IoT Platform:

Tutorial: Automated Path Discovery and Guidance

Table of Contents

Introduction 3

Things Needed 3

Assembling the Robot 4

Constructing the Maze 12

Robot Software Overview 16

References 17

| Revision history | | |
| --- | --- | --- |
| **Version** | **Date** | **Comment** |
| 1.0 | mm/dd/yy | Initial release |
|  |  |  |
|  |  |  |

# Introduction

In this tutorial, you will:

1. Learn to assemble the path-finding robot.
2. Construct a maze for the path-finding robot.
3. Understand the principles behind traversing paths and nodes.

# Things Needed

* Our previous tutorials:
  + Distance Sensor Arrays
  + IR Beacons and Receiver Arrays
  + Parallelism and Inter-Device Communication
* Hardware from our previous tutorials:
  + Intel Edison + Base Block + GPIO Block soldered to 5 ultrasonic distance sensors & UART connection + 9DOF Block + Battery Block
  + Intel Edison + Base Block + GPIO Block soldered to 5 ultrasonic distance sensors & UART connection + Battery Block
* Hardware from EE180D Lego Robot tutorial:
  + Lego Robot chassis with two motors soldered to Sparkfun Dual H Bridge BLock
* A PC or Mac,
* Poster paper or cardboard to construct a maze and IR beacon towers,
* Patience and an architectural spirit.

# Assembling the Robot

**Overview**

After learning how to construct the different individual components of the robot in the previous tutorials, it is now possible to put everything together and create the pathfinding robot system and an environment to test the system in. A picture of our robot can be seen below:

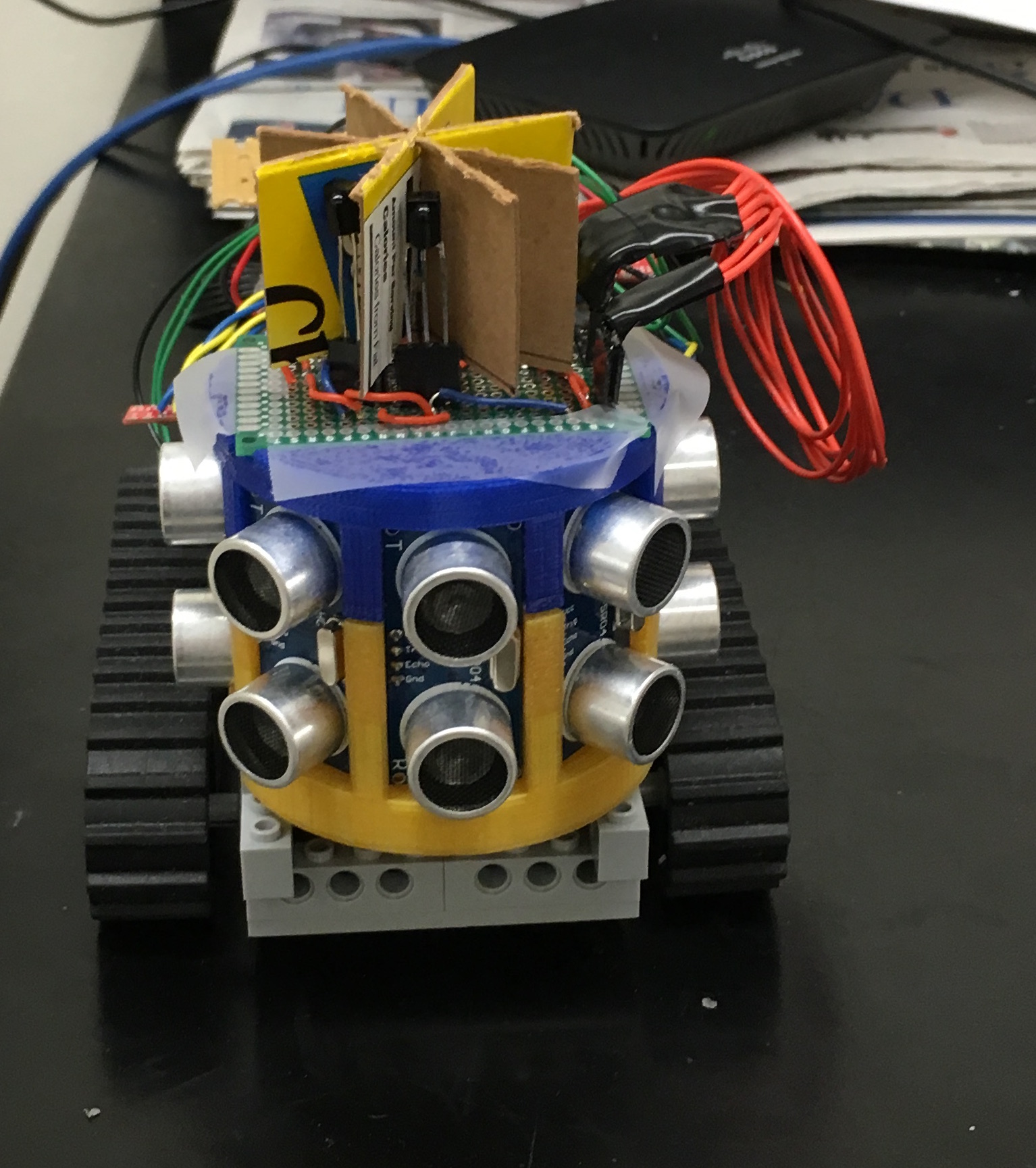
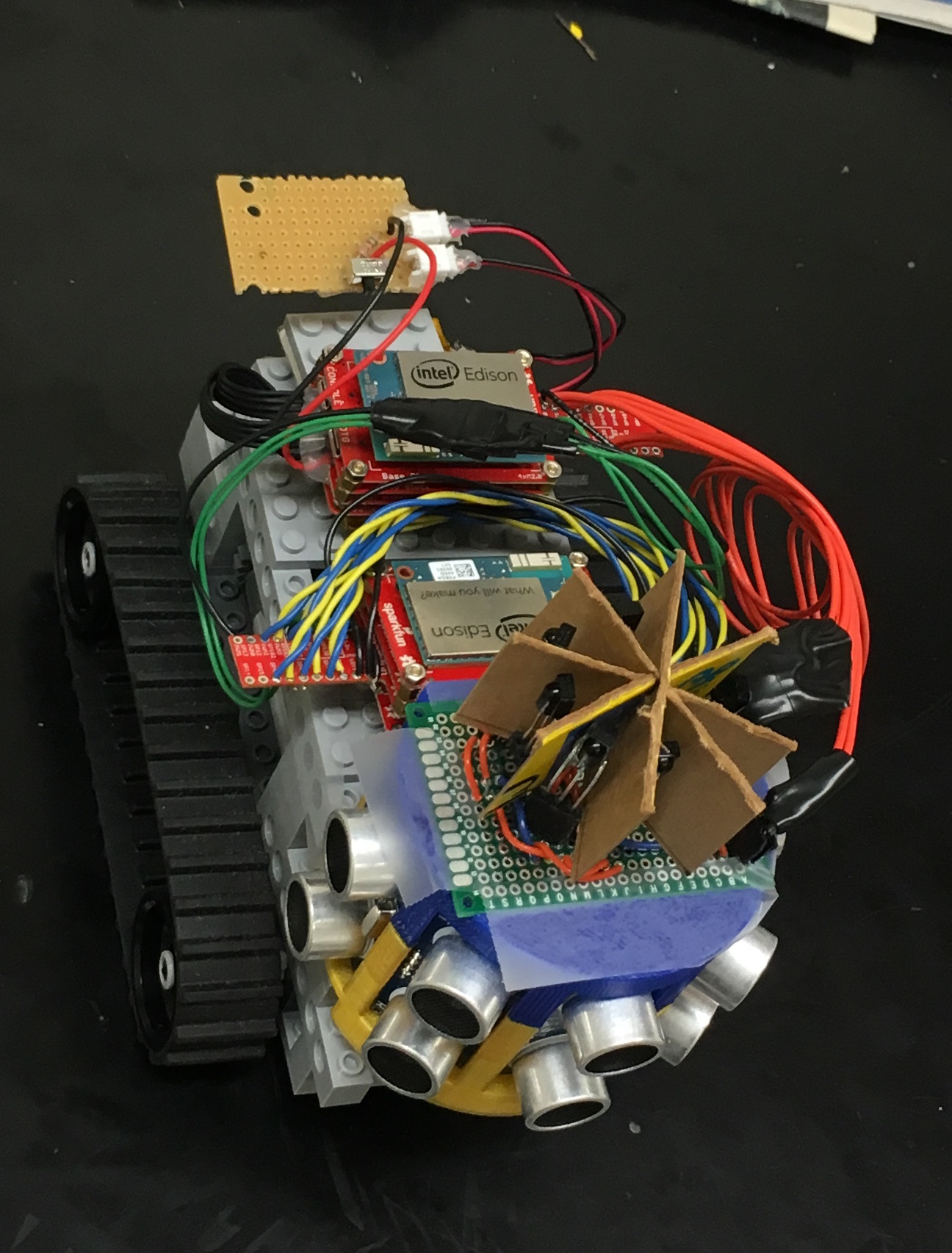
 

Figure 1: Fully Assembled Robot

The chassis of the robot is based on the EE180D Lego Robot. Follow the instructions for constructing the Lego robot and test that it is functional. Follow the distance sensor tutorial to create an array of distance sensors on a 3D printed mount. Next follow the IR receiver tutorial to create the IR receiver array for the robot on a 4 cm x 6 cm perfboard.

