Robot Assembling along a Boundary in Continuous Domain

Suraj Kumar
Roll-no - 21CS4128
Mtech Cse First Year
NIT Durgapur

April 2022

Abstract

This report is about how robots assembled in left boundary of rectangular region. Robots and obstacles are initially scattered in an unknown environment and they do not have direct communication among themselves. The algorithm guarantees successful assembling of all the robots on the left boundary of the given region within finite amount of time and without facing any collision during their movement. The intermediate distances among the assembled robots are not fixed. In this proposed algorithm, the robots follow the basic Wait -Observe-Compute-Move model together with the Full- Compass and Synchronous or semi - synchronous timing models.

Keywords: Robot swarm ,Assembling , Passive communication , Distributed algorithm

1 Introduction

Research works on swarm robots consider a group of relatively simple robots in various environments and coordinate the robots to perform desired task by using only local information. The main advantage of swarm is that , it can perform tasks that are beyond the capabilities of the individuals. For example, the ants move together in some particular direction in search of any food source and then they fetch the food portions in a coordinated manner which could have been an impossible mission for a single ant.

Problem of assembling a swarm of robots on a given line or shape is an interesting and very relevant problem in the area of swarm research. Assembling of robots on a common line may also be considered as one of the basic processing steps in solving different complex problems. Area partitioning may be mentioned as one such problem in which assembling of robots on a line has significant contribution, Area partitioning has several applications like scanning or coverage of a free space , lawn mowing and milling , sweeping, search and rescue of victims , space explorations etc.

2 Characteristics of Robots

The robots are usually represented as points. They may also be considered as unit discs. The point robots are assumed to be dimensionless.

On the other hand , sometimes the robots are assumed to be three dimensional bodies and are called *fat robots*. These fat robots, when they move, they leave a footprint on the ground. At any position , the footprint is a unit disc(circular). Because of their dimensions, the fat robots may create obstructions in other robots visibility and movement. In case of point robots also, some researchers assume line of sight obstruction by presence of a robot.

1. *Homogeneous vs Heterogeneous*
   In most cases robots are assumed to be homogeneous. That is, the robots are identical in all respect, especially; they have identical shape, size, computational capability, visibility, sensing capacity, speed. etc.

2. *Mobile*
   Robots are allowed to move freely on a plane.

   - *Rigid motion* Robot always reaches its destination wihout any halt in -between.
   - *Non-Rigid motion* In non-rigid motion , a robot may stop anywhere in-between before reaching its destination.

3. *Oblivious vs Non-Oblivious*
   Oblivious robots do not retain any information gathered in the past action. They carry out any computation based only on the present data.
   Non-Oblivious robots can use all information retained by them.

4. *Visibility Range and Sensing Range*
   Robots may have either unlimited or limited visibility.

   - *Unlimited visibility* Robot can view all other robots.
   - *Limited visibility* Robot can view only those robot which are located within its range of visibility.
     Area of visibility of a robot is circle of radius d where robot is at center.

   In case of unlimited visibility the line of sight of robots may be obstructed by presence of opaque obstacles.
   Sensing
   Robots are assumed to posses a sensing zone, a circular zone of radius, centred at the position of the robot. A robot can sense its surrounding and carry out some task within its sensing range. Sensing radius is usually much smaller than the visibility radius.

5. *State*
   Robots can have two states.

   - *Active* Robots are alive and executing its own job.
   - *Sleep* Robot does nothing live but not active. It is like power off state.

6. *Communication* There are three cases related to communication of robot.

   (a) Robots can have direct communication among themselves. They can directly exchange the information, their locations, states depending on those information robots decide their next course of action.

(b) In some cases robots are not able to communicate with other robots through message passing. They observe the position,actions of other robots then they decide their future action. This may be termed as *silent* or *passive communication*.

(c) *Restricted communication* In this type of communication robots can communicate only with visible robots and visibility between two robots get restricted due to presence of obstacle or any other robot or their line of sight.

