Potential Functions

This document will guide you through the practical work related to path planning algorithms based on potential functions. This lab has two parts:

- The first one consists on programming an attraction function (i.e., a *wave planner*) and finding the path from any starting position to the goal.
- The second part consist on implementing a *bushfire* algorithm and with it compute a *repulsive function*. To check that the repulsive function is working, you need to normalize both the attraction and the repulsive functions and merge them together (i.e., add them). Doing that, it will be possible to solve the path planning problem while keeping a certain distance with respect to the obstacles.

All the code has to be programmed in Python.

Grid map environment

We are going to use grayscale images to define our grid map environments. In Python, the `PIL` library can be used to load a map image and the `matplotlib` library can be used to show it. Transform the image to a 2D `numpy` array to simplify its manipulation. Note that when a grayscale image is used as an environment, 0 is black and it is used to represent an obstacle while 1 (or 255) is white and it is normally used to represent the free space. Therefore, we need to binarize the loaded image to ensure that there are no intermediate values as well as to invert the values to have 0 as open space and 1 as obstacles, that is the *standard* definition of *free* and *occupied* space in motion planning.

Python code snip

```python
import numpy as np
from matplotlib import pyplot as plt
from PIL import Image

# Load grid map
image = Image.open('map0.png').convert('L')
grid_map = np.array(image.getdata()).reshape(image.size[0],
image.size[1])/255
# binarize the image
grid_map[grid_map > 0.5] = 1
grid_map[grid_map <= 0.5] = 0
# Invert colors to make 0 -> free and 1 -> occupied
grid_map = (grid_map * -1) + 1
# Show grid map
plt.matshow(grid_map)
plt.colorbar()
plt.show()
```

The following maps with the proposed *start* and *goal* loacations are provided:

grid map name	start	goal
map0.png	(10, 10)	(90, 70)
map1.png	(60, 60)	(90, 60)
map2.png	(8, 31)	(139, 38)
map3.png	(50, 90)	(375, 375)

Part 1

Attraction function: Wavefront planner

An attraction function creates an attraction potential from any point in the `gridMap` to the `goal`. This potential field can be the distance between each cell in the `gridMap` to the `goal` taking into account the obstacles in the environment. You can use the Euclidean distance (better solution & higher mark):

$$D_{Euclidean} = \begin{bmatrix} 2.8 & 2.4 & 2. & 2.4 & 2.8 \\ 2.4 & 1.4 & 1. & 1.4 & 2.4 \\ 2. & 1. & Goal & 1. & 2. \\ 2.4 & 1.4 & 1. & 1.4 & 2.4 \\ 2.8 & 2.4 & 2. & 2.4 & 2.8 \end{bmatrix}$$

or a 4-Point/8-Point connectivity function (easier to implement & lower mark).

$$D_4 = \begin{bmatrix} 4. & 3. & 2. & 3. & 4. \\ 3. & 2. & 1. & 2. & 3. \\ 2. & 1. & Goal & 1. & 2. \\ 3. & 2. & 1. & 2. & 3. \\ 4. & 3. & 2. & 3. & 4. \end{bmatrix} \quad D_8 = \begin{bmatrix} 2. & 2. & 2. & 2. & 2. \\ 2. & 1. & 1. & 1. & 2. \\ 2. & 1. & Goal & 1. & 2. \\ 2. & 1. & 1. & 1. & 2. \\ 2. & 2. & 2. & 2. & 2. \end{bmatrix}$$

Use the `wavefront` algorithm to compute the attraction function. This algorithm is detailed in the book *Principles of Robot Motion, Howie Choset et al.*, and a pseudocode is shown next:

```
wavefront_planner_connect_4(map, goal)
  motions = [left, right, up, down]
  value = 2
  map[goal] = value
  queue = [goal]
  while queue not empty
    value++
    new_queue = []
    for p in queue
      for m in motions
        if isValid(p + m, map)
          map[p + m] = value
          new_queue.push(p + m)
        endIf
      endFor
    endFor
```

```
        queue = new_queue
    endWhile
    return map
```

The isValid function checks that the position p + m is inside the map and the value of map[p + m] do not corresponds to an obstacle.