For the robot, two Edisons will be required, and for the beacons, one Edison is required for each beacon. A diagram of the entire system can be found below:

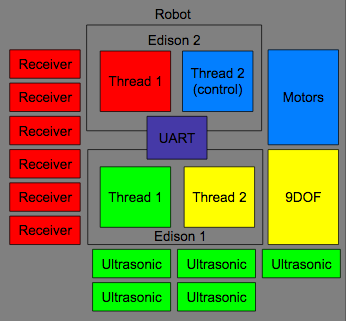


Figure 2: Diagram of Systems on the Robot

Edison 1 collects data from the ultrasonic and 9 degree of freedom (9DOF) sensors and sends it over UART (see *Parallelism and Inter Device Communication* Tutorial) to the Edison 2. Edison collects 2 data from the IR receivers and receives data from UART, and uses this data to control the robot.

The following instructions describe how to combine all the systems together into the robot.

**Robot Chassis and Motor**

To assemble the robot, first acquire the necessary parts and follow all the instructions given in the *Robotics* tutorial provided by the EE180D class ***except*** for the instruction about supplying power to the robot motors on page 21 (**Important!**).

In the Robotics tutorial, the tutorial lists two methods of supplying power to the robot:

Figure 3: Two methods of supplying power to the h-bridge block, from the Robotics Tutorial.

1. There are two options to supply power to the SparkFun® Dual H-bridge block:
   * 1)  Connect an external power source to VIN and GND pins.
   * 2)  Short VSYS–>VIN to supply power from VSYS.
2. We choose the second option for our robot.

The tutorial chooses the second option for the robot, but instead we will choose the first option, so do NOT complete the step to short VIN to VSYS.

In other words, do **NOT** complete this step:

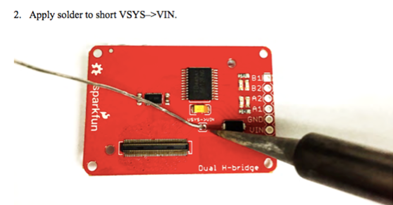


Figure 4: Instruction on page 21 of the Robotics tutorial that should not be followed

After completing the other steps of the tutorial, you should have an H-bridge block with the motors correctly soldered to it as follows:



Figure 5: Picture from Robotics Tutorial of the H-bridge block with Motors correct connected to it

For our robot to actually be able to move and traverse our maze with various obstacles like bumpy ramps, we will need to supply more power to the robot motors. Using a perfboard, JST connectors, headers and a switch, connect two rechargeable lithium ion 3.7V batteries (Battery 1 and Battery 2 in the circuit depicted in Figure 7) in series for a total voltage supply of 7.4V to the H-bridge and motors. See the diagrams on the following page for additional clarification on how to complete this step:

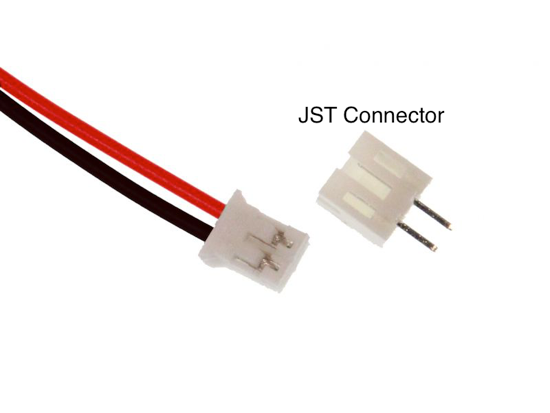


Figure 6: JST connector

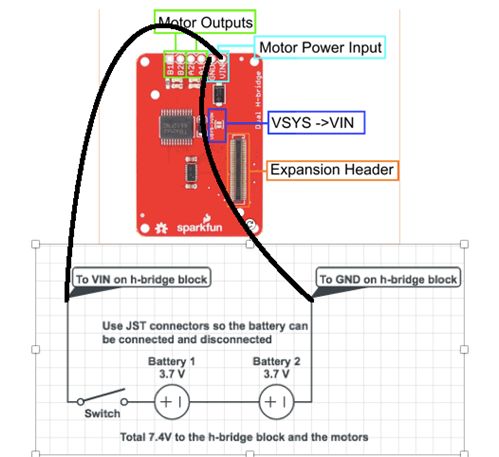


Figure 7: Circuit to connect two Batteries in series

Now, the H-bridge block is completely connected.

