Last week you met the Scribbler robot and saw how to use pseudocode to control it. This week, you finally get to be in charge. By applying your knowledge of pseudocode, you'll explore the robot's capabilities and use it to investigate physical world phenomena.

If you get stuck at any point, feel free discuss with another student or a TA. However, you are not allowed to copy another student's program or answers.

You should hand in your lab report at the beginning of lecture on Tuesday, February 28. Include the following:

- **Printouts of the 9 numbered programs described below**
- **The 3 pictures that you draw with the Scribbler in Part I**
- **Responses to questions printed in bold (number them by Part/Experiment)**

You can print your programs directly from Scribbler Control Panel, or copy and paste them into a program like Microsoft Word. (Click "Copy as Text" from the Edit menu.)

Part I: Motion by Dead-reckoning

You can move the robot along a pre-determined path by specifying the motor speed and duration of each leg of the journey. This kind of navigation is called *dead reckoning*. The robot's basic movement commands all work this way (except that the Scribbler uses motor power to approximate speed.) One drawback of dead reckoning is that any error in the measurement of speed or direction accumulates over time, causing the robot's position to become less and less accurate.

In this section you'll use dead reckoning to make the Scribbler draw some simple figures. You'll also see how to apply basic geometry to plan the robot's course.

Before you begin, calibrate your robot's motors to ensure that it can drive straight. Your TA will provide instructions.

  Experiment 1 — Draw a square

  Instructions required: Forward, Spin Right, Pause, Do for *n* times

  Program the Scribbler to draw a square with sides approximately 4 inches in length. Use trial and error to get the distances and angles right. Use a loop to avoid writing the same instructions four times.

Tips for accurate drawing: Tape the paper to your lab table so it doesn't slip. Always test the robot on the paper, since angles and distances may come out differently on other surfaces. Always insert a 0.2 second pause between motor commands to give the robot time to come to a stop.

Experiment 2 — Draw a star

Instructions required: Forward, Spin Right, Pause, Do for *n* times

Program the Scribbler to draw a five-point "pentagram" star, as show. Again, use a loop to avoid writing the same instructions multiple times.

Each vertex has an interior angle of 36°. Use this fact and the time you ran the motors to make a 90° angle in Experiment 1 to estimate how long to run the motors for the angles of the star.

36°

Start and end here.
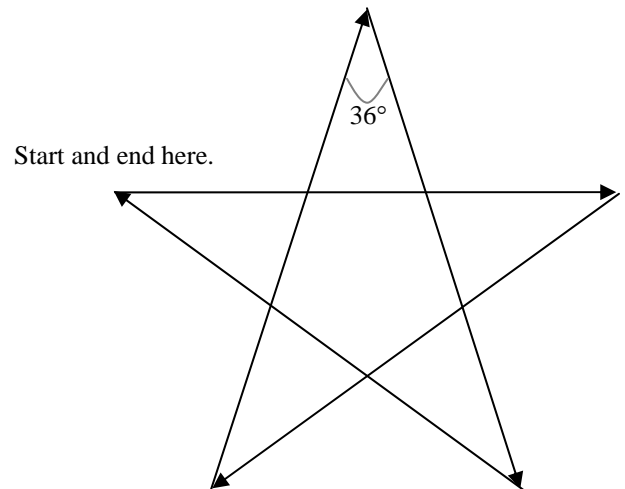
To draw more complicated pictures, you'll need to take advantage of the Scribbler's Advanced Motor Controls. These allow greater control over the speed of each wheel.

Click on the *Motor* command in Scribbler Control Panel, and check the box next to "Enable Advanced Motor Controls." Notice the additional sliders that become available. You can use these to set the speed of each motor from 100% (full power, forward) to -100% (full power, reverse). Try clicking each of the Basic Motor Control commands: Forward, Reverse, Left Turn, Right Turn, Left Spin, Right Spin. Note the Advanced Motor Control speed settings corresponding to each basic command.

