

ROB301 Project Report



Team 9

Christopher Sheedy, Lauren Adolphe,
Shahin Jafari

12/6/2017

Introduction

The objective of this project was to build a robot that could autonomously navigate its way through a maze. Equipped with the Lego Mindstorms EV3 Educator Vehicle and 2 weeks in the lab, there were limited resources and time to work with. Given these constraints, we focused on creating an effective design with little complexity so as to ensure both our completion of the robot and its completion of the maze.

On the hardware side, the robot used the EV3 brick connected to two motors and a single sensor. The differential steering was predictable and reliable, while the chosen gyroscope sensor provided useful measurement feedback on the robot's direction. On the software side, we implemented a graph and breadth-first search algorithm, which guaranteed the shortest path. Our robot then was able to navigate through the maze by navigating through each of the nodes from the aforementioned graph with modular turning and forward motion commands.

Our robot successfully navigated the maze on the second run with only a single instance of human intervention. This indicated the strength of the path-planning algorithm. However, error mitigation and correctional abilities were overlooked in the simplicity of the chosen design.

Robot Platform Design

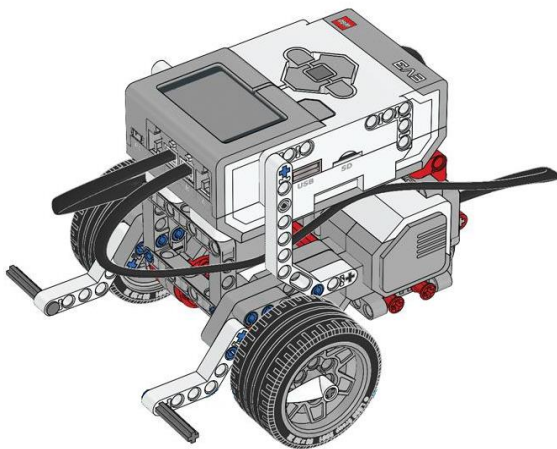


Figure 1, Overview of the designed robot

The mobile base that the team used was the LEGO MINDSTORMS EV3 Educator Vehicle (above left), with minor modifications. The two appendages at the front of the robot were removed to reduce the footprint of the robot, reducing the possibility of collision with the walls of the maze.

The Educator Vehicle uses tripod differential steering. This method of steering has two independently controlled wheels, and a third unpowered omnidirectional bearing at the back for support. By adjusting the speed and direction of the wheels the robot can change its center of rotation.

A single gyroscope was attached to the mobile base to take readings on the angular position of the robot. The complete mobile platform with gyroscope is pictured (above right).

Overview of Solution Strategy

The proposed solution aimed to keep the operation as simple as possible. The team, therefore, designed the robot to be able to find a path at the beginning of the operation. Having the path in hand, a pose to pose function was written to take the robot from an initial pose and brings it to a final pose and allow it to turn on the spot. By iterating through the list of the points found in the path planning and applying the pose to pose function on them, the robot was able to navigate through the maze.

Design Methodology

Path planning

As per the requirement of the project, a path planning algorithm that generates the set of accessible points given the initial and final positions was required. To find the shortest path, an adjacency matrix was used. The map was rendered as a 11x11 matrix originally, where each tile had 5 points (up, middle, down, left, right). Since the first and last row and the first and last column were all representing a wall, they were omitted from the matrix and the final adjacency matrix dimension was 9x9, where each point representing a wall had the value of 1, and the available spots were marked with 0. The point (0,0) was defined to be the top-left corner and the (9,9) was defined to be the bottom-right corner.

