

16-782, Fall'17, Planning and Decision-making in
Robotics

Homework I:

Planning for a robot trying to catch a target DUE: Sept 27th (Wed) at 11:59PM

Description:

In this project, you are supposed to write a planner for a non-holonomic 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 but I would highly recommend using c/c++ for this homework). 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,  
    float robotposeX,  
    float robotposeY,  
    float robotposeTheta,  
    float goalposeX,  
    float goalposeY,  
    PrimArray mprim,  
    int *prim_id  
)  
{
```

The map of discrete 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 $\langle \text{robotposeX}, \text{robotposeY}, \text{robotposeTheta} \rangle$ and the current target pose is given by $\langle \text{goalposeX}, \text{goalposeY} \rangle$. Note that the target is only two dimensional whereas the robot state has three dimensions.

The planner function should return what the robot should do next using the `*prim_id` which is the index of the motion primitive with respect to the motion primitives array `PrimArray`. Specifically, the robot has 5 motion primitives defined with respect to a grid of resolution 0.1 units. It implies then 1 cell (discrete space) corresponds to 0.1 units (continuous space). Note that the map is also defined on the same resolution e.g an `m`x`n` size map in discrete space

corresponds to $(m*0.1 \times n*0.1)$ units in continuous space. These primitives are: small step forward, long step forward, forward and left turn, forward and right turn and small step backward. The robot can have 8 discrete orientations and the PrimArray would contain 5 primitives for each of these 8 orientations. Corresponding to one of these primitives, the returned pointer should have a value ranging from 0 to 4 (1 to 5 for Matlab code). The robot is allowed to move on this so called lattice graph and hence is constrained to move according to the predefined motion primitives. 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 (in discrete space). But the target can only move in four directions on the 2-d grid. 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 you need to provide the start and goal locations (in continuous space) and the map.

```
>> robotstart = [1 1 0];  
>> targetstart = [20 10 0];  
>> runtest('map3.txt', robotstart, targetstart);
```

Same map but more difficult to catch the target

```
>> robotstart = [25 25 0];  
>> targetstart = [40 40 0];  
>> runtest('map3.txt', robotstart, targetstart);
```

Much larger map and more difficult to catch the target

```
>> robotstart = [70 80 0];  
>> targetstart = [85 170 0];  
>> 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. The current visualization would not depict the orientation of the robot).

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 contained in a .zip folder with the name “hw1_andrewid” by sending them to me via email(fahad.islam@fulbrightmail.org).

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

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

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