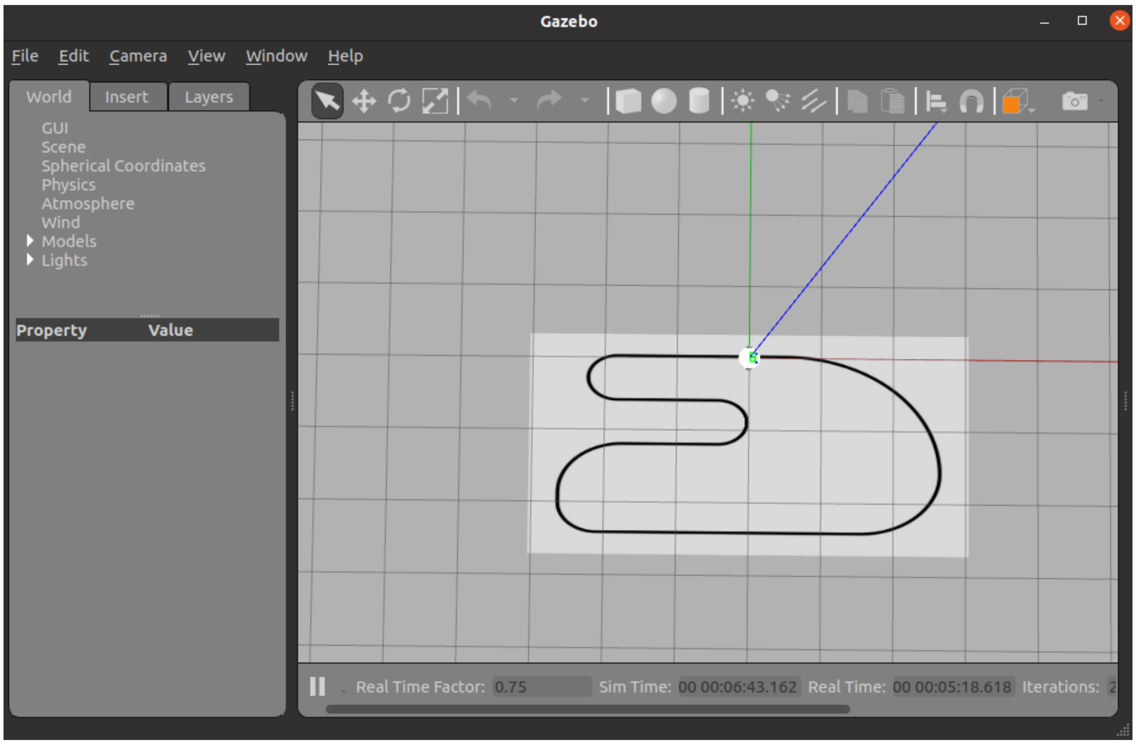
The Line Follower

In this tutorial, we're going to do quite a lot, and it's going to be challenging. We've given you quite a lot of direction here, so you should be able to figure out how to do it. You're going to need a lot of help from the counselors. This is the hardest program you've written so far this camp. It will be challenging, but that's okay! This is how you learn.

We're going to tell you what this program does so you can see where this is all going.

You'll be using the `racetrack.launch` launch file for these exercises, which looks like this:



The first program that you will write will allow you to see the numbers returned by the line follower. It lets you explore the raw, numerical data that the device sees, and how each number is in a position corresponding to the line. What I've said will make sense once you've tried it out.

The second program that you write will allow you to **threshold** the data coming off of the line follower, to determine what is a line and what is not. It will use the LCD on the robot to print asterisks wherever the line is under the line follower, and dashes wherever there isn't.

That will look something like this.

```
----**----------
```

If it looks like that, there's a line most of the way to the left of the robot.

```
**--------------
```

If it looks like that, there's a line all of the way to the right of the robot.

```
--------------**
```

Under the line that looks like that, it will print the number that is the value of the threshold. You will learn about thresholds in this exercise.