0	1	0	0	0	0	0	0	0
0	1	0	1	1	1	1	1	0
0	1	0	1	0	0	0	0	0
0	1	0	1	0	1	1	1	0
0	0	0	1	0	1	0	0	0
1	1	0	1	0	1	0	1	1
0	0	0	1	0	1	0	1	0
0	1	1	1	0	1	0	1	0
0	0	0	1	0	1	0	0	0

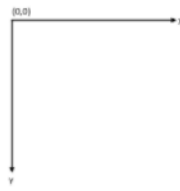


Figure 2, Matrix representation of the maze

The breadth-first-search algorithm was chosen for path finding, since the maze graph did not have any weights or direction and therefore, BFS was guaranteed to generate the shortest path. The following is a summary of the algorithm:

- Define allowable movements to be in cardinal directions. Therefore Δx and Δy can be ± 1 .
- Set every node to be unvisited.
- Append the first point (initial position) to a queue q.
- Iterate until q is empty: (while loop)
 - Dequeue a node n from q.
 - If n has the coordinates of the endpoint, stop and generate the path!
 - Otherwise, check all the cardinal adjacents of n that are possible points to go to.
 - If they haven't been visited before, set their parent to be n.
 - add them to q.
- Repeat

Once the robot reached the destination, the program exited the while loop and it backtracked the nodes starting from the destination. Each node was inserted as the first element of a reference array list in every iteration and the pointer was set to the node's parent. This backtracking was performed until the pointer reached the initial point and inserted it into the referenced array list. At the end of the path finding function, the reference array list of nodes that represented the shortest path with their coordinates was returned.

Pose to pose movement

This function was designed to control the movement of the robot. Accepting 2 parameters, initial and final position, it directed the robot from the initial pose to the final pose by first, making the robot aligned with the shortest line connecting the initial and final positions and then, move the robot straight and along the line.

A rotation was the only required action for the robot to face the final position since the robot was made to be able to turn on the spot. Based on the coordinates and allowable directions definition, the following turning scenarios were considered:

Δx	Δy	Angle to rotate to
0	1	270
0	-1	90
1	0	0
-1	0	180

Figure 3, Angle of rotation determination

At the beginning of the operation, a function (firstAngle) was designed to bring the robot to align to 0° of the predefined coordinates system. The function's input was an angle, and it rotated the robot counterclockwise until the gyroscope read the input angle. By doing so, the robot was guaranteed to start reading the angles from the 0° of the coordinate system. Due to time constraint, the team did not fix the odd definition of the coordinate systems (90° was set to be located at (0,-1) as opposed to standardized coordinates at (0,1)) but accounted for it in the function input, which ultimately hindered the readability of the overall code.

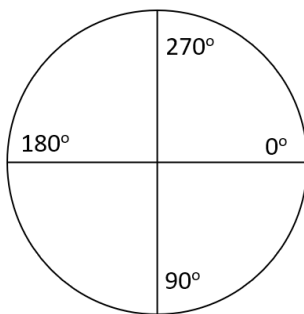


Figure 5, Initial angle definitions

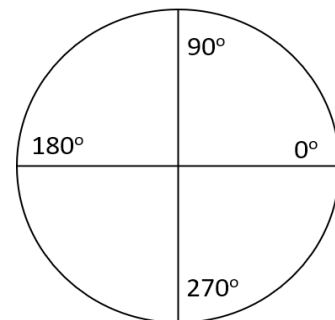


Figure 4, Angle definitions after initialization

At the end, a function (finalAdjust) was designed to adjust the robot to the final orientation. The coordinate system and the angles defined were exactly the same as the cartesian system and the team did not need to account for discrepancies.

Since all the points defined were placed in half-foot increments, the translational motion was achieved by rotating the wheels precisely 316° . The desired wheel rotation was originally determined by dividing the half-foot distance by the circumference. This calculation was then confirmed through testing and calibration.

Demonstration Performance

On the first run, an issue we had been experiencing with the gyroscope caused the robot to oversteer on the first rotation, causing it to collide with the wall, and eliminating any possibility of completing the run without adjustment.

On the second run, the gyroscope behaved as expected after a reset of the EV3 Brick. The robot successfully navigated the maze, but a significant amount of error built up on the final stretch, requiring an adjustment near the end of the demonstration. After the adjustment, the robot successfully made it to the final pose.

