

Task 1: Configuration Space (40 points)

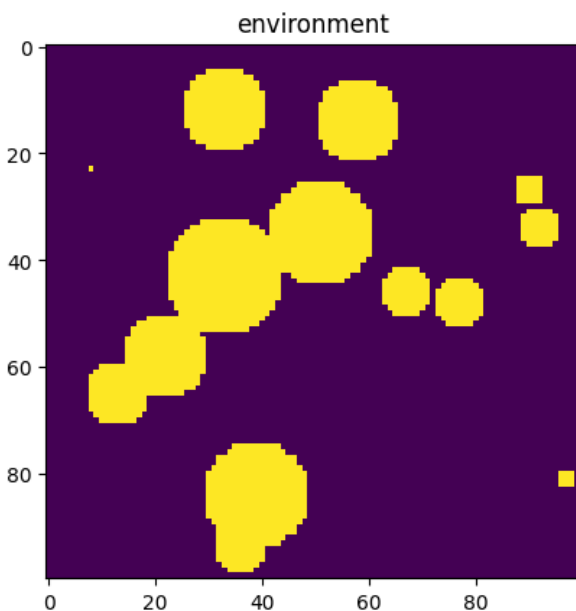
For this task, you will write a Python script or note-book that generates the configuration space, including the obstacle regions and free space. The Workspace is a rectangular grid area (image) of 100×100 , such that

$$W = 0, 1, \dots, 99 \times 0, 1, \dots, 99 \subset \mathbb{Z}^{0+} \times \mathbb{Z}^{0+}.$$

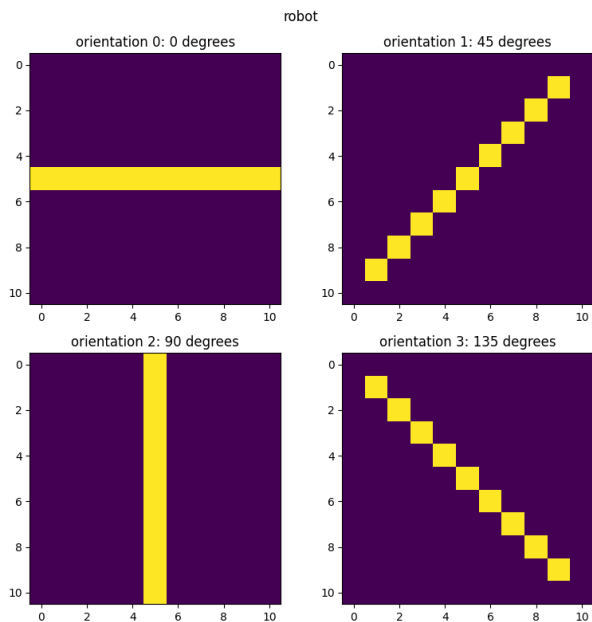
The robot is a “rod” object, whose configuration is $q = [x, y, \theta]$, taking into account that x, y are the image coordinates, therefore they are integer values in the grid and the orientation θ is also discretized into several values. All the required data can be obtained from data_ps1.npz. This numpy data file consists of the environment map `['environment'](np.ndarray([100,100]))` an image with values 0 to indicate free space and 1 to indicate an obstacle. The second component is the robot/agent, whose shape is included in `['rod'](np.ndarray([11,11,4]))`. There are 4 dimensions to represent the different discretization orientations - the robot rotated by 0, 45, 90, 135 degrees positions.

A (10 pts) Visualize from the given data the workspace and the different rod configurations for each discretized orientation. Comment on the given discretized values for orientation.

