// Nader Mahmoud

// Nawaf Abdullah

// CMPSC 201 SU 2016

// Date 8-04-16

// HW #11

// MATLAB Robot Simulation Final Project

1.0 Team Info

- Team members:
  - Nader Mahmoud
  - Nawaf Abdullah (Uploader)

- Communication platforms used
  - Microsoft OneDrive (very essential)
  - Gmail
  - Snapchat
  - Skype

2.0 Objective

- Description of the project’s goal:

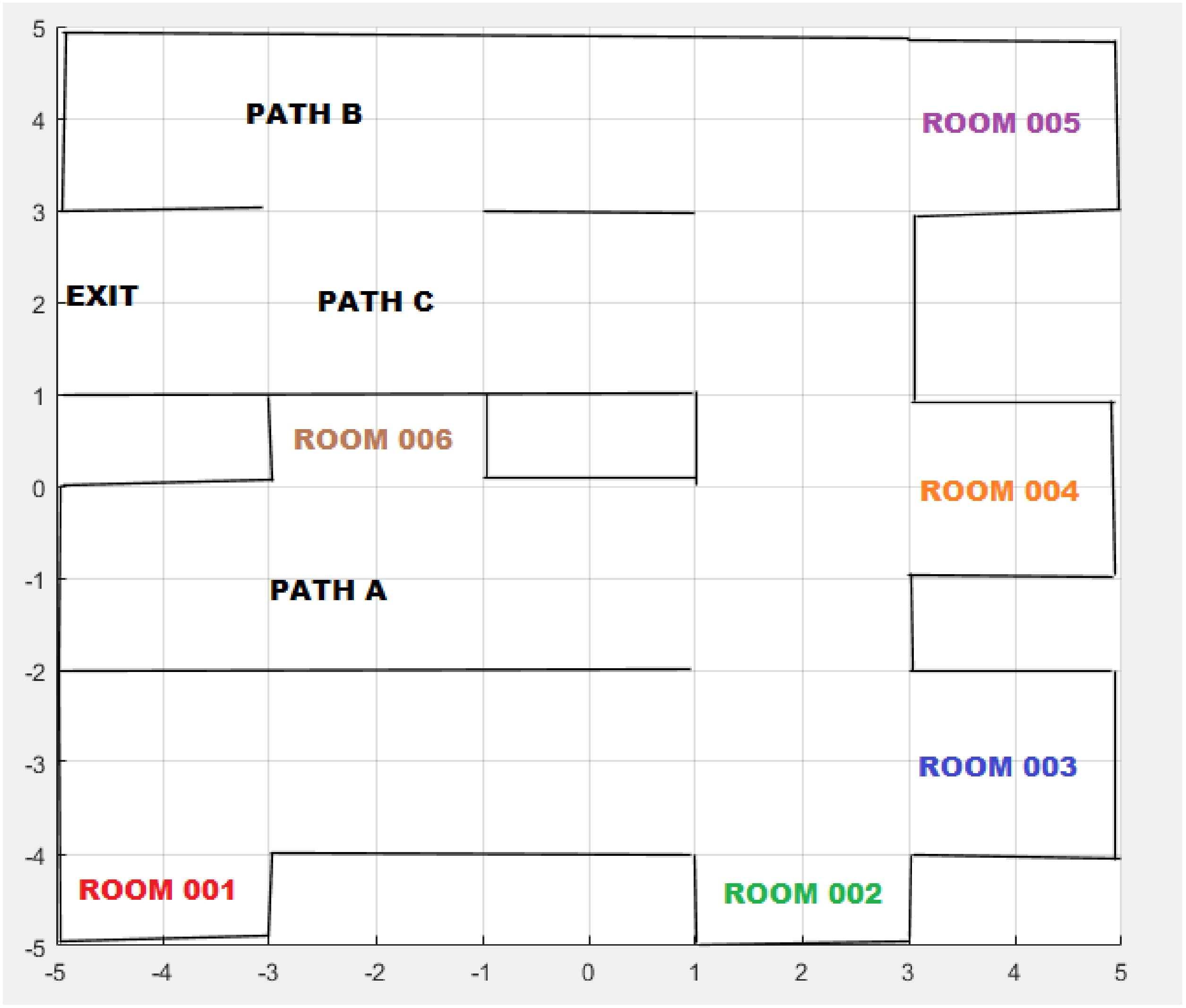
In this MATLAB simulation, we are attempting to program a robot to move through a maze, using the following features:

- Sonar Sensors
- Ability to make turn decisions based on which sensor (left, right, front, or back) is giving the largest reading (i.e. farthest from a wall).

