CS116/PHYS116
Robotic Design Studio
Fall, 2010

Challenges

Below is a list of challenges that you will be asked to complete during the first 7 weeks of the course. You should document each of the challenges in your design blog.

Cricket Programs

Challenge 1: Kinetic Sculpture

Day 1 (Tuesday, Sept. 7) With your partner, build a *very* simple "kinetic sculpture" out of LEGO parts and other materials. Your sculpture should be controlled by a *PicoCricket* and a simple *PicoBlocks* program. It should include one or more kinds of actuators (*e.g.*, motor, colored lights, sound box, or numerical display) and one or more kinds of sensors (*e.g.*, touch sensor, light sensor, sound sensor, or resistance sensor). For example, a shadow cast on a light sensor might trigger turn a motor to turn on or a loud clap might cause music to play on a sound box.

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

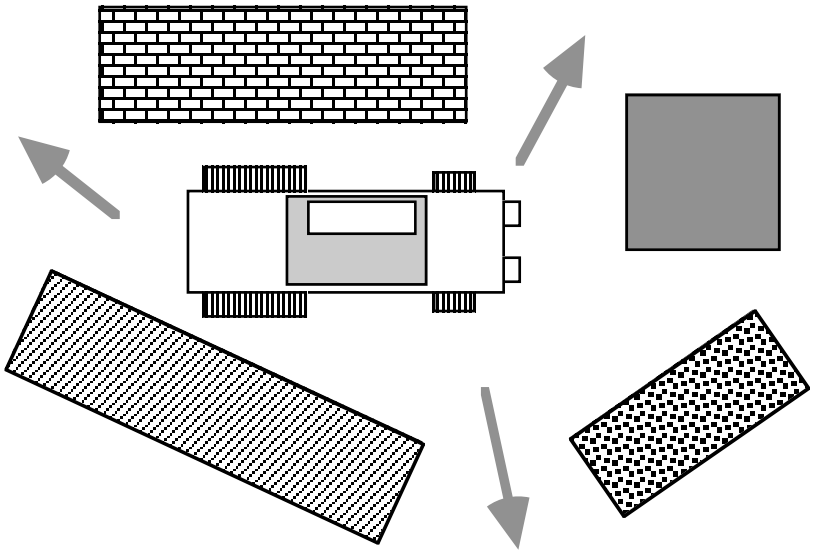
Day 2 (Friday, Sept. 10.) Here, and for almost every other challenge in the course, you should go through several iterations before you achieve the behavior you desire. This is not a bad thing, but an expected aspect of the design process that offers rich opportunities for learning. In your design blog, 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.

In class I will show you several "big ideas" about writing computer programs. One of these ideas is "procedural abstraction". Another is the difference between a "level triggered" event and a "edge triggered" event. I'd like your modified sculpture to include both these ideas. That is, your *PicoBlocks* program should make use of at least one procedure call that simplifies the readability of the program, and you should explain in your design blog whether the action in your sculpture is "level triggered" or "edge triggered". (If you're feeling ambitious, try to incorporate both kinds of triggering in your sculpture.)

Challenge 2: Simple SciBorg Behaviors

1. bounce: Write a program that makes your *SciBorg* go forward until it bumps into an obstacle, then back up for one second.

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



3. going the distance: Program your *SciBorg* to drive a distance of 1.5 meters over various terrain. Use three different approaches based on 1) elapsed time, 2) feedback from the counters, 3) using a reflectance sensor to detect when you've reached the "finish line". Which approach works best?

Challenge 3: *SciBorg* Line Follower

In class I demonstrated a *SciBorg* that was programmed to follow a black line of electrical tape on a white surface using feedback from two reflectance sensors. Your goal in this challenge is to get *your SciBorg* to follow such a line.

Start by writing an English description of the `follow-line` algorithm that explains the behavior you observe in the model *SciBorg*. Then try implementing your algorithm in code.

Can you get your *SciBorg* follow a white line on a black surface?

Challenge 4: Simple Line-follower Modifications

Here you are asked to predict the consequences of some simple modifications to *SciBorg*'s `follow-line` program. Think carefully about your predictions and record them in your design blog. 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". Predict what will happen if the blackness threshold in the program is set at the minimum value (0) and at the maximum value (1000). When testing your prediction, record the range of blackness thresholds in which *SciBorg* exhibits the "normal" line-following behavior.

