Onur Calisir

UID: 605-489-394

CEE / MAE M20

Mar 15th 2022

Final Project

1 Introduction

The goal of this problem is to develop a script that simulates a simple robotic mapping model. The robot is placed into a define map with a center obstacle, and an exit box. A stochastic process coupled with targeted driven variables will be used to help navigate said robot to the exit point. The robot must navigate from the initial position, initial constant speed to the exit traveling only in straight lines. If it encounters any obstacle, it will be deflected at same angle with the same speed. The only decision is at what angle with respect to the horizontal axis should the robot be initially set off. The mathematics involved in such calculation contains far too many variables and would not scale well if additional obstacles are introduced. Therefore, each run will be tested stochastically by randomizing the angle of departure between 0° and 90° over numerous trials to find the optimized one.

2 Model and Theory

Our Main Script will use 6 unique functions had will be set up to simulate the conditions:

1. **Draw_cir**: plots a solid circle, given x location, y location, radius, and color
2. **Draw_rec**: plots empty rectangles: given x location, y location, x size, y size and color.
3. **Plot_map**: uses draw_cir and draw_rect to plot the simulation map that include the map boundaries, the center wall as the obstruction, the robot in the shape of a solid circle, and the exit as a empty rectangle.
4. **check_collisions**: function checks for whether the robot has collided with any obstacles, i.e. the outer boundaries or the center wall.

5. **update_map**: function updates one iteration of the map based on collision type. The code would update the ball position if no collision occurred; otherwise, it would only update the velocity to indicate the reflection of the robot bouncing off the obstacle.
6. **run_trial**: function incorporates the aforementioned functions to run one trial of the robot movement and returns the time it took for the robot to reach the exit box. It is considered that the robot reached the exit box if its center passes the boundary of the box's edge.

3 Methods and Pseudo-code

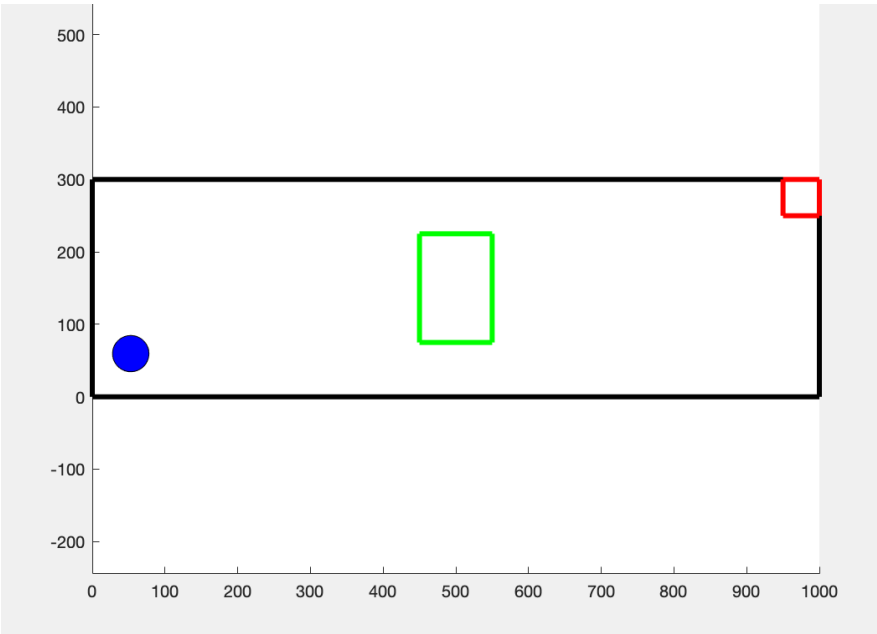
The flow of calculation should be as follows:

1. Select the initial conditions for the ball, and boundary conditions, ie the # of which run you want to simulate
2. Create the position matrices for map, wall, and exit
3. Set trial parameters
4. Create an storage array for the trials to optimize the angle
5. Iterate twice, the first one as the 10 bins, second one as the 10000 iterations
6. Set the initial conditions for the ball
7. Use run_trial function to simulate the process and calculate t=time_to_exit
8. From iterations calculate the best angles for each bin
9. Average the bins to get the best overall angle for a particular trial number
10. Print the best angle and time_to_exit

