MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.099—Introduction to EECS I
Fall Semester, 2006

Assignment for Week 6

- Issued: Thursday, October 12
- Also includes post-lab homework due before class on Tuesday, Oct. 17

Phototropism

Now we have two different, interesting kinds of sensors on our robots: sonars for measuring distance, and photoresistors for detecting light. We've already seen that we can measure the light by measuring the resistance of the photoresistor and use that to control the robot. Now, we'll put two photoresistors on the robot and use the difference in light in the two eyes to control the robot.

1. To do in lab on October 12

Biclops

Get a robot and one of the head platforms from last week. Using lego, velcro, spit, and baling wire, mount a protoboard horizontally on the front of the robot's head, facing forward, so that when you put two photoresistors on the board, they can act as "eyes" for the robot.

Since last week, we found that the NI box has another mode, with a different internal resistor network configuration. Our latest version of the `NIDAQ` software puts the box into this new mode, which is actually more straightforward to deal with. We have given you a program, `doublereading.py` that reads the voltages (compared to ground) at both `AI0` and `AI1` and prints them out.

- What reading do you get, in this new mode, when the voltage difference between `AI0` and ground is zero?
- Build a copy of your circuit from last week for measuring the resistance of the photoresistor. Does it still work?

Make two copies of your circuit from last week, or a different one, as long as it lets you read a good range of light values, on your protoboard, with the photodetectors mounted on opposite sides of the board. Connect the outputs of the circuits to ports "AI0" and "AI1" of the NI box. Be sure you're getting good readings from both ports.

We would like to use the difference in light values in the robot's "eyes" to decide which way to turn, in order to move the robot closer to the light. You will probably find that adding a "nose" (actually, a divider sticking out between the eyes), will result in a more useful difference between light values of the eyes.

Checkpoint: 2:00 PM

- Demonstrate a working biclops head, from which you can get two light readings. Show that it is set up so that you can tell the difference (just by looking at the printed-out values) between a situation when the light is to the right of the robot and one in which it's to the left.

Interacting with light

Valentino Braitenberg, in his book *Vehicles*, provides a nice way of thinking about simple robots that move in reaction to sensor input. Imagine a robot with two light sensors and two wheels, where you can make connections between the light sensors and the wheels.

Changing control abstractions

We've been using a control abstraction for our robots in which we command separate forward and rotational velocities. For this part of the lab, we'd like, instead, to generate velocity commands for the wheels separately. This is a nice example of changing the "abstraction" we use to think about how the robot works.

Thing about how to convert from commanded left and right wheel velocities to a forward velocity and a rotational velocity. Write a procedure that takes two values `leftVel` and `rightVel`, intended as velocities for the left and right wheels, as input and makes a call to our old `motorCommand` procedure.

A Tale of Four Vehicles

Now, we'll consider four different ways of connecting lights to wheels. Think about each of these first, and predict what it will do. Then choose two of them to implement and test.

There are two simple ways of connecting a light sensor to a motor:

- **Direct:** so that an increase in the light means an increased velocity;
- **Inverted:** so that an increase in the light means a *decreased* velocity.

There are two simple ways of connecting the light sensors to the motors:

- **Straight:** left sensor connected to left motor and right sensor connected to right motor;
- **Crossed:** left sensor connected to right motor and right sensor connected to left motor.

Now, pick the two most interesting of those behaviors, and get your robot to do them. You'll need to put the laptop on the robot to test these things out, but the laptops have a short battery life; so try to remember to keep your laptop plugged in when the robot isn't rolling around.

Checkpoint: 3:00 PM

- Demonstrate your program for changing the control abstraction.
- Explain the expected behavior of each of the 4 combinations of behavior above.
- Demonstrate your robot doing the two behaviors you picked. What makes them work well or poorly?

Seeking the Light

Using either the original control abstraction or the new one you just made, make a really good light-seeking behavior. The robot should move toward the light reliably and stop when it's very close.

Checkpoint: 3:30 PM

- Demonstrate your light-seeking behavior.

Obstacle Avoidance

Our next goal is to integrate light seeking with obstacle avoidance, so that your robot will be attracted to the light, but will still avoid some obstacles in its path. One way to do this is to our utility function model: you could make a new utility function for the `seek-light` behavior and add it to the old utility function for the `avoid` behavior. Alternatively, you could just try to write a single program that considers both the sonar readings and the light sensor readings, and tries to drive the robot toward the light without running into things.

Checkpoint: 4:00 PM

- Demonstrate your robot chasing light while avoiding obstacles.

Local Optima

You might have found that your program from the previous section was prone to local optima. That is, that your robot could find an obstacle directly between itself and the light that was beckoning to it. Some kinds of local optima can be escaped by adding a little bit of randomness to the robot's behavior (why)? But a robot trapped in a cul-de-sac, as shown in figure 1, will have to be smarter to escape.