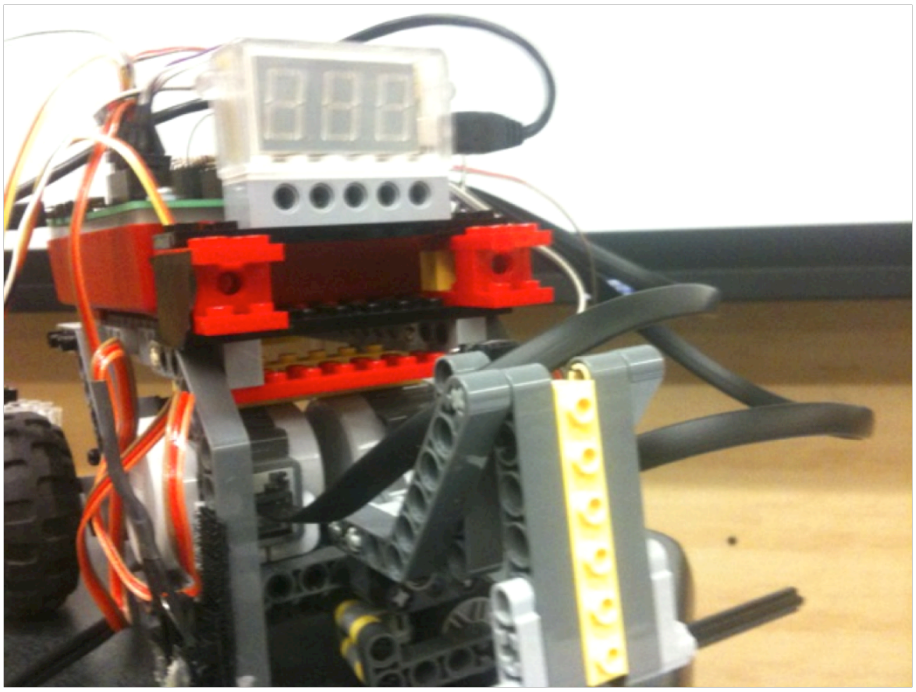
b. Swapping Sensors Assume that the blackness threshold is reset to its initial setting. Predict what will happen to the `follow-line` behavior if you swap the connectors in sensor ports 1 and 2.

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 5: More *SciBorg* Behaviors

Below are specifications for the behavior of two new *SciBorg* programs. Working with your partner, implement (*i.e.*, write *PicoBlocks* code for) and test these programs and include the code in your design blog. Include brief text to explain how your programs work.

> **1. follow-light**: Using two light sensors placed at the front of your *SciBorg*, program SciBorg to "home in" on a bright light source placed about 10 feet away. Assume that your *SciBorg* starts off with a randomly chosen orientation with respect to the light. (Look at the sample *SciBorg* pictured below for guidance on how to mount the light sensors; it's critical that they have "blinders" so that each light sensor can only look out at a relatively narrow range of angles.)



Hint: Base your light finding algorithm on testing the difference between the two light sensor values.

Extra (Optional) Challenges: 1) Have your SciBorg stop moving when there is insufficient light. 2) Use special infrared photodetectors to home in on a lit candle.

2. 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)? Also, try using feedback from the counters to drive straight.

Challenge 6: Sensor Interaction

On your own, or in a small group, write the following two *PicoBlocks* programs, in which switches 1 and 2 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. Test your programs to make sure they work, and include them in your design blog.

> **1. ab-on-off:** Motor `a` is on when switch 1 is pressed and is off otherwise; motor `b` is on when switch 2 is pressed and is off otherwise.
>
> **2. ab-reverse:** Pressing switch 1 reverses motor `a` on and off and pressing switch 2 reverses motor `b`.

Note: `ab-on-off` is relatively easy, but `ab-reverse` is *very* hard if you don't use the second wand. However, if you're up for the extra challenge, there is a way to implement `ab-reverse` in a single stack. (*Hint:* you will need to use the "variables" feature found in the *My Blocks* category.)

LEGO Structures and Mechanisms

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: 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 9: 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 on a 3 meter course.. You should use one of the gray rectangular motors that does *not* have internal gearing. This will encourage 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 three class sessions to work on this challenge. During the second class there will be a trail run in which you will pit your vehicle against others on the 3 meter course. During the third class the final competitive event will be held. You should document each iteration of your design in your design blog.