One use of Advanced Motor Controls is to make the Scribbler turn along a path with a specific radius. Suppose you wanted to draw a circle with radius *r*, as shown at right. If the distance between the robot's wheel is $2w$, then in one full circle the left wheel will travel a distance of $2\pi(r + w)$, while the right wheel will travel a distance of $2\pi(r - w)$. Say it takes *t* seconds to draw the circle. Then: *left wheel speed* $= 2\pi(r + w) / t$, and *right wheel speed* $= 2\pi(r - w) / t$. This yields the proportion:

$r + w$
Left wheel radius

$r - w$
Right wheel radius

r
Pen radius
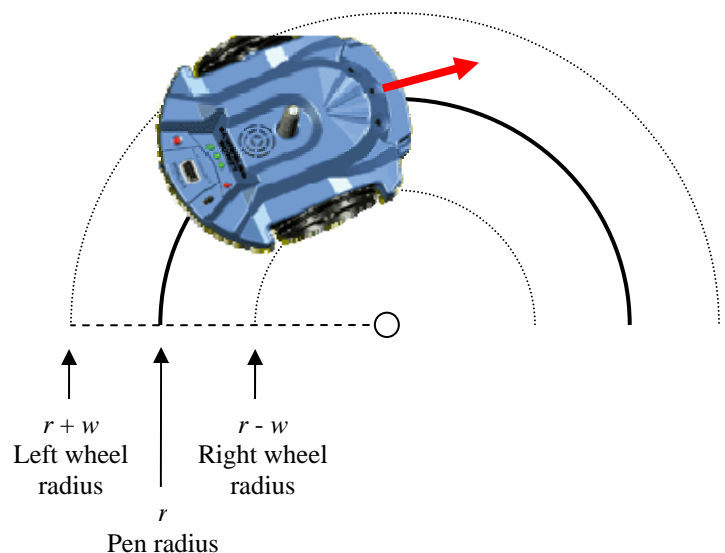
$$\frac{right\ speed}{left\ speed} = \frac{r-w}{r+w}$$

Measure *w* (half the distance between the wheels). Assume the left motor speed is 100% and solve the proportion to find the speed for the other motor that results in a curve with a 6 inch radius. What about a 3 inch radius? A 1 inch radius?

Experiment 3 — Draw a circle

Instructions required: Advanced motor controls

Program the Scribbler to draw a circle with a radius of approximately 5 inches. Use algebra to set the motor powers. Use trial and error to make sure the circle is complete without retracing too much of the line. Your program should use only a single instruction.

Part II: Motion with Sensor Feedback

Another way to direct a robot's motion is to use feedback from its sensors. In contrast to dead reckoning, sensor feedback allows the robot to limit the accumulation of positional error by continually updating its knowledge of its environment. In this section you'll use Scribbler's obstacle sensor and stall sensor to experiment with feedback-driven motion.

Experiment 4 — Avoiding obstacles with the obstacle sensor

Instructions required: Forward, Spin Right, Do forever, If

Program the Scribbler to move around the room while avoiding obstacles. Whenever an obstacle is detected by the obstacle sensor, change the robot's direction before proceeding forward.

Hint: Move the robot forward for a very short time, check the status of the obstacle sensor, and repeat.

As you saw last week, the obstacle sensor will not detect objects at very close range, and it cannot tell exactly how close an obstacle is. These limitations make the obstacle sensor ill-suited for navigating in confined spaces. The *stall sensor* should be used instead.

When the robot is applying power to its motors but they are prevented from turning because the robot it hitting an obstacle, we say the motors have *stalled*. The stall sensor detects this condition.

Up to this point, all the motor instructions you've used specify a duration for the movement. They switch the motor on, wait for some period of time, and then switch the motor off. The stall sensor can't be used with instructions like these, since the motors

aren't considered to be stalled anymore if they have been turned off. To utilize the stall sensor, you'll need to control the motors in a different way.

Say you want to drive the robot forward for 5 seconds. One way to write this is:

      Move Forward for 5s   // Turn both motors on for 5 seconds, then turn them off

An alternative way to say the same thing is to specify an "infinite" duration for the motor command (by setting *duration* to 0 in Scribbler Control Panel). This turns the motors on and leaves them on continuously, just like the LED ON command leaves the LED on. Your pseudocode can do other things, like test sensor values, before switching the motors off by using the Stop command or issuing another move command.

Here's how to express the same instruction using a continuous motor command:

      Move Forward   // Turn both motors on continuously (duration = 0)
      Pause 5s
      Stop   // Turn both motors off

There's one other important detail about the stall sensor. Every time you issue a move command, the stall sensor resets, and it needs to wait a short time to see whether the motors are stalled. This means that the sensor won't give accurate results if you test it too soon the robot starts to move. To avoid this problem, insert a 0.1 second pause after each continuous move command.

Experiment 5 — Avoiding obstacles with the stall sensor

Instructions required: Forward (continuous), Pause, Spin Right, Do Forever, If

Rewrite the pseudocode from Experiment 4 to use the stall sensor instead of the obstacle sensor. You'll need to use continuous-style movement commands. If the motors stall, make the robot change direction and then resume moving forward.