The result of this algorithm is a new map (a 2D matrix) with the same size than the original grid map in which each cell (or pixel) contains the distance to the goal taking into account the obstacles.

An example of execution is shown next. In this map 0's represents free space, 1's are obstacles and the 2 indicates the goal position:

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 1. 1. 1. 1. 0. 0. 0.]
[0. 0. 1. 1. 1. 1. 1. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 2. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Running the wavefront planner with connectivity 4, the following attraction function is obtained:

```
[17. 16. 15. 14. 13. 12. 11. 10. 11. 12.]
[16. 15. 14. 13. 12. 11. 10.  9. 10. 11.]
[17. 16.  1.  1.  1.  1.  1.  8.  9. 10.]
[16. 15.  1.  1.  1.  1.  1.  7.  8.  9.]
[15. 14.  1.  4.  3.  4.  5.  6.  7.  8.]
[14. 13.  1.  3.  2.  3.  4.  5.  6.  7.]
[13. 12.  1.  4.  3.  4.  5.  6.  7.  8.]
[12. 11.  1.  5.  4.  5.  6.  7.  8.  9.]
[11. 10.  1.  6.  5.  6.  7.  8.  9. 10.]
[10.  9.  8.  7.  6.  7.  8.  9. 10. 11.]
```

Finding the path

Once the attraction function is implemented, finding a path from any cell to the goal is very simple. Wherever the robot starts, it moves to the neighbouring cell that has the smallest distance to the goal. The process is repeated until the robot reaches the goal.

In some potential field functions it is possible to get trapped in some local minima. Find a solution to avoid getting trapped.

Exercise

Implement the following algorithms:

- `wavefront` planner: Given a `grid_map` as an image and a `goal` position, the output of this algorithm is an `attraction function` that takes into account the obstacles in the environemt.
- `find_the_path`: Given an `attraction function` and a `start` position, the algorithm returns the minimum `path` from `start` to `goal`.

Part 2

Bushfire algorithm

Potential fields motion planning algorithms use attraction fields to move from the starting point to the goal, as we have seen previously, but also repulsive fields to avoid passing close to any obstacle. The **bushfire** is one of the algorithms that can be used to easily compute the distance to an obstacle. Imagine that your grid map is defined as follows, being *0s* the free space and *1s* the obstacles:

$$grid\_map = \begin{bmatrix} 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 1. & 1. & 1. & 1. & 0. & 0. \\ 0. & 1. & 1. & 1. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. \end{bmatrix}$$

The result of applying the `bushfire` algorithm to this grid map is:

$$bushfire = \begin{bmatrix} 3. & 3. & 3. & 3. & 3. & 3. & 3. \\ 2. & 2. & 2. & 2. & 2. & 2. & 3. \\ 2. & 1. & 1. & 1. & 1. & 2. & 3. \\ 2. & 1. & 1. & 1. & 1. & 2. & 3. \\ 2. & 2. & 2. & 2. & 2. & 2. & 3. \\ 3. & 3. & 3. & 3. & 3. & 3. & 3. \\ 4. & 4. & 4. & 4. & 4. & 4. & 4. \end{bmatrix}$$

> Notice that here we do not use the Euclidean distance to simplify the computations (diagonals are at distance 1 instead of $\sqrt{2}$). You are free to improve that.

Repulsive Function

Thanks to the **bushfire** algorithm we know the distance from any cell (q) to an obstacle ($D(q)$). To compute a repulsive function we can use the following equation where Q is the desired radius of repulsion with the obstacles.

$$rep(q) = \begin{cases} 1 & D(q) = 1 \\ 4 \cdot (\frac{1}{D(q)} - \frac{1}{Q})^2 & D(q) \leq Q \\ 0 & D(q) > Q \end{cases}$$

Then for each pixel in the resulting matrix after applying the *bushfire* algorithm it is possible to compute the above repulsive function. This will produce a repulsive function in which free areas will be seen as valleys and obstacles as mountains. The zones between obstacles and free space will have a slope defined by Q. Note that this is an exponential function and therefore, the *repulsive effect* decreases very quickly.

