Gauntlet Challenge

QEA Spring 2020

Lilo Heinrich

Introduction

The gauntlet is a simulated environment with obstacles (boxes), walls, and the Barrel of Benevolence (BoB). The challenge is to autonomously guide a Neato robot from the origin of the map through the obstacles and to gently tap the BoB. There are three levels of missions:

- Level 1: the locations and dimensions of the obstacles and BoB are all given
- Level 2: the BoB location and radius is given but obstacles must be found using LIDAR
- Level 3: the radius of the BoB is given, obstacles and BoB must be found using LIDAR

Where levels 2 & 3 may be completed either through dynamic updating or just an initial scan from the LIDAR. I chose to take on level 3.

GitHub repository link: https://github.com/liloheinrich/Gauntlet

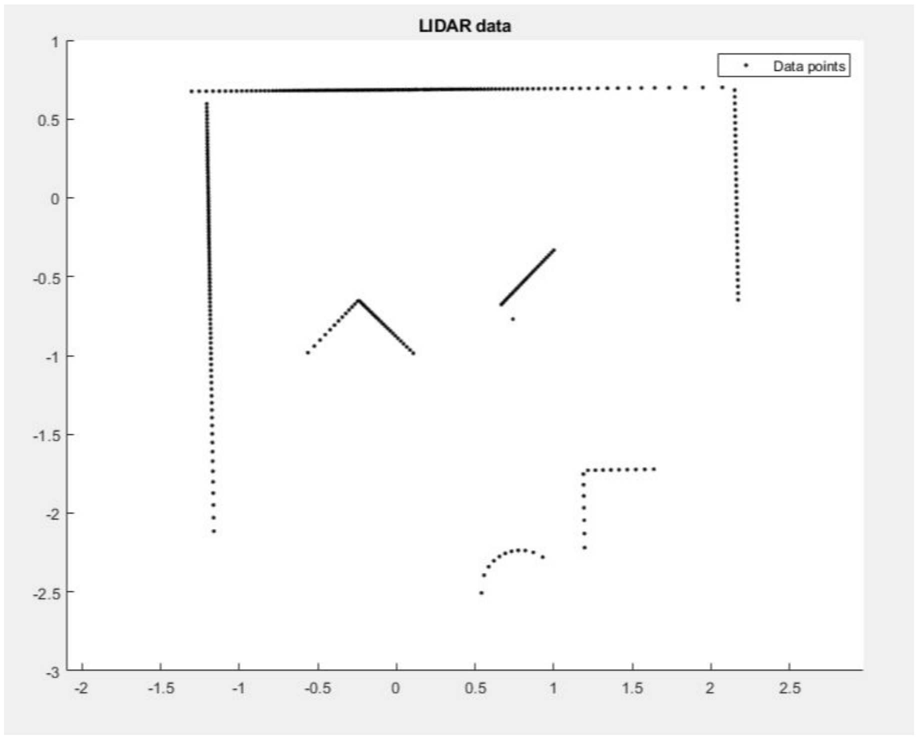
Approach

My general approach was to use only an initial scan. I figured out that running the circle detection and recalculating the potential field was going to be too computationally expensive causing the program to seriously slow down, as well as too complex for me to implement right now.