**Placing the Edisons and Sensors on the Robot**

Now, it is time to put the Edisons, sensors, and the batteries on the robot. First secure the two batteries and the battery connection circuit any way you wish with Legos and double sided tape onto the robot.

Next, assemble Edison 1’s stack. Start with a battery block that has a Lego piece taped to it using double sided tape so it can be connected to the robot. On top of the battery block place a GPIO block that has been connected to the ultrasonic sensors instructed previous tutorials. On top of the GPIO block place a 9DOF block and finally a base block with the Edison on the very top.

To assemble Edison 2’s stack, again start with a battery block with a Lego piece. Next on top of the battery block, place the H-bridge block that is already correctly connected to the motors and the batteries as instructed above. On top of the H-bridge block, place the GPIO block that is connected to the IR Receiver array instructed in previous tutorials. On the very top, place a base block with the Edison.

Connect the two Edison’s using the UART communication protocol described in previous tutorials. And place the two Edison stacks onto the robot.

Next, print the bottom mount from the *Ultrasonic Distance Sensor Array* tutorial. Additionally, print the top mount which attaches the IR receiver array on top of the distance sensors, found at <https://github.com/zhgary/UCLA_EE180D_Robot/blob/master/CAD/receiver%20mount.stl>. Attach a Lego to the bottom of the piece using hot glue so it can be secured on the robot. Attach the ultrasonic sensors and receiver array on the front of the robot.

