Cohort Session 3, Week 9
Robotic Boundary Follower

Objectives
1. Implement simple state machines that make robots perform simple tasks.
2. Experiment with robotic sonar sensors.
3. Construct sophisticated state machines that make robots maintain distances from obstacles and follow boundaries.

Please work on this lab in a group of **two or three**. Be sure to email your partner all the modified code, printouts and data. You may have to use them during your exams.

1 Equipment & Software

Download software for this lab from Week 9 on **1D Project website**.

Each group should have:

1. A robot with cable attached.
2. A white styrofoam board with bubble-wrap on one side.
3. `smBrain.py`, which specifies a simple robot 'brain' using the state machine class in the Python module `sm`.
4. `boundaryWorld.py`, which specifies a virtual world for a robot simulator.

WARNING!
- If the robot travels too fast and gets away from you, pick it up quickly to stop it from colliding into anything.
- The robot is to be placed on the floor at all times. Do NOT put it on a table, lovingly adore it on your lap, balance it on your fingertips, etc.
- Take **EXTREME CARE** when attaching and removing the cable from the robot's socket – it does not take too well to rough handling.

2 Warm-up: Turn in Place

A 'brain' is a Python program that specifies the behavior of a robot.

Tasks
1. Run a brain in a robot simulator named *soar*.
2. Write a brain to rotate the robot in place.
3. Run the brain on a real robot.

Instructions: On your laptop, do the following.

1. **Run a brain in the *soar* simulator.**
   a. In a terminal window, type `soar &` to launch the robot simulator.
   b. Click soar's **Simulator** button, navigate to `boundaryWorld.py`, and open it This loads the virtual world specified by `boundaryWorld.py` into our robot simulator.
   c. Click soar's **Brain** button, navigate to `smBrain.py`, and open it. This loads the state machine specified by `smBrain.py` into the robot simulator. That state machine describes the actions that the robot will take in response to the information it receives via its sonar sensors about the virtual world around it.
   d. Click soar's **Start** button, and let the robot run for a little while.
   e. Click soar's **Stop** button.
   f. The graph that is produced shows the path that the robot followed ('slime trail') while the brain was running. You can just close the window. (If you don't want the brain to produce the path, simply set the `drawSlimeTrail` argument in the `RobotGraphics` constructor in `smBrain.py` to `False`).
2. **Write a brain to rotate the robot in place.**
   a. In a terminal window, type `idle &` to open up an Idle environment or `canopy &` to open up Canopy environment.
   b. Click Idle's **File** menu, select **Open...**, navigate to `smBrain.py`, and click **Open**. If you use Canopy, select **Text Editor**, and select the file to open.
   c. The state machine that controls the robot's actions is defined by the `MySMClass` definition. Think of this state machine as taking sensory data as input, and returning as output instructions to the robot on how to behave. The `io.Action` object that is output by the `getNextValues` method of the `MySmClass` tells the robot how to change its behavior, and has two important attributes:
      - `fvel`: specifies the forward velocity of the robot (in meters per second).
      - `rvel`: specifies the rotational velocity of the robot (in radians per second), where positive rotation is counterclockwise.
   d. Find the place where the velocities are set in the brain, and modify them so the robot rotates in place.
   e. Save the file.
   f. Go back to the `soar` window and click the **Reload Brain** button.

g. Run the brain by clicking **Start** and then **Stop**.

3. **Run the brain on a real robot.**
   a. Connect the robot to your laptop by plugging the cable into your laptop's USB port.
   b. Turn on the robot (its switch is on its underside).
   c. Click soar's `eBot` button to select the robot. Its sonar sensors will start producing a ticking sound.
   d. One member of the group should be in charge of keeping the robot safe. Take care to prevent the cable from tangling in the robot's wheels. **If the robot starts to get away from you, pick it up, and turn it off.**
   e. Click soar's **Start** button.

3 Sonars

Task
Investigate the behavior of the robot's sonar sensors. **Do not spend more than 10 minutes experimenting with the sonars. When you are done, ask a staff member for a checkoff.**

The `inp` argument to the `getNextValues` method of `MySMClass` is an instance of the `io.SensorInput`. It has two attributes, viz., `odometry` and `sonars`. For this lab, we shall use the `sonars` attribute, which contains a list of 6 numbers representing readings from the robot's 6 sonar sensors at its front (we will not be using the sensors at its back), each giving a distance reading in meters. The first reading in the list (index 0) is from the leftmost sensor, and the last reading (index 5) is from the rightmost sensor (from the robot's perspective).

Instructions:

a. Set both forward and rotational velocities to 0, and uncomment the line `print inp.sonars[3]` in the `step` method. Reload the brain and run it. It will print the value of `inp.sonars[3]`, which is the reading from one of the forward-facing sonar sensors.

- From how far away can you get reliable distance readings?
- What happens when the closest thing is farther away than the above?
- What happens with things very close to the sensor?

b. 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 each beam corresponds to a 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, inform a staff member.

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

4 Keep your Distance, Buddy!

Task
Write a brain to keep the robot a fixed distance from an obstacle.

