Cohort Session 3, Week 9
Robotic Boundary Follower

Objectives
1. Implement simple state machines that make robots perform simple tasks.
2. Experiment with robotic infrared sensors.
3. Construct sophisticated state machines that make robots follow boundaries.

Please work on this lab in a group of **five**. 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 set of papers with black color region printed on one of the sides. This is used to construct the boundary.
3. `smbrainsim.py`, which specifies a simple robot 'brain' using the state machine class in the Python module `sm`.
4. `boxworld.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.
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 simulator.**
   a. Open `smbrainsim.py` using any text editor. This file contains a template to run a robot simulator utilizing Python’s turtle module.

   b. Notice that the template imports the file `boxworld.py` which contains the virtual world definition. The convention is to name the world as `thymio_world` inside any world file and to import it in your brain file. Notice that the variable `thymio_world` is used to initialized the world when creating the object in `smbrainsim.py`.

   c. To define the behaviour of the state machine, one has to define `start_state` and `get_next_values()`. The template `smbrainsim.py` shows you how to write these codes in order to rotate the robot in place.

   d. Run the file `smbrainsim.py`. A window will appear drawing the boundaries and rotate the robot in place. To stop, close the window.

2. **Write a brain to move the robot in a spiral.**
   a. Edit the file `smbrainsim.py`.

   b. 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 `get_next_values` 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.

   c. Find the place where the velocities are set in the brain, and modify them so the robot moves in an outward spiral clockwise.

   d. Save the file.

   e. Run the brain file.

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. 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.**

d. Copy the file `smbrainsim.py` to a new file called `smbrainreal.py`.

e. Fine the line that contains `ThymioSMSim(MySM, thymio_world)` and change it to `ThymioSMReal(MySM)`.

f. Save the file `smbrainreal.py` and run it.

3 Infrared Sensors

Task
Investigate the behavior of the robot's infrared sensors. **Do not spend more than 10 minutes experimenting with the infrared sensors.**

The `inp` argument to the `get_next_values` method of `MySMClass` is an instance of the `io.Input`. It has several attributes, viz., `prox_horizontal`, `prox_ground`, `temperature` and `accelerometer`. For this lab, we shall use the `prox_ground` attribute, which has three attributes `delta`, `reflected`, and `ambiant`. Each of this attribute is a list of two elements, one for the left and the other one for the right ground sensors.

Instructions:

a. Set both forward and rotational velocities to 0, and uncomment the lines `ground = inp.prox_ground.delta`, and the following lines:

```
ground=inp.prox_ground.delta
left=ground[0]
right=ground[1]
print(left ,right)
```

Use this code to print the values of the delta in the ground proximity sensors. Change to reflected or ambiant to get the other values. Answer the following:

- What are values of `delta`, `reflected`, and `ambiant` when the robot is on a white floor?
- What are values of `delta`, `reflected`, and `ambiant` when the robot is on a black floor?
- What are values of `delta`, `reflected`, and `ambiant` when the left sensor on the white floor and the right sensor is on the black floor?

4 Do Not Cross, Buddy!

Task
Write a brain to stop the robot at the boundary.

Instructions:

a. Edit the `get_next_values` method of `MySMClass` to make the robot move forward to approximately at the boundary between the white and the black. Use the ground proximity sensor to detect the boundary. (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).**

Checkoff 1
Demonstrate your stopping robot at the boundary 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 use its ground sensor `io.Input.prox_ground` to detect the color of its ground.
- The robot should move straight ahead when there is nothing nearby.
- As soon as it detects a black ground, the robot should rotate left and follow black ground's boundary with the black region on the right hand side of the robot.

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 2
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. Open `boundarybrain.py` and modify it to implement the state machine defined by your diagram. Make sure that you define a `start_state` attribute and a `get_next_values` 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. Test it on an actual robot.

Checkoff 3
Demonstrate your boundary follower to a staff member. Explain why it behaves the way it does. **Remember to mail your code to your partner!**