4 Calculations and Results

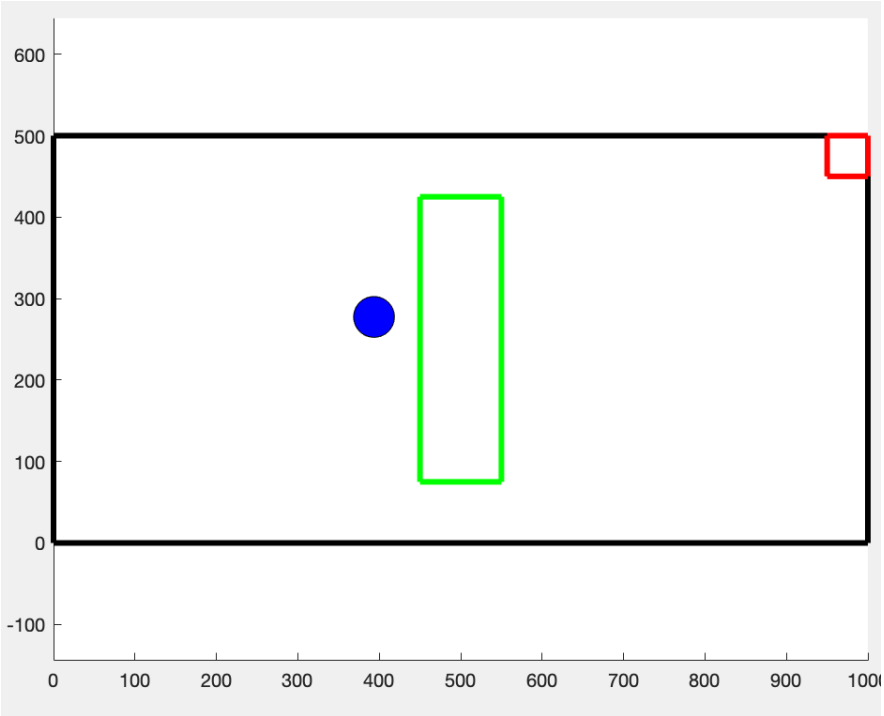
Trial #1

```
best_angle=6.893355  time_to_exit=25.400000
```



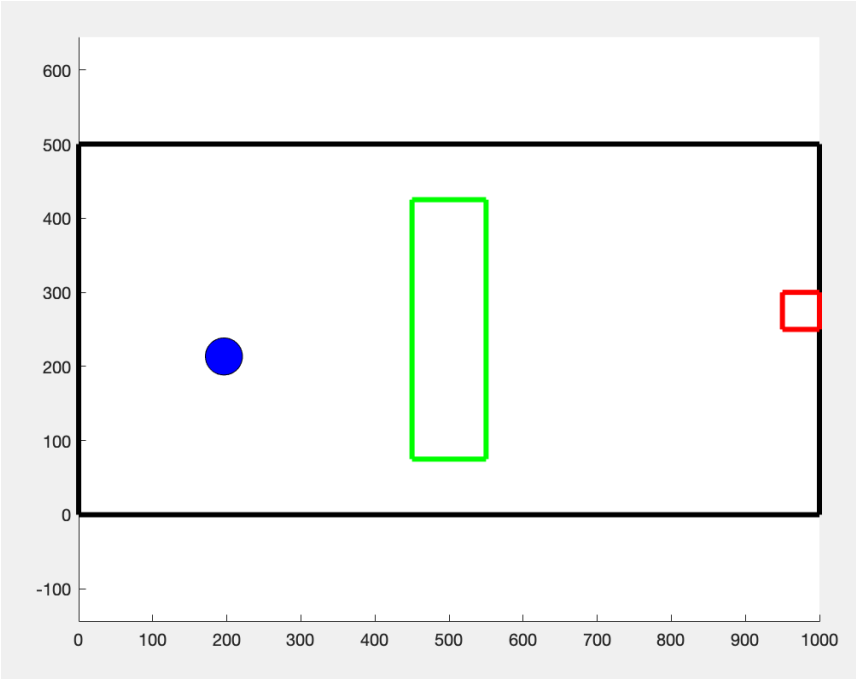
Trial #2

best_angle=7.117305 time_to_exit=41.400000



Trial #3

```
best_angle=35.732794 time_to_exit=122.700000
```



5 Discussions and Conclusions

From our calculations, it can be concluded that trial #1 takes the least amount of time to reach the exit box when launched with the best angle. The time it takes to exit nearly doubles for trial #2 and fivefolds for trial #5. It can be assumed that having a closer packed boundary, ie smaller y-size helps the robot exit much faster. It can also be assumed that having the exit box on the top corner is much more efficient than having it in the middle of the right map boundary. The robot is more likely to hit the corners than the wall itself at a certain position. That said, at the best angle, increasing the velocity yields in quicker exit for trial #3 ,but increases the time to exit by threefold for trial #1, and it doesnt effect trial #2 substantially. Overall we can conclude that even if the process is the same, alternating the variables for the map positions or the initial velocity of the ball yields in greatly different values, and the great discrepancy and complexity of the problem becomes more apparent. Even the slightist error yields in great differences and this is in general what makes AI systems or self-driving cars, so complex and challenging