Lab 1 - Maze Solving

CMSC477

Bradley Dennis

Jack Mirenzi

Table Of Contents

Introduction

In this assignment, we combined and implemented our PID robot movement controller as well as our Dijkstra solver to navigate a known maze. In addition to these, we utilized the included April tags on the walls of the maze to orient ourselves while navigating and to determine our location in the maze. To do this, we used Python to both solve and track our path through the maze and to communicate/command the robot.

Block Diagram

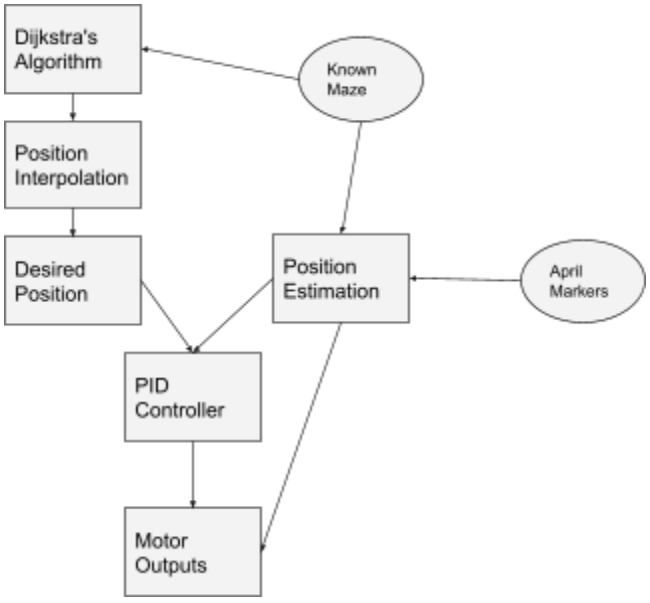

Figure 1. Architecture of our Maze Solving Algorithm

Methodology

I. Maze modifications

Starting with the maze itself, the raw data provided through this maze must be modified to give proper allowances and clearances for our robot.The first thing to do was to increase the resolution of the maze by a factor of 5 giving us 5 nodes per unit square in the real maze. This gave us more options to further modify the maze to our benefit. The primary way of doing this is through a ‘padding’ function in which all the walls are expanded by a certain number of units (we chose 4) to force our shortest path finding algorithm to steer slightly away from walls and give our robot more margin. One issue we ran into was that after giving a padding of 4, the shortest path was no longer through the middle of the maze, but instead around the top. To combat this, there was a physical

modification of the maze to block off the top route so as to keep our project in the same spirit of the lab.