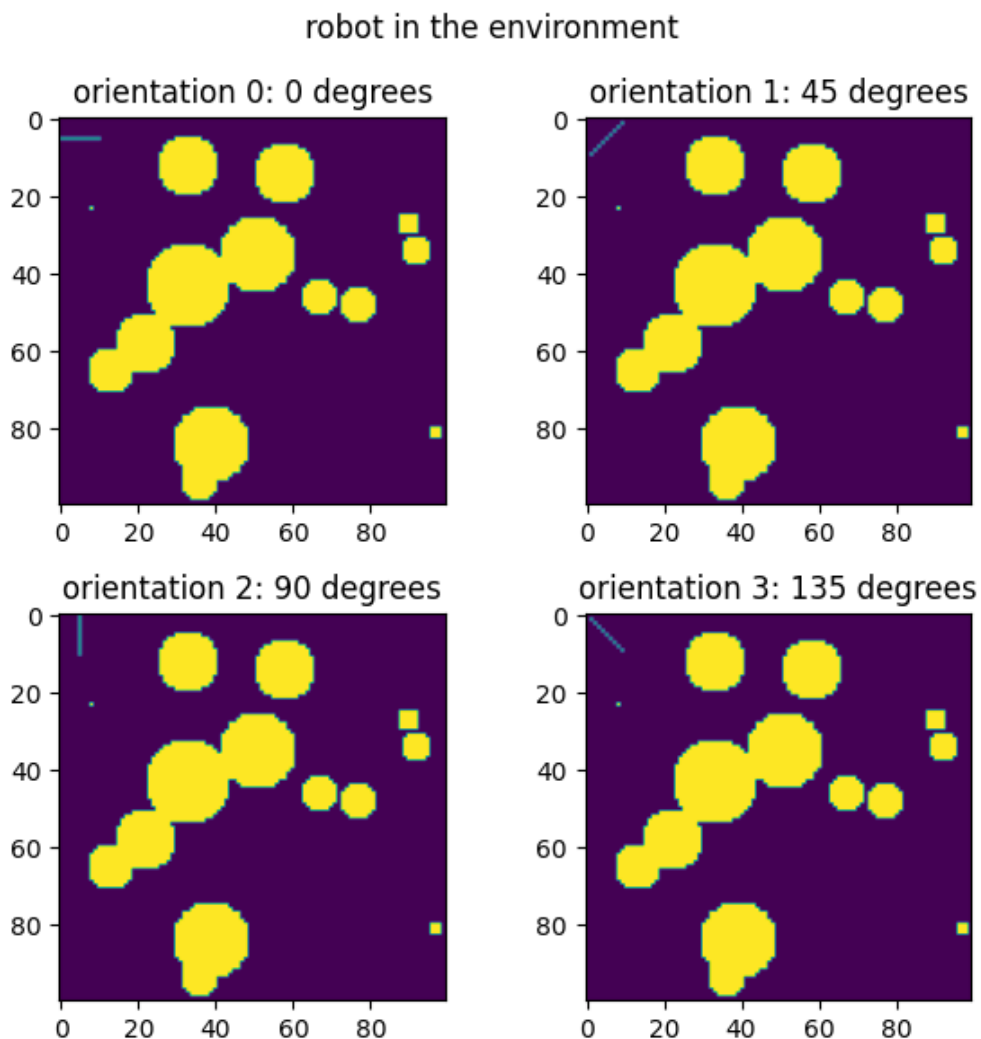
Workspace:



rod onfigurations:



B (10 pts) Visualize the environment together with the object. For this, you may want to use the function `plot_environment` from *utils.py* and select any valid configuration value for the rod.