In some case robots are assumed to have $O(1)$ (constant) number of coloured lights which are used to transmit specific messages to other robots.

7. *Autonomous* - Robots in a swarm may work autonomously or in a team. Autonomous robots take their decision and complete their job independent of others. On the other hand, robots in a team may need to depend on each other ; follow the rule set by the team leader or a specific team member.

3 Existing Models

Fundamental objective of research in swarm robotics to identify minimal sets of properties essential for the robots under which certain problems are solvable.

3.1 Computational Models

Wait-Observe-Compute-Move also known as a *CORDA* model.
Computational cycle consists of a sequence of three basic steps.

1. *Observe* - It gets snapshot of its surrounding depending on it's visibility range.
2. *Compute* - It process the data collected in observe step and compute next destination.
3. *Move* - Robots actually move to its destination as calculated in the compute phase. In some case observation might led a robot not a change its position in move step in such case robot remain idle. On reaching the destination, robot again start a new computational cycle. A robot executes sequence of computational cycles until the solution of the problem is obtained.

3.2 Model for Synchronicity

1. *Synchronous model* - Robots have identical clocks and are active in every cycle.
2. *Semi - Synchronous model* - Robots operate according to the same cycle, but need not be active in every cycle and may not be active in all computational cycle.
3. *Asynchronous model* - It operates on independent computational cycles of variable lengths. The robots may not share any common clock.

Robot can not be in sleep state for infinite time during the execution of any task.

3.3 Model for local co-ordinate system of the Robots

1. *Full - compass* - Direction and orientation of both axes of the local co-ordinate systems are common to all robots.
2. *Half - compass* - Direction of both axes are common to all , but the positive orientation of only one axis are common (i.e. robots may have different views of the positive orientation of the other axis).
3. *Direction-only* - Direction of both axes are common for all robots, but orientation of the axes may be different.
4. *Axes-only* - Direction of both axes are common to all but the orientation of the axes may differ. The robots do not agree on which of the two axes is the x-axis and which are in the y-axis.
5. *No-compass* - There is no agreement among local co-ordinate system.

4 Assumptions and Models

- Robots and obstacles are initially scattered.
- Robot posses *silent* or *passive communication*
- The intermediate distance among robots are not fixed.
- Robot follows
  - CORDA model
  - Full compass
  - Synchronous/ Semi - synchronous
- Obstacles are assumed to be opaque horizontal line with negligible width.
- Robots have unlimited visibility.
- Robots are oblivious (retain only constant / small amount of information during execution).
- No two robots occupy the same position.
- Obstacle length can not exceed the length of rectangular length.
- Robots are identical and homogeneous with respect to computational power.
- They are autonomous in the sense that there is no central control.
- All robot execute same algorithm independently.
- Robots are capable to move freely on a plane. They follow *Rigid motion* (compute step without any halt-in-between).

5 Algorithm

x = x co-ordinate of Robot
y = y co-ordinate of Robot
x1 = x co-ordinate of Obstacle
y1 = y co-ordinate of Obstacle
Here Obstacle means other Robot or horizontal line static obstacle

```
if Robot is on left boundary then
    Pass
else
    if There is no obstacles/robots in left of robot then
        x ← 0
    else
        if Robot in bottom boundary then
            if Obstacles in top boundary then
                y = ((breadth − y1//2) ∗ x)//length
            else
                y = (y1 ∗ x)//length
            end if
        else if Robot in Mid Region then
            if Obstacle Above Robot then
                if Obstacle in Top boundary then
                    y = (((y1 − y) ∗ x)//(2 ∗ length)) + y
                else
                    y = (((y1 − y) ∗ x)//length) + y
                end if
            else
                if Obstacle in bottom boundary then
                    y = −((y ∗ x)//2 ∗ length) + y
                else
                    y = ((y1 − y) ∗ x)//length + y
                end if
            end if
        else
            if Obstacle in bottom boundary then
                y = −((y ∗ x)//(2 ∗ length)) + y
            else
                y = ((y1 − y) ∗ x)//length + y
            end if
        end if
    end if
end if
```