3.0 Obstacles course

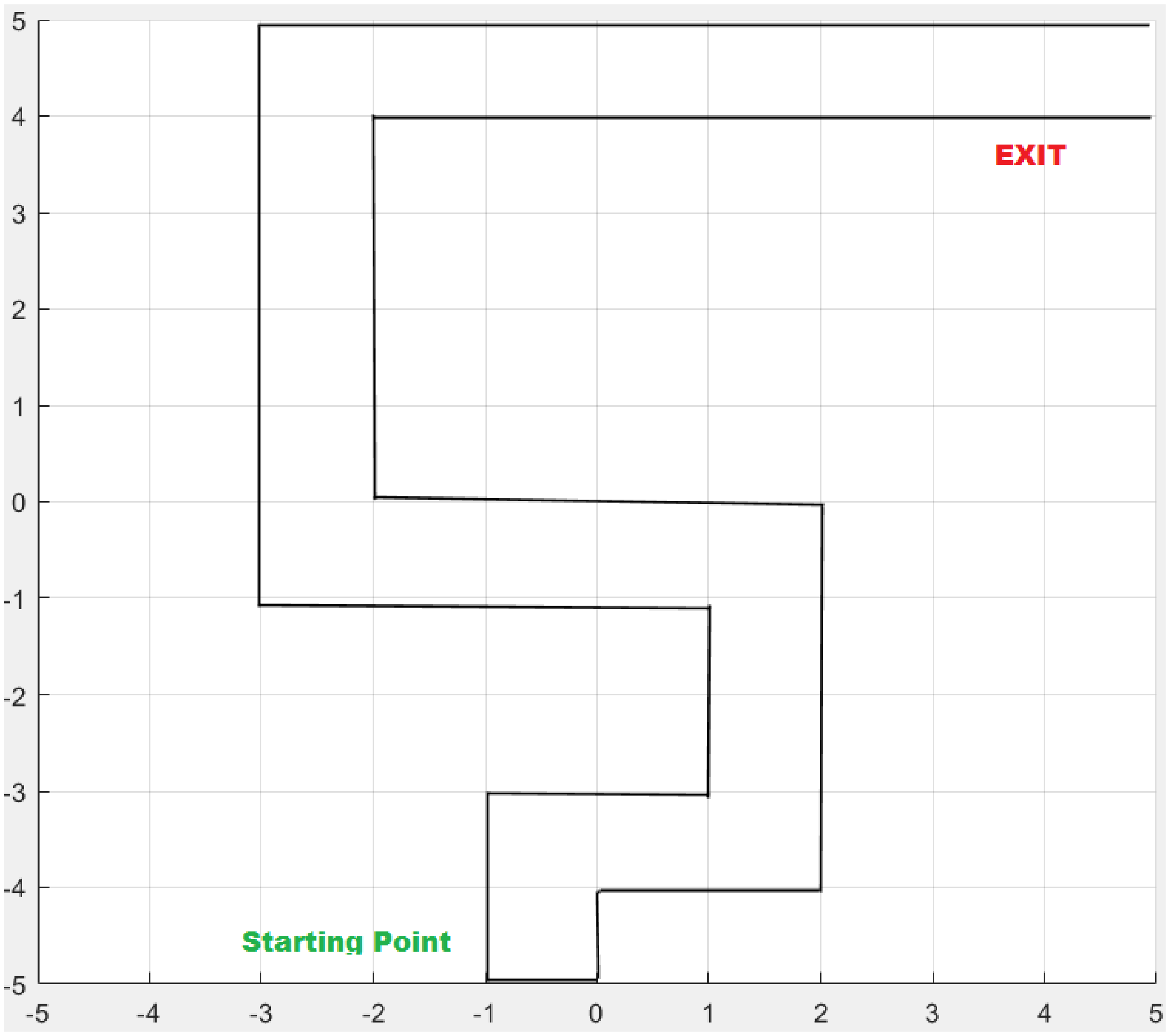
- Screenshots of the maps used for the simulation:

  Map for Test Case #1:
  This map has been designed to have multiple “rooms” (ROOM 001 – 006) that the robot may be going through, and there are 3 paths that are almost identical in length (PATH A,B and C).