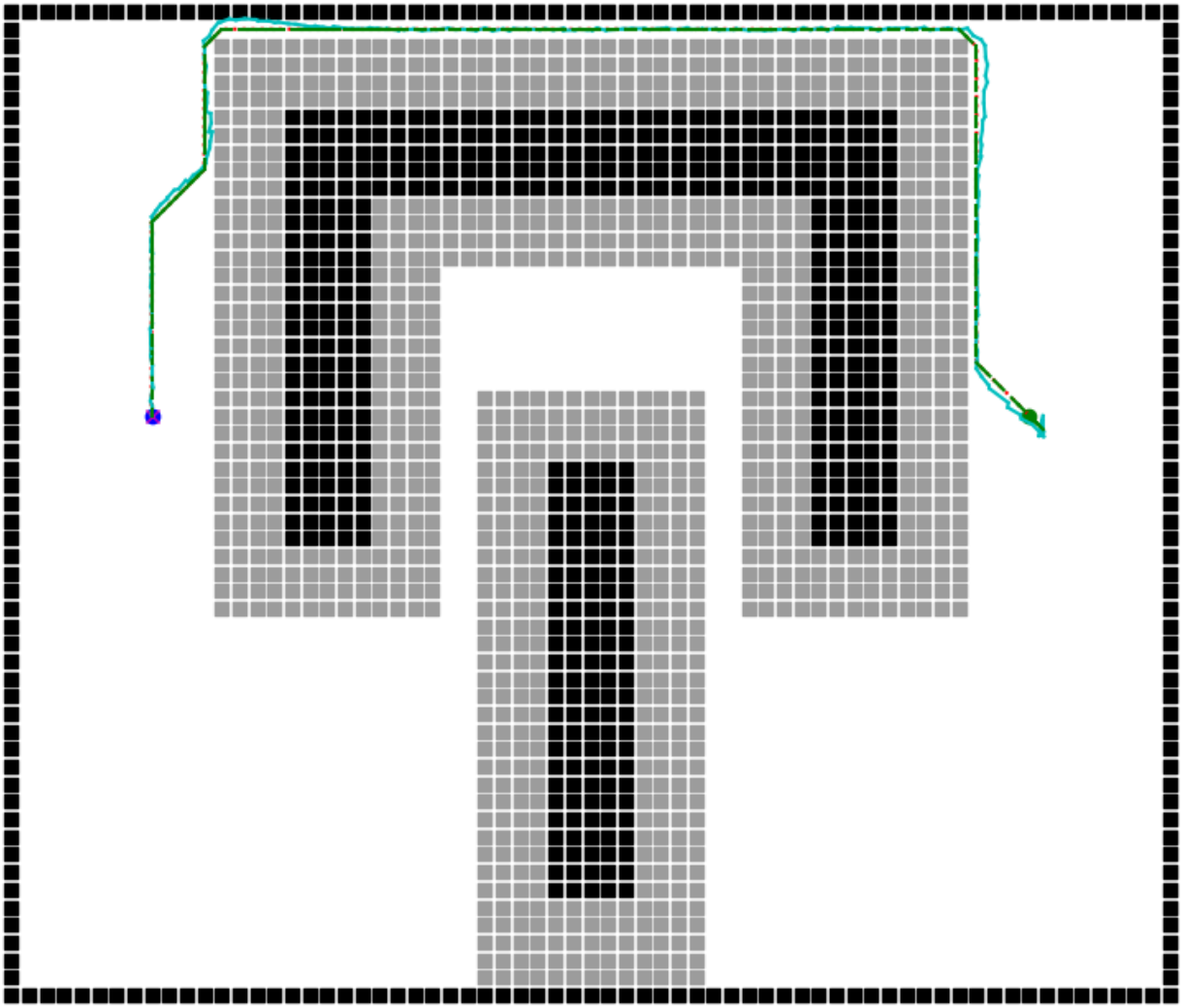

Figure 2. New Shortest Path Requiring Manual Intervention

II. Maze solver

The primary system for solving this maze was through implementing Dijkstra's algorithm with the robot sitting at the start of the maze. It uses weighted edges with values of one for edges and 1.41 for diagonals. The output of this solver is a list of tuples corresponding to the forward order of nodes required to visit.

III. Path Interpolator

Our method for following this arguably discrete and coarse path was to interpolate between points to allow us to have discrete time steps on a much smaller scale than the

factor of 5 between square units. This is done through a linearization of a set, hardcoded, timescale between nodes. This gives a set velocity of one node per unit time and is scaled so diagonals take 1.41 times as long. Doing so in the same function we use to pull our robots' real world position gives us the ability to use a pseudo tracking point that is constantly updating. This point is then used by our PID controller as it’s setpoint.

IV. PI Controller and Verification Simulator

Our PI controller uses all three parameters to provide quick response to turns and tracking errors that result in zero steady-state tracking error as can be seen in Figure 3 through the simulated controller path. This controller takes the setpoint provided by the interpolator and the robots true position converted to maze coordinates and computes the responses to the 3 degrees of freedom based on offset error. This equates to the robot tracking (from its perspective) a non-moving point at 0,0 while having variable and often nearly constant disturbances.

In order to verify our controller during testing, we used a moving point with random seeded errors added into positions each time step to provide a crude analog to the real world robot. This allowed us to quickly test and modify PI control and observe its response to the expected path.