6 Code

Code 1: Code for Main Function

```python
from Robot_package_2 import *
from Environment import create_environment,set_obstacle
import multiprocessing
import matplotlib.pyplot as plt

if __name__ == "__main__":
  manager = multiprocessing.Manager()
  final_pos = manager.list()

  # Create environment
  length , breadth = create_environment()
  r = []

  # Enter detail of obstacle in environment
  obstacles = set_obstacle()
  print("obstacles are.. \n",obstacles)

  count = int(input("Total robot = "))
  calc_inal_loc(count,length,breadth,obstacles)
  for i in range(0,count):
    name = 'r'+str(i)
    r.append(Robot(name,colors[i%11],inal_loc[i]))

  # Representing obstacles
  for i  in range(0,len(obstacles)):
    plt.axhline(y = obstacles[i][1]/breadth, xmin = obstacles[i][0]/length,
                xmax = (obstacles[i][0] + obstacles[i][2])/length ,
                color = 'r',linestyle = '-')

  # Representing robot
  for i in range(0,count):
    plt.axhline(y = inal_loc[i][1]/breadth ,
                xmin = inal_loc[i][0]/length,
                xmax = (inal_loc[i][0] +1)/length,
                color = 'b',linestyle = '-')

  plt.xlabel('x - axis')
  plt.ylabel('y - axis')
  plt.show()

  for x in range(0,count):
    r[x].mydetails(x)
  print()


  # Assemble all the robot in left boundary
  '''
```

```
for x in range(0,count):
  r[x].move_to_left(length,breadth,x,obstacles)
'''
process_robot = []
for i in range(count):
  process_robot.append(multiprocessing.Process(target =
     move_to_left ,
                                                             args = (length,
                                                                breadth,i,
                                                                obstacles,
                                                                inal_loc,
                                                                final_pos )))

for p in range(count):
  process_robot[p].start()

for q in range(count):
  process_robot[q].join()

print()
final_pos.sort()
#print()
#print("pos = ",final_pos)
for i in range(0,count):
  inal_loc[i] = [final_pos[i][1],final_pos[i][2]]

for x in range(0,count):
  r[x].mydetails(x)

print()
plt.show()
# Representing obstacles
for i  in range(0,len(obstacles)):
  plt.axhline(y = obstacles[i][1]/breadth, xmin = obstacles[i][0]
     /length,
              xmax = (obstacles[i][0] + obstacles[i][2])/length ,
              color = 'r',linestyle = '-')

# Representing robot
for i in range(0,count):
  plt.axhline(y = inal_loc[i][1]/breadth ,
              xmin = inal_loc[i][0]/length,
              xmax = (inal_loc[i][0] +0.4)/length,
              color = 'b',linestyle = '-')

plt.xlabel('x - axis')
plt.ylabel('y - axis')

plt.show()
```

Code 2: Code for Environment package

```
def create_environment():
  global length,breadth,x_axis,y_axis
  print("Enter data for environment: ")
  length = int(input("Enter length: "))
  breadth = int(input("Enter breadth: "))
  print("Co-ordinate of rectangular area (4 - corners) is:")
  print("[0,0]  {}  {}  {}   in clockwise direction".format
       ([0,breadth],[length,breadth],[length , 0]))
  print()

  return length,breadth

def set_obstacle():
  obstacles = []
  obstacle_no = int(input("Enter number of obstacle "))
  print("for obstacle enter three parameter ")
  print("x_axis y_axix length ")
  print()
  while obstacle_no > 0:
    obs = list(map(int,input().split()))
    assert obs[2] < length - 10, "Length of obstacle will be less than length of environment"
    obstacles.append(obs)
    obstacle_no -= 1

  return obstacles
```

