Homework 4: Python Programming I - Sense, Compute, Act

Due: October 2, 2014

You must hand in your code on Owlspace and bring printouts to class. **DO NOT** make us print your code, that will cost you 25% of your grade. All of your code should be contained in a single python file called PS04_*netid*.py, where *netid* is your netid. Be sure to comment your code so that the graders will know what you are trying to do. Please be sure to write on your printed code and your python file who you worked with, and who had what role.

Note: You can't do this homework in one night, especially if you are new to programming. Start early, and come to office hours and tutorial. Don't get stuck. One of the keys to engineering is experimentation. While it is important for you to think through what your software should be doing, it is equally important for you to try things out. Do not be afraid to have your robot run code that you know is not exactly correct. You will very likely learn something from the attempt that will help you improve your code. If you wait for your program to be perfect before trying it out, you will have a very hard time completing this assignment:

Think, program, test, repeat!

1 Driving in a Square

For the first half of this problem set, you will program your robot to drive in a square.

1.1 Write motion helper functions

Finish the four motion helper functions:
`move_stop(time)`,
`move_forward(time)`,
`move_rotate_right(time)`, and
`move_rotate_left(time)`.
Each of these functions takes one argument, `time`. They move the wheels in the proper directions for the desired time to produce motion. Note that we have defined a global variable, `MOTOR_PWM` to use as the PWM value in all of these helper functions. This way, if you want to change the PWM, you can do so in one place. Don't use a PWM of more than 80 unless you have a lot of space, or your robots could impact things harder than we would prefer.

Hand-in: Write `move_stop(time)`**,** `move_forward(time)`**,** `move_rotate_right(time)`**,** `move_rotate_left(time)`**. (2 pts each)**

1.2 Test your helper functions

It is important to test your modules before you build them into larger programs. Test each of your helper functions with the `move_test()` function.

Hand-in: Use the `move_test()` **function to test all of your motion functions. Verify that each function is moving the robot in the proper direction. (4 pts)**

1.3 Square motion

Use your motion functions to write `square_motion()`, which will drive the robot in a 1 foot $\times$ 1 foot square. Your answer must use a for loop, and your robot must use `move_stop()` after the square is finished. It might be easier to write this part twice, first without the for loop, then see what you've duplicated and add the for loop later.

Hand-in: Use the motion helper functions and a for loop to write `square_motion()`**. (5 pts)**

2 Driving Towards Light

For the second half of the problem set, we will use these motion functions and the light sensors to drive the robot towards light.

2.1 Write `light_diff`

This function should read the two front light sensors, then return the difference between the left sensor and the right sensor: diff = left − right.

Hand-in: Write `light_diff()` **(3 pts)**

2.2 Test `light_diff`

We've given you a test function, `light_diff_test()` that is more complicated that you might expect. Use it to test your function. Note: You do not need to modify `light_diff_test()`. Do not aim the flashlight at the robot until it starts printing data.

Hand-in: Test your `light_diff()` **function. Record the values when you have a flashlight aimed at the left sensor, between the two front sensors, and then at the right sensor. Explain why we are storing the initial diff value in** `diff_start` **(2 pts)**

2.3 Move towards light

Use your `light_diff()` function to complete `light_follow()`. We've given you a bit of starter code. Use an `if, elif, else` structure and your motion primitives from Section **??** to make the robot drive towards the light.

Hand-in: Use the motion helper functions, `light_diff()`**, and a** `if, elif, else` **structure to write** `light_follow()` **(10 pts)**

3 Avoiding Obstacles with the Bump Sensors

Driving towards things is only half of what robots do. Now let's drive *away* from things. In the third half of the problem set, we will move away from obstacles we run into using the bump sensors.

3.1 Test `bump_test()`

We've given you three nifty functions: `bump_left_get_value()`, `bump_front_get_value()`, `bump_right_get_value()`. They each return a boolean variable indicating if the bump sensor is pressed from the indicated direction. First, let's test these helper functions. Write a function `bump_test()` that uses an infinite while loop to read, then print the values of these three functions every 50 milliseconds. Print the values on a single line so they are easy to read. You can print multiple variables to a single line like this:

```
a = 4
b = 3.14159
c = True
print a, b, c
```

Hand-in: Write the `bump_test()` function. This function must use an infinite while loop and have a 50 ms delay. Can the more than one bump function return `True` at the same time? Is this surprising? (5 pts)

3.2 Moving away from walls

Finish `bump_avoid()` to move the robot away from collisions. We've given you a bit of starter code. Use an `if, elif, elif, else` structure and your motion primitives from Section **??** to finish this. This answer will look similar to the answer from Section **??**.

Hand-in: Use the bump sensor helper functions and a `if, elif, elif, else` structure to write `bump_avoid()` (10 pts)

4 Avoiding Obstacles with the IR Sensors

Running into walls is sooo last problem. In the fourth half of this problem set, we will avoid walls altogether using the IR system to detect them from a distance.

4.1 Test `obstacle_detect`

We've given you a nifty function: `obstacle_detect()`. It returns a tuple of boolean variables indicating where the obstacle is relative to the front of the robot. Test this helper function. Write a function `obstacle_detect_test()` that uses an infinite while loop to print the value of `obstacle_detect()` every 50 milliseconds. Move your hands around the robot and watch the output. The range is pretty far, you will need to get your robot away from things around you to see the program work. **Note: this question might be easy. Like one additional line easy.**

Hand-in: Write the `obstacle_detect_test()` function. This function must use an infinite while loop and have a 50 ms delay. Note how nifty the output from `obstacle_detect()` is. Record the average range it detects your hand when in front of the robot (5 pts)

4.2 Avoiding walls

Use `obstacle_detect()` to move the robots away from walls. We've given you a bit of starter code. Use an `if, elif, else` structure and your motion primitives from Section **??** to finish this. This answer will look similar to the answer from Section **??**, copy and paste your code and modify it, don't start from scratch. We have given you the code to call `obstacle_detect()` and unpack the tuple into three variables: `obs_front, obs_left, obs_right`. After they have been unpacked, you can forget about the tuple, and use these booleans just like any other variable.

Hand-in: Use the motion helper functions, `obstacle_detect()`, and a `if, elif, else` structure to write `obstacle_avoid()` (10 pts)