Problem_1

March 15, 2020

Problem 1: Simulation

1 Introduction

To solve the questions in this problem, I buit a simulator that simulate the movement of the robot. The simulator is more general, modular, and can handle different scenarios.

In teh directory Modules, the two important classes are `class Robot` and `class World`. The first, defines fully the robot and its dynamics, the second defines the environnement and handles the collisions and the clock.

The robot dynamic is assumed to be:

$$\dot{x} = v\cos(\theta(t))$$
$$\dot{y} = v\sin(\theta(t))$$

while θ is the heading control that is chosen randomely if we detect a collision and $v = .1m/s$ is the robot velocity that we assume is constant. To implement this dynamic, the finite difference descretization is used. This later results into:

$$x(t+\Delta t) = x(t) + \Delta t v\cos(\theta(t))$$
$$y(t+\Delta t) = y(t) + \Delta t v\sin(\theta(t))$$

where, now, Δt is the time step descretization chosen $\sim 0.9 sec$.

The environnement is defined in the class World. Basically, any environment has a rectangular shape that defined by its boundaries and characterize by its bottom_left and top_right corner edges. Similiray, it is possible to add an arbitrary number of rectangular obstacles, defined by there boundaries. Then, when the environnement starts to play the clock is increased by Δt amount at each step and the robot is moved using the dynamics explained above. If a collision is detected, a new heading angle, that we call controller is sampled from a unifrom distribution $\theta \sim \mathbb{U}(0, \frac{\pi}{2})$. Note, that this is an open loop controller.

Throughout this problem, we will use the Monte-Carlo simulation to compute the different values. Recall that, by the Law of Large Numbers, an estimmator of the mean can be computed as follow:

$$\tilde{g}_N = \frac{1}{N}\sum_{i=0}^{N} g(x_i)$$

such that $\lim_{N\to\infty} \tilde{g}_N = \mathbb{E}(g)$

Lastely, to estimate the coverage proportion, we need to descretize the space as well. Therefore, the `class World` contains an instance of the `class Grid`. This later, handles the update of each cell after the robot passes by that cell. Hence, computing the proportion of the updated cells to the number of the whole cells gives the coverage proportion.

1.1 Part A : Possible path that the robot could take after 5 minutes of operation.

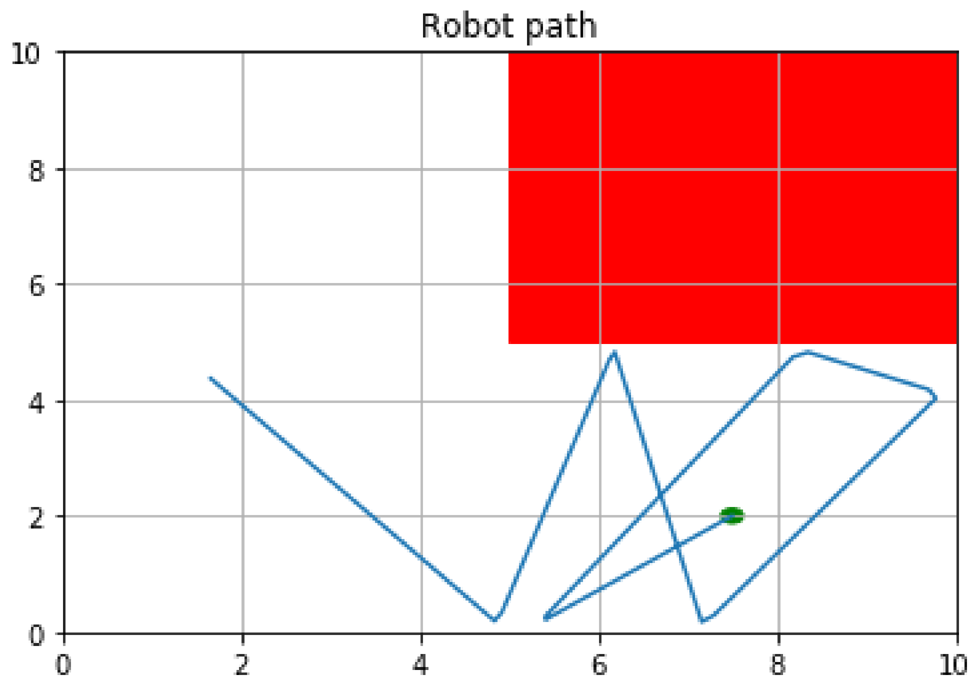
The following code, shows how to use the `class World` and `class Robot`, to simulate a simple movement within a finite horizon in time. <