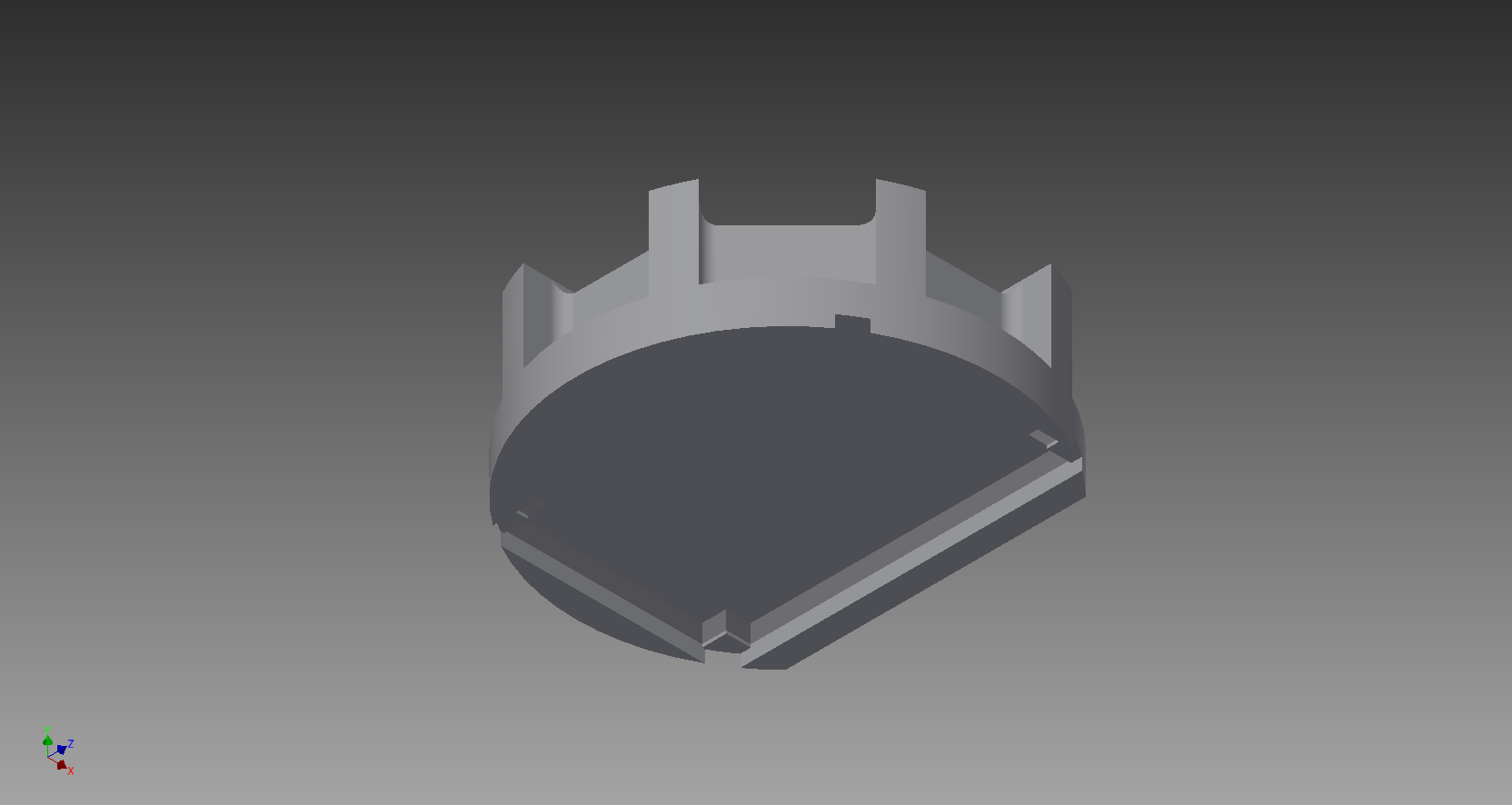


Figure 8: Receiver to Distance Sensor Mount

Pictured on the following page is how we placed all the components onto our robot:

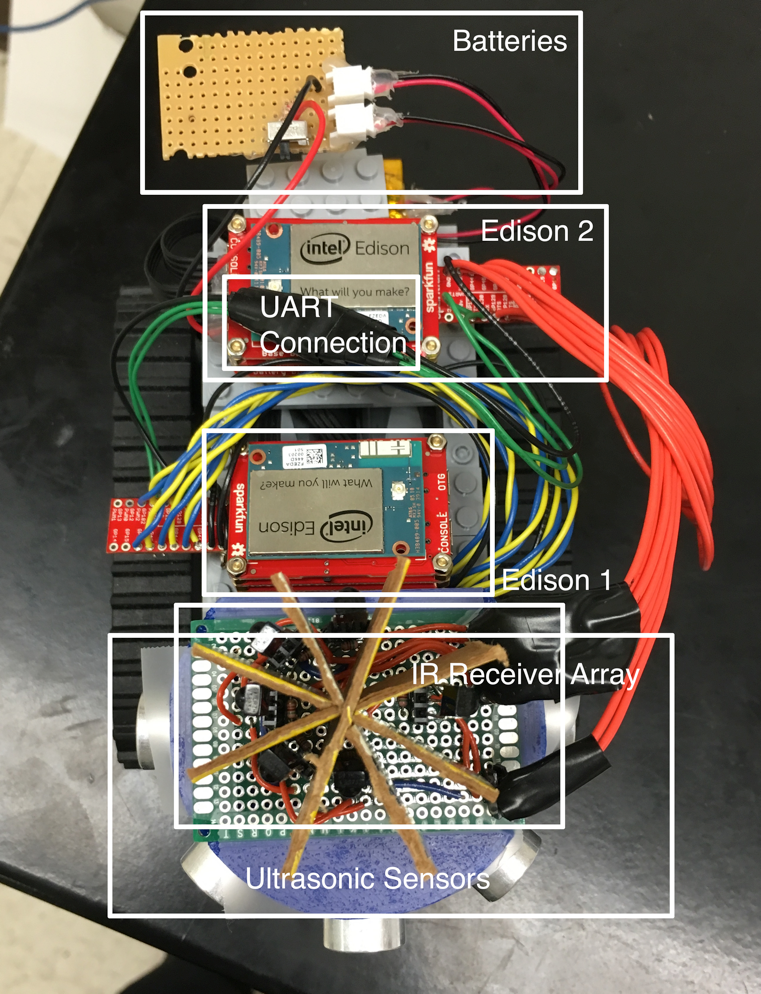


Figure 9: Robot and positions of all elements

Finally, we can upload code to the Edisons.

1. For Edison 2, obtain the code from <https://github.com/zhgary/UCLA_EE180D_Robot/tree/master/Pathfinding%20Robot%20Control>. Compile the code using **$ gcc -lmraa -pthread everything7.c lego\_robot.c -I . -o everything7**. This program is the main control program for the robot.
2. Edison 1 will contain the same code as used in the *Parallelism and Inter Device Communication* Tutorial. However, it would be convenient for the Edison to automatically run the program on startup. Move the program into the home directory of the root account. Type the following into a file named “startup.sh”:  
     
   **#!/bin/bash  
   ~/pthread2**  
     
   Then execute the following commands:  
     
   **$ cp startup.sh /etc/init.d/  
   $ chmod +x /etc/init.d/startup.sh  
   $ update-rc.d startup.sh defaults**
3. Test robot movement by running **$ ./everything7** on Edison 2. The program should create two threads, then wait for input. Type a single character and press Enter to send a command to the robot.
   1. ‘x’ stops the robot
   2. ‘f’ makes the robot move forward for one second
   3. ‘b’ moves the robot backwards
   4. ‘l’ makes the robot swivel left for one second
   5. ‘r’ is for swivel right
4. Check that the commands move the robot in the correct direction. If not, that means that the motors were connected to the H-bridge in a different way than what was expected by the program. This requires either resoldering the motors to the H-bridge or editing “lego\_robot.c” to match the wire configuration on your robot.
5. If the robot is brought near an IR beacon, Edison 2’s program should display data from the beacon.
6. Once you have completed the maze in the next section, you may set the robot in the maze and send these commands:
   1. ‘p’ puts the robot into path navigation mode, and the robot will start to move forward in the path it is placed on.
   2. ‘c’ puts the robot into beacon mode where the robot circles clockwise around the beacon at an intersection, and takes the nth exit (n starts at 0).