Instructions:

a. Edit the `getNextValues` method of `MySMClass` to make the robot move forward to approximately 0.5 meters of an obstacle (e.g., the bubble-wrapped styrofoam board) in front of it, and maintain that distance even if the obstacle moves back and forth. (To avoid unnecessary headaches, do not change any other part of the brain.) **Do not set the forward velocity higher than 0.2 (or lower than -0.2).**

b. Debug your code in soar with the virtual world specified in `boundaryWorld.py`. (Click on soar's **Simulator** button and select `boundaryWorld.py`. Reload your brain.)

c. After your code runs without a hitch on soar, run it on the real robot by clicking soar's **eBot** button.

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

5 Following Boundaries

Task
Build a more sophisticated brain (state machine) that makes the robot follow boundaries in the following manner:

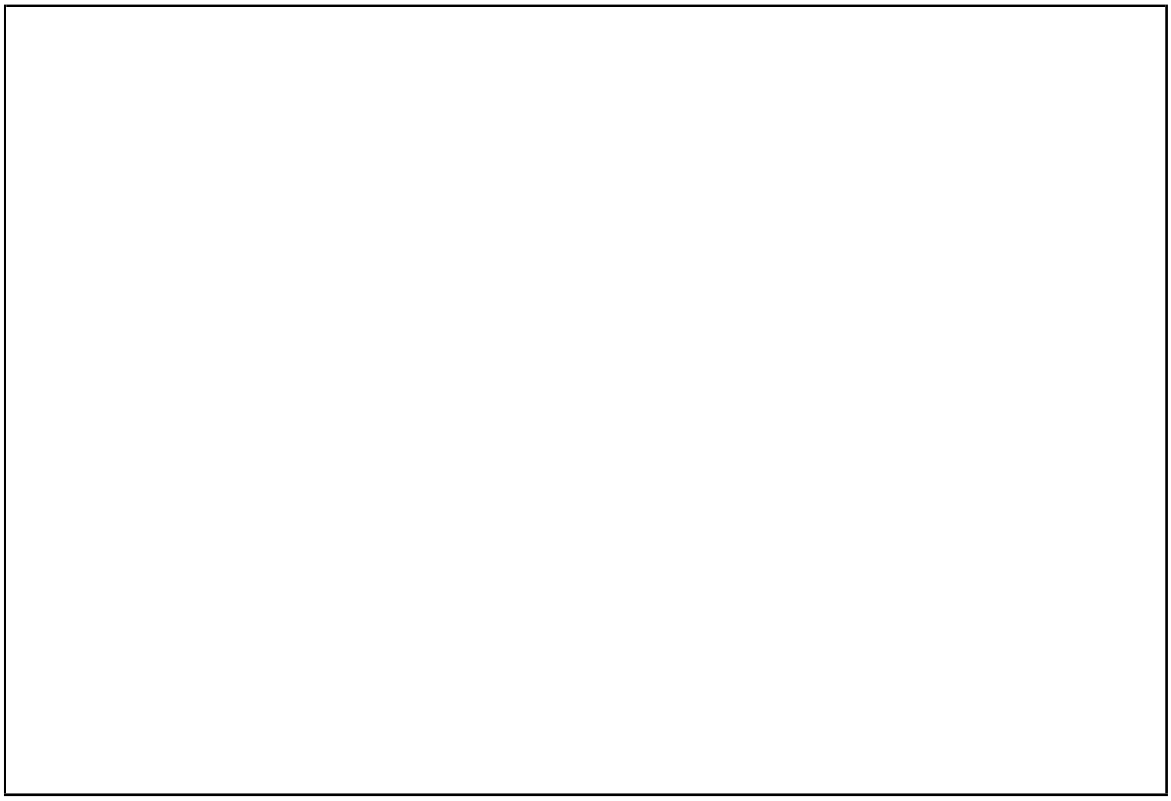
- The robot should move straight ahead when there is nothing nearby.
- As soon as it reaches an obstacle, the robot should follow the obstacle's boundary, keeping the obstacle on its right at a distance of between 0.3 and 0.5 m.

Instructions:

a. Draw a state-transition diagram that describes each distinct situation (state) during boundary following. In each state, clearly show the desired output (action) and next state in response to each possible input (sonar readings). Here are some hints:

- Start by considering the case of the robot moving straight ahead through empty space.

- Then think about the input conditions that it could encounter and the new states that could 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 member. **Clearly** show the conditions for every state transition, and the actions that are associated with each state.

b. Copy your current `smBrain.py` file to `boundaryBrain.py` (with **Save As** in Idle or Canopy), and modify it to implement the state machine defined by your diagram. Make sure that you define a `startState` attribute and a `getNextValues` method. To debug, add print statements that show the relevant inputs, the current state, the next state, and the output action.

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

c. Record the path (slime trail) of the simulated robot following a sequence of walls,

demonstrating its ability to handle outside and inside corners. (Going around very sharp corners or hairpin turns, such as the L in `boundaryWorld.py`, is not required, but will make you look ultra-cool to your labmates and the staff.)

Checkoff 4
Demonstrate your boundary follower to a staff member. Explain why it behaves the way it does. **Remember to mail your code to your partner!**