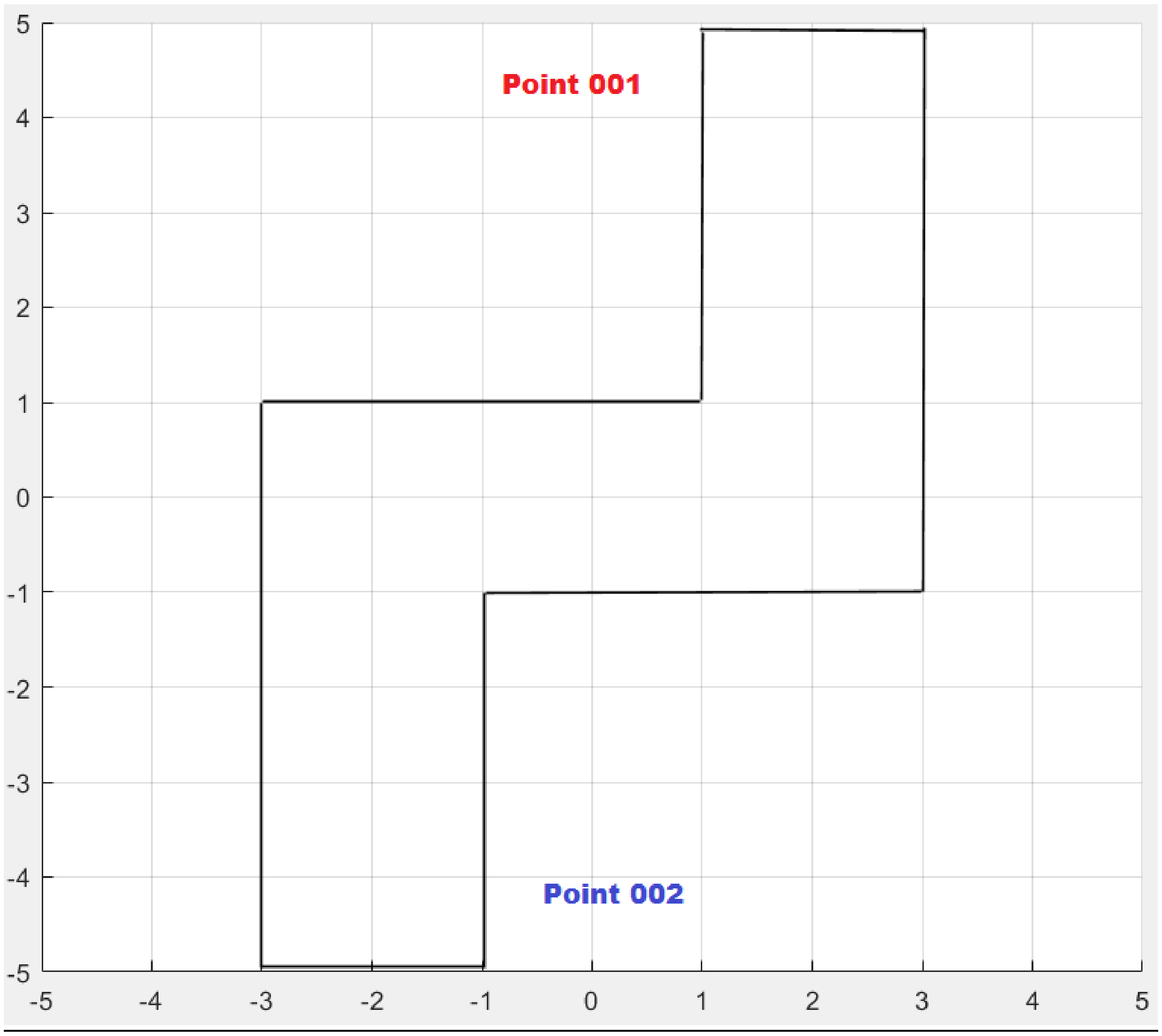
Map for Test Case #2:

This map was designed to be narrower, having only way from the starting point to the exit.



