Project One Report

CSE-4360-001 Robotics

November 2, 2022

Preston Mann
1001677093
Bryan Nguyen
1001719605

Objective:

Our objective is to use A* as a path finding algorithm in order to navigate a 10X16 cell grid with objects placed in front of a goal. We are allowed to use some basic sensors however we did not use any sensors and only kept track of our kinematics by moving our motors for a fixed amount of time. The A* algorithm used euclidean distance to measure a cost when looking for the shortest distance.

Robot Structuring:

We went with an easy to build structure that had no sensors and only uses the motors. The structured look like the follow image(figure 1)

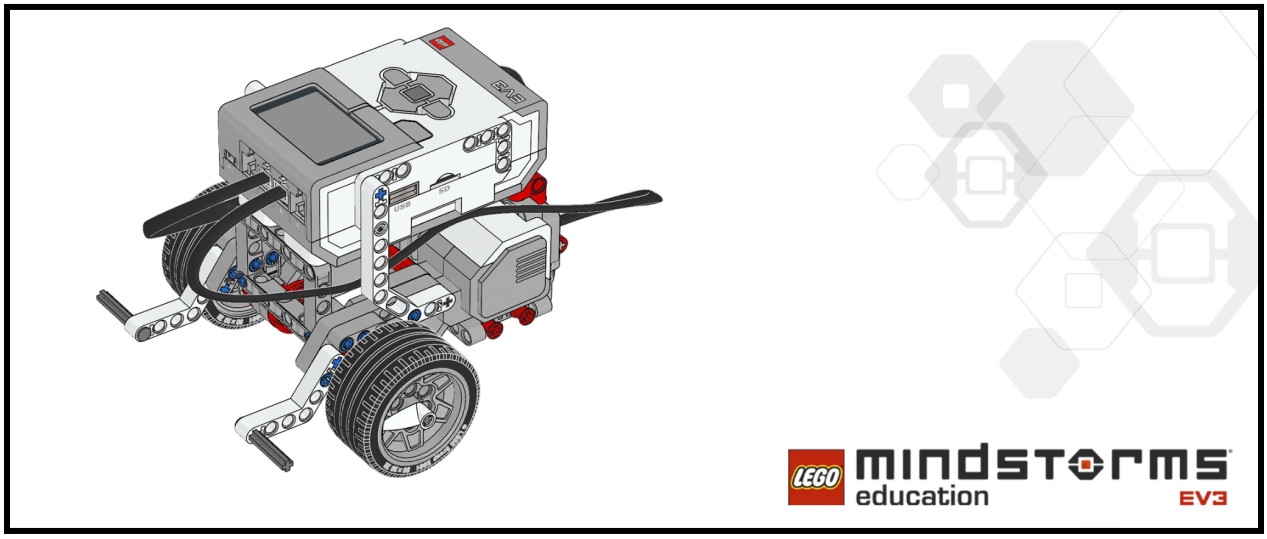


Figure 1: A example image of how our robot looks

We found this design from the EV3 website

https://education.lego.com/v3/assets/blt293eea581807678a/bltc8481dd2666822ff/5f8801e3f4f4cf0fa39d2fef/ev3-rem-driving-base.pdf

The benefits for this design can be summed as follows.

- 2 motors makes finding the kinematics and dynamics of the robot easy
- Stable turning due to the size
- Only took one hour to make this robot
- No sensors were added which allowed less work in the code for implementations

Main components.

- Two wheels and two motors
- Rotating ball for turning

Motor Controls:

Since we have 2 wheels with 2 independent motors we had to figure out a way to keep track of our vehicle kinematics. We defined functions for going forward where both motors would run at the same time in the same direction. To do this we would calculate a movement speed and a movement time by doing experiments. To get the movement speed we used the 300/(wheel diameter * pi)*360 where we determined the wheel diameter to be 120 to give a speed that worked best for our turning and forward movement. We got the movement time by allowing the robot to move forward and adjusting the movement time until the robot moved exactly 1 tile in distance. We also did this a movement going diagonal which needs to move slightly longer however in the final demo product we no longer used diagonal movements.

For turns we would simply move one motor in the negative direction and the other in the positive direction giving us a perfect turn. We only needed to find the turn time needed to go 90 degrees with a fixed turn speed which was half of the movement

speed. We did this for turning right and turning left at 90 degrees and 45 degrees however the 45 degree turn was never performed in our demo. We spent a lot of time trying to perfect the 90 degree turn with offset values to account for any friction on the floor and overall possible drift from not doing a perfect 90 degree turn. In our experiments 6/10 times we would be within less than 1 tile from our goal whereas 4/10 times we would drift 1 or more tiles from the goal.