Code 3: Code for Robot package

```
import random
import math

class Robot:
    def __init__(self,name ,colour,position = [0,0], angle_moved = 0):
        self.name = name
        self.colour = colour
        self.position = position
        self.angle_moved = angle_moved


    def mydetails(self,j = 0):
        print("Robot:  " , self.name)
        print("color is ",self.colour)
        print("Location is ", inal_loc[j])
        #print("Location is ", self.position)
        #print("Total angle moved ",self.angle_moved)
        print()

    # Function to move the robot from one place to
    #another at an angle theta degree
```

```
    def move(self,position,distance,angle,prev_angle):
        prev_angle += angle
        print("{} steps at {} degree".format(distance,angle))

        if angle != 90 or angle != 270:
            self.position[0] += distance * round(math.cos(math.
                radians(prev_angle)),3)
        self.position[1] += distance * round(math.sin(math.radians(
            prev_angle)), 3)

        print("current location is : ",self.position)
        print("Angle moved : ",prev_angle)

        return self.position , prev_angle

#This function will check whether there is a obstacles /
# robot in left side of robot or not

def no_obstacles(x,y,inal_loc,obstacles):
  for obj in obstacles:
    if obj[1] == y and obj[0] < x :
        return False

  for pos in inal_loc:
      if y == pos[1] and x > pos[0]:
        return False
  return True

#This function will return the position of robot
# Whether in the top or bottom boundary or in  mid region

def find_position(breadth,y):
  if y == 0:
    return 'Bottom boundary'
  if y == breadth:
    return 'Top boundary'
  return 'Mid Region'

# This function will find nearest obstacle

def obstacle_Co_ordinate(x,y,breadth,obstacles,inal_loc):
  minm = float('inf')
  x_cord = x
  y_cord = y

  for obj in obstacles:
    if obj[1] != y:
      if minm > abs(obj[1] - y):
        minm = abs(obj[1] - y)
        x_cord = obj[0]
        y_cord = obj[1]
```

```
    for pos in inal_loc:
      if pos[1] != y:
        if minm > abs(pos[1] - y):
          minm = abs(pos[1] - y)
          x_cord = pos[0]
          y_cord = pos[1]

    # if no obstacle is in above or below the robot
    # but obstacle is in left of robot
    if x_cord == x and y_cord == y:
      if y != breadth:
        y_cord = breadth - y
      else:
        y_cord = breadth - y//2
    return x_cord , y_cord



# This function will assemble the robot in left
# boundary of environment

def move_to_left(length,breadth,j,obstacles,inal_loc,final_pos):
    print()
    print(" j = ",j)
    print("initial ",inal_loc)
    x,y = inal_loc[j][0],inal_loc[j][1]

    '''
    if robot is on left boundary
    we don't need to do any thing
    '''
    if x == 0:
      print(" j1 = ",j)
      inal_loc[j][0] , inal_loc[j][1] = x,y
    else:
      print(" j2 = ",j)

      # if there is no obstacles/robots in left of robot
      if no_obstacles(x,y,inal_loc,obstacles):
        print(" j3 = ",j)
        x = 0
      else:
       Robot_position = find_position(breadth,y)

       # Co-ordinate of nearest obstacle
       x1 , y1 = obstacle_Co_ordinate(x,y,breadth,obstacles,inal_loc)


       '''
       If Robot in bottom Boundary
```

```
    '''
    if Robot_position == 'Bottom boundary':

      # If obstacle in top boundary
      if y1 == breadth:
        print(" j4 = ",j)
        y = ((breadth - y1//2)*x)//length
      else:
        # Obstacle not in top boundary
        print(" j5 = ",j)
        y = (y1*x)//length



    elif Robot_position == 'Mid Region':
      '''
      If Robot in Mid Region
      '''

      # Obstacle Above Robot
      if y1 > y:

        # obstacle in Top boundary
        if y1 == breadth:
          print(" j6 = ",j)
          y = (((y1 - y)*x)// (2*length ))+ y
        else:
          # obstcale not in top boundary
          print(" j7 = ",j)
          y = (((y1 - y)*x)// length )+ y


      else:
        # Obstacle below Robot

        # Obstacle in bottom boundary
        if y1 == 0:
          print(" j8 = ",j)
          y = -((y*x)//2*length) + y
        else:
          print(" j9 = ",j)
          y = ((y1 - y)*x)//length + y



    else:
      '''
      If Robot in Top Boundary
      '''

      # if obstacle in bottom boundary
```

```
       if y1 == 0:
         #print("y*x = ",(y*x))
         print(" j10 = ",j)
         y = -((y*x)//(2*length)) + y
       else:
         # if obstacle not in bottom boundary
         print(" j11 = ",j)
         y = ((y1 - y)*x)//length + y



  print(" j12 = ",j)
  x = 0
  print(" j = {}, x = {} y = {}".format(j,x,y))
  print()
  final_pos.append([j,x,y])



colors = ["red","blue","green","violet","orange","brown",
         "purple","cyan","indigo","yellow","grey"]

inal_loc = []
def is_valid(x,y,obstacles,inal_loc):
    if [x,y] in inal_loc:
        return False

    for obs in obstacles:
        if y == obs[1]:
            if obs[0]-5 <= x <= obs[0] + obs[2]+5:
                return False
    return True

def calc_inal_loc(robot_no , length,breadth,obstacles):
    print()
    print("Enter 1 or 2 for follwoing")
    print("1 Intialise position of robot randomly")
    print("2 Manually give the initial position")
    choice = int(input("pick: "))

    if choice == 1:
        while robot_no > 0:
            x_pos = random.randint(0,length)
            y_pos = random.randint(0, breadth)
            temp = [x_pos,y_pos]
            if is_valid(x_pos,y_pos,obstacles,inal_loc):
                inal_loc.append(temp)
                robot_no -= 1
        #print(inal_loc)
    else:
        print()
```

```
224         print("Enter x axis and y axis in follwoing way")
225         print("75 25")
226         print("x_axis <= {} and y_axis <= {}".format(length,breadth)
            )
227         print()
228         for i in range(0,robot_no):
229             print("Enter x and y axis for robot{}".format(i+1))
230             x_pos , y_pos = list(map(int,input().split()))
231             inal_loc.append([x_pos,y_pos])
```