[58]:
```
from Modules import Robot, World, Smart_Robot

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
```

[43]:
```
dt = .5 # time steps
r_radius = .25/2# robot radis
r_pos = (7.5,2,5) #robot init position as shown in the problem

robot = Robot(x0=r_pos[0],
              y0=r_pos[1],
              th0=4.9*np.pi/4,
              v=.1,
              radius=r_radius,
              th_nsamples = 100)

#Define the world as a rectangle
world = World(robot,
              bottom_left = (0,0),
              top_right = (10,10),
              dt=dt,
              ncells = 150)

#Add a rectangular obstacle where the robot cannot go
world.add_RedZone(bottom_left = (5,5),
                  top_right = (10,10))



# world.run(50*60)

# world.reinitialize(random_init_pos=True)
world.conditional_run(lambda world: world.time<5*60,  world ) #Run the sim while␣
↪defining the stop criteria. In this case, it is t<5min
```

```
##Post processing
#The following will plot the path

pos = np.array(robot.pos)
fig,ax1 = plt.subplots(1,1)

# # Create a Rectangle patch that represent the No go zone and cicle
rect = patches.Rectangle((5,5),5,5,linewidth=1,edgecolor='r',facecolor='r')
circ = patches.Circle(r_pos, radius=r_radius,edgecolor='g',facecolor='g')
ax1.add_patch(rect)
ax1.add_patch(circ)

ax1.plot(pos[::5,0],pos[::5,1])
ax1.axis([0,10,0,10])
ax1.grid()
ax1.set_title("Robot path")
fig.show()
```

```
/Users/massimacbookpro/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:47: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
```

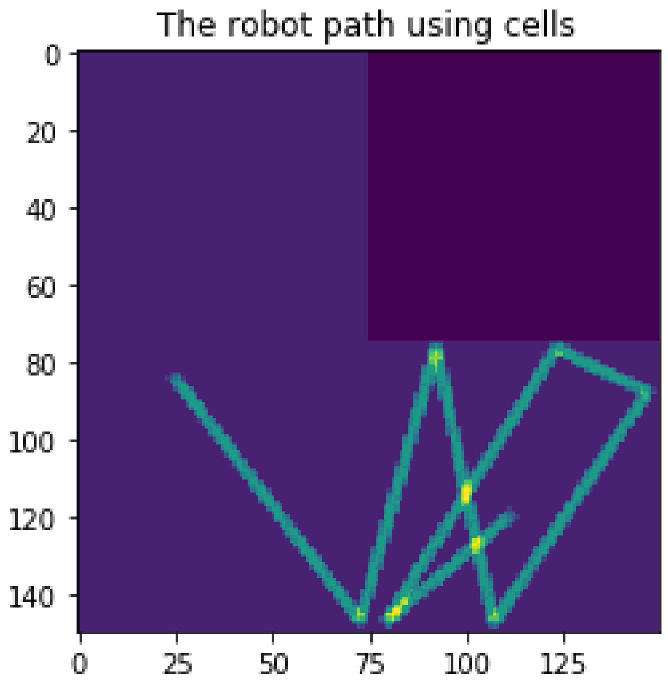


The figure above, shows the path of the robot within 5min horizon time. The red zone represent the no go zone or the obstacle.

In the figure bellow, we will render the cells.

[44]:
```
fig,ax2 = plt.subplots(1,1)
Z = world.grid.get_cells() #get the cells


ax2.imshow(np.rot90(Z))
ax2.set_title("The robot path using cells")
# ax.patch.set_facecolor("red")
```

[44]:
```
Text(0.5, 1.0, 'The robot path using cells')
```



The figure above shows more details about the path. The colored cells or area represent the cells that the robot has covered. Moreover, the cells color show how ofter the robot covered this area. For instance, the more the color tends to yellow, the more it means that this cell is visited more often.

Therefreo, computing the number of these colored cells allows us to estimate the coverage proproption or the cleaned area. This is done in the method `World.get_coverage`.

