CS115/PHYS115
Robotic Design Studio
Wintersession 2009

Challenges

Below is a list of challenges that you will be asked to complete during the first 6 or 7 class meetings. You should document each of the challenges in your design journal.

Challenge 1: Kinetic Sculpture

With one or two other partners, build a *very* simple kinetic sculpture out of Lego parts that is controlled by a *PicoCricket* with a simple *PicoBlocks* program. Your sculpture should include two different kinds of actuators (*e.g.*, motor, colored lights, sound box, or numerical display) and two different kinds of sensors (*e.g.*, touch sensor, light sensor, sound sensor, or resistance sensor). For example, one sensor might turn a motor on while the other might cause music to play on a sound box.

Include in your design journal a description of the sculpture (including a rough sketch and/or copies of digital photos) and the *PicoBlocks* program.

Here, and for almost every other challenge in the course, you may need to go through several iterations before you achieve the behavior you desire. This is not a bad thing, but an expected aspect of the design process that offers rich opportunities for learning. In your design journal, you should document each iteration of a design, indicating what worked well, what didn't work so well, and what you learned from the experience.

Challenge 2: How Does SciBorg Follow a Line?

The first menu selection on a *SciBorg's Handy Board* is `follow-line`. Executing the program causes the *SciBorg* to follow a black line. The goal of this challenge is to figure out *how* it follows the line – *i.e.*, to determine the algorithm used by the `follow-line` program.

You should work in teams with two or three members. Each team should have one *SciBorg*, a sheet of large paper, and a black marker (you may need to share markers between teams). Perform experiments with the *SciBorg* to deduce the algorithm employed by `follow-line`. In your design journal, document your hypotheses, experiments, and conclusions. You should include an English description of the `follow-line` algorithm that explains all the behaviors you observe, including not only the line-following behavior but also the song-playing behavior.

Pay special attention to figuring out which sensors and actuators participate in the behavior. Recall that the analog sensor display mode (which you get by turning the knob on the *Handy Board* past menu item 7) is a particularly good way to monitor analog sensors. You might also want to experiment with different colors of markers and papers. Can *SciBorg* follow a red line? What happens if you place *SciBorg* on black paper? On the rug?

Challenge 3: Simple SciBorg Modifications

Here you are asked to predict the consequences of some simple modifications to *SciBorg*. Think carefully about your predictions and record them in your design journal. Later (maybe even the next day), test your predictions on an actual *SciBorg* and record your observations. Explain any discrepancies between a prediction and an observation.

a. Changing the Blackness Threshold. *SciBorg's* notion of what constitutes a black line is determined by a "blackness threshold". You can add 10 to this threshold via the `black+10` menu option and subtract 10 via the `black—10` menu option. (Each can be executed multiple times, and each indicates the threshold number that results from the operation.) Predict what will happen if the blackness threshold is set at the minimum value (0) and at the maximum value (250). When testing your prediction, record the range of blackness thresholds in which *SciBorg* exhibits the "normal" line-following behavior.

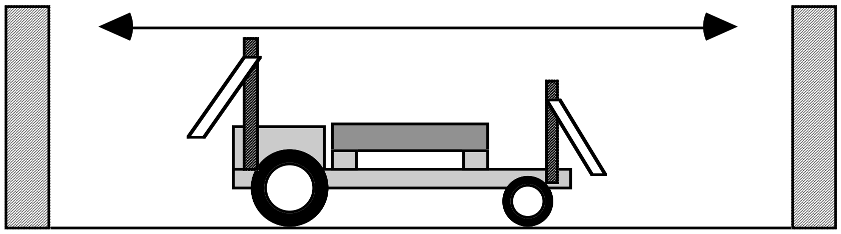
b. Swapping Sensors Assume that the blackness threshold is reset to its initial setting. Predict what will happen to the `follow-line` behavior if you swap the connectors in analog sensor ports 0 and 1.

c. Swapping Motors. Assume that the blackness threshold and sensor ports are reset to their initial settings. Predict what will happen to the `follow—line` behavior if you swap the connectors in motor ports `a` and `b`.

