CamJam EduKit Robotics - Obstacle Avoidance

Project Obstacle Avoidance

Description You will learn how to stop your robot from bumping into things.

Equipment Required

For this worksheet you will require:

- Your robot with the ultrasonic sensor attached to the front (see Worksheet 6).

How it Works

In Worksheet 6 you learnt how to detect how far in front of your robot an object was using the ultrasonic sensor. In this worksheet you are going to get the robot to avoid driving into obstacles.

Here is the logic for how the robot will work in this worksheet:

- The robot will drive forwards until it detects that there is an object a few centimetres in front of it.
- The robot will reverse and then turn right.
- The robot will then drive forwards again.

Code

The code will be based on the code you completed at the end of Worksheet 7, so lets copy it:

```
cd ~/EduKitRobotics/
cp 7-pwm2.py 9-avoidance.py
nano 9-avoidance.py
```

Add variables for the ultrasonic sensors' GPIO pins to the section where the motor GPIO pins are defined:

```
# Define GPIO pins to use on the Pi
pinTrigger = 17
pinEcho = 18
```

Add variables to set how close to an obstacle the robot can go, how long to reverse for and how long to turn right for. You should change these variables to suit your robot.

```
# Distance Variables
HowNear = 15.0
ReverseTime = 0.5
TurnTime = 0.75
```

And add the GPIO setup line to the others:

```
# Set pins as output and input
GPIO.setup(pinTrigger, GPIO.OUT)  # Trigger
GPIO.setup(pinEcho, GPIO.IN)      # Echo
```

You're going to add another function that will return whether the ultrasonic sensor sees an obstacle.

```
# Return True if the ultrasonic sensor sees an obstacle
def IsNearObstacle(localHowNear):
    Distance = Measure()

    print("IsNearObstacle: "+str(Distance))
```

```
    if Distance < localHowNear:
        return True
    else:
        return False
```

In the code above, you pass in a variable to say how close you can get to an obstacle. It also uses another new function that does the measurement using the ultrasonic sensor.

```
# Take a distance measurement
def Measure():
    GPIO.output(pinTrigger, True)
    time.sleep(0.00001)
    GPIO.output(pinTrigger, False)
    StartTime = time.time()
    StopTime = StartTime

    while GPIO.input(pinEcho)==0:
        StartTime = time.time()
        StopTime = StartTime

    while GPIO.input(pinEcho)==1:
        StopTime = time.time()
        # If the sensor is too close to an object, the Pi cannot
        # see the echo quickly enough, so it has to detect that
        # problem and say what has happened
        if StopTime-StartTime >= 0.04:
            print("Hold on there!  You're too close for me to see.")
            StopTime = StartTime
            break

    ElapsedTime = StopTime - StartTime
    Distance = (ElapsedTime * 34326)/2

    return Distance
```

You also need to write a function that will make the robot avoid the obstacle by moving back a little, then turning right.

```
# Move back a little, then turn right
def AvoidObstacle():
    # Back off a little
    print("Backwards")
    Backwards()
    time.sleep(ReverseTime)
    StopMotors()

    # Turn right
    print("Right")
    Right()
    time.sleep(TurnTime)
    StopMotors()
```

The code that controls the robot then replaces everything between the comment `# Your code to control the robot goes below this line` and the end of the code.

```
try:
    # Set trigger to False (Low)
    GPIO.output(pinTrigger, False)

    # Allow module to settle
    time.sleep(0.1)

    #repeat the next indented block forever
    while True:
        Forwards()
        time.sleep(0.1)
        if IsNearObstacle(HowNear):
            StopMotors()
            AvoidObstacle()

# If you press CTRL+C, cleanup and stop
except KeyboardInterrupt:
    GPIO.cleanup()
```

This code runs the commands within the `try:` section until the Ctrl+C keys are pressed.

Within the `try:` is a `while True:`. Since True is always true, the code within that while will run forever – or until Ctrl+C is pressed.

Within the `while True:`, the code makes the robot go forward until it is `HowNear` cm from an obstacle. When it is, the robot reverses and turns right.