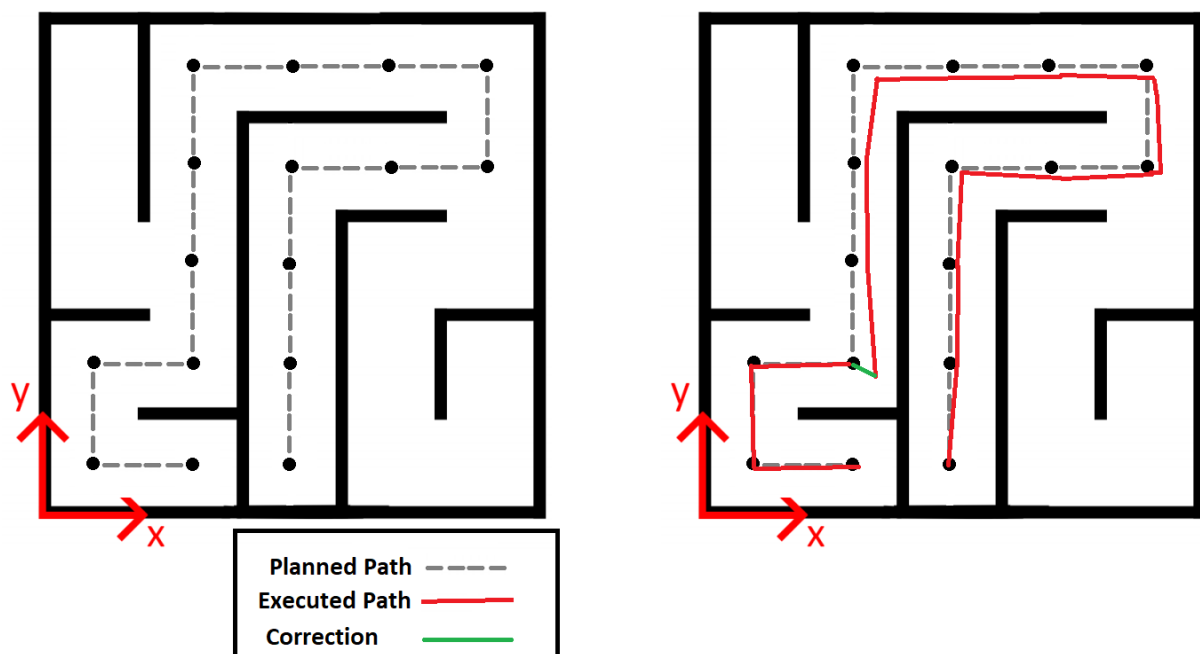


Figure 6, Summary of execution

Potential Improvements

One of the major sources of improvement can be done on path finding function. As mentioned earlier, the (0,0) was the top-left corner, and because of the gap between 2 lab sessions,

this subtlety was lost in translation, costing our team valuable time to figure it out. The cardinal angles were also different at the beginning and again, it took time for us to fully understand its impact on the navigation. As a result, a lot of time was wasted where we could've used it for calibration.

Another improvement can be done in translational motion code. We designed it in terms of the rotation of the wheels, which was not very accurate due to imperfections on the floor's surface. As discussed earlier, the error built up, resulting in a non-negligible deviation from our desired trajectory.

Our robot was very dependent on the initial position for calibration of the gyroscope. If we didn't place it correctly in the middle of a tile facing one of the cardinal angles, it would've failed in initial adjustment and as a result, our robot would deviate from the desired trajectory in every turn. Furthermore, since we did not wrap the turning angles to angles between 0 and 180, the robot's position was in higher risk of a change due to the tight space and potential interference of the walls with the wires and roughness of the ground. A potential improvement can be done by writing a wrapper function, that wraps the angle thus containing the rotation to the range of $[0, \pm 180]$.