[29]:
```
print("The percentage of the room that is cleaned by the robot within 5min is␣
↪{}%".format(world.get_coverage()*100))
```

```
The percentage of the room that is cleaned by the robot within 5min is
8.847407407407408%
```

It is important to note the value above is just a simple. To obtain, a better estimate we need to sample different the simulation for several times, as shown bellow:

```
[28]: n_samplings = 800
      cover_pourcentage = 0

      for i in range(1,n_samplings+1):
          world.reinitialize(random_init_pos=True) # reinit the position, the cell and␣
       ↪the time of the simulation.
          world.conditional_run(lambda world: world.time<5*60,  world)
          cover_pourcentage += world.get_coverage()
          if i%100 == 0:
              print("run {}: The percentage of the room that is cleaned by the robot␣
       ↪within 5min is {:0.2f}% ".format(i,cover_pourcentage/i*100))
```

```
run 100: The percentage of the room that is cleaned by the robot within 5min is
9.11%
run 200: The percentage of the room that is cleaned by the robot within 5min is
9.06%
run 300: The percentage of the room that is cleaned by the robot within 5min is
9.04%
run 400: The percentage of the room that is cleaned by the robot within 5min is
9.02%
run 500: The percentage of the room that is cleaned by the robot within 5min is
9.02%
run 600: The percentage of the room that is cleaned by the robot within 5min is
9.02%
run 700: The percentage of the room that is cleaned by the robot within 5min is
9.02%
run 800: The percentage of the room that is cleaned by the robot within 5min is
9.01%
```

Hence, by looking at the result above, we can say that the percentage of the room that is cleaned by the robot within 5min is $\sim 9\%$.

1.2 Part B: The average of how long it takes the vacuuming robot to clean 75% of the room.

As above, to estimate the average time to clean or cover 75% of the whole room, we use the Monte-Carlo method. To do so, we will sample new initiale positions of the robot and compute the time that it takes to cover 75% of the area, then we compute the time. This will be repeated for N times untill the average time $\tilde{t} = \frac{1}{N}\sum_{i=1}^{N} t_i$ converges.

```
[49]: n_samplings = 150
sum_time = 0
ratio = .75

for i in range(1,n_samplings):
    world.reinitialize(random_init_pos=True) # reinit the whole scenario (with␣
 ↪new init position and cells)
    world.conditional_run(lambda world: world.get_coverage()<ratio,  world )
    sum_time += world.time
    if i%30 == 0:
        print("run {}: average time to cover {}% of the area is {:0.2f} min".
 ↪format(i,ratio*100,sum_time/i/60))
```

```
run 30: average time to cover 75.0% of the area is 77.49 min
run 60: average time to cover 75.0% of the area is 77.34 min
run 90: average time to cover 75.0% of the area is 77.92 min
run 120: average time to cover 75.0% of the area is 77.82 min
```

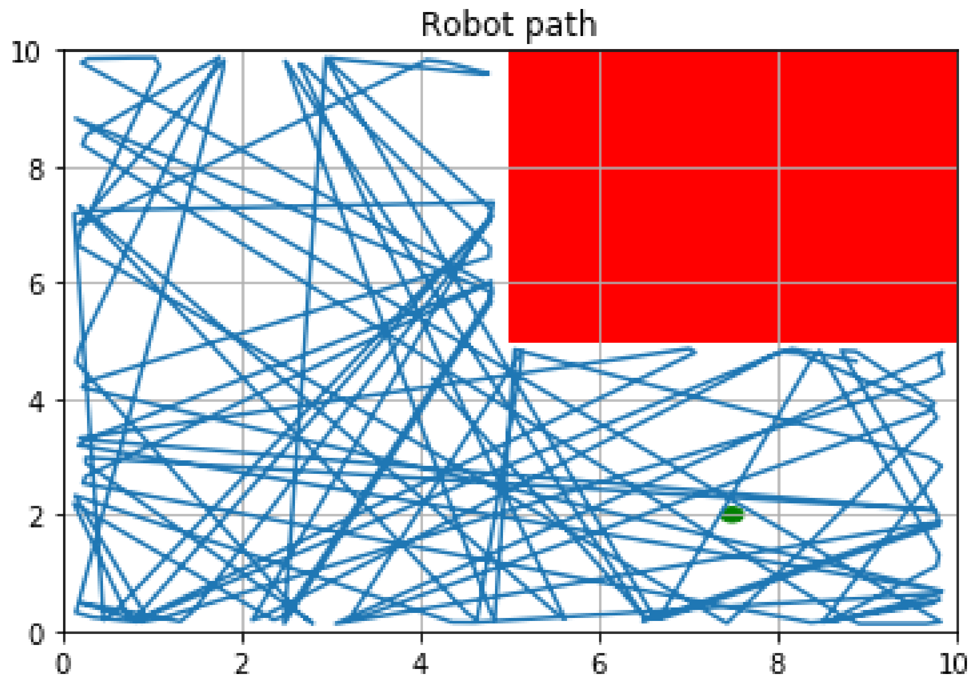
Hence, by looking at the result above, we can say that average time to clean 75% of the area is ~ 77 min, that is, 1h 17min.