Part III: Lights and Sound

As you saw in Lecture 2, a rapidly blinking light source appears to the eye to be completely steady. This phenomenon is called *flicker fusion*. Psychophysicists define the *flicker fusion threshold* as the frequency at which a person detects flicker in 50% of trials. The threshold varies among individuals and is affected by the brightness of the light, the location on the eye, and other factors. You can use the Scribbler to observe the flicker fusion effect and measure the flicker fusion threshold.

Experiment 6 — Measure your flicker fusion threshold

Instructions required: LED, Pause, Do Forever

Write a Scribbler program that repeatedly blinks the LEDs on and off. Insert equal length pauses after the LEDs turn on and after they turn off. Start with very brief pauses—0.001 seconds.

1. **Run your program and watch the LEDs. Do they appear to flicker? If not, alter the program to double the pause times, and try again. Repeat this process until you can see the LEDs flicker.**

2. **What is the briefest pause time where you can see flicker? What frequency does this correspond to?** (*Frequency* $= 1 / time$. Here, *time* is the length of one complete on-off cycle.)

3. **Movies are displayed at 24 frames per second and TV at 29.97 frames per second, though these are not the rates at which the images flicker. Use Google or Wikipedia to research how these imaging technologies increase the flicker rate to exceed the flicker fusion threshold. Write a few sentences about what you find.**

Experiment 7 — A Scribbler siren

This is a warmup for next week's lab, when you'll study how computers process sound and music.

Instructions required: Do forever, Sound

Program the Scribbler to sound a siren by repeating two different tones. Experiment with different frequencies and durations to produce a realistic effect. Try frequencies that are close together and ones that are farther apart. Characteristics of the robot's speaker and the resonance of its plastic case cause some frequencies to sound louder than others. Choose loud ones for your siren.

Part IV: Non-determinism and Randomness

The robot's sensors detect the condition of the physical world, but there is another condition that can be used in branches and loops: the outcome of a random "coin toss." Of course, the robot doesn't toss an actual coin; rather, it uses a mathematical function with random-looking outputs to simulate a coin toss. A future lecture will revisit randomness in computation. Here we do a few experiments.

If you toss a fair coin, it will come up heads or tails with about equal likelihood. Is the robot's "coin" fair?

Experiment 8 — Testing the coin toss

Instructions required: If, Forward, Back

Program the Scribbler to toss a coin. If the coin toss comes up heads, move the robot forward a few inches; otherwise, move it back a few inches.

Run this program 10 times. Count how many times the robot moved in each direction.

Imagine a staggering drunk on his way home from the bar. He is so inebriated that, with each step, he moves forward with probability p and backwards with probability $1 - p$. This kind of motion is a simple example of a *random walk*. Random walks are used as a model in an amazing variety of contexts, including physics (Brownian motion), economics and finance (share prices), and biology (bacterial motion). You can command the Scribbler to follow a very simple random walk and investigate some of the properties of this motion.

Experiment 9 — Random walk

Instructions required: Do for n times, If, Forward, Back, Pause

Clearly, 10 samples is too little information to judge whether the coin toss is fair. Test 50 times instead. Rather than running the program 50 times by hand, create a new program that adds a loop around the instructions in Experiment 9.

1. **If the Scribbler's coin toss is fair, we expect that the *average* distance from the robot's starting position to its final position will be zero. However, in any individual trial, the robot is unlikely to end up exactly where it started. If the distance traveled after one coin toss is d, how far from the starting position will the robot finish most of the time?**

2. **Place the robot on the floor and mark its starting point with a piece of tape. Run the program and measure the actual distance.**

3. **Try this test three more times. How accurate was your prediction?**

4. **For each trial, use the distance you recorded to determine about how many flips came up heads and how many came up tails.**

5. **Does the robot's coin toss appear to be fair? Do you have enough information to say with confidence? Briefly respond.**

<u>**Extra Credit**</u>**: (Requires some familiarity with probabilistic thinking.)**
The coin toss is a way to make a uniformly random selection between two choices. But what if we want to choose uniformly among three choices?

Experiment 10 — Choosing uniformly among three choices

Instructions required: Do forever, If, LED, Pause

Write a program that, every second, randomly picks one of the Scribbler's three LEDs and turns it on for 1s. Make sure each LED has probability exactly 1/3 of being chosen. (Hint: As a first step, you could come up with a method so that the probabilities are very very close to 1/3, rather than exactly 1/3.)