Your code should now look like this:

```
# CamJam EduKit 3 - Robotics
# Worksheet 9 – Obstacle Avoidance

import RPi.GPIO as GPIO # Import the GPIO Library
import time # Import the Time library

# Set the GPIO modes
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Set variables for the GPIO motor pins
pinMotorAForwards = 10
pinMotorABackwards = 9
pinMotorBForwards = 8
pinMotorBBackwards = 7
# Define GPIO pins to use on the Pi
pinTrigger = 17
pinEcho = 18

# How many times to turn the pin on and off each second
Frequency = 20
```

```
# How long the pin stays on each cycle, as a percent
DutyCycleA = 30
DutyCycleB = 30
# Setting the duty cycle to 0 means the motors will not turn
Stop = 0

# Set the GPIO Pin mode to be Output
GPIO.setup(pinMotorAForwards, GPIO.OUT)
GPIO.setup(pinMotorABackwards, GPIO.OUT)
GPIO.setup(pinMotorBForwards, GPIO.OUT)
GPIO.setup(pinMotorBBackwards, GPIO.OUT)
# Set pins as output and input
GPIO.setup(pinTrigger, GPIO.OUT)  # Trigger
GPIO.setup(pinEcho, GPIO.IN)      # Echo

# Distance Variables
HowNear = 15.0
ReverseTime = 0.5
TurnTime = 0.75

# Set the GPIO to software PWM at 'Frequency' Hertz
pwmMotorAForwards = GPIO.PWM(pinMotorAForwards, Frequency)
pwmMotorABackwards = GPIO.PWM(pinMotorABackwards, Frequency)
pwmMotorBForwards = GPIO.PWM(pinMotorBForwards, Frequency)
pwmMotorBBackwards = GPIO.PWM(pinMotorBBackwards, Frequency)

# Start the software PWM with a duty cycle of 0 (i.e. not moving)
pwmMotorAForwards.start(Stop)
pwmMotorABackwards.start(Stop)
pwmMotorBForwards.start(Stop)
pwmMotorBBackwards.start(Stop)

# Turn all motors off
def StopMotors():
    pwmMotorAForwards.ChangeDutyCycle(Stop)
    pwmMotorABackwards.ChangeDutyCycle(Stop)
    pwmMotorBForwards.ChangeDutyCycle(Stop)
    pwmMotorBBackwards.ChangeDutyCycle(Stop)

# Turn both motors forwards
def Forwards():
    pwmMotorAForwards.ChangeDutyCycle(DutyCycleA)
    pwmMotorABackwards.ChangeDutyCycle(Stop)
    pwmMotorBForwards.ChangeDutyCycle(DutyCycleB)
    pwmMotorBBackwards.ChangeDutyCycle(Stop)
```

```
# Turn both motors backwards
def Backwards():
    pwmMotorAForwards.ChangeDutyCycle(Stop)
    pwmMotorABackwards.ChangeDutyCycle(DutyCycleA)
    pwmMotorBForwards.ChangeDutyCycle(Stop)
    pwmMotorBBackwards.ChangeDutyCycle(DutyCycleB)

# Turn left
def Left():
    pwmMotorAForwards.ChangeDutyCycle(Stop)
    pwmMotorABackwards.ChangeDutyCycle(DutyCycleA)
    pwmMotorBForwards.ChangeDutyCycle(DutyCycleB)
    pwmMotorBBackwards.ChangeDutyCycle(Stop)

# Turn Right
def Right():
    pwmMotorAForwards.ChangeDutyCycle(DutyCycleA)
    pwmMotorABackwards.ChangeDutyCycle(Stop)
    pwmMotorBForwards.ChangeDutyCycle(Stop)
    pwmMotorBBackwards.ChangeDutyCycle(DutyCycleB)

# Take a distance measurement
def Measure():
    GPIO.output(pinTrigger, True)
    time.sleep(0.00001)
    GPIO.output(pinTrigger, False)
    StartTime = time.time()
    StopTime = StartTime

    while GPIO.input(pinEcho)==0:
        StartTime = time.time()
        StopTime = StartTime

    while GPIO.input(pinEcho)==1:
        StopTime = time.time()
        # If the sensor is too close to an object, the Pi cannot
        # see the echo quickly enough, so it has to detect that
        # problem and say what has happened
        if StopTime-StartTime >= 0.04:
            print("Hold on there!  You're too close for me to see.")
            StopTime = StartTime
            break

    ElapsedTime = StopTime - StartTime
    Distance = (ElapsedTime * 34326)/2

    return Distance
```

```
# Return True if the ultrasonic sensor sees an obstacle
def IsNearObstacle(localHowNear):
    Distance = Measure()

    print("IsNearObstacle: "+str(Distance))
    if Distance < localHowNear:
        return True
    else:
        return False

# Move back a little, then turn right
def AvoidObstacle():
    # Back off a little
    print("Backwards")
    Backwards()
    time.sleep(ReverseTime)
    StopMotors()

    # Turn right
    print("Right")
    Right()
    time.sleep(TurnTime)
    StopMotors()

# Your code to control the robot goes below this line
try:
    # Set trigger to False (Low)
    GPIO.output(pinTrigger, False)

    # Allow module to settle
    time.sleep(0.1)

    #repeat the next indented block forever
    while True:
        Forwards()
        time.sleep(0.1)
        if IsNearObstacle(HowNear):
            StopMotors()
            AvoidObstacle()

# If you press CTRL+C, cleanup and stop
except KeyboardInterrupt:
    GPIO.cleanup()
```

Once you have typed all the code and checked it, save and exit the text editor with “Ctrl + x” then “y” then “enter”.

Running the Code

Place your robot on the floor near a wall or other obstacles and start the program by typing the following into the terminal window:

```
python3 9-avoidance.py
```

The robot should then approach the wall until it gets close. It should then reverse and turn right.

Tuning the Robot

As mentioned in previous worksheets, every robot is different. You should therefore experiment with the distance variables (`HowNear`, `ReverseTime`, and `TurnTime`), as well as the speed of the robot, if necessary (`Frequency`, `DutyCycleA` and `DutyCycleB`).

Challenge

Instead of just reversing and going right, make your robot turn left, measure the distance to the next object, then turn right and do the same and drive in the direction that has an obstacle furthest away.