Notes:

* While navigating the maze, the robot will move with a jerky motion. This is feature of the robot to prevent overheating if a motor stalls and to allow the robot to climb more difficult obstacles. It was found that slowly pulsing the Lego motors to prevent overheating also provided more instantaneous force. Adjust this behavior by modifying the variables “motor\_reset\_period\_u” and “motor\_reset\_duty\_u” in the code.
* While it was originally planned for the robot to automatically map the maze, that feature was only partially implemented. You must program the robot to take the correct exits at intersections by editing the source code. “everything7.c” has an array named “maze” which is a list of numbers denoting how many exits to skip at successive intersections. Alter this array to program the robot to traverse the desired paths, and alter the variable “totalTurns” to reflect the length of the array.

# Constructing the Maze

**Concept**

To demonstrate the path-finding robot’s capabilities, construct a maze with multiple paths between multiple nodes. IR beacons will be placed at large intersections to help the robot determine its location and the number of possible branching paths at that intersection. With some paths clearly less favorable than others, the robot will ultimately determine which path is the most time-efficient between any two nodes. A general idea of such a maze is seen in the following figure, with IR beacons, bumpy paths, smooth ramps, and bumpy ramps labeled accordingly.

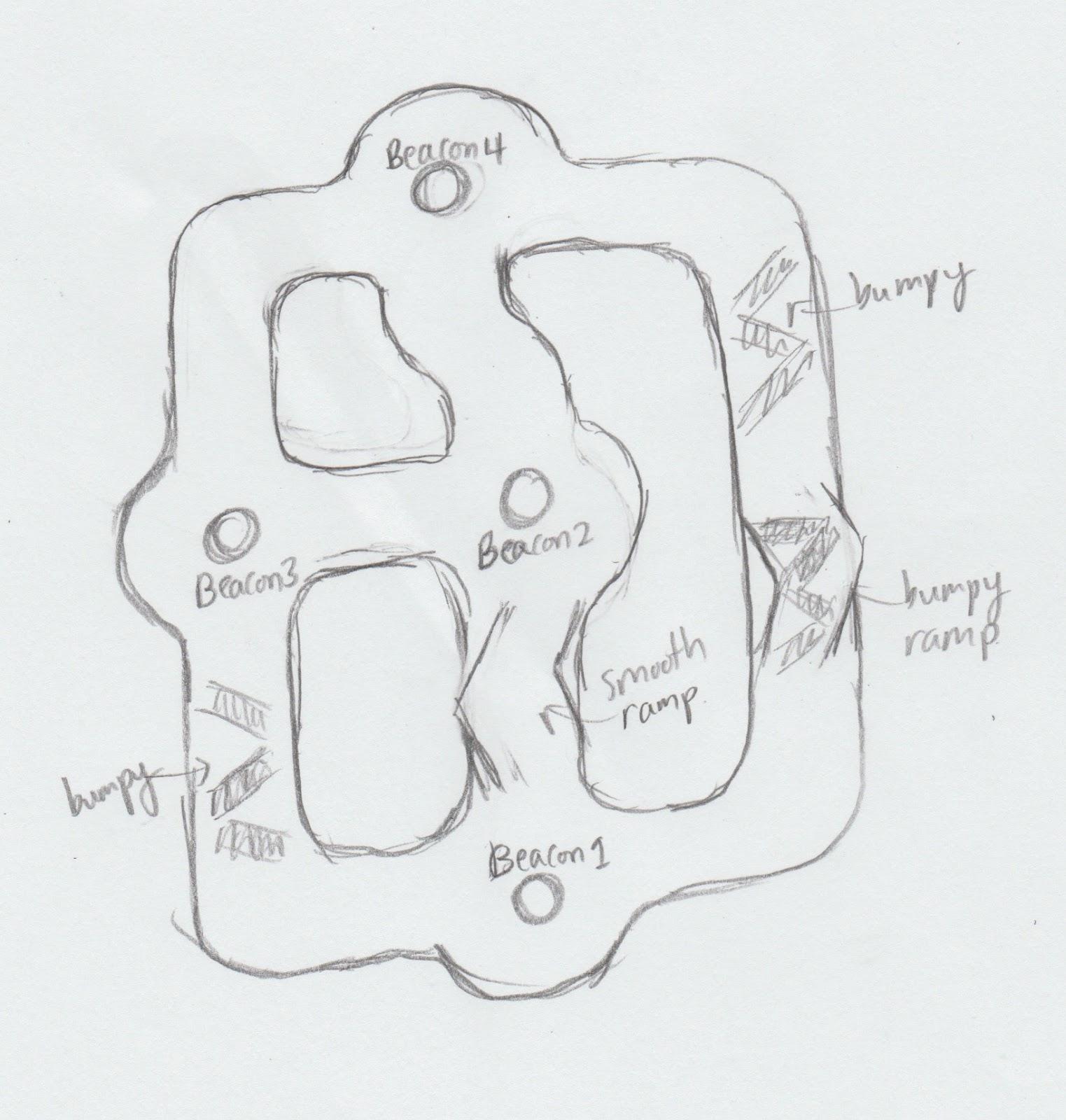


Figure 10: One possible maze plan for testing the robot

**Specifications**

The maze has one limitation in its topology so the robot may use an algorithm to map the topology of the maze by itself. **Any two beacons can only have one path connecting them.**

While maze dimensions do not need to be exact, the robot’s programming works best if the dimensions are within a certain range.

* Paths should have a width of around 16cm, or slightly less than twice the width of the robot.
* Beacon towers should be slightly less than the width of the robot or around 8-10 cm.
* Intersections should have a distance of around 16cm between the beacon tower and the walls.
* The walls of the path and intersections should not have any sharp corners; they should not curve tighter than a radius of 5cm.
* The walls of the tower should be high enough so that they will block IR LEDs and receivers if the robot does not have line of sight to an intersection.