6.1 Code Explanation

6.1.1 Code for Main Function

Line 1 to 4
Importing various package
Robot package and environment package are self written package
matplotlib package is for plotting graph.

Line 7 - 8
manager list will return the list of final co ordinate of robot.
It is difficult to return the list after multiprocessing so manager list is used.

Line 10 to 15
We create environment and get the length and breadth.
Set the Obstacle.

Line 18 to 19
Enter the number of robot
calc_inal_loc function are taken from other package (*Robot_package_2*)
It will initialise the position of robot.

Line 20 to 22
r is a list in which we are storing robot object.
This robot object include three feature Robot name , Robot color and it's initial position.

Line 24 to 39
This code segment is used for representing the robot and obstacle in 2 d plane . Horizontal line represent the obstacle while point represent the robot.
Argument passed in axhline are *y* , *xmin* , *xmax,color, linestyle*. Since this graph will plot horizontal line . y is for y co ordinate , *xmin* is starting index of x co-ordinate and *xmax* is final index of x co-ordinate. *xmin* and *xmax* are helpful in plotting obstacle.
color is for color of line and line style is for selecting different type of line style.

Line 41 to 43
This segment is used for displaying the every object of robot.

Line 46 to 60
This segment is used for multiprocessing.
process_robot is a list that store the multiprocessing object. In the *target* we have to pass the function which will be followed by objects for multiprocessing. *args* have the argument of *target* function as a tuple.
Later we have to start all the process and join them.

Line 63 to 67
final_pos is a list which has final y co-ordinate of every robot and robot id. We sort the list and update the inal_loc for every robot which is now final position.

