Planning and decision making in robotics: 16782
Homework -1
Write up

1. Approach for solving this homework:
    a. Used vanilla A* algorithm with 8 connected grid heuristic (as discussed in class): $\min(\Delta x,\Delta y)\sqrt{2} + [\max(\Delta x,\Delta y) - \min(\Delta x,\Delta y)]$
    b. Goal position changes dynamically. We look ahead few steps to decide the goal state because planning takes time in which the target would have already moved. How much we look ahead also changes based on the previous path length (to be exact, by number of steps it takes between two replans).
    c. We also replan occasionally and not every time planner function is called. Replanning happens after robot covers half the length of the previous trajectory to the previous goal. This seems reasonable so that the robot doesn't go towards the outdated goal in case this outdated goal is in a different direction than previous goal. All the times replanning doesn't occur, we just step onto next in our previous calculated path.
    d. c(s,s') is modified from the values given in the map because otherwise planner takes a lot of time. So if the map value is greater than 2000 then, I keep it to be 50 and otherwise it is 1. I played with these values and going lower didn't help too much with reducing cost value of the maps but it reduced the speed of running code a lot.
    e. I also tried weighted A* but results were too variable and not robust with increasing and decreasing weight value and highly depends on the environment
    f. I also tried using backward A* as a heuristic but it takes more time as compared to the above heuristic and therefore (even though more accurate as it compensates for the obstacles) was not used.
    g. Note: these values have been played with and I am sure we can come up with some analytical meaning behind those values but it was out of scope for me to think about it for this assignment.
    h. Run code using:

        mex -O CFLAGS="\$CFLAGS -std=c++11" planner.cpp aStar.cpp

        In matlab, just run runtest file like you would.
2. Data Structures used:
    a. Sets have been used for openSet which needs to be prioritized according to custom defined comparator (in astar.hpp) which first prioritizes according to f-value, then x coordinate, and then y coordinate. This was chosen because it is feasible to search this set unlike priority queue which in my knowledge cannot be searched directly since all the elements cannot be accessed.
    b. ClosedSet does not need to be ordered and hence an unordered set was used which uses hash function to store the keys which makes it very fast (logarithm) to search in.
    c. Other data structures include vector, struct and class

3. Heuristics used:
    a. Implemented euclidean and 8 connected grid heuristic (see 1a)
4. Efficiency tricks:
    a. Consistent use of pointers. Passing by reference. Unordered Set - storing coordinates as strings as they are unique.
    b. Static variables in the class and planner function (so that I don't have to plan every time planner is called)
5. Memory Management:
    a. Use of pointers and releasing nodes from sets before termination so that there is no memory leak
    b. Least use of static variables in the class
    c. No nested for loops
6. Results:

Map 1:

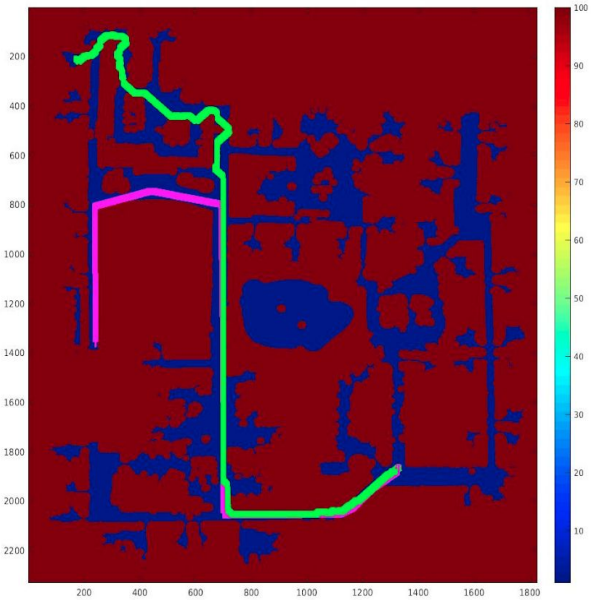
RESULT:

target caught = 1

time taken (s) = 2958

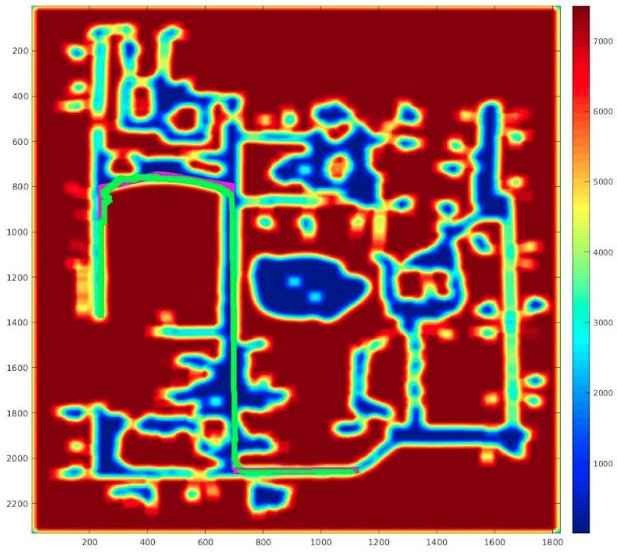
moves made = 2948

path cost = 2958



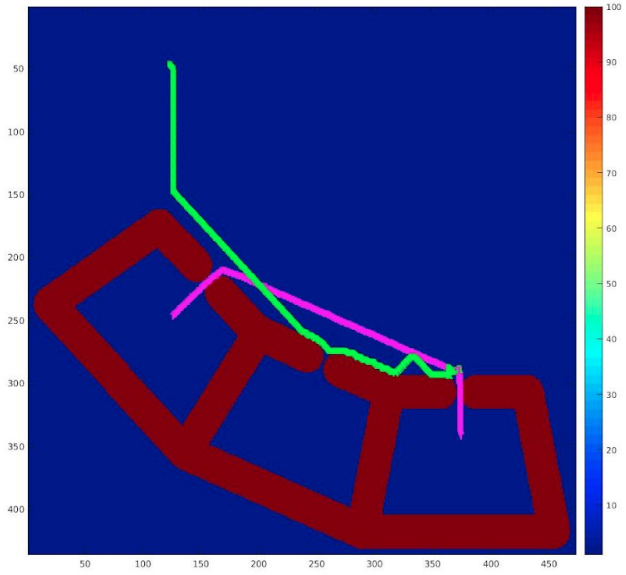
Map 2:
RESULT:

target caught = 1
time taken (s) = 2603
moves made = 2602
path cost = 3872825



Map 3:
RESULT:

target caught = 1
time taken (s) = 359
moves made = 358
path cost = 359



Map 4:

RESULT:

target caught = 1

time taken (s) = 431

moves made = 372

path cost = 59420