One effective method for getting out of such jams is the "Bug 2" algorithm[1]. The idea is illustrated in figure 2. Here's the algorithm:

- Compute the line m from start to goal

[1]It is often better, but sometimes much worse than another algorithm called "Bug 1". For more info see `http://www.cs.jhu.edu/~hager/Teaching/cs336/Notes/Chap2-Bug-Alg.pdf`

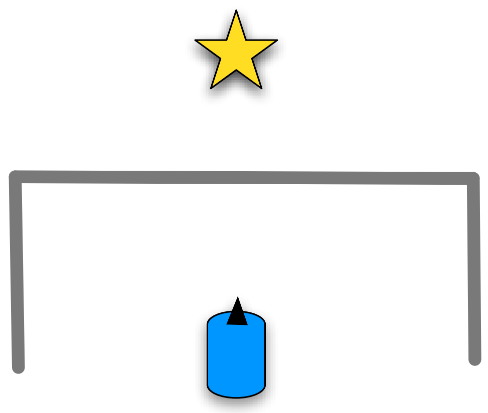
Figure 1: Robot in a cul-de-sac.

- Follow that line until obstacle is encountered
- Follow the obstacle (in either direction) until you encounter the m line again, *closer to the goal*
- Leave the obstacle and continue toward the goal

This algorithm is hard to implement on a real robot; think about why. But maybe you can gain inspiration to implement your own algorithm that can get your robot out of a cul-de-sac. Try to make it as generally useful as you can.

Checkpoint: 4:45 PM

- Explain why Bug2 would be hard to implement on our robots.
- If we hadn't added the *closer to the goal* requirement in the Bug 2 algorithm, what would it have done on the environment in figure 3? Try it with the robot turning to the right when it hits the obstacle, and with it turning left.
- What would your algorithm do on that domain? (Don't worry if it doesn't work well).
- Demonstrate and explain your program for getting out of cul-de-sacs.

2. To do before October 17

What to turn in

Carefully, coherently written answers to the questions from the last checkpoint. Include a description of your approach to avoiding cul-de-sacs and how it compares to the Bug 2 algorithm. If you didn't get your implementation to work, then just describe what you were going to do, as clearly

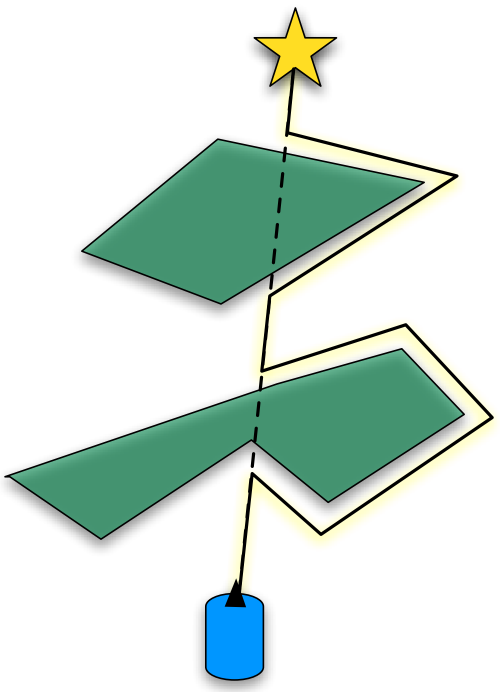

Figure 2: The Bug2 algorithm.

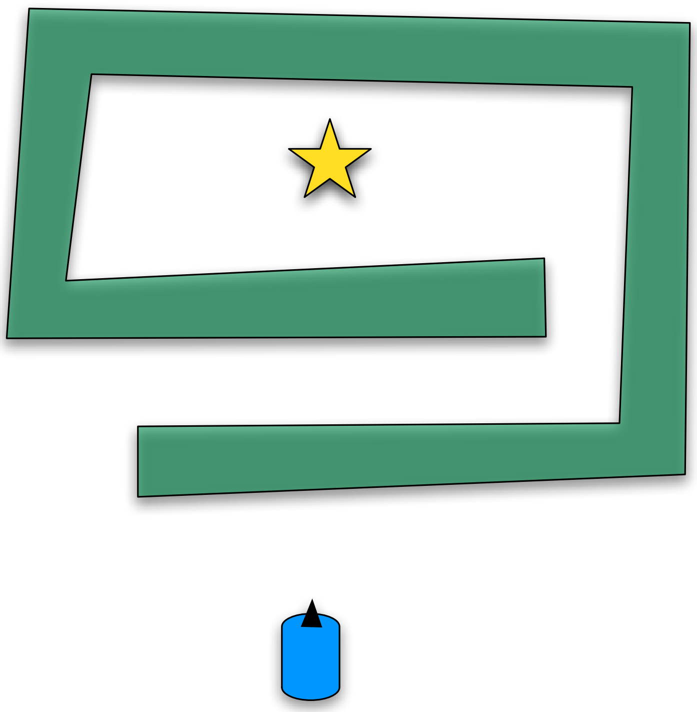

Figure 3: What would Bug2 do in this domain without the *closer to goal* test?

and carefully as possible, and what behavior you think it would have generated. Give a configuration of obstacles for which your algorithm behaves stupidly (or prove that it never does, and you can publish a paper!).

Concepts covered in this assignment

Here are the important points covered in this assignment:

- More practice with circuits and programming
- Integrating new sensors into the robot
- Interesting behavior arising with simple control rules
- Local navigation methods for mobile robots