Line 69 to 70
Printing the detail of every robot along with it's final position.
Line 73 to 90
Again this code is used to plot obstacle and robot similar to *Line 24 to 39*

6.1.2 Code for Environment Package

Line 1 to 11 create_environment()
This code snippet is used for creating environment. It will ask user to enter length and breadth of 2D plane and display four co-ordinate of plane. Also return length and breadth to calling function.
Line 13 to 25 set_obstacle()
We are asking user to set the obstacle manually.
For any obstacle there will be three parameter *x-axis , y-axis* and *length*. Starting position for obstacle will be *[x-axis , y-axis]* and final position for robot will be *[x-axis , y-axis + length]*.
We use assert function to check final position of obstacle should at least 10 less than the length of field. This is because there should be some space for robot to move if required.

6.1.3 Code for Robot package

Line 1 to 2
Random package is used for generating random number and math package is used for different math operation used throughout the code.
Line 4 to 9
We create Robot class . Initialize robot object with name, colour, position and angle_moved. Position will display the current position of robot.Angle_moved will keep track of total angle moved by that particular robot object this will help to calculate the final co-ordinate of robot when we use polar co-ordinate system.
Line 12 to 18 mydetails(self, j = 0)
This code is used for displaying the various attribute of robot object. We can get position of robot by two way.
Using *self.position* and another way is *inal_loc[j]*. For the second method we are taking an extra argument j.
Throughout the code we use Cartesian coordinate system so we comment the angle moved part.
Line 20 to 33 move(self, position, distance, angle, prev_angle)
This code snippet is used to move the robot object according to polar co-ordinate.
If angle is 90 degree or 270 degree x-axis will not change otherwise $x = x + x \times cos\theta$. Similarly y axis will become $y + y \times sin\theta$.
Round function is used to round the value up to 3 decimal place otherwise it will give value up to 6 decimal place.
Later this function will return position and angle to calling function.
Line 35 to 46 no_obstacles(x, y, inal_loc, obstacles)
This function will check whether there is an obstacle in left side of robot or not. If there is an obstacle this function will return False otherwise it will return True. Obstacle means either horizontal line obstacle or another robot.
from 39 to 41.
We check every obstacle whether the y axis of that obstacle is same with y axis of robot. If they are equal then we check for x axis . If x axis of obstacle is less than x axis of robot it means

obstacle are in left side of robot.
Same logic is applied with robot position.

Line 48 to 56 find_position(breadth, y)
This function will return the position of robot whether the robot is in top or bottom boundary or in mid region.
Here we have to check y axis of robot. If y axis is equal to zero it means robot is in bottom boundary. If y axis is equal to breadth it means robot is in top boundary. Otherwise robot is in mid region.

Line 58 to 86 obstacle_Co_ordinate(x, y, breadth, obstacles, inal_loc)
This code snippet is used to find nearest obstacle to the particular robot.
We initialize *minm* with infinity. For every obstacle we find the absolute difference of y axis of robot and obstacle. If *minm* is greater then that difference we update the *minm* with that difference. Also store the x cord and y cord of that obstacle in *x_cord* and *y_cord* respectively.
Similar logic is applied to robot position. Since *inal_loc* has position of every robot we keep on comparing with *minm* and update it if required.
If no obstacle is in above or below the robot but there is a obstacle in left of robot.We check if y is not equal to breadth the *y_cord = breadth - y* otherwise *y_cord = breadth - y//2*.
We have to update the *x_cord* and *y_cord* because this function will always return value of *x_cord* and *y_cord* and if there is no obstacle above or below it will return the co ordinate of robot itself in that case program will fail to assemble all the robot properly.

Line 90 to 188 move_to_left(length, breadth, j, obstacles, inal_loc, final_pos)
move_to_left is the function which is passed at the time of multiprocessing. This function use the other function no_obstacles, find_position , obstacle_Co_ordinate .

x, y have x cord and y cord of robot respectively.
Line 103 to 105
This code snippet is for case when robot is already on left boundary. Here we don't need to do anything.
From line 106 to 188 cover the case where robot is not in left boundary.
Line 110 to 112 . This code is for the case where there is no obstacle in left of boundary. In this case robot simply move to left in one step.
Line 113 to 188 cover the case where there is a obstacle in left of boundary.
In line 114 we get the Robot position.

