Homework 5a

ME 570 - Prof. Tron

2020-11-20

In this homework you will implement a roadmap based on a visibility graph.

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: Visibility graph roadmap

In this problem we will consider a world with four polygons. More in detail, we will use a array of structures `world` with dimension [`NPolygons` $\times$ 1] with two fields:

- `vertices` (dim. [2 $\times$ `NVertices`]): the vertices of a polygon using the convention of Homework 1.
- `idxVertices` (dim. [1 $\times$ `NVertices`]): `indexes` containing a unique index for each vertex in each polygon. These indexes will become the indexes of the vertex in a `graphVector` structure for the visibility graph.

Explicit values for this structure are given in the file `polygonWorld.mat`. This data represents an approximation of the sphere world used in Homework 3. As such, the first polygon in `world` is an hollow polygon.

You will first implement basic functions to plot the polygons in `world`.

Question provided 1.1. Modify the function `polygon_plot(_)` from Homework 1 to accept a third, optional argument `indexes` (dim. [1 $\times$ `NVertices`]) which contains an index number for each vertex in the polygon. If this argument is passed, display the indeces must be displayed beside the corresponding vertices.

Question provided 1.2. Implement a function to visualize a polygon world

> `polygonworld_draw(world,xGoal)`
> Description: Uses `polygon_plot(_)` from Homework 1 to draw the polygonal obstacles together with a `*` marker at the goal location.
> Input arguments
> - `world` (dim. [`NPolygons` $\times$ 1], type `struct array`), `xGoal` (dim. [2 $\times$ 1]): same as defined for the `polygonWorld.mat` file.
>
> Requirements: The function should check if any of the endpoints of `edge` is inside the polygon, or if `edge` is in collision with any of the edges of the polygon.

Use this function to visualize the contents of the file.

Question **code** **1.1.** Make a function that checks for collisions between an edge and a polygon.

> `[flagIsCollision]=polygon_isCollisionEdge(edge,vertices)`
>
> Description: Uses the function `[flagIsCollision]=edge_isCollision(_)` from Homework 1 to check whether an edge is in collision with a polygon.
>
> Input arguments
>
> - `edge` (dim. `[2 × 2]`): coordinates of the vertices defining an edge (i.e., a line segment).
> - `vertices` (dim. `[2 × NVertices]`): coordinates of vertices defining a polygon.
>
> Output arguments
>
> - `flagIsCollision` (, type `logical`): `true` if the edge intersects with the polygon, `false` otherwise.

Question **code** **1.2.** Implement a function that checks the visibility between a point and every vertex in the polygon world.

> `[idxVisibleVertices]=visibility_isVisible(world,x)`
>
> Description: Returns the indexes of vertices in `world` that are visible from the point `x`. In practice, your function should iterate over polygon, and perform the following for each vertex in the current polygon:
>
> - Check for self-occlusions using `polygon_isSelfOcclusion(_)` with respect to the current polygon.
> - Check for edge-intersections using `polygon_isCollisionEdge(_)` and iterating over all the edges defined by `world`.
> - If both checks do not reveal occlusions, the index corresponding to the vertex should be added to `idxVisibleVertices` (using the information from `world.indexes`).
>
> Input arguments
>
> - `world` (dim. `[NPolygons × 1]`, type `struct`): struct array containing the polygonal world as described above.
> - `x` (dim. `[2 × 1]`): a point in the environment (which could correspond to a vertex of one of the polygons as a particular case).
>
> Output arguments
>
> - `idxVisibleVertices` (dim. `[NVisibleVertices × 1]`): an array containing the list of vertices in the world visible from `x`, identified by their corresponding index specified in the field `indexes` of `world`

Question **provided** **1.3.** Make a function to test the world visibility function.

`visibility_isVisible_test(_)`
Description:

1) Load the structure `world` and the variable `xStart` from the file `polygonalWorld.mat`

2) For each for each point (column) x in `xStart`:

- Open a new figure.
- Call `polygonworld_draw(_)` to display the polygonal world.
- Call `visibility_isVisible(_)`.
- Plot lines between x and every visible vertex.

Requirements: For this function, it is helpful to first concatenate all the arrays `vertices` and `indexes` into a single one, so that it is easy to map the indexes returned in `idxVisibleVertices` to the actual coordinates.

Question report **1.1.** Include the five plots generated by `visibility_isVisible(_)` in your report.

Question code **1.3.** Implement a function that creates a visibility roadmap in the form of a `graphVector`

`[graphVector]=visibility_graph(world)`
Description: Uses `visibility_isVisible(_)` to build a `graphVector` structure representing the visibility of each vertex in `world` with respect to all the others.
Input arguments

- `world` (dim. `[NPolygons × 1]`, type `struct`): struct array containing the polygonal world as described above.

Output arguments

- `graphVector` (dim. `[NVerticesTotal × 1]`, type `struct`): A struct array, of the format previously described, describing a visibility graph. Use the indexes contained in `world().indexes` to establish the mapping between vertices in the world and elements of `graphVector`.

Question code **1.4.** Implement a planner based on the visibility roadmap.

`[xPath]=visibility_search(graphVector,xStart,xGoal,world)`
Description: This function performs the following steps:

1) optional Check if `xEnd` is visible from `xStart`. In this case, return `xPath=[xStart xEnd]` and exit.

2) Use `visibility_isVisible(_)` to add two nodes to the graph `graphVector` corresponding to `xStart` and `xGoal`. **Important note:** In addition to adding two elements to `graphVector`, the function will have to modify the `neighbors`

and `neighborsCost` fields that are already in the `graphVector` array to keep the graph symmetric.

3) Use `graph_search(_)` to find and return a path from `xStart` to `xGoal`.

Input arguments

- `graphVector` (dim. [`NVerticesTotal` × 1], type `struct`): a struct array, of the format previously described, describing a visibility graph.
- `xStart` (dim. [2 × 1]), `xGoal` (dim. [2 × 1]): the coordinates of the initial and final locations.
- `world` (dim. [`NPolygons` × 1], type `struct`): struct array containing the polygonal world as described above.

Output arguments

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

Question report **1.2.** Test your visibility-roadmap planner.

`visibility_search_test(_)`
Description:

1) Loads the variables `world`, `xStart` and `xGoal` from the file `polygonalWorld.mat`.

2) Calls `visibility_graph(_)` to build a visibility graph representation of the environment.

3) For each starting location (column) in `xStart`, calls `visibility_search(_)` to find a path to `xGoal`.

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

Include the output of your function in your report.

Question report **1.3.** The last question is similar to the sphere world problem you encoutered in Homework 3 (the polygonal environment is a crude approximation of the continuous spheres, but the start and goal locations are identical). Comment on the relation between the results obtained from the two problems in the two homework sets.

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 1.2: This function is very similar to `polygon_isVisible(_)`, except that `visibility_isVisible(_)` needs to check for collisions with *all* edges in the world, not only the current polygon.