C (10 pts) Create the C-space for the 2D environment map. For this, plot all the images corresponding to each of the orientations by using collision checking.

hint: you might want to look at the library `scipy.signal` and use function

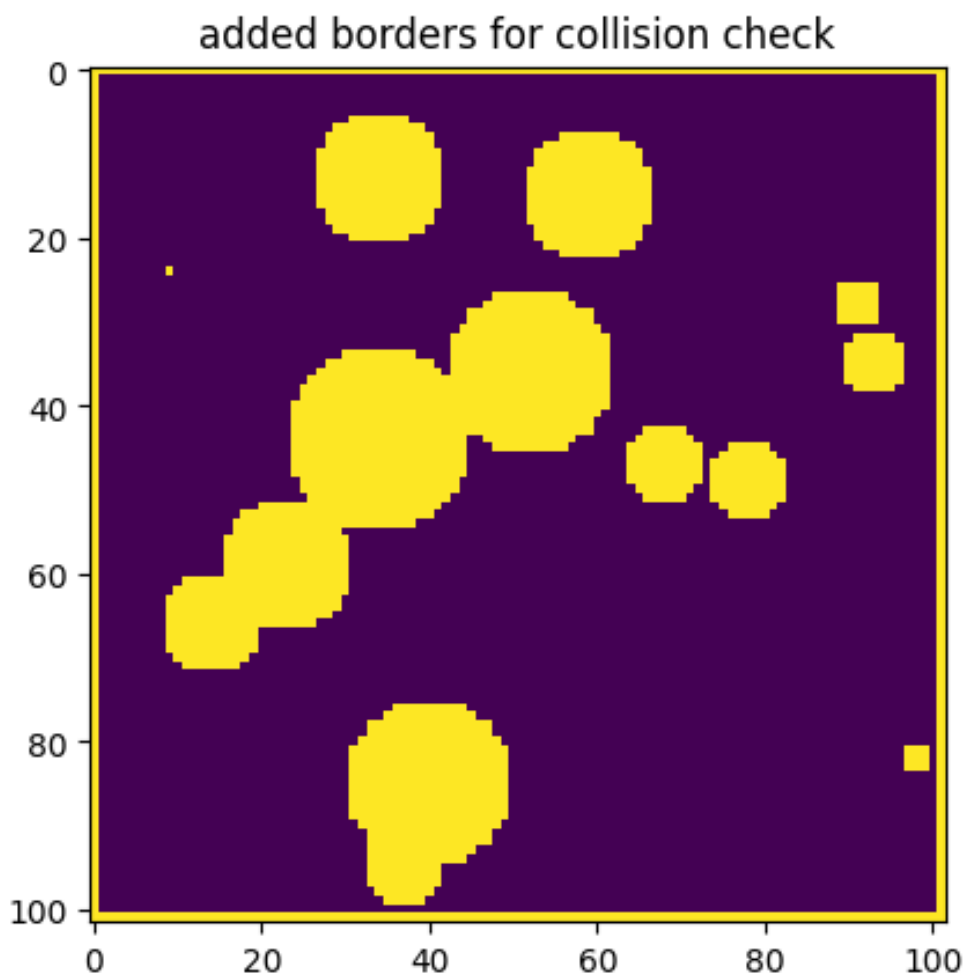
```
signal.convolve2d(env_map, kernel, boundary='symm', mode='same')
```

to check for collisions.

hint: you may want to use `normalize_image(img)` from *utils.py* to normalize created space to $[0, 1]$, since after convolutions, values are not exactly 0 and 1 (this will be useful for task 2).

