Design Lab 2: Controlling Robots

Get a lab laptop, in the Terminal window, type:

```
> athrun 6.01 update
```

Warning: if your robot starts to go too fast or get away from you, ***pick it up!!***

Be sure to mail all of your code to your partner and remember to do the upload questions on the tutor that relate to this lab.

1 Simple Brains

1. Get a lab laptop and a robot and a serial cable (per pair of students).
2. **Run a brain in the simulator.**
   a. In the Terminal window, type `soar &`.
   b. Click soar's **Simulator** button, choose `tutorial.py`, click Open.
   c. Click soar's **Brain** button, navigate to `Desktop/6.01/lab2/designLab/smBrain.py`, and click Open.
   d. Click soar's **Start** button, and let the robot run for a little while.
   e. Click soar's **Stop** button.
   f. Notice the graph that was produced; it shows a 'slime trail' of the path that the robot followed while the brain was running. You can just close the window. (If you don't want the brain to produce a slime trail, you can set the `drawSlimeTrail` argument to the `RobotGraphics` constructor to be `False`).
3. **Modify the brain and run it.**
   a. Click Idle's **File** menu, select **Open...**, navigate to `Desktop/6.01/lab2/designLab/smBrain.py`, and click **Open**.
   b. The `Action` object returned as the output by the `getNextValues` method of the `MySmClass` has two attributes that are important to us:
      - `fvel` specifies the forward velocity of the robot (in meters per second)
      - `rvel` specifies the rotational velocity of the robot (in radians per second), positive rotation is counterclockwise

c. Find the place where the velocities are set in the brain, and then modify it so that it makes the simulated robot rotate in place.

d. Save the file.

e. Go back to the soar window and click the **Reload Brain** button

f. Run the brain by clicking the **Start** and then the **Stop** buttons.

4. **Run it on the robot**

a. Connect the robot to your laptop, making sure the cable is tied around the handle in the back of the robot.

b. Power on the robot, with a switch on the side panel.

c. Click soar's Pioneer button, to select the robot.

d. Load the brain you want.

e. Click soar's Start button.

f. One partner should be in charge of keeping the robot safe. Keep the cable from getting tangled in the robot's wheels. **If the robot starts to get away from you, pick it up!!** Then, turn it off using the switch on the robot.

2 Sonars

The `inp` argument to the `getNextValues` method of `MySMClass` is an instance of the `soar.io.SensorInput` class, which we have imported as `io.SensorInput`. It has two attributes, `odometry` and `sonars`. For this lab, we will just use the `sonars` attribute, which contains a list of 8 numbers representing readings from the robot's 8 sonar sensors, which give a distance reading in meters. The first reading in the list (index 0) is from the leftmost (from the robot's perspective) sensor; the reading from the rightmost sensor is the last one (index 7).

Now we will investigate the behavior of the sonar sensors. **Don't spend more than 10 or 15 minutes on this.**

- Modify the brain so that it sets both velocities to 0, and uncomment the line `plotSonar(3)`. Reload the brain and run it. You should see a continuous plot of the values of `inp.sonars[3]`, which is one of the forward facing ones.
- From how far away can you get reliable distance readings? What happens when the closest thing is farther away than that?
- What happens with things very close to the sensor?
- Does changing the angle between the sonar transducer and the surface that it is pointed toward affect the readings? Does this behavior depend on the material of the surface? Try bubble wrap versus smooth foam core.
- Now, comment out `plotSonars(3)` and set the `sonarMonitor` argument to the `RobotGraphics` constructor to be `True`. Reload the brain and run it. This will bring up a window

that shows all the sonar readings graphically. The length of the beam corresponds to the reading; red beams correspond to "no valid measurement". Test that all your sonars are working by blocking each one in turn. If you notice a problem with any of the sensors, talk to the staff.

Checkoff 1. Explain to a staff member the results of your experiments with the sonars.

Make the robot move to within 0.5 meters of an obstacle and keep it at that distance, even if the obstacle moves back and forth. Do this by editing the `getNextValues` method of `MySMClass`; there is no need to change any other part of the brain. Debug it in simulation, then run it on a real robot.

Checkoff 2. Demonstrate your distance-keeping brain on a real robot to a staff member.

3 Following Boundaries

Our goal now is to build a state machine that controls the robot to do a more complicated task:

1. When there is nothing nearby, it should move straight forward
2. As soon as it reaches an obstacle in front, it should follow the boundary of the obstacle, keeping the right side of the robot between 0.3 and 0.5 meters from the obstacle.

Draw a state-transition diagram that describes each distinct situation (state) during wall-following and what the desired output (action) and next state should be in response to the possible inputs (sonar readings) in that state. Start by considering the case of the robot moving straight ahead through empty space and then think about the input conditions that you encounter and the new states that result. Think carefully about what to do at both inside and outside corners. Remember that the robots rotate about their center points. Try to keep the number of states to a minimum.

Checkoff 3. Show your state-transition diagram to a staff mamber. Make clear what the conditions on state transitions are, and what actions are associated with each state.

Copy your current `smBrain.py` file to `boundaryBrain.py` (you can do this with **Save As** in idle), and modify it to implement the state machine defined by your diagram. Make sure that you define a `startState` attribute and a `getNextValues` method.

Try hard to keep your solution simple and general. Use good software practice: do not repeat code, use helper procedures with mnemonic names, try to use few arbitrary constants and give the ones you do use descriptive names.

To debug, add print statements that show the relevant inputs, the current state, the next state, and the output action.

Record a slime trail of the simulated robot following a sequence of walls; make sure that it can handle outside and inside corners. Can you handle very sharp corners, such as the L in `tutorial.py`?

Checkoff 4. Demonstrate your boundary follower to a staff member. Explain why it behaves the way it does.

6.01 Introduction to Electrical Engineering and Computer Science I
Fall 2009