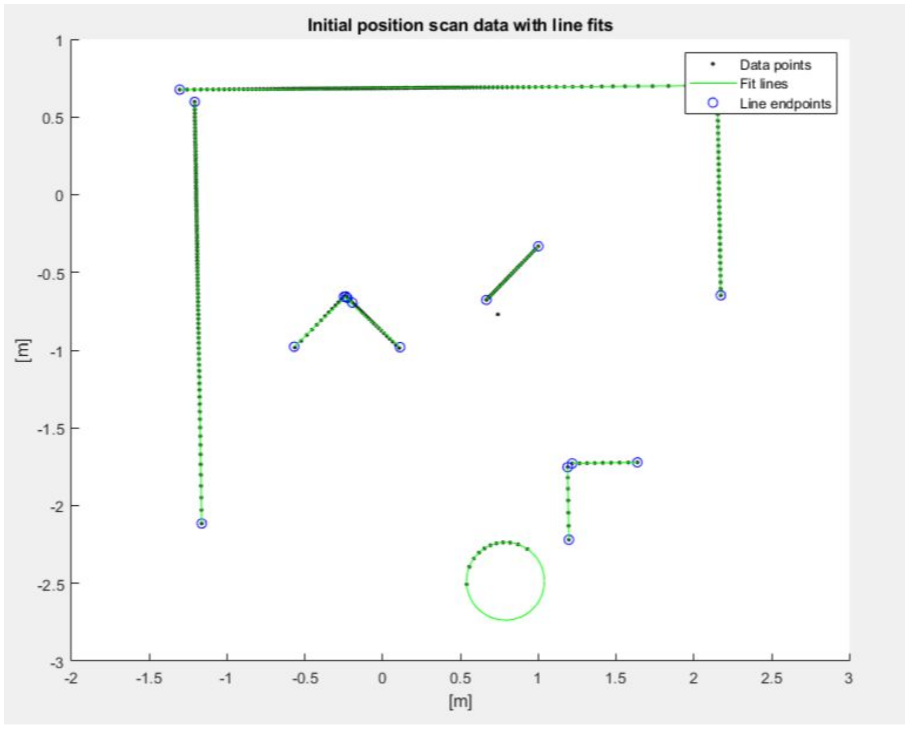
I also initially tried taking the polynomial best fit of the path points, turning it into a parametric equation, and running it through my Bridge of Doom parametric path follower code, but it wasn't working very well and so I decided to go for a simpler approach to speed up the process.

The five main steps to my method of completing the challenge were:

