Project Report

Design of the Robot

The robot is designed using two motors attached on the sides to provide forward motion and two ball casters on the back to provide balance. The motors and casters were attached to the robot through different points to try and reduce any shaking while the robot moved. The set of wheels used were 3.5cm thick and had a diameter of 7cm. This set was chosen because it had the least wear on it and its material was in the best condition, in comparison to the other wheels.

Initially, the group wanted to use the set of wheels that had treads that mostly resembled the ones on bicycles. This set was a lot thinner and had a greater diameter of about 8cm. During testing, this set of wheels performed better than the first set regarding turning, making sharper and more consistent turns. However, the major issue that ultimately had the group decide against the set was the effects on the robot as it moved forward. Because the treads were significantly thinner than the other wheels and alternated between different sides of the wheel as it rotated, the robot had a turbulent forward movement pattern, despite moving relatively slow. This would cause the robot to slowly veer off course during test runs.

The only sensor attached to the robot is the gyroscope. It should be noted that the accuracy of the gyroscope readings is inconsistent.

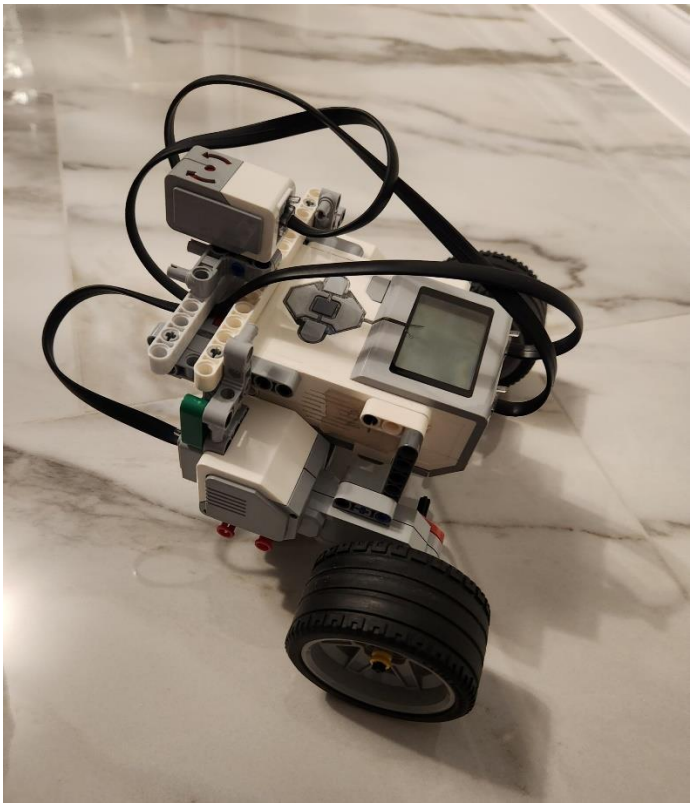


Figure 1. Final Design of the Robot

Group 21 - Michael Nguyen, Muhammad Muawiz Farooqi, Alfredo Reina Corona
Project 1 Report

Pathsearch Strategy

The strategy that the team decided to use was a navigation function, which is a class of potential functions. The constraints of this strategy include:

- Goal is the minimum
- Obstacles are maxima
- There are no other local extremes

Because of these constraints, this strategy forms complete and correct path planners. The group decided to implement an adjusted version of Manhattan distance on a regular grid as a navigation function. The Manhattan distance is calculated by warping around the obstacles, ensuring that there are no local minima where the robot can get stuck in. This strategy was picked because the obstacles, end goal, and starting position of the robot are already on a coordinate system. In addition to not having the robot trapped inside a local minimum, this strategy is easy to compute and implement. The Manhattan distance is calculated starting from the goal position. The Manhattan distance of the goal is set to **0**, and the obstacles are set to **1000**. The value of the obstacles was picked as an arbitrary value large enough to be greater than any possible expected Manhattan distance value at any position in the 10 x 16 grid. With the goal starting at **0**, the highest possible Manhattan distance of free spaces would be **24** and is impossible to reach as high as **1000**.