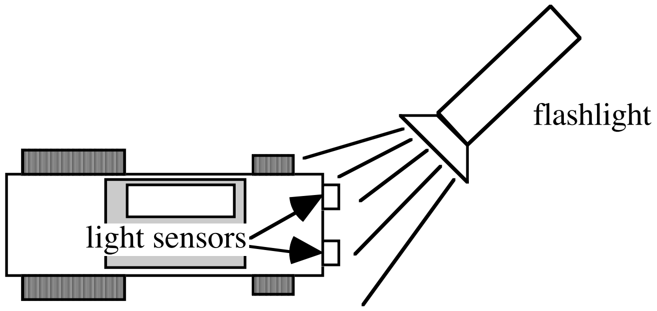
Challenge 4: New *SciBorg* Programs

Below are specifications for the behavior of four new *SciBorg* programs. Working in teams of two or three members, implement (*i.e.*, write *Handy Logo* code for) and test all four of these programs and include the code in your design journal. Remember to use comments to document any aspects of your code that need explanation.

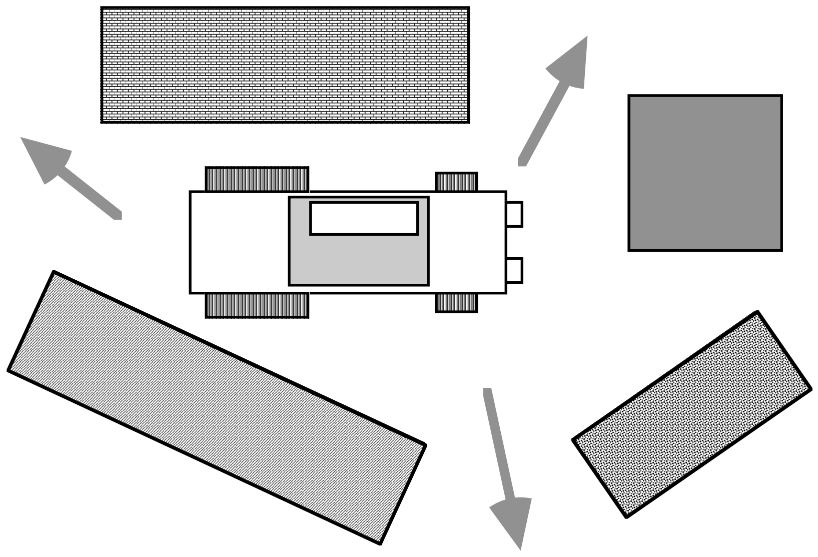
1. ping-pong: Using the front and back bumper sensors, program *SciBorg* to bounce back and forth between two walls or obstacles.



2. follow-light: Using the light sensors at the front of *SciBorg*, program SciBorg to follow a moving flashlight. *Hint*: test the difference between the two light sensor values. *Extra Challenge*: Have the *SciBorg* stop moving when there is insufficient light.



3. escape: Program *SciBorg* to find its way out of a field of obstacles. The `random` procedure is quite helpful here.



4. sobriety: *SciBorg's* line following algorithm causes it to zig-zag back and forth a lot. Modify the line following behavior to reduce the number of zig-zags. That is, try to make *SciBorg* go as straight as possible when following a straight line segment. Does this change the behavior of *SciBorg* at a dead end (*i.e.*, a line that just stops)?

Challenge 5: Sensor Interaction

On your own, or in a small group, write the following two *Handy Logo* programs, in which switches 7 and 8 independently control motors **a** and **b** (respectively). In both programs, the switches can be pressed in any order and any number of switches (zero, one, two) may be pressed at once. Do not use any concurrency commands (e.g. `launch`, `forever`, `when`, `every`). Test your programs to make sure they work, and include them in your design journal.

1. ab-on-off: Motor **a** is on when switch 7 is pressed and is off otherwise; motor **b** is on when switch 8 is pressed and is off otherwise.

2. ab-toggle: Pressing switch 7 toggles motor **a** and pressing switch 8 toggles motor **b**.

Note: `ab-on-off` is relatively easy, but `ab-toggle` is *very* hard. One hint we can give is that you will need to use “global variables” to solve `ab—toggle`. Programs like `ab-toggle` are an excellent motivation for the concurrency commands we will study at the end of the second week of the course (see Challenge 12).

Challenge 6: Indestructible Box

By yourself, build a LEGO box that holds at least two red or black "weight bricks" and can be consistently dropped (at least twice in a row without any tweaks) from a height of 2 meters without coming apart. Some of the idioms described in the handout on *Building Strong LEGO Structures* and also Fred Martin's *The Art of LEGO Design* (accessible from the course handout page) are particularly helpful in this challenge. Demonstrate your indestructible box to Robbie or Lyn and write up a brief description of your design in your design journal. As usual, you can expect to go through several iterations before you achieve the goal.