**Intersections**

To create the intersections, first place an entire, square piece of poster paper flat on a surface. Trace over the pattern seen in the following figure:

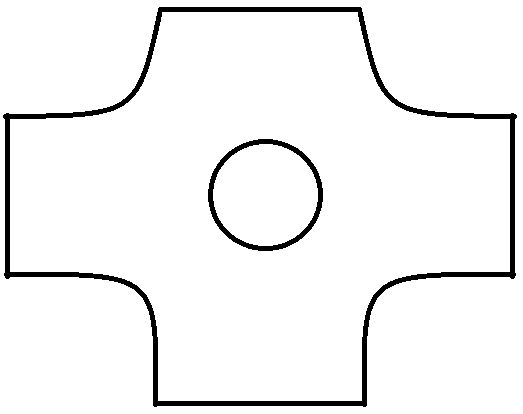


Figure 11: Intersection Pattern

Next, fold paper to make walls, and place them on the base in the pattern seen above. The IR beacon will be placed in the circle in the center of the figure, and each of the four sides will be a possible branching path. Since the IR beacon is composed of IR LEDs in a circle configuration, the data will be transmitted in virtually all directions from the center of this intersection, ensuring that the robot will be able to accurately determine its location. Make sure there is enough space between the curved edges and the IR beacon for the robot to pass through, since it will have to circle the beacon. Please refer to Figure 13 to see a fully constructed example.

**Paths**

We used poster paper to create the maze and the walls were 16cm high with paths of 18cm wide. Walls of any height are okay as long as the walls are high enough for the ultrasonic sensor to detect. To connect multiple path components, secure the connection of the sides of the paper with tape or paper clips.

**Obstacles**

In order for there to be an optimal path, there must be paths that are clearly more time-consuming to traverse. To do this, we add obstacles such as bumps, ramps, or bumpy ramps to increase the time it takes for the robot to traverse that path. To create bumps, we suggest tightly rolling two pages of newspaper, then taping them down onto the surface, and covering them with a smooth piece of paper to help the robot move over them. For ramps, we suggest folding paper to create the ramp shape, then stuffing newspaper underneath the ramp to support the weight of the robot as it traverses over it.

**IR Beacon Towers**

To make an IR Beacon tower you will be assembling the same electronic hardware as specified in the *IR Receiver Array Tutorial*, and using the *beacon\_sender* program.

Since the IR receiver array is mounted on top of the robot, the IR beacon LED array must be elevated in order for the receiver to receive the signals. To do this, we suggest constructing a cylindrical tower and placing the IR beacon on top. We rolled a piece of cardboard in a cylinder, and then added a lid for the IR beacon to sit on. The towers must of a height so the IR LEDs are at the same height as the IR Receiver array. We used towers of 11cm high; the actual IR LEDs will be slightly higher.