Before any pathfinding took place, the environment was discretized using the *bfs_manhattan* function. This function returns a 2D array representing the workspace of the robot in terms of the Manhattan distance from the goal.

The chosen path from the *findPath* function returns a list of tuples, representing the path with the shortest possible travel distance from the starting position to the goal. As the robot travels down the path, if there is a tie between the next possible tiles, the priority of the directions is *Up*, *Down*, *Right*, and *Left*. The robot will prioritize moving vertically along the y-axis over horizontal movements along the x-axis.

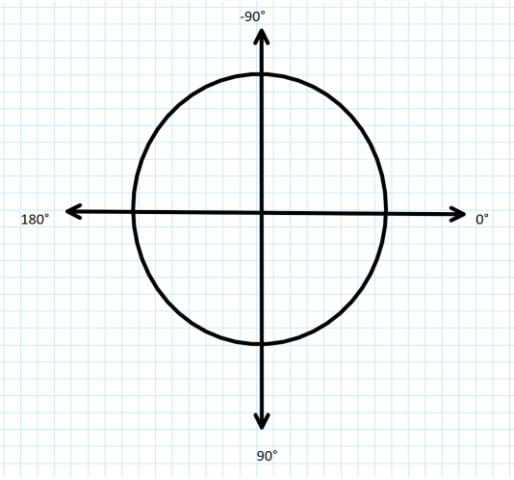
The *findPath* function returns the path as an array of directions for the robot to follow through iteration. The directions (*Up*, *Down*, *Right*, *Left*) are based on looking at the regular coordinate grid of the workspace. For the *Up* direction, the robot would move 1 square higher along the y-axis while the *Down* direction brings the robot 1 square lower. The *Left* direction moves the robot 1 square to the left along the x-axis while the *Right* direction moves the robot 1 square to the right.

Moving the Robot

The robot's movements are largely based on the direction returned from the *findPath* function. With each direction, besides the *Start* that is skipped, the robot first orients itself in the right direction proceeding to move forward. The side lengths of each square on the regular grid are 0.305m long. The robot moves forward in 2 steps. It first moves half a square forward before checking its orientation. If the orientation is unaligned and the robot detects the error, it corrects

itself before moving forward another half square. After it moves a full square, it'll check its orientation again before moving on to the next direction.

Throughout its run, the orientation of the robot is determined using the gyroscope. The orientation is based on the following figure. For example, if the direction is *Up* then the robot's target angle is -90°. The direction the robot turns depends on whether the current angle of the robot is less or greater than the target angle. This target angle is also what is used whenever the robot corrects its orientation. For each run, the robot will always start facing in the 0° direction.



Challenges Faced

The group faced some issues along the way. Early in the development cycle the angles the gyroscope would use to move in a direction were not clear to us, so we made an educated guess about their ranges; this caused the angles of adjacent tuns to overlap and cause the robot to enter an infinite cycle of moving from one direction to another. Also, throughout the development there was a bug in the code that would cause the robot to "adjust" at the end of every tile. This movement was unpredictable and would often cause the robot to veer off-path.

For every challenge with the software, it seemed as if there was one for the hardware as well. The main physical issue throughout the whole project was the choice of wheels. We started off with the basic 5cm wheels but soon realized that their width-to-grip ratio was not ideal. They were too wide, so they easily picked up dust and slipped on the tile but were not coarse enough to provide the needed grip. We then tried to go with the 8cm wheels. Theoretically, these wheels would be the best choice, as they would disperse the weight over a smaller area which would minimize slippage, but again, we found out that this was not the case. The 8cm wheels did help minimize slippage, but because of the protruding tread, it would shake the whole vehicle and veer it off the path without any change detected on the gyroscope.

The next biggest problem was the inconsistency of the materials. The motors would fire off at separate times even when prompted not to, and this would start the vehicle on the wrong path almost immediately. Similarly, some of the wheels seemed more used than others, so we had to

find ones that matched evenly to minimize the slippage. The sensitivity of the gyroscope was also incredibly inconsistent on what values it reads as the robot moves. This can be seen in how different the robot moves between consecutive runs. It was the main reason why a correction function was implemented.