Sensors

Challenge 10: Build Your Own Sensor

Scratch is a new programming language that is similar to *PicoBlocks*. *Scratch* makes it easy to create your own interactive stories, animations, games, music, and art -- and share your creations on the web.

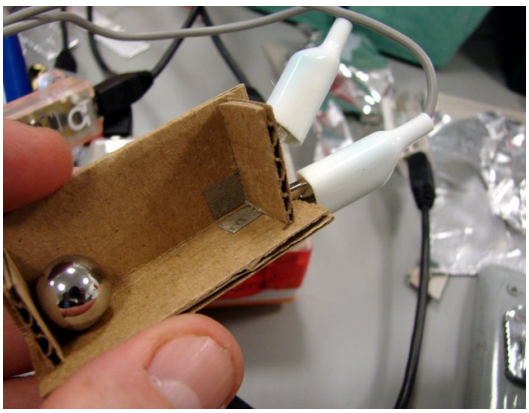
The *PicoBoard* contains a variety of sensors that allows *Scratch* projects to interact with the real world. In addition to a collection of built-in sensors (light, sound, slider, button), the *PicoBoard* also has four connections for homemade "resistive" sensors.

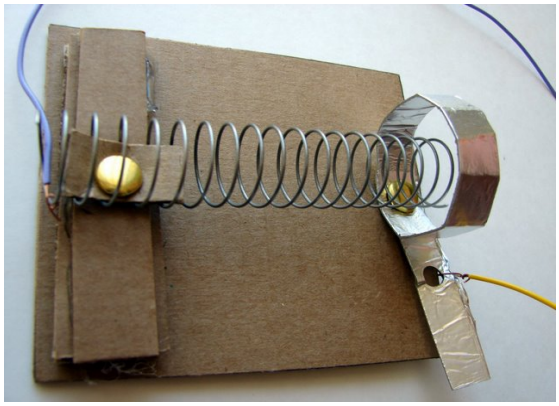
In this challenge I'd like you to build your own "resistive" sensor, connect it to a *PicoBoard*, and use your sensor to interact with a *Scratch* project. For ideas look at the document posted at:

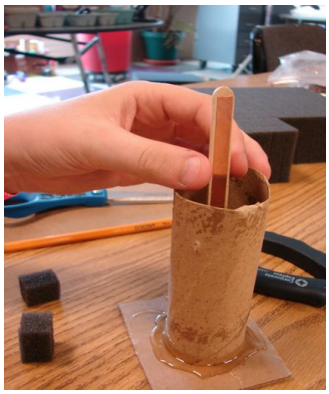
http://www.picocricket.com/pdfs/Getting_Started_With_PicoBoards.pdf

There are also some nice sample projects in the *Scratch Projects* folder. (Look in the *Sensors and Motors* subfolder.) And you should also look at the user-created projects posted at the *Scratch* website:

http://scratch.mit.edu/







Challenge 11: Personal Fabrication

In ancient times (pre-1980 or so) the making of book or a high quality documents required skilled professionals and expensive equipment like printing presses and typesetters. Then came a revolution in which a new generation tools like word processors and laser printers suddenly allowed just about anyone to create high-quality printed material.

Today we're on the threshold of a similar revolution in the fabrication of physical objects. A new generation of "rapid prototyping" tools such as laser cutters and 3D printers is allowing a wide range of users to make their own things. These tools allow you to design an object on a computer and then "print out" the three dimensional object in a variety of materials.

In the *Engineering Studio* we are fortunate to have three such rapid prototyping tools: a laser cutter that uses a high power laser to quickly cut and or engrave plastic and wood parts with very high precision, a 3D printer that three dimensional objects of any shape out of plastic, and a CNC milling machine that can fabricate parts out of a variety of materials, including brass and aluminum.

In this challenge I'd like you to use a simple drawing program like *Adobe Illustrator* of *Google Draw* to design a a simple object for decorating the door the *Engineering Studio*. Then use the laser cutter to print out the part.

Challenge 12: 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 13: 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 200. (Remember, higher numbers correspond to less light reaching the light sensor.) She gets her robot working perfectly with the following *PicoBlocks* code:

```
to react-to-shadow
    waituntil [sensor1 > 200]
    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*.