Lab 1: Reactive Behaviors and State Machines

CSCI 3302: Introduction to Robotics

The goals of this lab are to understand

- Basic reactive behaviors such as attraction and avoidance
- How to compose multiple behaviors into more complex ones
- The notion of state and basic ways of implementation

You need:

- **A functional Python 3.7 or 3.8 development environment. Do not use Python 2.**
- A group programming IDE is not mandatory but recommended. E.g. www.codeshare.io | http://collabedit.com/
- Webots robot simulation software (https://cyberbotics.com/). At least one computer per team able to run it smoothly (you will need a GPU).

Overview

A very simple way to program a robot is to directly tie its sensor input to wheel motion. Examples include obstacle avoidance, for example "turn right if you see an obstacle in front", light following or avoidance, or line following. One type of these robots is a "Braitenberg Vehicle". It would be quite limiting, however, if robots would always exhibit the same behavior when presented with stimuli. To achieve more complex behaviors, robots need to switch between different operational modes based on the context they're in. In the first part of this lab, you will play with different standard behaviors. You will then create a simple finite state machine and switch between different behaviors to accomplish a complex task.

Instructions

Each group must develop their own software implementation and turn in a lab report. **You are encouraged to engage with your lab partners for collaborative problem-solving**, but are expected to understand and be able to explain everything in your write-up. If your group does not finish the implementation by the end of the class, you may continue this lab on your own time as a complete group.

Part 1: Introduction to Webots

1-A) Review the Getting Started with Webots section on the Cyberbotics website: https://cyberbotics.com/doc/guide/getting-started-with-webots

1-B) Follow the first three tutorials on the Webots website to familiarize yourself with the software interface and components: https://cyberbotics.com/doc/guide/tutorials

1. First Simulation: https://cyberbotics.com/doc/guide/tutorial-1-your-first-simulation-in-webots
2. Modifying the Environment: https://cyberbotics.com/doc/guide/tutorial-2-modification-of-the-environment
3. Modifying the Appearance of Objects: https://cyberbotics.com/doc/guide/tutorial-3-appearance
4. Creating a Robot Controller: https://cyberbotics.com/doc/guide/tutorial-4-more-about-controllers

[End Week 1 of Lab]

Part 2: World Creation and Controller Programming

Create an environment with the following in it:

1) 3m x 3m rectangular floor surrounded by walls.
2) An e-puck robot in the center
3) Two obstacles (O1 and O2) of your choosing, on opposite sides of the e-puck.
   O1 must be in front of the robot, at least 15cm away
   O2 must be further than 15cm away from the robot, directly behind the robot.

Implement a state machine-based controller that recreates the following behavior:

> The robot drives forward. The robot continues until its sensor reads a value from one of its front distance sensors showing that it's within about 0.05m of the detected object. Once at that distance, the robot turns 180 degrees and drives until its front sensor says it is within about 0.05m of another object. Once arriving at within 0.05m of the object, it will rotate clockwise until the left distance sensor (ps5) reads <0.05m. Finally, it will continually drive forward as long as the left distance sensor reads <0.05m, otherwise it will stop in place and wait forever.

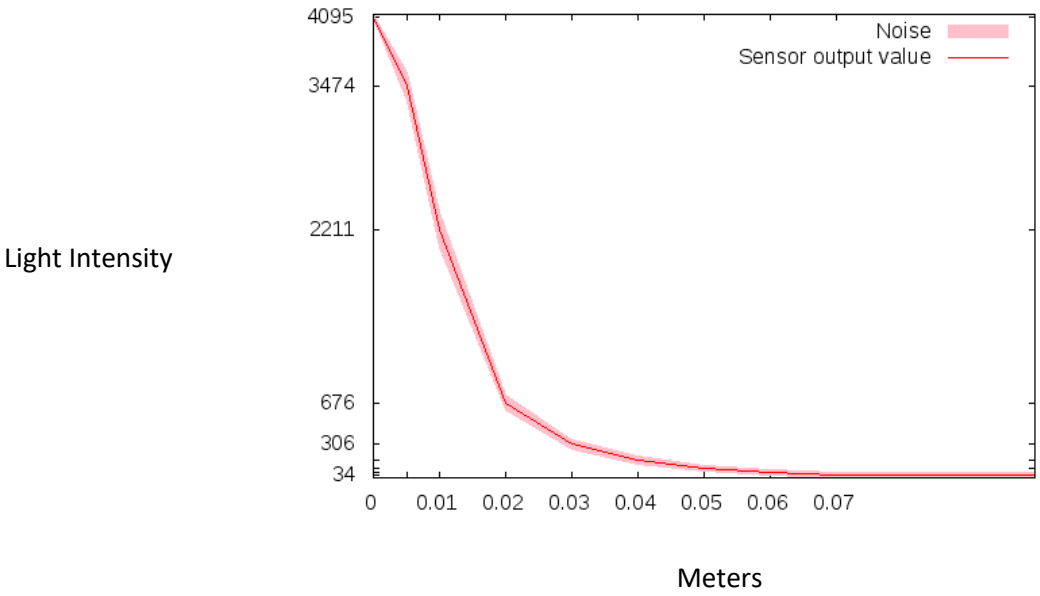
Note: **robot.timeStep** function is what advances time in the world and you shouldn't arrest the control from the main loop in any case.

HINT: Planning your answer for Part 3 - #3 before writing code will make this easier.

HINT: You can manually time out how long the robot should actuate its wheels to turn around, or you can use the distance sensors behind it to tell when you've finished turning.

HINT: The distance sensor doesn't return a distance value, it returns an intensity measurement: you will need to convert the returned value to a distance, which will only be approximate given the low accuracy of this sensor.

You can use the lookup table below to map distance sensor values to distances:



HINT: The distance sensor on the e-Puck is very noisy! You can right-click the robot and click "Show Robot Window" to view sensor readings. You can then place the robot near obstacles and see what the sensor values are to use as thresholds in your code.

HINT: There are many ways to pause the robot while still in the main loop. One could be to set the wheel velocity to 0. We have provided one such way to achieve that through the following code snippet which might be helpful for Part 2.

This function will let you "pause" execution of your controller for duration seconds. We can't use time.sleep() within the simulator because simulation time can be paused and run at speeds other than real-time. This snippet assumes that your robot object is named robot. You can provide a 'tiny' duration which in a loop can create a pause given the appropriate state.

```
def sleep(duration):
    # Waits for duration seconds before returning.
    global robot
    end_time = robot.getTime() + duration
    while robot.step(TIME_STEP) != -1 and robot.getTime() < end_time:
        pass
```

Part 3: Lab Report

Create a report that includes/answers the following:

1. The names of everyone in your group, and your team's name

2. Screenshots of your world from Tutorial 1, 2, 3, and 4.

3. A drawing of your state machine. Make sure all the states and transitions are labeled and that it is faithful to your implementation.

4. Screenshot of the world you created with the obstacles clearly visible.

5. Were you able to get your controller to complete the task? If not, which parts failed? Why?

6. A statement indicating whether you have worked with Python before, and if so, describe your experience.

7. How much time did you spend programming **Part 2** of this lab?

Please submit a zip file containing your Lab Report in PDF form, your Webots world file (.wbt), and Controller file (.py) on Canvas. We only need one submission per group.