Robot Motion Planning Capstone Project

Plot and Navigate a Virtual Maze

Project Description

This project takes inspiration from Micromouse competitions, wherein a robot mouse is tasked with plotting a path from a corner of the maze to its center. The robot mouse may make multiple runs in a given maze. In the first run, the robot mouse tries to map out the maze to not only find the center, but also figure out the best paths to the center. In subsequent runs, the robot mouse attempts to reach the center in the fastest time possible, using what it has previously learned. This video (Youtube) is an example of a Micromouse competition. In this project, you will create functions to control a virtual robot to navigate a virtual maze. A simplified model of the world is provided along with specifications for the maze and robot; your goal is to obtain the fastest times possible in a series of test mazes.

Project Requirements

Requirements

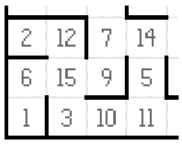
- Python > 2.7.X
- NumPy

Environment Specifications

Maze Specifications

The maze exists on an *n* x *n* grid of squares, *n* even. The minimum value of *n* is twelve, the maximum sixteen. Along the outside perimeter of the grid, and on the edges connecting some of the internal squares, are walls that block all movement. The robot will start in the square in the bottom- left corner of the grid, facing upwards. The starting square will always have a wall on its right side (in addition to the outside walls on the left and bottom) and an opening on its top side. In the center of the grid is the goal room consisting of a 2 x 2 square; the robot must make it here from its starting square in order to register a successful run of the maze.

Mazes are provided to the system via text file. On the first line of the text file is a number describing the number of squares on each dimension of the maze *n*. On the following *n* lines, there will be *n* comma-delimited numbers describing which edges of the square are open to movement. Each number represents a four-bit number that has a bit value of 0 if an edge is closed (walled) and 1 if an edge is open (no wall); the 1s register corresponds with the upwards-facing side, the 2s register the right side, the 4s register the bottom side, and the 8s register the left side. For example, the number 10 means that a square is open on the left and right, with walls on top and bottom (0*1 + 1*2 + 0*4 + 1*8 = 10). Note that, due to array indexing, the first data row in the text file corresponds with the leftmost column in the maze, its first element being the starting square (bottom-left) corner of the maze.



Robot Specifications

The robot can be considered to rest in the center of the square it is currently located in, and points in one of the cardinal directions of the maze. The robot has three obstacle sensors, mounted on the front of the robot, its right side, and its left side. Obstacle sensors detect the number of open squares in the direction of the sensor; for example, in its starting position, the robot's left and right sensors will state that there are no open squares in those directions and at least one square towards its front. On each time step of the simulation, the robot may choose to rotate clockwise or counterclockwise ninety degrees, then move forwards or backwards a distance of up to three units. It is assumed that the robot's turning and movement is perfect. If the robot tries to move into a wall, the robot stays where it is. After movement, one time step has passed, and the sensors return readings for the open squares in the robot's new location and/or orientation to start the next time unit.

More technically, at the start of a time step the robot will receive sensor readings as a list of three numbers indicating the number of open squares in front of the left, center, and right sensors (in that order) to its "next_move" function. The "next_move" function must then return two values indicating the robot's rotation and movement on that timestep. Rotation is expected to be an integer taking one of three values: -90, 90, or 0, indicating a counterclockwise, clockwise, or no rotation, respectively. Movement follows rotation, and is expected to be an integer in the range [-3, 3] inclusive. The robot will attempt to move that many squares forward (positive) or backwards (negative), stopping movement if it encounters a wall.

Scoring

On each maze, the robot must complete two runs. In the first run, the robot is allowed to freely roam the maze to build a map of the maze. It must enter the goal room at some point during its exploration, but is free to continue exploring the maze after finding the goal. After entering the goal room, the robot may choose to end its exploration at any time. The robot is then moved back to the starting position and orientation for its second run. Its objective now is to go from the start position to the goal room in the fastest time possible. The robot's score for the maze is equal to the number of time steps required to execute the second run, plus one thirtieth the number of time steps required to execute the first run. A maximum of one thousand time steps are allotted to complete both runs for a single maze.

Project Flow

Task Specifications

You can find the starter code for the project linked in an archive here. Starter code for the project includes the following files:

- `robot.py` - This script establishes the robot class. This is the only script that you should be modifying, and the main script that you will be submitting with your project.
- `maze.py` - This script contains functions for constructing the maze and for checking for walls upon robot movement or sensing.
- `tester.py` - This script will be run to test the robot's ability to navigate mazes.
- `showmaze.py` - This script can be used to create a visual demonstration of what a maze looks like.
- `test_maze_##.txt` - These files provide three sample mazes upon which to test your robot. Feel free to create your own mazes using the specifications above.

To run the tester, you can do so from the command line with a command like the following: `python tester.py test_maze_01.txt`. You should perform no modifications to `maze.py` or `tester.py`; the script you should be modifying is `robot.py`. If you create additional scripts, make sure that you include them in your submission.

Write-Up

As part of the project submission, in addition to your Python script(s), you will provide a write-up that documents your project from start to finish. Due to the nature of this project, some sections will require specific responses. Guidelines for these sections are below:

1. **Data Exploration:** Use the robot specifications section to discuss how the robot will interpret and explore its environment. Additionally, one of the three mazes provided should be discussed in some detail, such as some interesting structural observations and one possible solution you have found to the goal (in number of steps). Try to aim for an optimal path, if possible!

2. **Exploratory Visualization:** This section should correlate with the section above, in that you should provide a visualization of one of the three example mazes using the `showmaze.py` file. Your explanation in **Data Exploration** should coincide with the visual cues from this maze.

3. **Benchmark:** You will need to decide what you feel is a reasonable benchmark score that you can compare your robot's results on. Consider the visualization and data exploration above: If you are allowed one thousand time steps for exploring (run 1) and testing (run 2), and given the metric defined in the project, what is a reasonable score you might expect? There is no right or wrong answer here, but this will help with your discussion of solutions later on in the project.

4. **Data Preprocessing:** Because there is no data preprocessing needed in this project (the sensor specification and environment designs are provided to you), be sure to mention that no data preprocessing was necessary and why this is true.

5. **Free-Form Visualization:** Use this section to come up with your own maze. Your maze should have the same dimensions (12x12, 14x14, or 16x16) and have the goal and starting positions in the same locations as the three example mazes (you can use `test_maze_01.txt` as a template). Try to make a design that you feel may either reflect the robustness of your robot's algorithm, or amplify a potential issue with the approach you used in your robot implementation. Provide a small discussion of the maze as well.

6. **Improvement:** Consider if the scenario took place in a continuous domain. For example, each square has a unit length, walls are 0.1 units thick, and the robot is a circle of diameter 0.4 units. What modifications might be necessary to your robot's code to handle the added complexity? Are there types of mazes in the continuous domain that could not be solved in the discrete domain? If you have ideas for other extensions to the current project, describe and discuss them here.

Make sure that near the top of your document it clearly states that your submission is for the "Plot and Navigate a Virtual Maze" project, as there are many options for the capstone project for the Nanodegree program.

Project Evaluation

Your evaluator will review your submitted robot code and question responses using the capstone project rubric. Your robot's code will be evaluated against versions of `maze.py` and `tester.py` as provided by the starter code archive. In addition to the three test mazes provided in the starter code, the robot will be tested against three additional mazes created under the same maze specifications.

Resources and Links

Supporting Course

Artificial Intelligence for Robotics (cs373)

Additional Links

APEC Micromouse Contest Rules 2015