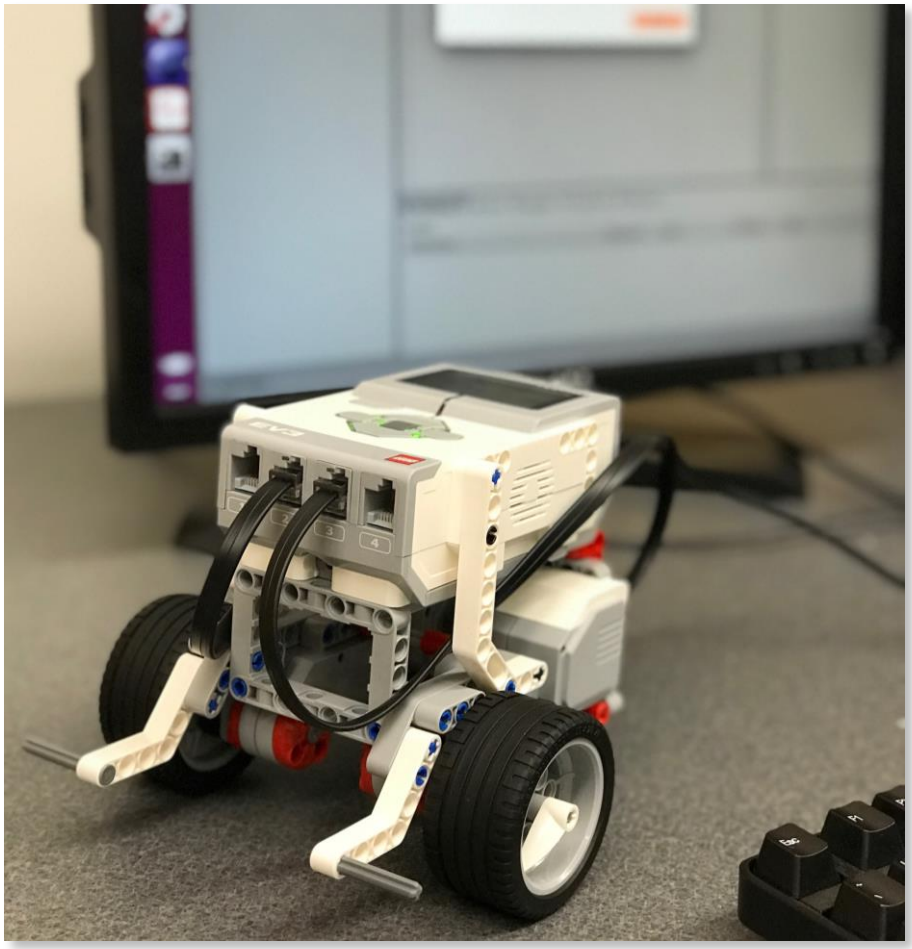
Team: Mindstorms
Dr. Manfred Huber
Robotics (CSE 5364)
15 November 2017

Autonomous Robots

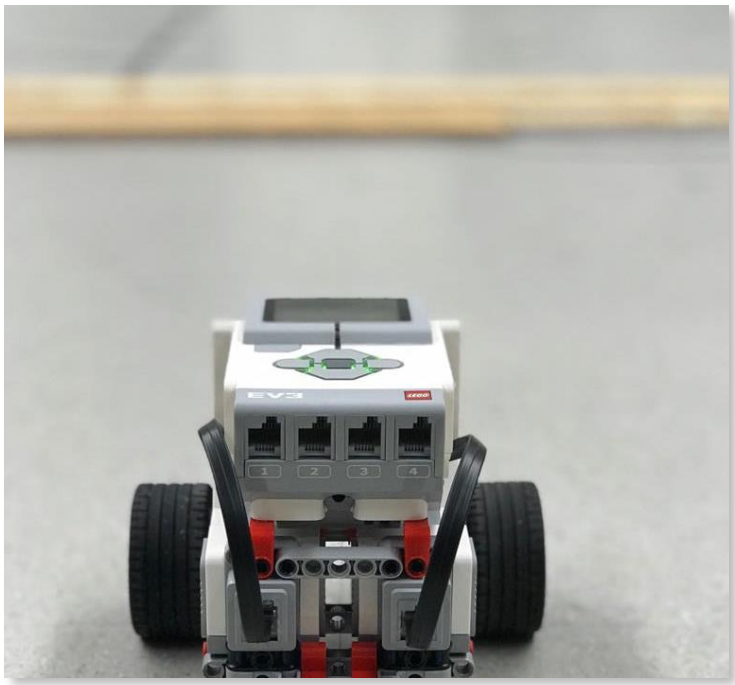
Navigating a known Obstacle Course with Lego Robot



Objective

Navigate a mobile robot through an obstacle course to a goal location. The start position of the robot as well as the locations of all obstacles and of the goal are given before the robot is started.

“Moving at slower speeds improved the overall precision drastically.”



Setup

The workspace for the robot is a rectangular area, 3.66 m x 3.05 m in size. Obstacles are squares 0.305 m x 0.305 m in size placed in the workspace. The goal is a black circle with a radius of 0.305 m.

The Project

Robot Design

The task is a very simple one: navigate from start position to goal position and avoid any obstacles on the way. For this, we implemented a sturdy two wheeled robot with motors on each. A small third wheel at the back was mounted for extra support. The EV3 block is supported by the base and is mounted on the top of robot.

A flap on the outside of each wheel was placed that gives a good idea on how much the wheel has rotated. This design principle came in handy in the initial stages to understand the error while changing directions of the robot.

Navigation Strategy

In the given setup, we are not working with any sensors. However, we are provided with positions of all the obstacles, start and goal coordinates. The problem becomes solvable when we map the real world into a matrix of two dimension.

Hence, the workspace was mapped to a two-dimensional array of 12 columns and 10 rows - each value corresponding to a tile in the real world.

Next, we start computing the distance to goal for each tile, starting with the goal position. Obviously, the goal position gets the value of 0. Since we must avoid all obstacle positions, each obstacle is given a value of -1.

Eventually, using Manhattan distance and Breadth First Traversal, the entire grid (2D array) is set with an integer value representing the distance to goal from that position/tile.

The algorithm then begins with the start position and compares the neighboring tiles for a value which is less than the current (lesser value depicts closer position to the goal)

In case of no path found, the robot simply stays at start position and displays an appropriate message.

To minimize error caused by change in direction, the robot prioritizes the current direction of traversal. In case of a higher value for a neighbor in the same direction, the algorithm then looks randomly for a neighbor with lower distance to goal in a different direction. The path planner breaks down the path in a series of 3 directional moves (forward, left or right).

When traversing in a direction, the motor cuts off the power just before it reaches the destination tile. It then computes the exact distance left to traverse and makes error corrections (usually small magnitudes) if need be. This correction is made by calculating the offset between the desired rotation value and current motor rotation value. The required rotation value has been calculated using a web-tool on LEGO website.

The Code

Attached as a project alongside