I started with addition of borders to the environment with

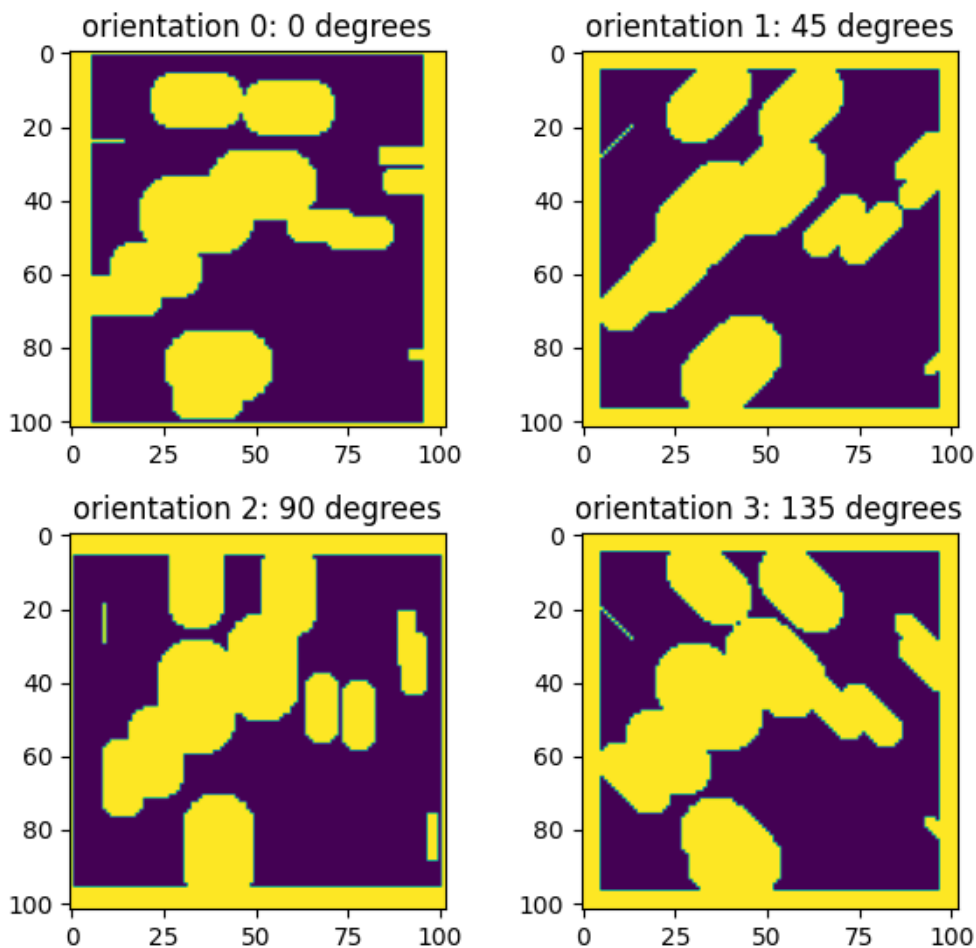
```
np.pad(environment, pad_width=1, constant_values=1)
```



Then I convolved it with `rod` as a kernel and normalized:

```
convolved_env = convolve2d(environment_edged, kernel,  
                             mode='same', boundary='symm')  
normalize_image(convolved_env)
```