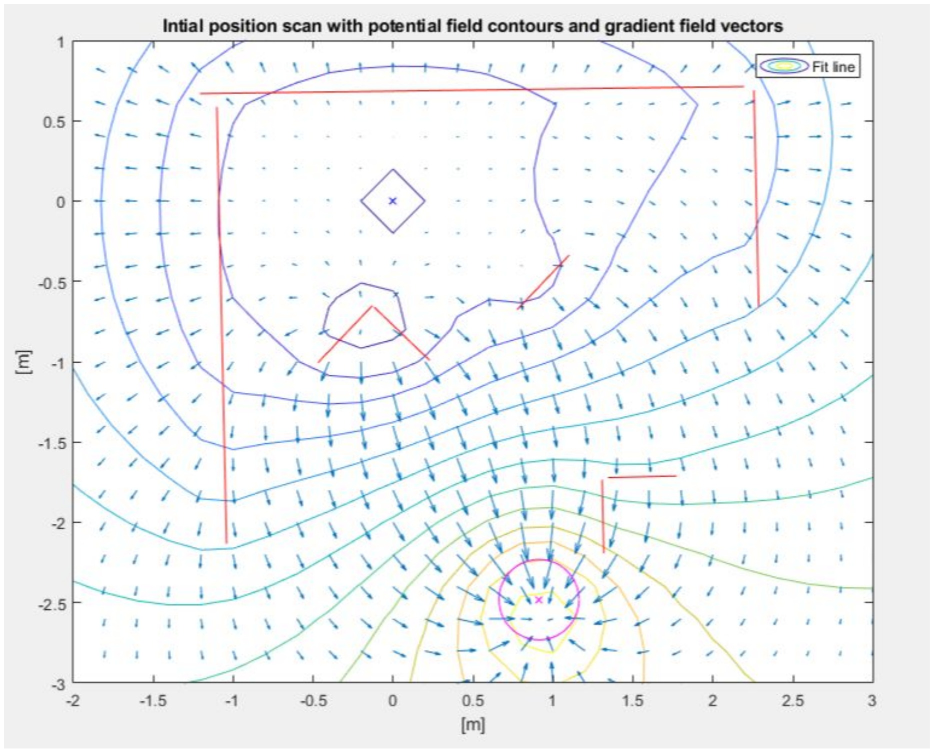
1. Scan: collect LIDAR scan data



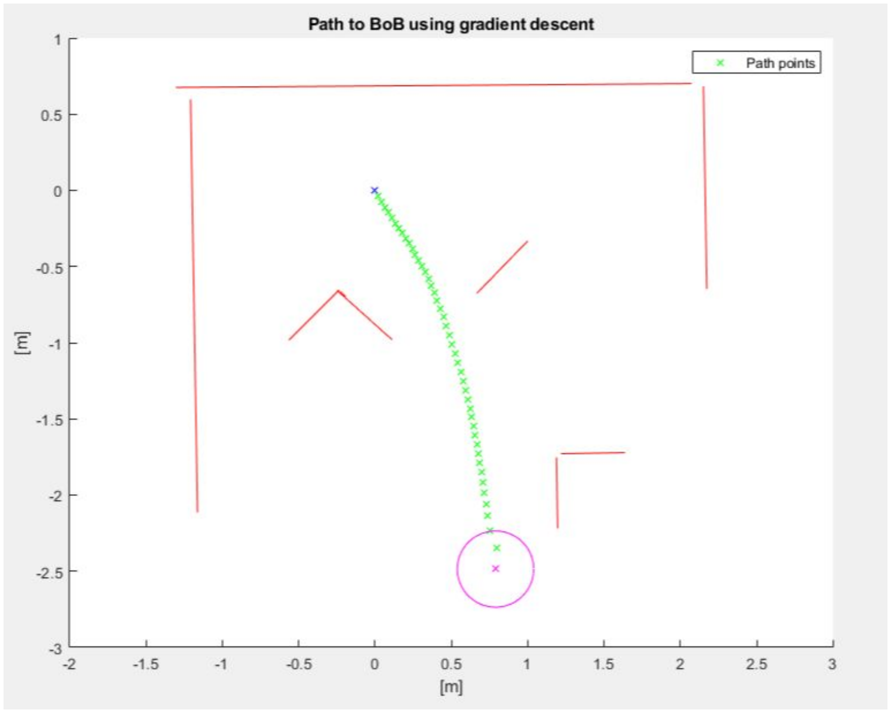
2. Fit: run circle detection for center coordinate, then find obstacles as line segments



3. Path:
    a. create potential field with sources across line segments and sinks along BoB circle

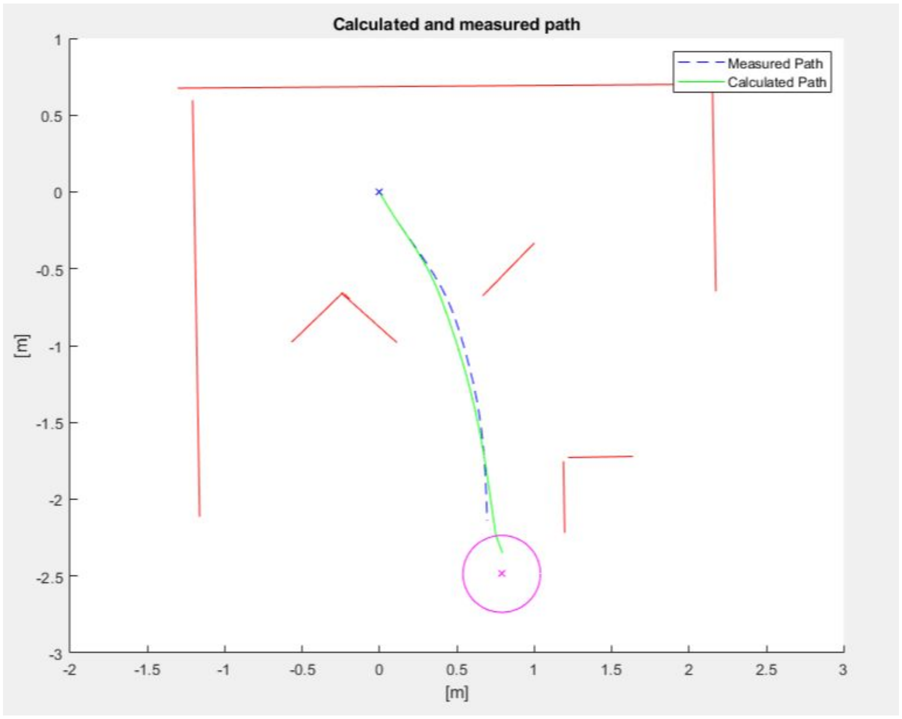


b. run gradient descent to calculate path to local maximum



4. Drive: use the change in heading to feed into the angular velocity in order to roughly follow the calculated points path (also: record encoder data as Neato drives)
    a. Calibrate coefficients for the angular velocity, forward velocity, and time increment
    b. Results: the neato drove the path in **4.405 seconds** according to my program
        i. *Video link:* https://youtu.be/aL6x6IM77Cw
        ii. Or find it in the github repository linked above

5. Analyze: reconstruct path from encoder data, compare it to calculated path


Calculated and measured path
Measured Path
Calculated Path
[m]
[m]