- If robot is in bottom boundary . It will execute code from line 123 to 132.
- If robot is in mid region. It will execute code from line 136 to 163.
- If robot is in Top boundary. It will execute code from line 167 to 180.

Different cases are already explained in Algorithm section of this report.
We append the final position of every robot along with robot object index in *final_pos*.

Line 192 to 193
We manually store some color name in color list. Although actual algorithm consider the homogeneous robot. This color is only of representation purpose. There is only eleven color in list. If we have more then eleven robot we can use *mod 10* function to assign color to the robot.

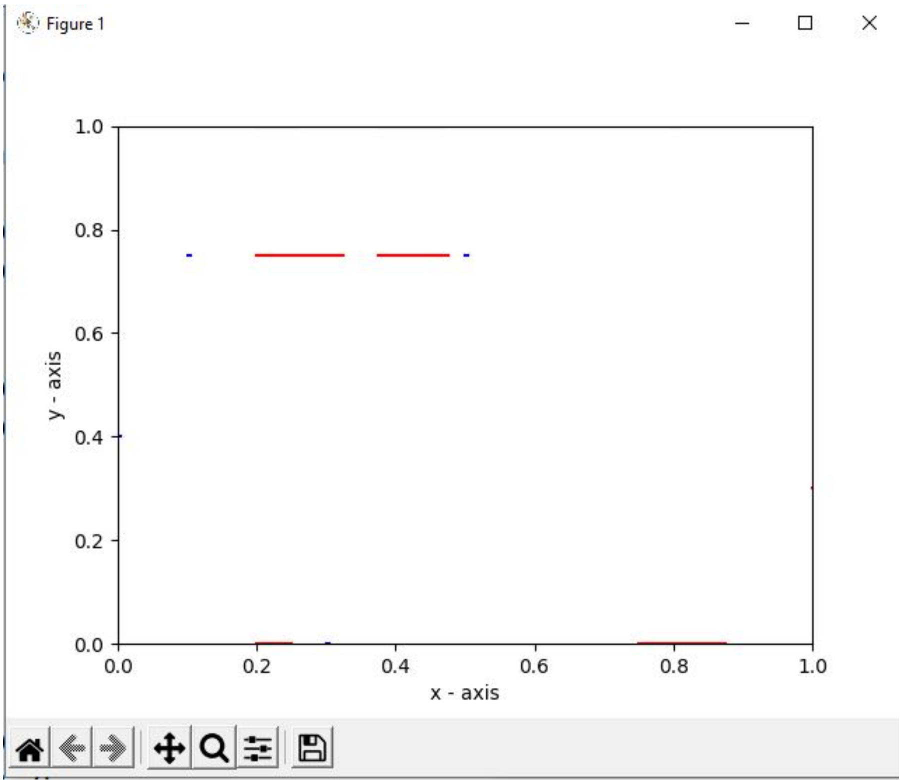
Line 195 to 204 is_valid(x, y, obstacles, inal_loc)
This function is used when we try to initialize the robot randomly. This function will ensure that two robot don't get the same co-ordinate. It also ensure that there should be at least difference of 5 unit distance in between obstacle and robot.

Line 206 to 231 calc_inal_loc(robot_no, length, breadth, obstacles)
This code snippet is used to initialise the position of robot. Here either we can manually give the initial position to robot or randomly generate some arbitrary position. Choice 1 is for randomly generating the position otherwise we have to manually enter the position.
For choice 1, for *x_pos* we randomly pick any point from 0 to length, for *y_pos* we randomly pick any point from 0 to breadth. Then we check whether the coordinate are valid or not. If valid we append in *inal_loc*.
For manual entry we simply enter the x coordinate and y coordinate separated by a single space for every robot and append it to *inal_loc*.