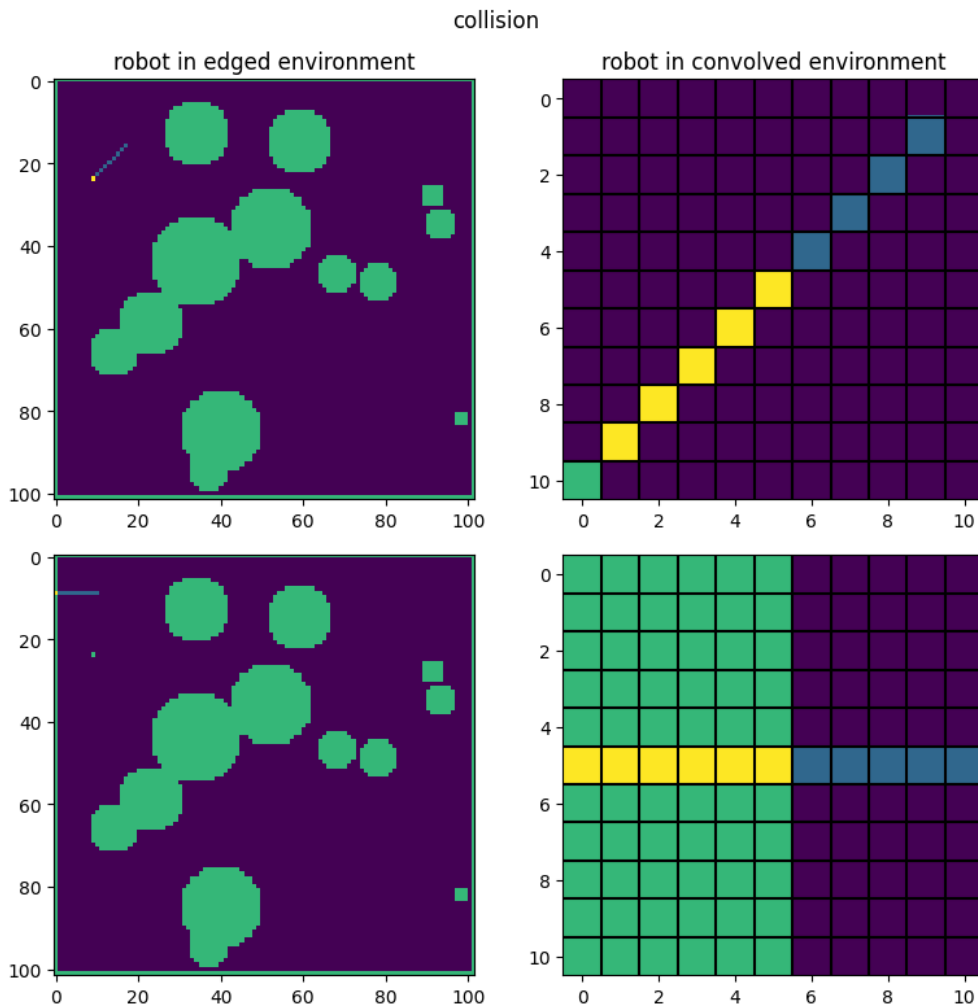
environment convolved with rod array for prediction of collision



D (10 pts) Comment on the obtained C-space with the previous method. What is the size of the C-space?

The size of C-space (called `check_collision_envs` in my case) is (102,102,4) due to padding for borders.

We can check the work of convolved environment with a few examples:



on the left we see that **collision** has occurred and on the right we see that in convolved environment the center of the rod has intersected with boundaries. So now to check whether robot collided or not we just need to check whether the center of the robot has intersected with boundaries or not.

Task 2: A star Algorithm (60 points)