Path Finding Algorithm:

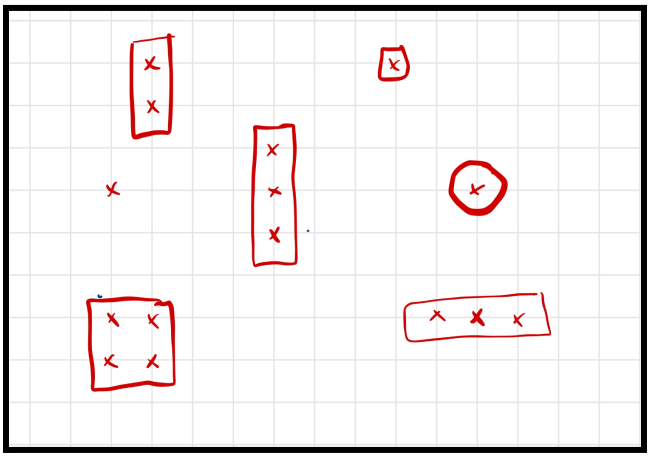
For our path finding algorithm we based it off of a star. In a star the algorithm uses 3 values F, G, and H inside a node structure which depend on the distance from the starting position and the end position. The F value is the sum of the G and H values. G is the distance from the starting block or node and the current block. H is the heuristic which means the distance from the current to the end block. It uses the euclidean distance formula to calculate the distances. For our mapping we decided to use a 2-D array which is convenient since each “index” maps to exactly one tile. Since we were given the values for the start, end, and obstacles we used a ‘0’ to show the blocks that did not occupy an obstruction and a ‘1’ to represent the obstacles. When the rover first starts running it evaluates all the neighboring blocks around it, adds it to an open list and finds the 3 values we just discussed. Except when looking at the neighboring blocks it ignores it if there is a 1 in the current block in a 2-D array signifying an obstacle. From there it moves to the best neighbor by evaluating its F value and records the previous block as its parent. If it has been evaluated, look at the G score and compare it to the new G score. If the G is less it means a more optimized path so change the parent to

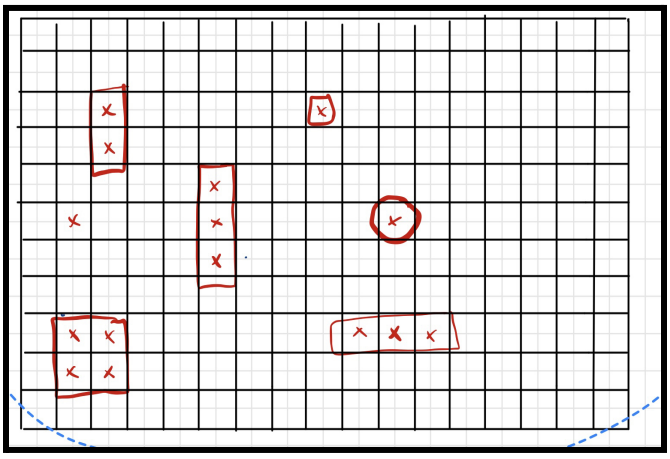
the best one. The algorithm finishes when you either evaluate all the blocks on the map or when the end point is added to the closed. If that is the case you start with the end point and find the parent repeating until you are at the starting position. Then reversing that path to reorder the sequence.

Finally, to decide if it was going forward, backwards, or turning we took the current block and the next block, subtracted them and then based on the difference it decided what to do next. For the distance calculation we added .305 for every forward movement and the diagonal was supposed to be the hypotenuse, but instead we have .305 * 2 for two forward movements.

Problems and Solutions:

The first problem that we ran into was where the starting, end, and obstacles sat within a tile. If everything sat in the middle of the tiles everything would've been perfect with a 2-D array structure, but everything is located at the intersections of 4 tiles. To compensate for this, we increased the 16x10 array with one extra row and column and overlaid it where everything appears to be in the middle. Then put ‘1’ on the outside rows and columns to act as borders. The origin of the given map had (0, 0) at the bottom right corner while ours had it at the top left, we just found the corresponding value by subtracting the 10 with the given value.





With the many turns and rotations we have to do in the assignment it can be confusing keeping track of the orientation of the rover, so instead we made it turn back after every turn keeping its orientation facing the goal. This led to many other problems because since the rover had some inconsistency with the motors the turn was not always perfect, which led to the rover to accumulate errors when moving. One thing that resolved this issue was running the motor slower which reduced the offset in the angles we wanted. At first our plan to do diagonal turns was to rotate 45 degrees and go up by the hypotenuse to reach the other corner, but we found out quickly that it would hit the corners of the obstacles. Instead we opted to do the same thing instead implementing our 90 degree turns, so it would go forward, rotate then go forward again, landing in the same spot. After testing it out on the course, we found out that based on the order of rotating or turning first influenced if it was going to hit the corners of the obstacles, to solve this we had to look at to see what side the obstacle was on. From there, we then decided on the sequence of order for the ‘diagonal’ turns.