

Homework I:

Planning for a robot trying to catch a target

DUE: Feb 21st (Tue) at 11:59PM

Description:

In this project, you are supposed to write a planner for the robot trying to catch a target. The planner should reside in `planner.c` file (or `robotplanner.m` file if you prefer to write it in matlab). Currently, the file contains a greedy planner that always moves the robot in the direction that decreases the distance in between the robot and the target.

The planner function (inside `planner.c`) is as follows:

```
static void planner(  
    double* map,  
    int x_size,  
    int y_size,  
    int robotposeX,  
    int robotposeY,  
    int goalposeX,  
    int goalposeY,  
    char *p_actionX,  
    char *p_actionY  
)  
{
```

The map of size `x_size`, `y_size` contains information on what are obstacles and what are not. It should be accessed as:

```
(int)map[GETMAPINDEX(x,y,x_size,y_size)].
```

If it is equal to 0, then the cell $\langle x,y \rangle$ is free. Otherwise, it is an obstacle. Note that all coordinates start with 1. In other words, `x` can range from 1 to `x_size`. The current robot pose is given by $\langle \text{robotposeX}, \text{robotposeY} \rangle$ and the current target pose is given by $\langle \text{goalposeX}, \text{goalposeY} \rangle$.

The planner function should return what the robot should do next using the pointers `p_actionX` and `p_actionY`. Specifically, `*p_actionX` should be set to the desired change in X (-1, 0, or 1) and `*p_actionY` should be set to the desired change in Y (also, -1, 0, or 1). The robot is allowed to move on an 8-connected grid. All the moves should be valid with respect to obstacles and map boundaries (see the current planner inside `planner.c` for how it tests the validity of the next robot pose).

The planner is supposed to produce the next move within 1 second. Within that 1 second, the target also makes one move. But the target can only move in four directions. If the planner

takes longer than 1 second to plan, then the target will move by longer distance. In other words, if the planner takes N seconds (rounded up) to plan the next move of the robot, then the target will move by N steps in the meantime.

The directory contains few map files (map1.txt and map3.txt).

Here are few examples of running the tests from within the Matlab (in the same directory where all source files are placed):

To compile the C code:

```
>> mex planner.c
```

To run the planner

```
>> robotstart = [10 10];  
>> targetstart = [200 100];  
>> runtest('map3.txt', robotstart, targetstart);
```

Same map but more difficult to catch the target

```
>> robotstart = [250 250];  
>> targetstart = [400 400];  
>> runtest('map3.txt', robotstart, targetstart);
```

Much larger map and more difficult to catch the target

```
>> robotstart = [700 800];  
>> targetstart = [850 1700];  
>> runtest('map1.txt', robotstart, targetstart);
```

Executing runtest command multiple times will show that sometimes the robot does catch the target, and sometimes it does not. The letters R and T indicate the current positions of the robot and target respectively. (Sometimes, they may appear as if they are on top of a boundary of an obstacle, but in reality they are not. The letters are just much bigger than the actual discretization of the map.)

NOTE: to grade your homework and to evaluate the performance of your planner, I may use different and larger maps than the ones I provided in the directory, and a different strategy for how the target moves than the one in targetplanner.m. **The only promise I can make is that the target will only move in four directions and the size of the map will not be larger than 5000 by 5000 cells.**

To submit:

Submit three files by sending them to me via email(maxim@cs.cmu.edu):

1. robotplanner.m
2. planner.c (and other .h files you created that I need to compile)
3. ASCII file (.txt) with 1-2 paragraphs describing the approach you took for the planner (i.e, what algorithm and with what parameters).

NOTE: I should be able to just run `mex planner.c` to compile your code from within Matlab under Windows!

Grading:

The grade will depend on two things:

1. How well-founded the approach is. In other words, can it guarantee completeness (to catch a target), can it provide sub-optimality or optimality guarantees on the paths it produces, can it scale to large environments, can it plan within 1 second?
2. How fast (and how consistently) it catches the target