Challenge 7: *PicoCricket* Motion Modules

LEGO motors spin around in circles. But there are lots of other interesting kinds of motions that you might want to incorporate into your robot project. To get a sense of some of the possibilities, build at least one of the "motion modules" posted at:

http://www.picocricket.com/motion.html

Use your motion module and some craft materials to make a whimsical animated creature.

Challenge 8: Single Motor Racing Vehicle

In a group with two or three members, design a vehicle with a single motor, powered by a *PicoCricket,* that can carry a 1.0 kg weight as fast as possible. You should use one of the gray rectangular motors that does *not* have internal gearing. This will force you to experiment with building your own gear trains. It will be helpful to study the handout on *LEGO Gears and Motors* and also the section on gears in *The Art of LEGO Design*.

This is a non-trivial challenge that will require many design iterations on your part. You will have several days to work on this challenge. There will first be a test run in which you will pit your vehicle against others on a 3 meter course. On the following day final competitive event will be held. You should document each iteration of your design in your design journal.

Challenge 9: Everyday Sensors

Many sensors are embedded in machines, devices, pieces of equipment , etc. that you frequently use. In your design journal, make a list of ten sensors you can encounter on the Wellesley campus – e.g., in your dorm, classrooms, common areas, outside, etc. You need not be able to see a sensor in order to deduce that it exists. Try to avoid listing simple switch-like sensors, such as light switches, mouse buttons, faucet handles, etc. The sensor may be embedded in a piece of equipment or machine, such as an automobile or photocopy machine.

Challenge 10: Animal Sensors

Animals display an astonishing variety of behaviors. Many of these behaviors depend crucially on special-purpose sensors possessed by an animal. Find an animal sensor that intrigues you and write a few paragraphs in your design journal describing what the sensor is used for, how it works, and why it interests you. Remember that humans are animals, too. Try to use at least two sources of information, and be sure to give a bibliographic reference for each such source. It's OK to use newspapers, popular magazines and journals (*e.g.*, *Audubon*, *National Geographic*, *Natural History*, *Newsweek*, *Scientific American*, *Time*, *etc.*), and articles posted on the Internet, but you are also encouraged to check more "serious" journals (*e.g.*, *Nature*, *Science*) and books. You may use *Wikipedia* as a starting point, but it doesn't count as one of your references.

Challenge 11: Auto-Thresholding

Constance has built a robot that uses a light sensor as a "shadow detector" – when passersby cast a shadow on the robot, it springs into action. In the robot lab, she notices that whenever anybody walks in front of her robot, the light sensor reading goes above 20. (Remember, higher numbers correspond to less light reaching the light sensor.) She gets her robot working perfectly with the following *Handy Logo* code:

```
to react-to-shadow
    waituntil [(sensor 0) > 20]
    spring-into-action
end
```

On the day of the big robot exhibition, Constance moves her robot to a sunlit exhibition space. In the mid-afternoon, there is so much light that her robot never turns on when people pass by it. After the sun has set, her robot turns on whether or not people are passing by.

Of course, the cause of this heart-breaking behavior is the fact that Constance has "hard-wired" (*i.e.*, fixed as a constant) the threshold value used by her program. A threshold value that works well in one ambient lighting condition may fail miserably in other environments.

A much better strategy is *auto-thresholding*, in which the robot detects and records the ambient lighting conditions each time its main program starts, and uses this information to automatically determine the correct shadow threshold. Your challenge is to implement this strategy by hooking a light sensor to the Handy Board and writing code that can reliably detect shadows under a variety of ambient lighting conditions (e.g., robot lab with lights on and off, Sage lounge, outdoors on a sunny day). Your shadow detector can trigger any simple action of your choice – e.g., turn on a motor, beep, *etc*.

Challenge 12: Modularizing Behavior

On your own, write the following *Handy Logo* programs. Use the concurrency constructs discussed in class. Appreciate how difficult the programs would be if you could not use the concurrency constructs! (Compare to Challenge 5.)

1. `ab-toggle`: Pressing switch 7 toggles motor `a` and pressing switch 8 toggles motor `b`. The switches can be pressed in any order and any number of switches (zero, one, two) may be pressed at once. (This is the same problem as Challenge 5, #2, but here concurrency yields a simpler solution.)

2. `a-toggle-reverse`: Pressing switch 7 toggles motor `a` and pressing switch 8 reverses its direction.

3. `ab-onfor`: Pressing switch 7 turns motor `a` on for one second and pressing switch 8 turns motor `b` on for one second. Each switch should be active even when the other motor is on.