A major downfall of our robot's execution was error mitigation. As seen in our robot's path diagram, small deviations from the ideal were left uncorrected. As the robot progressed through the maze, these errors propagated and accumulated. To improve upon this, we could implement a simple PID controller. With the use of an ultrasonic sensor, we could measure lateral position from the wall and utilize these coordinates to implement the feedback control.

Conclusion

The robot was able to achieve the initial goal of this project, albeit with minimal human intervention. Due to the effective path-planning, the robot was consistently and reliably able to find a path between any two points in the maze. This process was executed efficiently at the beginning of each run, demonstrated by the minimal lag between pressing start and the robot starting.

On execution, the robot's strength was consistent angular direction, but it lacked translational accuracy. Given additional time, we would have implemented rotational wrapping and PID control for improved translational motion. With these enhancements, the robot could actively avoid the walls of the maze, thus enabling it to complete the course without human intervention.

Appendix

Robot Source Code

```
1. package csl_final_project;
2. import java.util.*;
3. import java.util.HashMap;
4. import lejos.hardware.Button;
5. import lejos.hardware.lcd.LCD;
6. import lejos.hardware.motor.Motor;
7. import lejos.hardware.port.SensorPort;
8. import lejos.hardware.sensor.EV3GyroSensor;
9. import java.util.ArrayList;
10. class node {
11.     int x;
12.     int y;
13.     int value;
14.     node parent;
15.     int visited = 0;
16.     public node(int x1, int y1, int val) {
17.         x = x1;
18.         y = y1;
19.         value = val;
20.     }
21. }
22. public class robot_control {
23.     public static EV3GyroSensor tilt = new EV3GyroSensor(SensorPort.S4);
24.     public static int sampleSize = tilt.sampleSize();
25.     public static float[] tiltsample = new float[sampleSize];
26.     public static ArrayList path(node[][] map, node start, node end) {
27.         ArrayList < node > path = new ArrayList < node > ();
28.         ArrayList < node > dir = new ArrayList < node > ();
29.         ArrayList < node > q = new ArrayList < node > ();
30.         ArrayList < node > visited = new ArrayList < node > (); //direction list
31.         node temp_node = new node(1, 0, 0);
32.         dir.add(temp_node);
33.         temp_node = new node(-1, 0, 0);
34.         dir.add(temp_node);
35.         temp_node = new node(0, 1, 0);
36.         dir.add(temp_node);
37.         temp_node = new node(0, -1, 0);
38.         dir.add(temp_node);
39.         start.parent = null;
40.         q.add(map[start.y][start.x]);
41.         node current = null;
42.         while (!q.isEmpty()) {
43.             current = q.get(0);
44.             q.remove(0);
45.             if (current.x == end.x && current.y == end.y) {
46.                 break;
47.             }
48.             for (int i = 0; i < dir.size(); i++) {
49.                 node direction = dir.get(i);
50.                 if ((current.y + direction.y >= 0 && current.y + direction.y <= 8) && ((current.x + direction.x >= 0 && current.x + direction.x <= 8)) && map[current.y + direction.y][current.x + direction.x].value != 1) { //System.out.print(current.y + direction.y); //System.out.println(current.x + direction.x);
51.                     node child = map[current.y + direction.y][current.x + direction.x];
```



```

52.         if (child.visited == 0) {
53.             child.parent = current;
54.             q.add(child);
55.         }
56.     }
57. }
58.     current.visited = 1;
59.     visited.add(current);
60. }
61. while (current.x != start.x || current.y != start.y) {
62.     path.add(0, current);
63.     current = current.parent;
64. }
65. path.add(0, current);
66. return path;
67. }
68. public static void gotoAngle(float angle) throws InterruptedException { // set speed
69.     Motor.B.setSpeed(50); // left
70.     Motor.C.setSpeed(50); // right
71.     /*     sampleSize = tilt.sampleSize();     tiltsample = new float[sampleSize];     ti
72.     lt.getAngleMode().fetchSample(tiltsample, 0);     */
73.     if (tiltsample[0] > angle) {
74.         Motor.B.forward();
75.         Motor.C.backward();
76.         while (tiltsample[0] > angle) { //sampleSize = tilt.sampleSize(); //tiltsample = n
77.             ew float[sampleSize];
78.             tilt.getAngleMode().fetchSample(tiltsample, 0);
79.             System.out.println(tiltsample[0]);
80.         }
81.     } else {
82.         Motor.B.backward();
83.         Motor.C.forward();
84.         while (tiltsample[0] < angle) { //sampleSize = tilt.sampleSize(); //tiltsample = n
85.             ew float[sampleSize];
86.             tilt.getAngleMode().fetchSample(tiltsample, 0);
87.             System.out.println(tiltsample[0]);
88.         }
89.     }
90.     Motor.B.stop(true);
91.     Motor.C.stop(true);
92. }
93. public static void firstAngle(float angle) throws InterruptedException { // set speed
94.     Motor.B.setSpeed(50); // left
95.     Motor.C.setSpeed(50); // right
96.     tilt.reset();
97.     int wait = 0;
98.     while (wait < 1000) {
99.         wait = wait + 1;
100.     } //sampleSize = tilt.sampleSize(); //tiltsample = new float[sampleSize];
101.     tilt.getAngleMode().fetchSample(tiltsample, 0); //System.out.println(tiltsample[0]);
102.     if (tiltsample[0] > angle) {
103.         Motor.B.forward();
104.         Motor.C.backward();
105.         while (tiltsample[0] > angle) { //sampleSize = tilt.sampleSize(); //tiltsam
106.             ple = new float[sampleSize];
107.             tilt.getAngleMode().fetchSample(tiltsample, 0);
108.             System.out.println(tiltsample[0]);
109.         }
110.     } else {
111.         Motor.B.backward();
112.         Motor.C.forward();

