Homework 5b

ME 570 — Prof. Tron

2020-11-20

In this homework, you will implement a variation of PRM, one of the sampling-based methods seen in class. Given the fact that the graph search algorithm was already implemented in Homework 4, for this homework you will mainly need to implement the sampling functions. If the assignment mentions a file not specifically provided, it refers to a file from a previous assignment (in fact, you will need functions from Homework 2, 3 and 4).

General instructions

Programming For your convenience, together with this document, you will find a `zip` archive containing Matlab files with stubs for each of the questions in this assignment; each stub contains an automatically generated description of the function and the function header. You will have to complete these files with the body of each function requested. The goal of this files is to save you a little bit of time, and to avoid misspellings in the function or argument names. The functions from the parts marked as **provided** (see also the *Grading* paragraph below) contain already the body of the function.

Please refer to the guidelines on Blackboard under Information/Programming Tips & Tricks/Coding Practices.

Homework report Along the programming of the requested functions, prepare a small PDF report containing one or two sentences of comments for each question marked as **report**, and including:

- Embedded figures and outputs that are representative of those generated by your code.
- All analytical derivations if required.

You can include comments also on the questions marked as **code**, if you feel it necessary (e.g., to explain any difficulty you might have encountered in the programming), but these will not be graded. In general, *do not* insert the listing of the functions in the report (I will have access to the source files); however, you can insert *short* snippets of code if you want to specifically discuss them.

A small amount of *beauty points* are dedicated to reward reports that present their content in a professional way (see the *Grading criteria* section in the syllabus).

Analytical derivations To include the analytical derivations in your report you can type them in LaTeX(preferred method), any equation editor or clearly write them on paper and use a scanner (least preferred method).

Submission

The submission will be done on Blackboard. Please upload a single ZIP file with your name, and containing both the PDF of the report, and all the code necessary to reproduce the figures in the paper.

Optional and provided questions. Questions marked as optional are provided just to further your understanding of the subject, and not for credit. Questions marked as provided have already a solution provided in the file accompanying this assignment, which you can directly use unless you want to answer it by yourself. If you submit an answer for optional or provided questions I will correct it, but it will not count toward your grade.

Hints

Some hints are available for some questions, and can be found at the end of the assignment (you are encouraged to try to solve the questions without looking at the hints first). If you use these hints, please state so in your report (your grading will not change based on this information, but it is a useful feedback for me).

Use of external libraries and toolboxes All the problems can be solved using Matlab's standard features. You are **not allowed** to use functions or scripts from external libraries or toolboxes (e.g., mapping toolbox), unless specifically instructed to do so.

Problem 1: Probabilistic Road-Maps (PRM)

The goal of this question is to implement a basic version of PRM. The version discussed below generates the samples and tries to connect them incrementally; another possible version would first generate all the samples first, and then generate the roadmap later (combinations of the two approaches are also possible).

We will use the sphere world from Homework 3 for testing. Please refer to that homework for a detailed description of the `world` data structure.

Question provided 1.1. This function implements a collision check against all spheres in the world.

> [`flag`]=`sphereworld_isCollision`(`world,x`)
>
> Description: Checks if the points with coordinates `x` is inside any of the spheres in `world`. This is done by iterating over all spheres, using the function `sphere_distance`( ) from Homework 3 on each one of them, and then combining the results.
>
> Input arguments
>
> - `world` (dim. [`NSpheres` × `1`], type `struct array`): see definition in Homework 3.
> - `x` (dim. [`2` × `NPoints`]): array with the coordinates of the sampled locations (one for each column) that need to be tested for collisions.
>
> Output arguments
>
> - `flag` (dim. [`1` × `NPoints`], type `logical`): `flag(iPoint)` is `true` if `x(:,iPoint)` is inside one of the spheres (i.e., it is a collision), `false` otherwise.

Question provided **1.2.** Create a function that samples from the free configuration space using rejection sampling.

