R: 30.0, PWM1: 6.0, PWM2: 9.0
Motion complete.
```

Figure 6: Backward Motion

Conclusion:

At the starting point, the robot turns slightly to the right due to the left motor rotating faster than the right. However, once feedback control is applied, it maintains a relatively straight path (though not perfectly straight). Toward the end, it also experiences a slight Turn.

Task F) Use what you have learned in the previous tasks to create a programm that will move the robot forward until it comes close to an obstacle (using the SR04 distance sensor to measure the distance), then turn right or left 90 degrees at random and continue moving until the next obstacle is encountered in the same fashion. Use the rotary encoders to turn exactly 90 degrees (as good as possible).

Answer:

```
import RPi.GPIO as GPIO
import time

# ------------------- GPIO Setup -------------------
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

# Pin configuration
LED = 21
push_button = 5

Motor1_PWM = 18
Motor1_IN1 = 17
Motor1_IN2 = 22

Motor2_PWM = 19
Motor2_IN1 = 24
Motor2_IN2 = 4

Encoder1_PIN1 = 16
Encoder2_PIN1 = 23

SR04_TRIG = 25
SR04_ECHO = 27

# Motor setup
GPIO.setup([Motor1_PWM, Motor1_IN1, Motor1_IN2, Motor2_PWM, Motor2_IN1,
    Motor2_IN2], GPIO.OUT)
pwm_motor1 = GPIO.PWM(Motor1_PWM, 100)
pwm_motor2 = GPIO.PWM(Motor2_PWM, 100)
pwm_motor1.start(0)
pwm_motor2.start(0)

# Inputs
GPIO.setup([Encoder1_PIN1, Encoder2_PIN1, push_button, SR04_ECHO], GPIO
   .IN)
GPIO.setup([SR04_TRIG, LED], GPIO.OUT)

# ------------------- Functions -------------------
```

```
def read_distance():
    GPIO.output(SR04_TRIG, False)
    time.sleep(0.01)
    GPIO.output(SR04_TRIG, True)
    time.sleep(0.00001)
    GPIO.output(SR04_TRIG, False)

    while GPIO.input(SR04_ECHO) == 0:
        pulse_start = time.time()
    while GPIO.input(SR04_ECHO) == 1:
        pulse_end = time.time()

    duration = pulse_end - pulse_start
    distance = duration * 17150
    return distance

def move_forward(pwm_base):
    # Motor1 (Left), Motor2 (Right)
    pwm_motor1.ChangeDutyCycle(pwm_base)
    GPIO.output(Motor1_IN1, GPIO.HIGH)
    GPIO.output(Motor1_IN2, GPIO.LOW)

    # Motor2 adjusted (Right motor needs more PWM)
    pwm_motor2.ChangeDutyCycle(pwm_base + 10)
    GPIO.output(Motor2_IN1, GPIO.HIGH)
    GPIO.output(Motor2_IN2, GPIO.LOW)

def stop():
    pwm_motor1.ChangeDutyCycle(0)
    pwm_motor2.ChangeDutyCycle(0)
    GPIO.output(Motor1_IN1, GPIO.LOW)
    GPIO.output(Motor1_IN2, GPIO.LOW)
    GPIO.output(Motor2_IN1, GPIO.LOW)
    GPIO.output(Motor2_IN2, GPIO.LOW)

def turn_left_90():
    # 90-degree turn based on encoder count (adjust for your robot)
    count_left = 0
    encoder_start = GPIO.input(Encoder1_PIN1)
    pwm_motor1.ChangeDutyCycle(0)
    GPIO.output(Motor1_IN1, GPIO.LOW)
    GPIO.output(Motor1_IN2, GPIO.LOW)

    pwm_motor2.ChangeDutyCycle(50)
    GPIO.output(Motor2_IN1, GPIO.LOW)
    GPIO.output(Motor2_IN2, GPIO.HIGH)

    while count_left < 15:  # You might need to calibrate this value
        encoder_now = GPIO.input(Encoder2_PIN1)
        if encoder_now != encoder_start:
            count_left += 1
            encoder_start = encoder_now
        time.sleep(0.01)

    stop()
    time.sleep(0.5)

# ------------------- Main Loop -------------------
```

```
while True:
    if GPIO.input(push_button) == GPIO.LOW:
        GPIO.output(LED, GPIO.HIGH)

        while True:
            dist = read_distance()
            print("Distance:", dist)

            if dist < 20:  # Obstacle threshold
                stop()
                time.sleep(0.5)
                turn_left_90()
                time.sleep(0.5)
            else:
                move_forward(50)  # Base PWM

            time.sleep(0.1)
    else:
        stop()
        GPIO.output(LED, GPIO.LOW)
        GPIO.cleanup()
```

Conclusion:

This task performs as expected — the robot stops upon detecting an obstacle ahead and initiates a turn. However, it fails to complete a full 90-degree turn due to slipping on the ground surface.

Task G) Make sure you close all open connections and pin assignments when closing the programs.

Answer:

To ensure all open GPIO connections are properly closed, I used the GPIO.cleanup() function at the end of the program