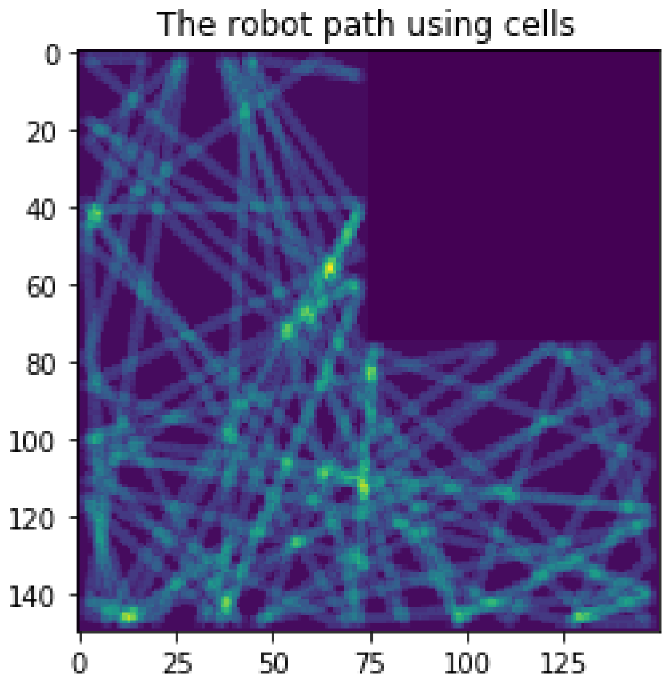
```
[51]: pos = np.array(robot.pos)
fig,ax1 = plt.subplots(1,1)

# # Create a Rectangle patch that represent the No go zone and cicle
rect = patches.Rectangle((5,5),5,5,linewidth=1,edgecolor='r',facecolor='r')
circ = patches.Circle(r_pos, radius=r_radius,edgecolor='g',facecolor='g')
ax1.add_patch(rect)
ax1.add_patch(circ)

ax1.plot(pos[::5,0],pos[::5,1])
ax1.axis([0,10,0,10])
ax1.grid()
ax1.set_title("Robot path")
fig.show()


fig,ax2 = plt.subplots(1,1)
Z = world.grid.get_cells() #get the cells


ax2.imshow(np.rot90(Z))
ax2.set_title("The robot path using cells")
# ax.patch.set_facecolor("red")
```

```
/Users/massimacbookpro/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:14: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
```

show the figure.

```
[51]: Text(0.5, 1.0, 'The robot path using cells')
```





The 2 figures above show how the path looks like when the robot covers 75% of the area

1.3 Part C: Upgraded version of the robot

As stated in the problem set, to upgrade the robot decision making, we use visual feedback to give more preference to non-clean area. To do this, we create a `class Smart_Robot`, which inheretate from the `class Robot`.

To implement the feedback strategy, we override the method `Smart_Robot::get_control(self,world,*args,**kwargs)`. To do so at each collision, we make a query of all the cell that are not yet covered and pick a direction toward one of them picked randomely. For more details, please see the file `Smart_Robot.py` in Modules.

```
[61]: dt = .9 #sampling time
r_radius = .25/2 # robot radis
r_pos = (7.5,2,5) #robot init position

##The NEW class is here
robot = Smart_Robot(x0=r_pos[0],
              y0=r_pos[1],
              th0=5*np.pi/4,
              v=.1,
              radius=r_radius,
              th_nsamples = 1000)

#Define the world as a rectangle
world = World(robot,
              bottom_left = (0,0),
              top_right = (10,10),
              dt=dt,
              ncells = 100)

#Add a rectangular No Go zone where the robot cannot go
world.add_RedZone(bottom_left = (5,5),
                  top_right = (10,10))

n_samplings = 150
sum_time = 0
ratio = .75

for i in range(1,n_samplings):
    world.reinitialize(random_init_pos=True) # reinit the whole scenario (with␣
 ↪new init position and cells)
    world.conditional_run(lambda world: world.get_coverage()<ratio,  world )
    sum_time += world.time
    if i%30 == 0:
        print("run {}: average time to cover {}% of the area is {:0.2f} min".
↪format(i,ratio*100,sum_time/i/60))
```

```
run 30: average time to cover 75.0% of the area is 60.29 min
run 60: average time to cover 75.0% of the area is 60.59 min
run 90: average time to cover 75.0% of the area is 60.44 min
run 120: average time to cover 75.0% of the area is 60.92 min
```

Similarily, by looking at the result above, we can say that average time to clean 75% of the area is $\sim$ 60 min, that is gain of 17min. This result is expected, because by adding the visual feedback, we can close the control loop and find better decisions.