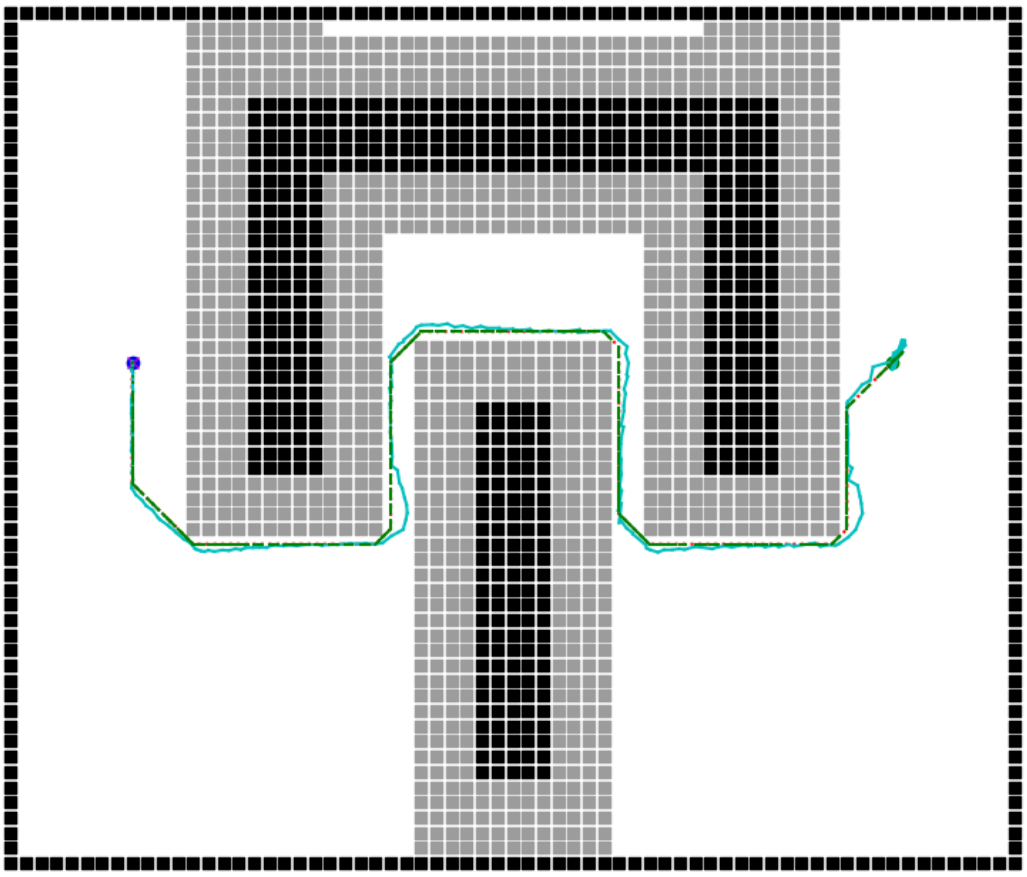

Figure 3. Simulated Run of Our PID Controller Using Random Error Input for Validation

V. World Coordinate and rotation determination

The World frame was set with an origin in the bottom left corner and the x axis along the bottom, the z axis along the side, and the y axis coming out of the paper. The April Tags positions and rotations were then recorded into a csv and a helper function was used to create a transformation matrix from the World Frame to each marker frame. Another helper function was used to take the pose from the camera to the marker and create a transformation matrix then return its inverse. This allowed the robot to create an estimation of its position and rotation in the world from any of the markers. Since there is slight warping in the camera a weighted average was used, where estimations from markers closer to the center of the camera were given a greater weight and markers past a specified threshold were ignored.

VI. Integration

The robot only had a good estimation of its position when it saw at least one April Tag. To ensure there was always one in sight a function of x was used to find the desired heading.

$$\theta_{des}(x) = \frac{\pi}{2} sin(\frac{\pi}{2}(\frac{x_{start} - x}{x_{end} - x_{start}})) - \frac{\pi}{2} \quad (1)$$

To make sure the robot completed the entire desired path, each desired point had to be reached - within a small tolerance - by the robo before the next point along the path was used. Additionally the responses from the PI controllers also had to be converted from the World Frame into x,y, and z velocities in the robot frame.

Results

As can be seen in Figure 4, our controller is capable of keeping up with the desired path but loses line of sight to any April tag and thus its reference for location determination somewhat often (seen by the gaps in the line for where the error is over .25 meters difference and the large straight lines where the error is less than .25 meters). These blackouts in the data input results in periods of time where the robot is trying to follow the line, but has a non-updating position. A quick option to remedy this in the future is to add a dead reckoning condition for tag loss, or at least until another tag can be seen and used for navigation. A more robust solution would be to implement full state estimation where the a Kalman Filter could be used to fuse position and rotation estimation from vision with orientation data from the IMU and velocity values from the motor commands.

One of the hardest hurdles for this project is the relatively tight spaces the robot is trying to fit through as the camera often loses navigational capabilities as the April tags are too close to be registered as useful markers (i.e., the start and end positions).