Combining the Attraction and Repulsive functions

Repulsive functions can be used in combination with an Attraction function. To combine them it is important to normalize their values:

- In the original Attraction function obstacles have value 1, the goal has value 2 and the other cells have a value greater than 2. Replace the obstacles value by the maximum value in the Attraction function + 1 and then, normalize the function between 0 (for the goal) and 1 (for the obstacles).
- The Repulsive function should be already normalized between 0 (farthest than Q from any obstacles) and 1 (obstacles).

Once normalized, is it possible to add both functions together. Then, run the already implemented *finding the path* algorithm on it to find a path that not only guides the robot to the goal but also that keeps it far from the obstacles.

Exercise

Implement the following algorithms:

- `bushfire` algorithm: Given a `grid_map` as an image returns a `map` with the distance of each cell to the closest obstacle. You can use connect-4, connect-8 or Euclidean distance to compute this distance.
- `repulsive_function`: Given the output of the `bushfire` algorithm and a radius of repulsion Q, it returns a `repulsion function`.
- `potential_function`: Normalize the attraction and repulsive function previously implemente and combine them. Once done find the path that goes from start to goal and plot it.

Submit

Once all these algorithms are implemented and tested individually, create a scripts following this interface:

```
$ ./potential_function_YOUR_NAME.py path_to_grid_map_image start.x start.y
goal.x goal.y Q
```

where

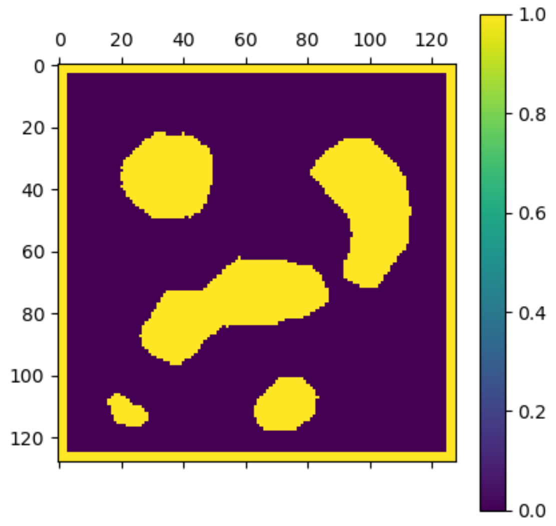
- `path_to_grid_map_image` is the path to the image that contains the grid map environmet.
- `start` is a 2D array containing `[start_x, start_y]` values that correspond to the pixel position in which the path has to start.
- `goal` is a 2D array containing `[goal_x, goal_y]` values that correspond to the pixel position in which the path has to end.
- `Q` is the radius of repulsion with the obstacles.

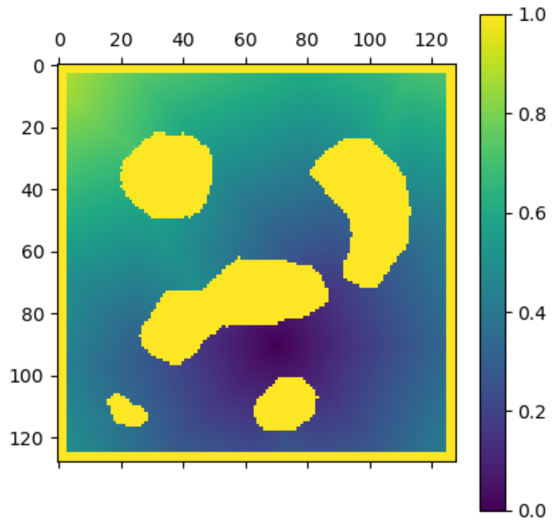
The output of this Python script has to be the path (list of x, y positions) and the following images:

- Attraction function
- Path from start to goal using only the Attraction function
- Bushfire
- Repulsive function
- Potential function (i.e., Attraction function + Repulsive function normalized)
- Path from start to goal using the combined Attraction and Repulsive functions

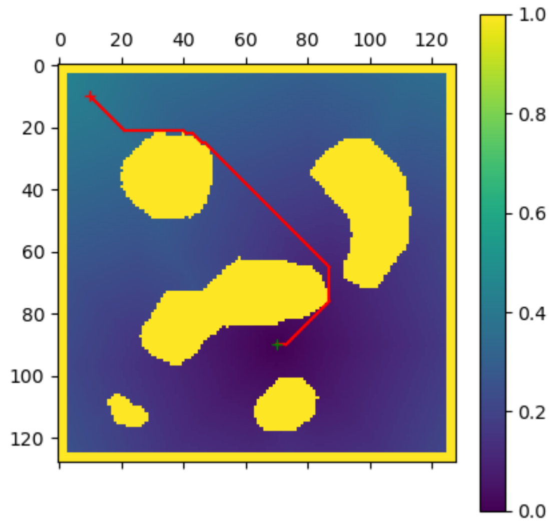
An execution example is shown next:

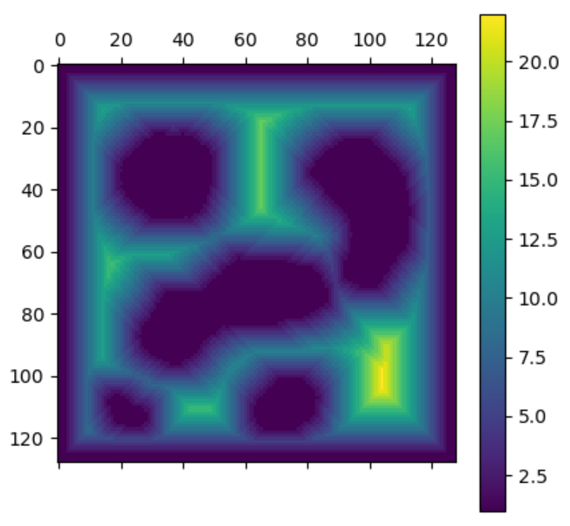
```
python potential_function_NARCIS.py ./map0.png 10 10 90 70 5

path = [(10, 10), (11, 11), (12, 12), (13, 13), (14, 14), (15, 15), (16,
16), (17, 17), (18, 18), (19, 19), (20, 20), (21, 21), (21, 22), (21, 23),
(21, 24), (20, 25), (20, 26), (21, 27), (20, 28), (19, 29), (18, 30), (18,
31), (18, 32), (18, 33), (18, 34), (18, 35), (18, 36), (18, 37), (18, 38),
(18, 39), (18, 40), (19, 41), (19, 42), (20, 43), (20, 44), (20, 45), (21,
46), (22, 47), (23, 48), (23, 49), (24, 50), (25, 51), (26, 52), (27, 53),
(28, 54), (29, 55), (30, 56), (31, 57), (32, 58), (33, 59), (34, 60), (35,
61), (36, 62), (37, 63), (38, 64), (39, 65), (40, 66), (41, 67), (42, 68),
(43, 69), (44, 70), (45, 71), (46, 72), (47, 73), (48, 74), (49, 75), (50,
76), (51, 77), (52, 78), (53, 79), (54, 80), (55, 81), (56, 82), (57, 83),
(58, 84), (59, 85), (60, 86), (61, 87), (62, 87), (63, 87), (64, 87), (65,
88), (66, 88), (67, 87), (68, 88), (69, 88), (70, 89), (71, 89), (72, 89),
(73, 90), (74, 89), (75, 89), (76, 89), (77, 89), (78, 89), (79, 88), (80,
87), (81, 86), (82, 85), (83, 84), (84, 83), (85, 82), (86, 81), (87, 80),
(88, 79), (89, 78), (90, 77), (90, 76), (90, 75), (90, 74), (90, 73), (90,
72), (90, 71), (90, 70)]
```