Our program says this right now.

```
----**----------
```

```
10
```

Just kidding. We want you to find the threshold for yourself. Ours is much bigger than 10.

Back to the Robot Documentation

The `TexBot` class still has a significant amount of functionality that we haven't yet touched, one of those is the use of the analog-to-digital converter

int readLineSensor(int sensorNum)

On our robot, we use a greyscale camera to develop a line sensor. We take the image from the sensor, and split it up into eight columns - one for each array position - then return the average amount of intensity for each section.

The value `sensorNum` determines which section of the sensor you are getting the intensity for. The range for intensity is [0,255].

Exercise 6.1.1: The Line Follower Test

- You'll be using the `racetrack.launch` launch file for these exercises.

For this exercise we want you to read the values off of the line follower, while running it over the line target. You're going to need to write a short program to do this.

Here's what your program should do.

- Read each of the values from the line follower using the `readLineSensor` function.
- Print each value onto the terminal.
- Print a space following each value from the line follower.
- Print a new line at the end of the 8 values.

When you hook this up, the numbers are going to scream past on the screen really quickly, so we suggest adding a `sleep` after the for loop to make everything much easier to read.

- The easiest way to implement this is with a for loop and the `int readLineSensor(int)` function.
- Count from 0 to 7 using the for loop.
- Inside the loop, read the value in the corresponding line sensor channel.
- `cout` that value and a space.
- After the for loop, use `cout << endl`

Try the Line Follower

- Move the robot around under the line target. You'll want to **rosrun** the `teleop_texbot` executable for this!
- You should notice that the numbers where the black line is present

are different from the numbers where there is no black.
  - How are they different?

Thresholding

If you did the last exercise properly, you should have noticed that ABOVE some value, you could assure that the number being returned by the line follower was the the line being picked up. Beneath some value, what you saw was the white surface.

The difference between the region where the line was and the region where there was no line is called a **threshold**. We are going to use a threshold to determine where the line is.

We're going to have you follow a line on a racetrack! What we'll do to account for this is write a small user interface where you can tune your threshold. While we're at it, we'll also experiment with the line follower and the LCD, and learn a bit about strings in C++.

Exercise 6.1.2

The next few exercises will build up to a small user interface that will allow you to see the line the way that your robot sees it!

In C++, Most Text is Just Special Integers

- Write a quick function, call it `void printAsterisks()`.
- Have `loop` call `printAsterisks`.

```
void printAsterisks(TexBot bot) {

}

int main(int argc, char **argv) {

    ros::init(argc, argv, "robot");

    TexBot bot;

    while(ros::ok()) {
        printAsterisks(bot);
        ros::spinOnce();

    }

    return 0;
}
```

It's going to be tempting to put a `usleep` in here somewhere. Do yourself a favor and resist that urge. Your program will work better and look better if you avoid this.

There's a type that we didn't tell you about way back in "Simple Math and User Input" called `char`.

`char` stands for **character**. Each `char` is 1 byte, meaning that it can represent 256 different numbers. Each `char` is `unsigned`, meaning that it cannot be negative. So, a `char` is a number from [0,255] which is used to represent a character.

One way to represent text is called a C string. This is different from a C++ `string`. A C string is an **array** of type `char`.

Every number from 0-255 is associated with a unique, printable character. This is called **ASCII**.

- ASCII - American Standard Code for Information Interchange
- There's a good article on Wikipedia about learning ASCII if you really want to, but it won't be especially relevant to what we're doing here, so, save that for after camp!

- Remember the `bot.lcd1` command? Use that in your program to print 2 asterisks ("*") on the top line of the LCD on your robot.
- Now try it using a C string.
  - Make a **global** variable to store your C string in. Call it `lineUI`.
  - Make `lineUI` 17 chars long.
  - Make a `for` loop to fill `lineUI`.

Global Variables

You're about to see your first example of a **global variable**. We've mentioned the concept of **scoping** a few times already, which is where one can access a particular variable. A variable that is global can be seen *everywhere* in the file.

