University of Texas at Arlington

Project #1: Navigation Robot with Pathfinding Method
Project Report

Leonardo Gregorio Ibarra

Mubtasim Ahmed Rakheen

Muhammad Haziq Zuhaimi

CSE-4360: Autonomous Robots

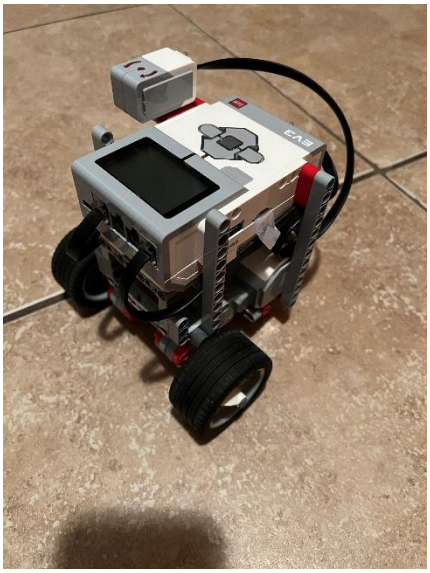
Manfred Huber

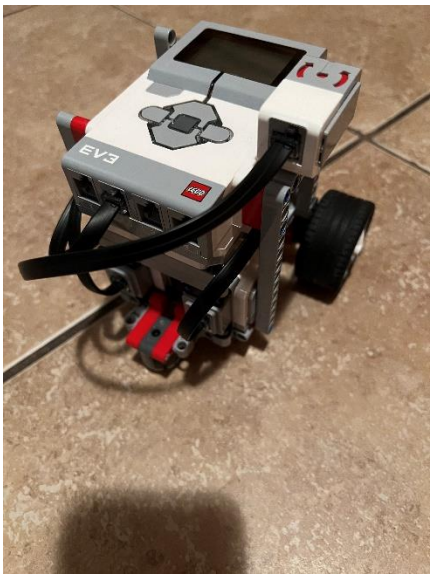
October 27, 2023

Robot Specification

Since the grid specification mentions that one tile space is 0.305 m long and 0.305 m wide, we determined that the robot should be able to fit in that configuration. Therefore, we decided to build a compact non-holonomic differential drive robot with two wheels being driven by motors, and a small metallic ball as the back wheel for better stability during movements and turns.

For sensors, the robot uses a gyroscope to be able to perform 90-degree turns. In addition, the robot motors contain encoders which helped in determining whether one motor was spinning faster than the other. For a better visual, the robot wheels have a flap in them to see the current angle of the wheels.





Method For Pathfinding

To create the best possible path route to take our robot from point A to point B in the minimum amount of distance, we decided to utilize the Manhattan distance algorithm. Since our configuration space is mapped as a grid with free space cells and obstacle cells sharing the same measurement, the Manhattan distance formula was a straightforward choice for our pathfinding method.

Python was our choice of programming language since our group was more familiar with the workspace environment. It was difficult and time-consuming when we first tried using the C-based system and Eclipse, so we opted for a more familiar setting.

How It Works

For our program to effectively choose a path, we had to program the obstacle course and pinpoint locations of the starting position, end position, and obstacle positions. The implementation of our grid consisted of the dimensions of 10 tiles by 6 tiles with each cell dimension being around 0.45 m. This resulted in a more cost-efficient implementation of our algorithm since there were fewer cells to consider than the original grid size.

After establishing the mapping of the obstacle course, we pinpointed the locations of the starting point, goal point, and obstacle locations. Each location would take up one cell and be marked differently from free cells. The cell that contained the starting point would be indicated as such and used as part of the path-planning algorithm. Obstacle locations were marked so it could be avoided as part of the criteria for the project.

Since we have established our finalized map with the required locations, the Manhattan Distance heuristic is introduced to label and draw potential paths that the robot can take to reach the goal.

The process begins by marking obstacle cells with a value of -1 so it would be recognized and avoided in the algorithm. After this, the algorithm will begin at the goal cell marked as 0 and begin to increment the distance of each neighboring cell. After each free cell has been marked with its designated Manhattan Distance value, the algorithm will start its search for the path that finds the goal cell with the least distance.

In addition to displacements, our robot needs to account for orientation to move in adjacent cells. The solution was to have a cardinal direction variable for each cell assigned when calculating the Manhattan Distance. The algorithm will then be able to determine whether to turn left or right before moving forward to the next cell given the starting orientation. This process will continue until the robot successfully reaches the goal location.

Challenges

Throughout the process of the project, we found that the robot doesn't exactly turn 90 degrees even with the gyroscope's data. Additionally, the robot would not drive forward straight constantly, even when the encoders determined that the motors were driving at the same speed. After constant experiments and research, we found out that the sensors can be very inconsistent with their readings, so we had to include error constants that would correct the angle rotation to be exactly 90 degrees. Finally, we figured out that changing the status of stopping to hold during the running function resulted in the robot moving straight with minimal error.