Configure the beacons to send the following 8 bit data:

* The 4 most significant bits contain the unique ID number of the beacon
* The 4 least significant bits contain the number of “gates”, or paths connecting to that intersection

Pictured below is one of our completed IR beacons: The batteries, Edison and GPIO block are inside the beacon tower underneath the lid.

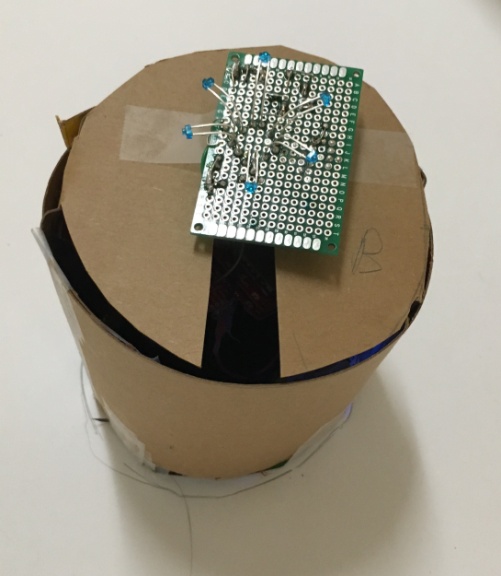


Figure 12: Fully constructed IR beacon

**Photograph**

Below is an image of the fully constructed maze with beacons placed in their proper locations.



Figure 13: Fully constructed maze from the diagram in figure 3

# Robot Software Overview

The Lego Robot navigates through the maze using distance sensor and IR receiver data. While it is navigating, it can collect 9DOF and time data which may characterize the path.

While Edison 1 contains the same code as used in a previous tutorial, Edison 2 adds navigation functionality to the robot.

Edison 2 mainly runs two threads. The IR receiver thread contains code adapted from the tutorial for the IR receiver arrays, while the other thread incorporates code from the UART tutorial and adds code for motor control.

The robot has many different states, but mainly two of them are used for autonomous movement: path mode and beacon mode. They are used when the robot is traversing paths between nodes, and when the robot is circling around a beacon at a node respectively.

**Path Mode**

In this mode, the robot navigates only using distance sensors. A threshold check (distance less than 20cm) is performed on the most recent set of values from each distance sensor, resulting in five Boolean values. Then the program uses what is essentially a lookup table (in the form of a switch statement) for the 5 bit value to determine how to turn the motors. Programming the control algorithm this way results in less effective pathfinding but avoids the time spent developing complex algorithms.

In path mode, the robot can switch states to beacon mode if an IR receiver detects a beacon signal and the distance sensor facing in the same direction as the IR receiver detects a nearby object: the beacon tower.

**Beacon Mode**

In this mode, the robot relies primarily on directional sensing from the IR receivers, and takes advantage of the distance sensors if it is able to. The robot checks the directions of the most recent three IR signals received and sees if the distance sensor for the same direction detected something. The robot then sets the motors depending on the presumed direction of the beacon. If no distance sensor detects anything in the direction of the most recent IR signals, then the robot follows a different lookup table.

As the robot circles counterclockwise around the beacon, the left distance sensor is used for check for exits, with a debouncing algorithm. Upon arriving at the correct exit the robot enters a third state in which it turns away from the beacon towards the path until its back faces the beacon. Then it goes to path mode.

**Mapping (unimplemented)**

Mapping the maze involves knowing which paths at an intersection lead to which other intersections, and how those paths are ordered when the robot circles counterclockwise.

One example of mapping logic is if a robot arrives at intersection C from intersection B, circles clockwise around and takes the first exit, and eventually arrives at intersection D, then it must be that for intersection C, the exit leading to intersection B is directly counterclockwise to the exit for D. Furthermore, if the robot arrives back at C from E, and C only has three exits, or “gates”, then the order of exits at intersection C must be B🡪D🡪E clockwise. To implement mapping we propose using linked lists (each node representing an exit or “gate”) to maintain partially complete mappings of intersections, and once the intersections are completely mapped, the nodes will have been connected into a circularly linked list.

Some possible structs for representing linked lists and paths are contained in the “paths.c” and “paths.h” files.

# References

1. <https://www.linux.com/learn/how-use-linux-command-line-basics-cli>
2. [https://learn.sparkfun.com/tutorials/sparkfun-blocks-for-intel-Edison---dual-H-bridge?\_ga=1.153286264.1711566730.1443483194](https://learn.sparkfun.com/tutorials/sparkfun-blocks-for-intel-edison---dual-h-bridge?_ga=1.153286264.1711566730.1443483194)
3. EE180D Robotics Tutorial