Build an Arduino Self-Driving Car

Lesson 8: Designing and Implementing an Algorithm

Estimated time: 2 hours

Introduction

In this lesson your students will design and code an algorithm to make their self-driving car autonomously navigate an artificial track or roadway. This will require using all the skills they have learned in previous lessons to control the car's motors, ultrasonic sensor, and infrared sensors.

Learning Objectives

- Understand what an algorithm is and how to represent one with pseudocode and a flowchart
- Define relevant parameters and desired behaviors for their car on the track
- Design an algorithm (using pseudocode and a flowchart) to implement these behaviors
- Implement and test the algorithm using real Arduino code

Preparation

- You will need a track or road set up for students to set their cars. See the "Track Creation Guide" document in the Welcome section for tips.
- Try this activity yourself before trying it with students so you are aware of any challenges they might encounter.

Materials

This lesson requires all the materials from previous lessons. No additional materials are required.

Procedure

Walk your students through the following steps or have them watch the video and follow along.

Step	Timestamp	Description
1	0:00	Demonstrate a working car for students.
2	0:27	Review example scenarios that could occur when driving on the track.
3	1:18	Discuss the differences between real autonomous cars and the Arduino cars.
4	2:05	As a class, agree on pre-defined parameters or "rules of the road" for your track (e.g., What do obstacles and lines represent? What should happen if two cars are driving at once and encounter each other?).
5	2:39	Using the self-driving car as an example, explain what an algorithm is and demonstrate how to represent one using pseudocode and a flowchart (see Appendix 8A).
6	5:53	Demonstrate how to represent functions with pseudocode and a flowchart.
7	6:41	Have students design their own algorithms based on the parameters and behaviors they defined earlier. You can decide whether you want them to represent their algorithm using pseudocode, flowcharts, or both.
8	7:04	Convert the algorithms to actual Arduino code. Students should have all the sections of code they need from previous lessons (e.g., code to control the motors, ultrasonic sensor, and infrared sensors). They will need to pay close attention to their use of **`if/else`** statements to implement their algorithms.

9	8:22	Upload and test the code. Students may need to iteratively change their code and re-test. For example, they may need to change parameters in the code (like motor speeds or sensor thresholds) or change their algorithm and the structure of the code itself. Emphasize that it's OK if their car does not work perfectly on the first try – this is why engineers test things!

Assessment

As you walk around the class:

- Confirm that students can represent their algorithms with pseudocode and/or flowcharts.
- Observe how students convert their algorithm into actual Arduino code.
- Observe how the cars perform when students test them. It is *not* critical that the cars work perfectly on the first try – the important part is that students are able to identify problems and work to fix them.

Appendix 8A: Pseudocode and Flowcharts

The following pseudocode represents an algorithm for a car that will automatically stay in a lane but will temporarily ignore the infrared sensor readings to drive around an obstacle if it detects one in the road.

```
Start program
Infinite loop:
        Is there an obstacle in front of the car?
                If yes, drive around it while ignoring IR sensor readings
        Does the right IR sensor see a line?
                If yes, turn left
        Does the left IR sensor see a line?
                If yes, turn right
        Otherwise, drive forward
```

You can also represent the algorithm using a flowchart. Flowcharts use different symbols (an oval for the program's start, diamonds for "decisions" or checking conditions, and rectangles for actions) and arrows to represent the flow of a program.