"Submission for Plotting and Navigating a Virtual Maze"

1. Document your work on the maze learning part of your robot. What was your approach to your robot's first run through a maze? What algorithms or techniques did you use to decide where to move the robot next? How did your robot decide to end the exploration run? Were there any approaches that did not work?

I originally started thinking based on the "Search" chapter of "AI for Robotics" where BFS, A* and DP approaches where suggested. I thought of following similar styles as we coded in the coding assignments there. I was thinking of using DP to find optimal path and store the path, so that in 2nd run I could just follow the optimal path. However, soon realised that "robot.py" has no knowledge of the full maze. It only gets the sensor values as it explores the maze and hence these approaches would not work. So any algorithm based on prior knowledge of maze will not work.

I then moved to some research and came to realise that in micro-mouse world, flood fill with many variants are popular. I first came across the paper http://web.cecs.pdx.edu/~edam/Reports/2001/DWillardson.pdf which talks about the approach. The broad approach that all such algorithms follow is to have two grids, one to learn the maze based on the sensor values as robot moves around. The initial configuration of the maze is just boundary walls with no internal walls. And second grid is used to store the most optimistic distance based on the discovered maze. As the maze initially has no internal walls, the distance grid is very optimistic. At each step, the robot uses the sensor to do follow:

1. Update the maze (the maze learned by robot and kept in its memory)
2. Do distance update – this is the place where there are multiple options
3. compute the move based on the best distance
4. repeat 1-3 until goal is hit

Step 2 – is a variation of generic flood fill approach. I first started using the brute force and then stack based algorithm as described in DWillardson's paper. However, it kept failing. My distance update would invariably land up going in a circular loop. I was not able to figure out the problem despite tracing the results.

At this point, based on some additional research, I came across a nice paper "Quantitative Comparison of Flood Fill and Modified Flood Fill Algorithms" by George Law published in International Journal of Computer Theory and Engineering, Vol. 5, No. 3, June 2013. This is available at the link "http://ijcte.org/papers/738-T012.pdf". This talks about a modified approach to stack based flood fill which offers a lesser number of iterations during the update of distance grid. The code submitted uses this approach.

Step 3 – compute the move – was done on simple logic of moving one step in the direction of the accessible neighbours with minimum distance value. In case of a tie, preference has been given to moving straight. Also in case the robot is stuck in a hole, it just turns clockwise by 90. I decided to not use the multi cell movement in first pass as the idea in first pass is to explore as much as possible. My current approach aborts the first run as soon as it reaches the goal.

Move computation, during 2nd pass also follows the same logic. Originally I tried to use multi step approach for 2nd run based on a greedy strategy of moving the cell which is accessible (as sensed by sensors) and is within a distance of 3 and has minimum distance value. However, it gave lower results for all the three mazes as compared to using one-step move approach in 2nd pass – same approach as the first one. I need to research this more. Few ideas that I have for future are:

1. During first run, after reaching goal, do more exploration (may be a fixed number of time steps). This could help discover more parts of the maze and then combined with a greedy strategy for 2nd run could offer a better performance 2nd time.
2. Instead of using a greedy strategy, use other strategies such as breadth-first-search or A* to find the shortest path to goal.

The end of exploration run was decided by checking if the robot reached the goal. Another better approach would have been to allow the robot to wonder around by moving to a neighbouring accessible cell which has minimum distance value and has not been visited before. And if this wondering leads to a situation that all accessible neighbours have already been visited, the exploration round ends. As a safety, the wondering should also finish after some fixed additional rounds after the robot has reached goal during the first run.

Q2 - Document your work on the fast-solution part of your robot. How did your robot make its second run through a maze? What algorithms or techniques did you perform? Did you have to make refinements and changes to your approach?

Already talked about it in Q1. More specifically, my idea of taking advantage of walking more than one steps based on the sensor and doing so on partially discovered maze gave me a worse running time. I tried to debug the issue and as of now my conclusion is that it is due to the reason that I terminate exploration immediately after reaching the goal. Additional exploration as described above in Q1, could lead to more maze being discovered. And then following a breadth first search for the discovered maze with the assumption that robot cannot visit the cells it did not visit in first run, could give me a faster solution. I plan to implement this some time in future.

Q3 - 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 could you make to your robot's code to handle this added complexity? Are there other extensions for this project that you can think of, and potential ways of approaching these extensions?

The current code is based the fact that there is no variability. All movements and turnings are executed perfectly. We would need to add sensors to make sure that we keep the robot in the middle of the path and some way to ensure that turns are as close as possible to being perfect. These changes are required to address the issue of a physical world.

Now moving on to a continuous world where the motion and turn are still being executed perfectly, we could still model the robot with step size of 1 unit. The other option would be to define the grid steps as 0.5 and then use a turn which is along with movement, in some kind of arc. This is assuming that robot has multiple sensors around.

Also such a model, may not need time to turn and hence preference given to moving straight could be dropped.

Another interesting maze could be a maze where the distance between neighbouring cells is not always one. Assume a grid which is 3 dimensional with some inclination along the path. Such a model could mean that algorithm to update a cell distance based on neighbouring cells may factor some value other than one.

Yet another interesting variation could be trying to find the goal without getting found by guard-robots who are also on prowl in the maze and who have full knowledge of the maze. Though I am not sure how to model such a problem. May be using some kind of adversarial on-line search could be the way to go. Seems like an interesting problem to think. Would be glad to have some pointers in this regard.