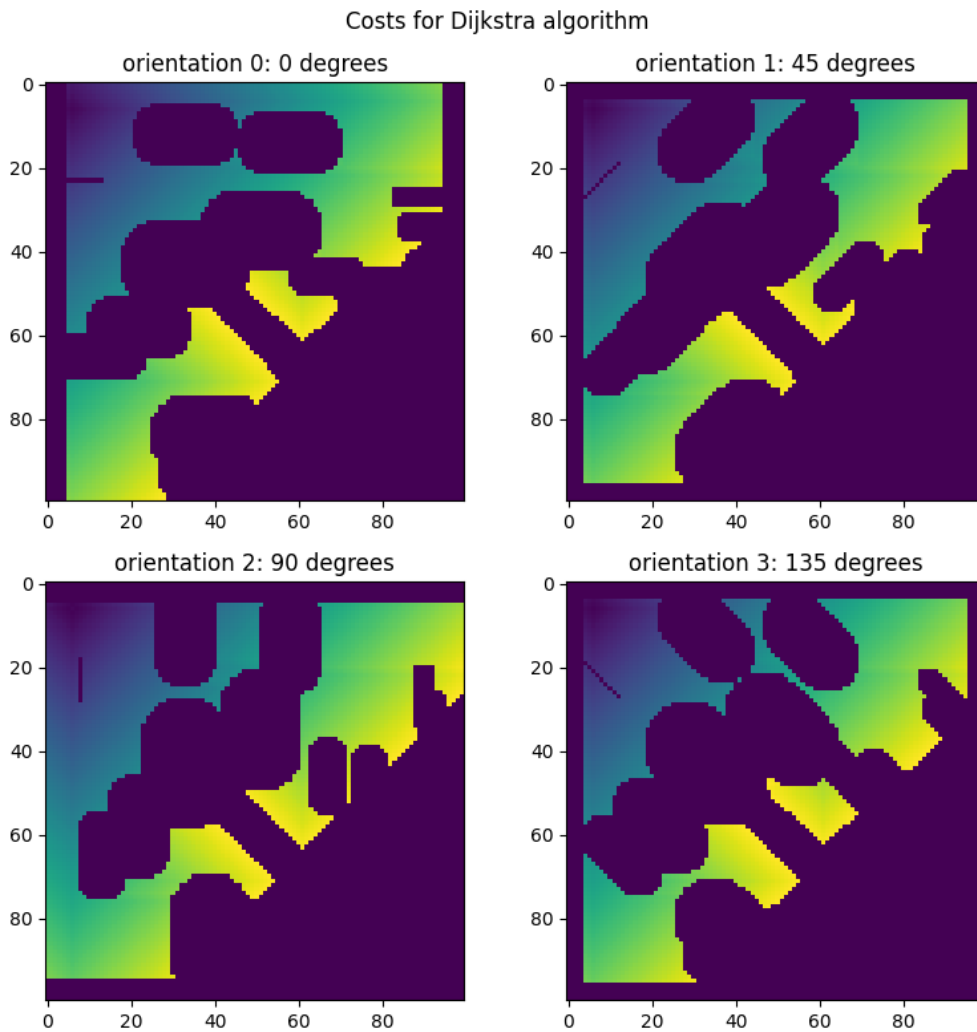
For this task, you will implement a graph search algorithm. The actions allowed in this problem are [moving up, down, left, right, rotate right, rotate left]. In total 6 actions, each of them has an assigned a cost of 1.

A (40 pts) You need to implement the A star algorithm and plan in the generated discrete C-space from the previous task.

The starting configuration of the agent is (6, 6, 2) and the goal configuration is (55, 55, 0). On this first iteration, use an heuristic function $h(q, q_G) = 0$, which is equivalent to the **Dijkstra algorithm**. Save the result of calculated plan in **rod_solve.mp4** using `plotting_result(environment, rod, plan)` from *utils.py*, where plan is list of rod states from start to goal.


hint: Track the number of visited states to avoid/debug potential issues with internal loops.

The goal was reached by the robot in 117 number of steps after 13628 cycles with Dijkstra algorithm. Below you can see the costs of the states explored by the algorithm.



B (10 pts) Change the heuristic function now to be $h(q, q_G) = L1$ norm of the x, y components. Comment on the changes, how many states have been visited compared to Dijkstra? What is the final cost? Comment on the results.

```
def h_l1(x,y, goal=goal):
    return int(np.abs(x-goal[0])+np.abs(y-goal[1]))
```

The goal was reached by the robot in 117 number of steps after 5607 cycles with A algorithm, L1 norm. *The number of cycles is around 50% less than for Dijkstra algorithm. Below you can see the costs of the states explored by the algorithm.*


C (10 pts) Propose an heuristic function $h(q, q_G)$ that includes orientation. Compare metrics with the previous results. Comment on the results

```
def h_l1_(x,y,z, goal=goal):
    return int(np.abs(x-goal[0])+np.abs(y-goal[1])+np.abs(z-
goal[2]))%3)
```

I tried to use the same l1 norm as in the previous task but I added difference along z axis too. I used %3 because if 3 rotations are needed (i.e. from orientation 3 to orientation 0) then the same can be done in 1 rotation. In the end, it didn't change anything - nor did it change heuristic values, nor did it change the total number of cycles:

The goal was reached by the robot in 117 number of steps after 5607 cycles with A* algorithm and modified L1 norm.

Costs for A*_ algorithm