Input

```
Enter data for environment:
Enter length: 200
Enter breadth: 100
Co-ordinate of rectangular area (4 - corners) is:
[0,0]  [0, 100]  [200, 100]  [200, 0]   in clockwise direction

Enter number of obstacle 7
for obstacle enter three parameter
x_axis y_axix length

40 0 10
150 0 25
200 30 15
75 75 20
40 75 25
40 100 12
75 100 18
obstacles are..
 [[40, 0, 10], [150, 0, 25], [200, 30, 15], [75, 75, 20], [40, 75, 25], [40, 100, 12], [75, 100, 18]]
Total robot = 5

Enter 1 or 2 for follwoing
1 Intialise position of robot randomly
2 Manually give the initial position
pick: 2

Enter x axis and y axis in follwoing way
75 25
x_axis <= 200 and y_axis <= 100

Enter x and y axis for robot1
60 0
Enter x and y axis for robot2
160 100
Enter x and y axis for robot3
100 75
Enter x and y axis for robot4
20 75
Enter x and y axis for robot5
0 40
```

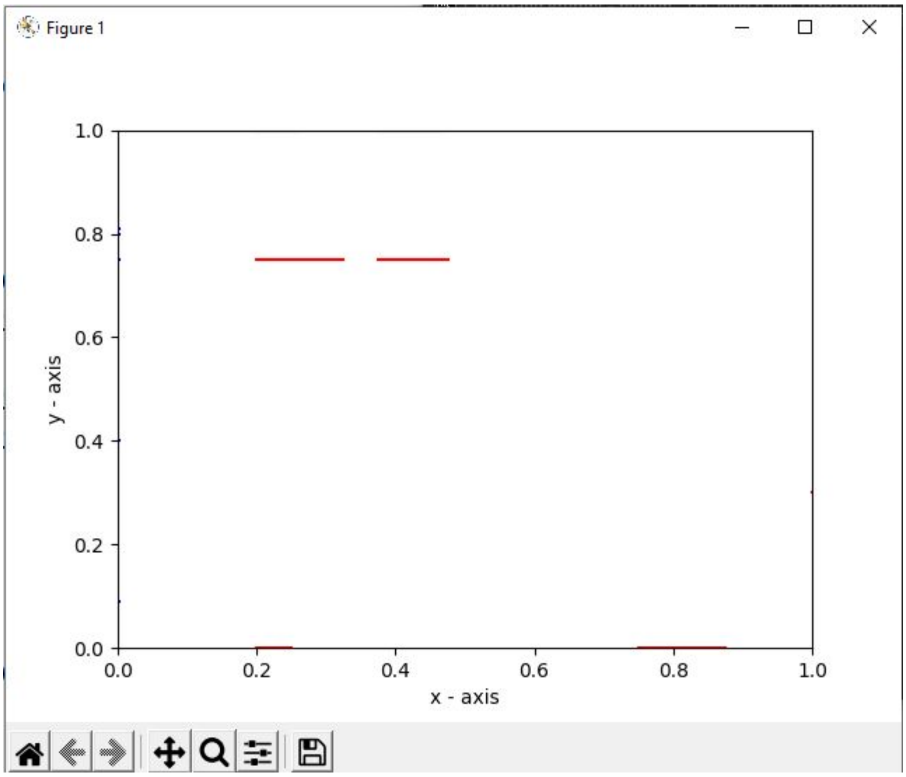
Initial Snapshot of 2d Field



Output

```
Robot:   r0
color is  red
Location is  [0, 9]

Robot:   r1
color is  blue
Location is  [0, 80]

Robot:   r2
color is  green
Location is  [0, 81]

Robot:   r3
color is  violet
Location is  [0, 75]

Robot:   r4
color is  orange
Location is  [0, 40]
```

Final Snapshot of 2d Field



7 References

https://www.youtube.com/playlist?list=PLeo1K3hjS3uub3PRhdoCTY8BxMKSW7RjN
https://www.youtube.com/watch?v=qiSCMNBIP2g&t=274s
https://link.springer.com/chapter/10.1007/978-981-10-1645-5_3