CMPE 8 Lab 6: The Stall Sensor and Subroutines

Lab Objectives

By the end of this lab you should be able to:

1. **Identify** how the robot’s stall sensor works.
2. **Use** subroutines to organize your code and make it more efficient.
3. **Create** a program that uses subroutines and the stall sensor to enable the robot to avoid objects and unstick itself.

The Stall Sensor

The Scribbler 2 is equipped with an internal stall sensor that can be used to tell the robot when it is stuck. The stall sensor works by monitoring the movement of the tail wheel, which is shown in the picture below.

If we examine the tail wheel of the robot, we will see that the wheel has three holes in it. The Scribbler 2 has an infrared (IR) LED on one side of the tail wheel and an IR diode on the other side. The holes in the wheel allow the infrared light from the IR LED to pass through the wheel and be detected by the IR diode. If enough time passes without the IR diode detecting infrared light from the IR LED, then the stall sensor sends a signal that the robot is stuck.

We can get information from the stall sensor by using the command: `s2.stalled`. This command returns a value of 255 if the robot is stuck and a value of 0 otherwise. Just like with the object detection sensor, we can assign the value returned by the `s2.stalled` command to a variable, and use an if/else statement with a condition based on the value of that variable to dictate the behavior of the robot.

Subroutines

In the previous lab, we learned how to make a simple program that controls the behavior of the robot based on information from one of its sensors. As the programs for the robot get larger and more complicated, we need a way to keep them organized. One of the ways we can do this is by using subroutines. We can identify a piece of code that we use multiple times and separate it from the main part of the program by putting it into a subroutine. Just like functions in
MATLAB, when we want to use the code in the subroutine we simply “call” the subroutine using the name that we gave it.

You might not have realized it, but we have been using subroutines since we began programming the Scribbler 2 robots. For instance, the command:

\[ s2.set\_leds(s2\#GREEN,s2\#GREEN,s2\#GREEN,s2\#NO\_CHANGE) \]

is actually a call to the subroutine

\[ set\_leds(left\_color,center\_color,right\_color,power\_color) \]

located inside the s2 object. In fact, all of the commands that start with “s2.” are calls to subroutines located inside the s2 object.

To create our own subroutines in the SPIN language, we first need to create a new PUB section in our program. Then we need to create a name for the subroutine and write the code we want the subroutine to execute each time it is called. Each subroutine we create needs its own PUB section and its own unique name. To demonstrate this process, let’s create a subroutine to get information from the robot’s object detection sensor. Our program from the last lab used the commands

\[
\begin{align*}
\text{leftSide} & := s2.obstacle(s2\#LEFT,20) \\
\text{rightSide} & := s2.obstacle(s2\#RIGHT,20)
\end{align*}
\]

to get values from the object detection sensor that represent whether or not an obstacle has been detected on either the left or right-hand side of the robot’s field of vision. Instead of having to write those two commands every time we want to check the object detection sensor, we can put them into a subroutine. Such a subroutine would look like this:

```spin
PUB ReadObstacleSensor
    leftSide := s2.obstacle(s2#LEFT,20)
    rightSide := s2.obstacle(s2#RIGHT,20)
```

We place this subroutine below the code of the main program. Then, whenever we want the program to check the object detection sensor, we just use the command `ReadObstacleSensor` in the main program and the variables `leftSide` and `rightSide` will be assigned values of 255 or 0 depending on whether or not an obstacle was detected. Those two variables can then be used in conditional statements in the main program just like before.

**Exercises**

1. Start by commanding your robot to continuously drive forward at full speed (i.e., 255). If the robot detects that it is stuck, turn on all three LEDs, have it back up at full speed for half of a second, turn 90 degrees to the left and then continue to drive forward.

2. Combine your code from exercise 1 with your program from last week (Lab 5). Your robot should now be able to detect and avoid obstacles and unstick itself when it stalls.

3. Using your combined program from exercise 2, create a subroutine called `LEDTurn(left,center,right)`. This subroutine should turn on the correct LEDs and
command the robot to turn the correct direction based on the values of the variables left, center and right. The three variables should take the values of either 1 or 0. You should use this subroutine when your robot detects an obstacle or when it senses that it is stuck. For example, if the robot detects an object on its left-hand side, you should use the command LEDTurn(1,0,0) to have the subroutine turn on the left LED and command the robot to turn 90 degrees to the right. **Hint:** Because the variables left, center and right are only used in the subroutine, you do not have to declare them in the VAR section.

**Note:** For this lab you don’t need to comment every line of your code. Only include enough comments to sufficiently explain your programs. Also, be sure to turn in the code used for each exercise.