```



```

109.         while (tiltsample[0] < angle) { //sampleSize = tilt.sampleSize(); //tiltsam
    ple = new float[sampleSize];
110.             tilt.getAngleMode().fetchSample(tiltsample, 0);
111.             System.out.println(tiltsample[0]);
112.         }
113.     }
114.     Motor.B.stop(true);
115.     Motor.C.stop(true);
116.     tilt.reset();
117.     wait = 0;
118.     while (wait < 1000) {
119.         wait = wait + 1;
120.     }
121. }
122. public static void finalAdjust(float angle) throws InterruptedException { // set sp
    eed
123.     Motor.B.setSpeed(50); // left
124.     Motor.C.setSpeed(50); // right
125.     /*      sampleSize = tilt.sampleSize();      tiltsample = new float[sampleSize];
    tilt.getAngleMode().fetchSample(tiltsample, 0);      */ //System.out.println(tiltsample[0]
    );
126.     if (tiltsample[0] > angle) {
127.         Motor.B.forward();
128.         Motor.C.backward();
129.         while (tiltsample[0] > angle) { //sampleSize = tilt.sampleSize(); //tiltsam
    ple = new float[sampleSize];
130.             tilt.getAngleMode().fetchSample(tiltsample, 0); //System.out.println(ti
    ltsample[0]);
131.         }
132.     } else {
133.         Motor.B.backward();
134.         Motor.C.forward();
135.         while (tiltsample[0] < angle) { //sampleSize = tilt.sampleSize(); //tiltsam
    ple = new float[sampleSize];
136.             tilt.getAngleMode().fetchSample(tiltsample, 0); //System.out.println(ti
    ltsample[0]);
137.         }
138.     }
139.     Motor.B.stop(true);
140.     Motor.C.stop(true);
141. }
142. public static void pose2pose(node current, node next) throws InterruptedException {
143.     /*      sampleSize = tilt.sampleSize();      tiltsample = new float[sampleSize];
    tilt.getAngleMode().fetchSample(tiltsample, 0);      */
144.     float relative_angle = tiltsample[0];
145.     float goal_angle = 0;
146.     int dx = next.x - current.x;
147.     int dy = next.y - current.y;
148.     if (dx == 1) {
149.         goal_angle = 0;
150.     } else if (dx == -1) {
151.         goal_angle = 180;
152.     } else if (dy == 1) {
153.         goal_angle = 90;
154.     } else if (dy == -1) {
155.         goal_angle = 270;
156.     } //System.out.println(goal_angle); //goal_angle = goal_angle - true_angle;
157.     gotoAngle(-1 * goal_angle);
158.     Motor.B.setSpeed(100); // left
159.     Motor.C.setSpeed(100); // right