In practice, we try not to make too many variables global, because this can lead to confusing behavior in more complicated programs. Sometimes, however, we want every part of our program to have access to a variable. In these cases, we choose to make our variables global.

```
char lineUI[17]; //global variable

void printAsterisks(TexBot bot) {

  for(int i = 0; i < 16; i++) {
    lineUI[i] = '*';
  }
  lineUI[16] = 0;
  bot.lcd1(lineUI);
}

int main() {
  //some parts of main are excluded in this example code
  printAsterisks();
  bot.lcd2(lineUI); //Notice how we can reference lineUI in bo
}
```

- What the heck is `'*'` ?
  - Single quotes surround a **character-literal** in C.
  - Just like a string can be surrounded in double-quotes (which is called a **string-literal**), a single character can be surrounded in single-quotes.
- You probably also noticed that I set `lineUI[16] = 0;`
  - C strings are **null terminated**. This means that you put a 0 at the end of the string to mean that the string is done.

Null termination allows us to have shorter or longer strings stored in the same memory. This means that if we only wanted lineUI to be 8 characters long, we'd just put a zero in `lineUI[8]`.

- Go ahead and run the program from the example above. Don't forget to add any declarations and things that I didn't put into the code snippet!

Modify the For Loop from above

In our UI, what we want is for the top line to show us where the line is. Eventually, there will be asterisks where the line is, and dashes everywhere else.

However, we run into a problem. There are 8 sensors on the line sensor, and 16 characters on the top line. So, we'll want 2 asterisks for every sensor on the line follower. If we have that, and ONLY asterisks turned on for each sensor on the line follower that sees a part of the line, then we will be able to see the line the way that the robot sees it!

- Modify the `for` loop from the example above to **iterate** through the loop 8 times so that only 8 asterisks show up on the LCD.
  - Don't forget to change your null terminator.
  - Your for loop should now count from 0-7. `int i = 0; i < 8`!
- Modify the `for` loop so that it only iterates 8 times, but still prints 16 characters.
  - Do this by making every time it iterates through the loop insert 2 asterisks into `lineUI`.
  - You can do this by multiplying `i * 2` when you fill in `lineUI`
  - That, however, will skip ever other entry in `lineUI`, so, make it also fill in `i * 2 + 1` with an asterisk every time it iterates through the `for` loop.
  - Try it out!

If these steps don't make sense, try just printing out the values of `i*2` and `i*2+1`. You should notice a pattern!

Exercise 6.1.3

Now we're going to write another UI. Copy your code from Exercise 6.1.2 into the new file (ex_6_1_3.cpp) and continue modifying in there.

- Add a new global variable, `double thresh` , and set it to zero.
- In `printAsterisks` print the value of `thresh` on line 2 of the robot's LCD.
- Add some code to the loop block that responds to button presses.
  - When button 1 is pushed, add 1 to `thresh` .
  - When button 2 is pushed, subtract 1 from `thresh` .
- Try it out. The number on the second line should rise when pressing the top button and drop when pressing the lower button.

Exercise 6.1.4

Now we're going to make it so you only see where the line is under the line follower, allowing you to see the line the way that your robot does.

- Make a global array of type `int` with 8 elements. Name it `lineSensor` .
- Modify the loop so that it reads the ADC using `bot.readLineSensor` .
  - Use a `for` loop so that it fills the `adc` variable.
- Modify the `for` loop in `printAsterisks` .
  - Every time `lineSensor[i] > thresh` , it should put 2 asterisks.
  - Every time `lineSensor[i]` is not `> thresh` , it should put 2 dashes.
- Run your program. Put the line target under the line follower. Push the up button until there are only asterisks under the line follower sensors that are over the line, and dashes everywhere else.
- Move the target around and see how the robot can now "see" the line.
  - Ideally, you only see the line in one place on the LCD, and it is only 2 asterisks wide.