[`xSample`]=`sphereworld_sample`(`world,...`)

Input arguments

- `world` (dim. [NSpheres × 1], type `struct array`): see definition in Homework 3.

Optional arguments

- `'distribution',distribution` : Selects the type of distribution to be used. Possible values for `distribution` are 'uniform' and 'Gaussian'.
- `'size',sz` : Size parameter for the distribution (default value: `sz=10`). For the `'uniform'` distribution, the resulting domain of the random variable is the square [− `maxSize`, `maxSize`]^2. For the `'Gaussian'` distribution, `sz` represents the variance.
- `'mean',mu` (dim. [2 × 1]): Mean of the distribution (default value: `mu=[0;0]`).

Output arguments

- `xSample` (dim. [2 × 1]): a sample in the free space. Can be empty if a sample cannot be generated in `nbTrials=1000` trials.

Description: Generate samples using the specified distribution, until `sphereworld_isCollision`( ) returns `false` for a sample, or a maximum number of trials `nbTrials` is reached.

Question code **1.1.** This function samples locations along a line between two samples, and checks if this line can be used to create an edge in the roadmap.

[`flag`]=`prm_localPlannerIsCollision`(`world,x1,x2,maxDistEdgeCheck`)

Description: Generates `NPoints` equispaced points on the line between `x1` and `x2`, such that the maximum distance between two consecutive samples is less than `maxDistEdgeCheck`. Note that the value of `NPoints` is determined by the distance between the two points and `maxDistEdgeCheck`.

Input arguments

- `world` (dim. [NSpheres × 1], type `struct array`): see definition in Homework 3.
- `x1` (dim. [2 × 1]), `x2` (dim. [2 × 1]): coordinates of the endpoints of the line on which to sample the locations.
- `maxDistEdgeCheck` (dim. [1 × 1]): maximum distance between sampled locations.

Output arguments

- `flag` (dim. [1 × 1], type `logical`): `true` if *any* of the sampled locations is in collision, `false` otherwise.

Requirements: This function returns an "all-or-nothing" result, meaning that the edge between `x1` and `x2` even if only one of the samples is in collision.

Question **code** **1.2.** The following function adds a single sample to the roadmap.

[`graphVector`]= `prm_addSample` (`graphVector,xSample,kNeighbors,world`)

Description:

1) Uses `graph_nearestNeighbors` ( ) that was provided in Homework 4 to find the `kNeighbors` nearest neighbors in the graph. You can start with `kNeighbors=3` , but you might want to test different values (see also Question report 1.1).
2) Adds a node at location `xSample` to `graphVector` .
3) Iterates over the neighbors using `prm_localPlannerIsCollision` ( ) to check if there is an edge between `x` and each neighbor. If an edge is found, it is added to `graphVector` : *remember to modify the entries relative to both* `x` *and the neighbor to keep the graph symmetric, and to also update the corresponding values of the field* `neighborCost` *for both vertices.* You can start with `maxDistEdgeCheck=0.1` , but you might want to test different values (see also Question report 1.1).

Input arguments

- `graphVector` (dim. [NNodes × 1], type `struct`): the structure describing the tree to extend, as specified in Homework 4.
- `xSample` (dim. [2 × 1]): location of the sample to add.
- `kNeighbors` : number of neighbors to consider for the sample.
- `world` (dim. [NSpheres × 1], type `struct array`): see definition in Homework 3.

Output arguments

- `graphVector` (dim. [NNodes+1 × 1], type `struct`): same as the corresponding input argument, but with an additional node (unless the extension fails).

Requirements: We first look for neighbors, and then add the sample, in order to avoid finding the sample as a neighbor of itself.

Question **code** **1.3.** Now it is time to implement the part of PRM that builds the roadmap.