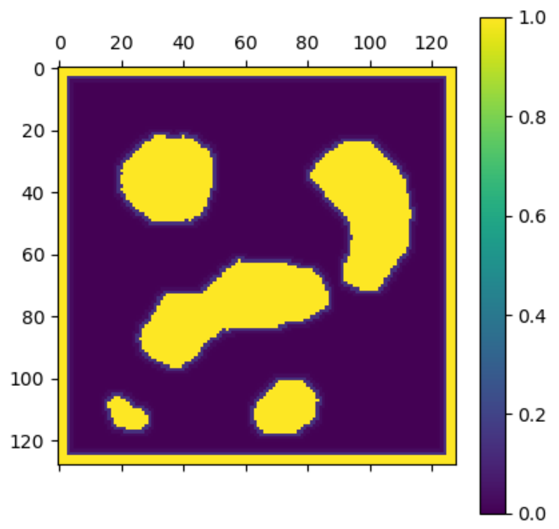


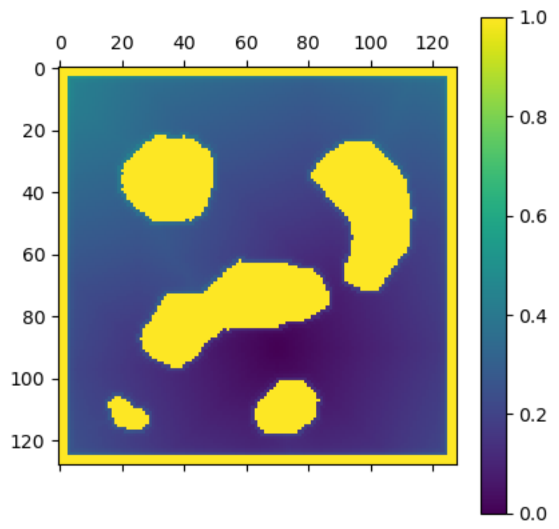
Original Grid map	Attraction function

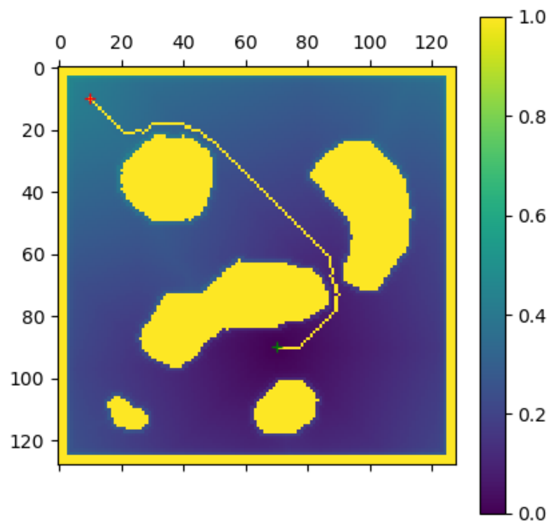




Path on Attraction function	Bushfire







Path on final Potential function

Submit a report in PDF and the Python script (only one file!!!). The script must have the name: `potential_functions_YOUR_NAME.py`. In the report, explain in detail, and with graphical information, the work done in all sections, show diferent execution examples, discuss about the (e.g., best valu of `Q` for a particular environment, diferences between Euclidean and connect-8/4 distance, ...). Explain also the problems you found. You might want to test your algorithm using other environments. **BE SURE** that your script follows the defined interface!

WARNING:

We encourage you to help or ask your classmates for help, but the direct copy of a lab will result in a failure (with a grade of 0) for all the students involved.

It is possible to use functions or parts of code found on the internet only if they are limited to a few lines and correctly cited (a comment with a link to where the code was taken from must be included).

Deliberately copying entire or almost entire works will not only result in the failure of the laboratory but may lead to a failure of the entire course or even to disciplinary action such as temporal or permanent expulsion of the university. Rules of the evaluation and grading process for UdG students.

Narcís Palomeras Last review May 2022.