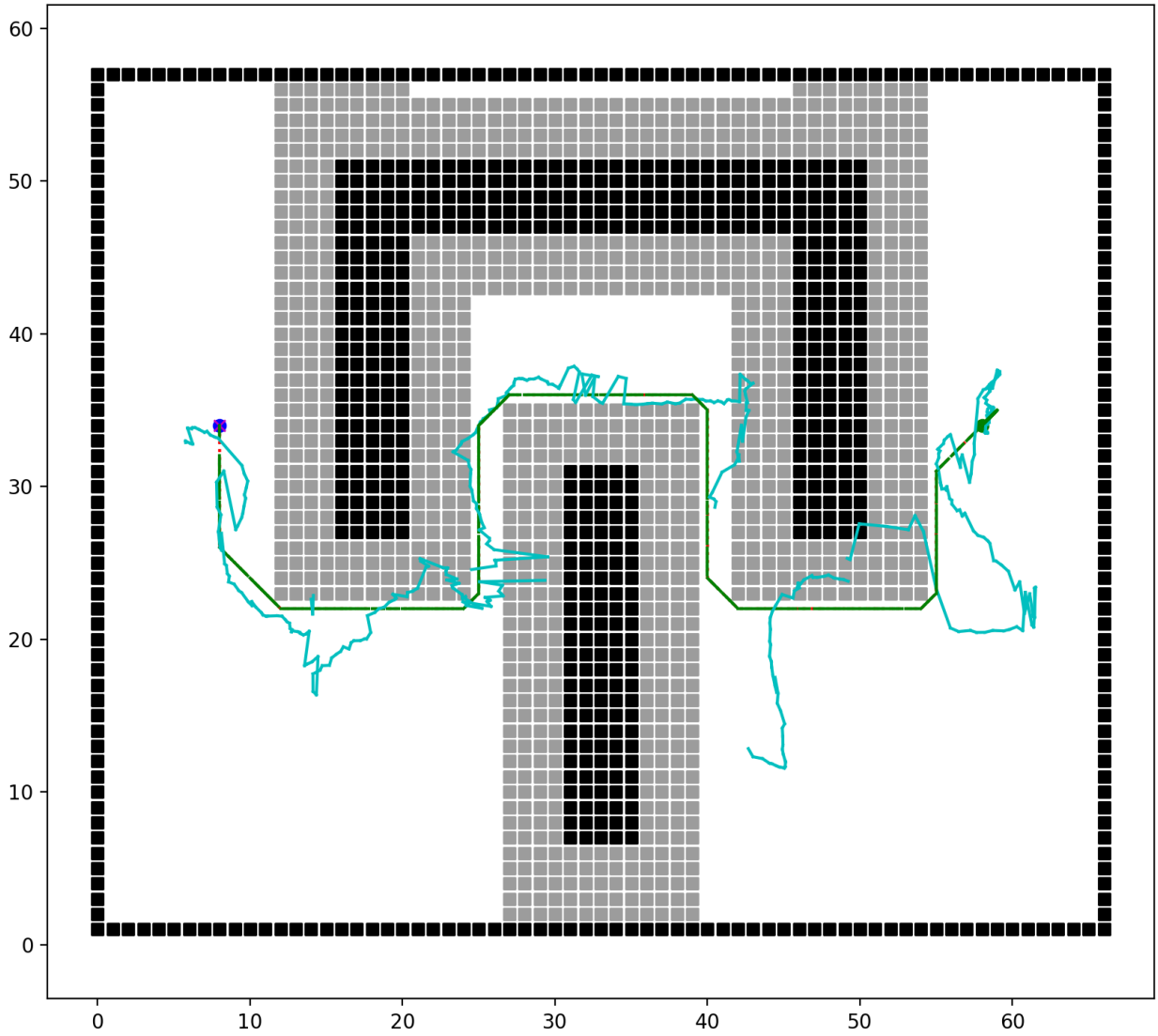

Figure 4. True Robot Navigation Path

Conclusion

Ultimately we created and implemented a functional navigation scheme that allows our robot to solve and follow the shortest path within the given maze while using April tags to determine its location in the world and maze coordinate frame. With some additional time and tweaks of our control scheme and position determination, we can make our path following algorithm significantly more accurate and even speed up the time to solve.

Code: Github Link

https://github.com/Bradd72/CMSC477/tree/main/Lab_1

CMSC 477

Video of our demonstration & Google Drive

Link to folder containing lab write-up and video of robot completing the maze → **Lab 1**