[`graphVector`]= `prm_buildGraph` (`world,NSamples`)

Description: This function performs the following steps:

1) Initialize the `graphVector` structure as an empty vector.
2) Uses the function `sphereworld_sample` ( ) to sample a random location `xSample` in the free space. Remember to handle the (very unlikely) case in which the sampling fails.
3) Uses `sphereworld_isCollision` ( ) on `xSample` . If `x` is not a collision, use

prm_addSample ( ) to add that sample.

4) Iterates until graphVector contains NSamples, or has sampled a maximum of 1000 locations.

Input arguments

- world (dim. [NSpheres × 1], type struct array): see definition in Homework 3.
- NSamples (dim. [1 × 1]): desired number of vertices in

optional Note that the function as described above might add vertices that are not connected to the roadmap. If you want, you can add an option to modify the steps above in order to prevent this from happening. How do you think it would change the behavior of the algorithm?

Question report 1.1. In this question you will test prm_buildGraph ( ) and try different values of NSamples.

prm_buildGraph_test ( )

Description: This function performs the following steps:

1) Loads world from sphereworld.mat.
2) Calls prm_buildGraph ( ) the graphVector for a roadmap.
3) Uses sphereworld_plot ( ) from Homework 2 to visualize the sphere world.
4) Uses the provided function graph_plot ( ) from Homework 4 to superimpose the roadmap in graphVector on the world.

Try different values of NSamples in the call to prm_buildGraph ( ), until you find a value high enough such that the roadmap reliably represents the topology of the space. You can also experiment here with different values of kNeighbors and maxDistEdgeCheck to use in the calls to graph_nearestNeighbors ( ) and prm_addSample ( ), respectively.

Question code 1.4. This question implements the second part of PRM, which uses the roadmap built by the function in the previous question.

[xPath]=prm_search(graphVector,world,xStart,xEnd)

Description: This function performs the following two steps:

1) Use prm_addSample ( ) to add two nodes to the graph graphVector corresponding to xStart and xEnd (note that here you don't necessarily have to use the same value of kNeighbors).
2) Use graph_search ( ) from Homework 4 to find and return a path from xStart to xEnd.

Input arguments

- graphVector (dim. [NVerticesTotal × 1], type struct): a struct array containing the roadmap in the format previously described.

- `world` (dim. [`NSpheres` × `1`], type `struct array`): see definition in Homework 3.
- `xStart` (dim. [`2` × `1`]), `xEnd` (dim. [`2` × `1`]): the coordinates of the initial and final locations.

Output arguments

- `xPath` (dim. [`2` × `NPath`]): array where each column contains the coordinates of the points of the path found from `idxStart` to `idxEnd`.

This function and its specification above are very similar to those in Question 3.5 from Homework 4 (`visibility_search`( )).

Question report 1.2. In this section you will test your PRM planner.

`prm_search_test`( )
Description:

1) Loads the variables `world`, `xStart` and `xEnd` from the file `sphereWorld.mat`.

2) Calls `prm_buildGraph`( ) to build a probabilistic roadmap representation of the environment.

3) For each starting location (column) in `xStart`, and each goal location (column) in `xGoal`, calls `prm_search`( ) to find a path.

4) Calls `sphereworld_draw`( ) and then overimposes all the trajectories from the start locations to the goal location.

Question optional 1.1. The path returned by PRM will likely be quite jagged. Implement a function [`xPathSmoothed`]= `path_smoother` (`xPath`) that implements a method of your choice for simplifying the path (you can look in the book for options).

Question optional 1.2. Modify and run `prm_search_test`( ) so that they run the respective planners multiple times on the same data, and show how the variance of the length of the paths found across different trials.

Problem 2: Homework feedback

Question report 2.1. Indicate an estimate of the number of hours you spent on this homework (possibly broken down by problem). Explain what you found hard/easy about this homework, and include any suggestion you might have for improving it.

Hint for question code 1.1: To generate the samples, you can use the function `line_linspace`( ) from Homework 2, with `a=x2-x1` and `b=x1` (what should you use for `tMin` and `tMax`?)

Hint for question code 1.2: If the graph is empty, you can just add the sample without any check.