```

```

160.         Motor.C.rotate(316, true); //318
161.         Motor.B.rotate(316, true); //318
162.         Motor.B.waitComplete();
163.         Motor.C.waitComplete();
164.     }
165.     public static node[][] graph1() {
166.         int x = 0;
167.         int y = 0;
168.         int val = 0;
169.         node[][] map = new node[9][9];
170.         int row = 0;
171.         int col = 0;
172.         for (row = 0; row < 9; row++) {
173.             for (col = 0; col < 9; col++) {
174.                 node temp_node = new node(col, row, val);
175.                 map[row][col] = temp_node;
176.             }
177.         }
178.         map[5][0].value = 1;
179.         for (row = 0; row < 4; row++) {
180.             col = 1;
181.             map[row][col].value = 1;
182.         }
183.         map[5][1].value = 1;
184.         map[7][1].value = 1;
185.         map[7][2].value = 1;
186.         for (row = 1; row < 9; row++) {
187.             col = 3;
188.             map[row][col].value = 1;
189.         }
190.         map[1][4].value = 1;
191.         map[1][5].value = 1;
192.         for (row = 3; row < 9; row++) {
193.             col = 5;
194.             map[row][col].value = 1;
195.         }
196.         map[1][6].value = 1;
197.         map[3][6].value = 1;
198.         map[1][7].value = 1;
199.         map[3][7].value = 1;
200.         for (row = 5; row < 8; row++) {
201.             col = 7;
202.             map[row][col].value = 1;
203.         }
204.         map[5][8].value = 1;
205.         return map;
206.     }
207.     public static void main(String[] args) throws InterruptedException {
208.         /* float angle1 = 90; float angle2 = 45; float angle3 = 90;
float angle4 = 0; gotoAngle(angle1); gotoAngle(angle2); gotoAngle(angle3);
gotoAngle(angle4); */
209.         while (!Button.RIGHT.isDown()) { //chill
210.         }
211.         /* node start = new node(1,4,0); node end = new node(0,4,0);
pose2pose(start,end,270); pose2pose(end,start,270); */ //path finding mai
n //node(x,y,value = 0)
212.         node start = new node(4, 8, 0);
213.         node end = new node(2, 8, 0);
214.         int init_angle = 180;
215.         int final_angle = 90;
216.         node[][] map1 = graph1();

```

```

217.         ArrayList < node > path_found = new ArrayList < node > ();
218.         path_found = path(map1, start, end);
219.         System.out.println("done planning");
220.         for (int i = 0; i < path_found.size() - 1; i++) {
221.             System.out.print("go from: ");
222.             System.out.print(path_found.get(i).x);
223.             System.out.print(",");
224.             System.out.print(path_found.get(i).y);
225.             System.out.print(" to: ");
226.             System.out.print(path_found.get(i + 1).x);
227.             System.out.print(",");
228.             System.out.print(path_found.get(i + 1).y);
229.             if (i == 0) {
230.                 firstAngle(init_angle);
231.                 pose2pose(path_found.get(i), path_found.get(i + 1));
232.             } else pose2pose(path_found.get(i), path_found.get(i + 1));
233.             System.out.print("    Done movement!");
234.         } //final adjustment
235.         finalAdjust(final_angle);
236.     }
237. }

```