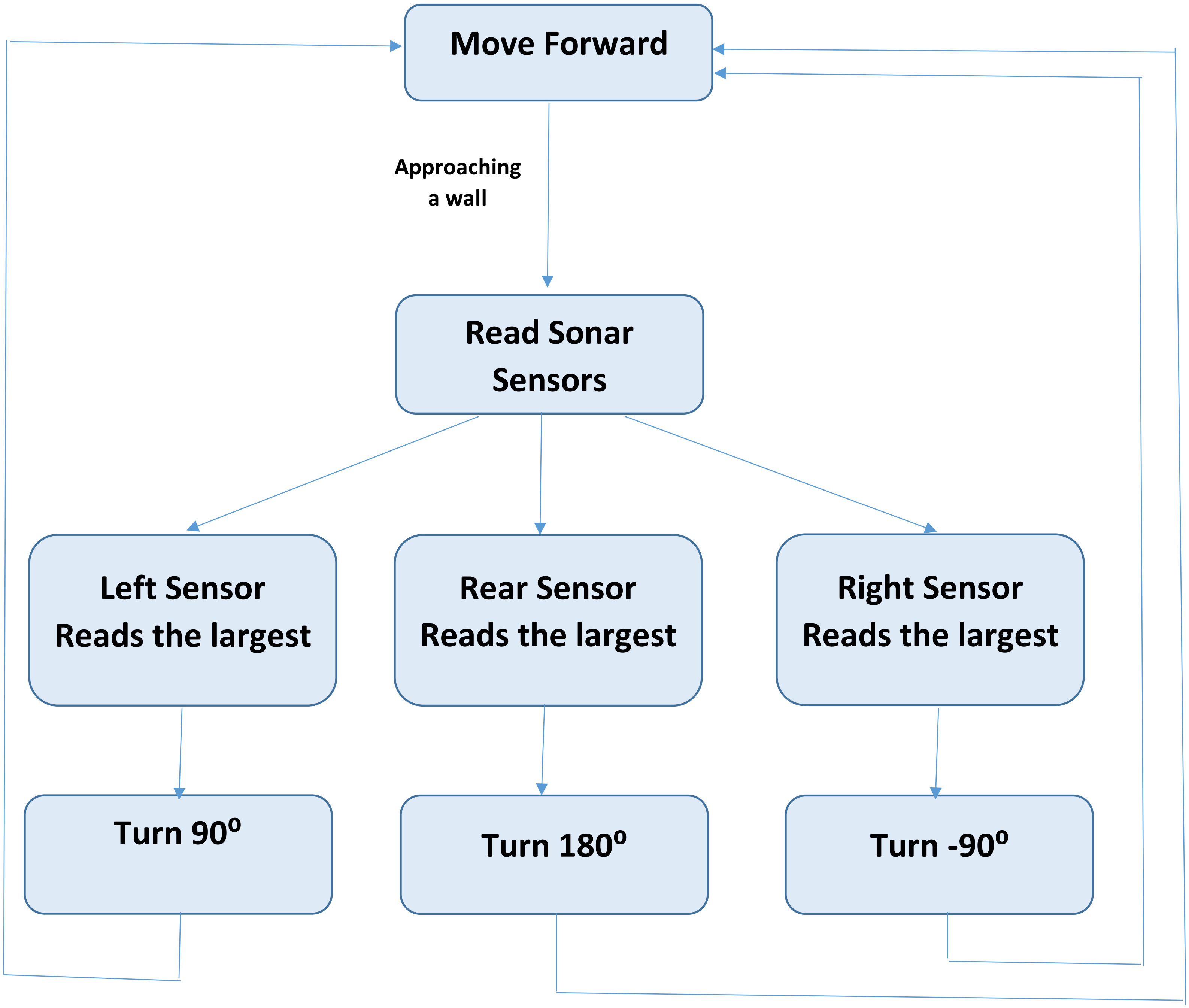
Map for Test Case #3:

The last map is just the same map used in H.W. 10, it is completely closed from both ends, and the objective here will be to see if the robot will keep going back and forth between point 001 and point 002.



4.0 MATLAB script

- The algorithm
  This is the basic algorithm for deciding the robot movements:



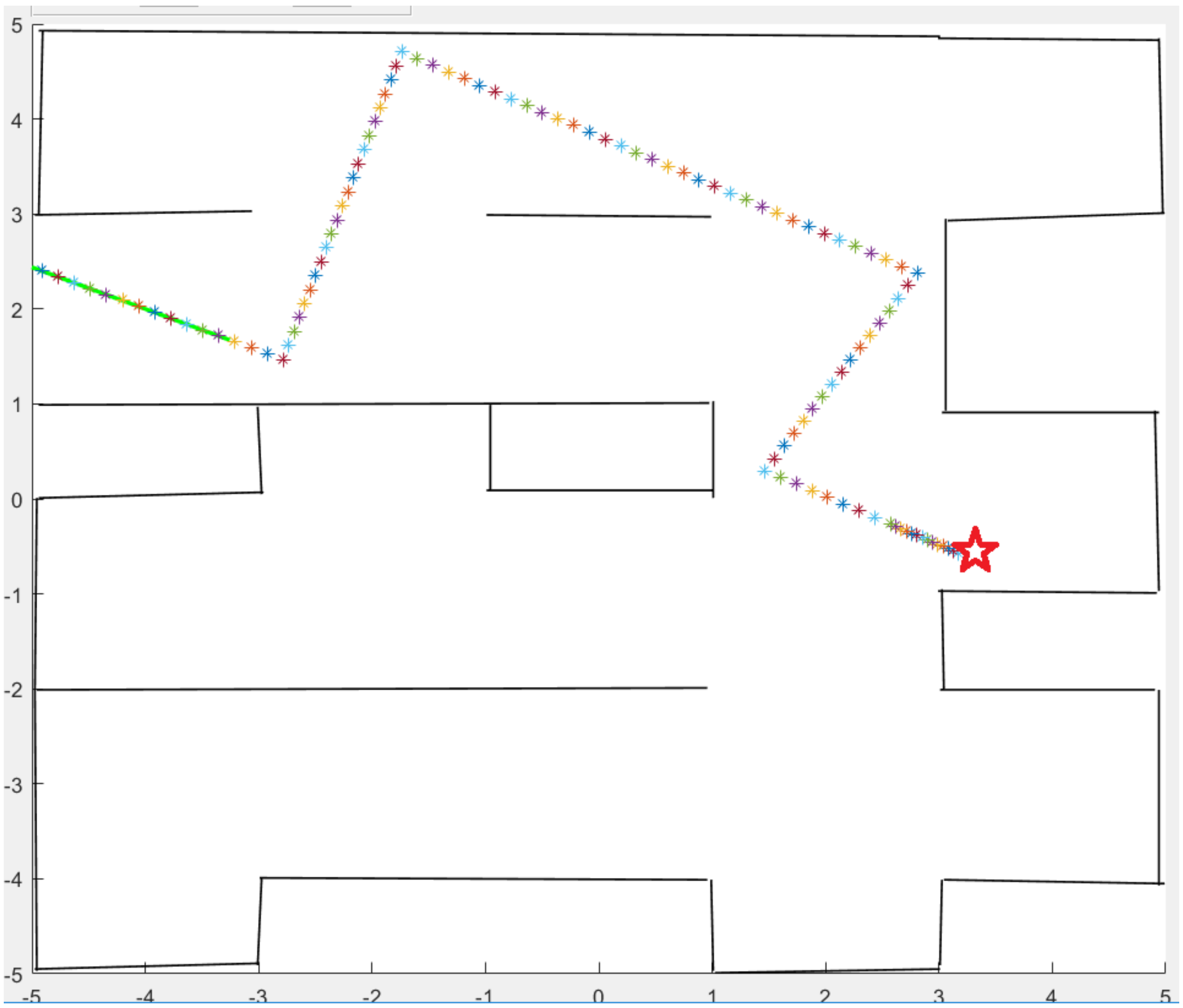
Move Forward
Approaching a wall
Read Sonar Sensors
Left Sensor Reads the largest
Rear Sensor Reads the largest
Right Sensor Reads the largest
Turn 90º
Turn 180º
Turn -90º

- The m-file script used for the MATLAB simulation with comments
  The script consists of just two code segments:

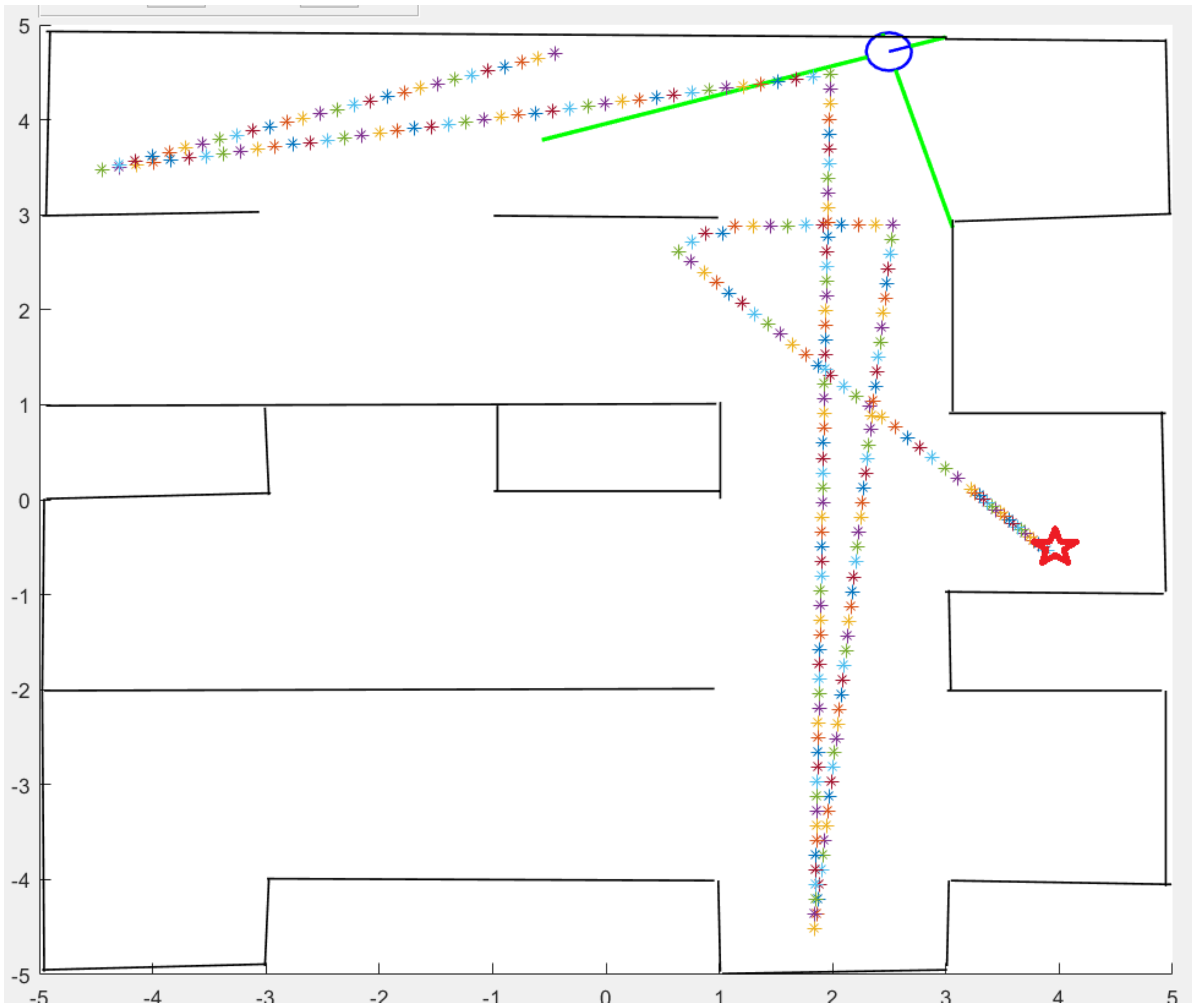
```
function MazeRunner(serPort)
SetDriveWheelsCreate(serPort, 0.5, 0.5)%Robot drives with a speed of
0.5m/s
while (1)
    dFront = ReadSonarMultiple(serPort, 2) %Front Sensor
    dRear = ReadSonarMultiple(serPort, 4) %Rear Sensor
    dRight = ReadSonarMultiple(serPort, 1) %Right Sensor
    dLeft = ReadSonarMultiple(serPort, 3)  %Left Sensor
    [x y th] = OverheadLocalizationCreate(serPort);
    plot(x, y, '*'); pause(0.1) %Paint path
    if   dFront < 0.5 %Stop wjhen wall is within 0.5 meter
        break
    end
end


if dRear > dRight && dRear > dLeft
    turnAngle(serPort, .2, 180);
elseif dRight > dRear && dRight > dLeft
    turnAngle(serPort, .2, -90);
elseif dLeft > dRear && dLeft > dRight
    turnAngle(serPort, .2, 90);
elseif dLeft == dRear
    turnAngle(serPort, .2, 90);
elseif dRight == dRear
    turnAngle(serPort, .2, -90);
else
    turnAngle(serPort, .2, 90)
end
```

5.0 Screenshots of test cases

- Test case #1:



In this test case, the robot functioned without any problems, starting in Room 004 (indicated by the red star), it didn't make contact with any of the walls, and ended up exiting the map.

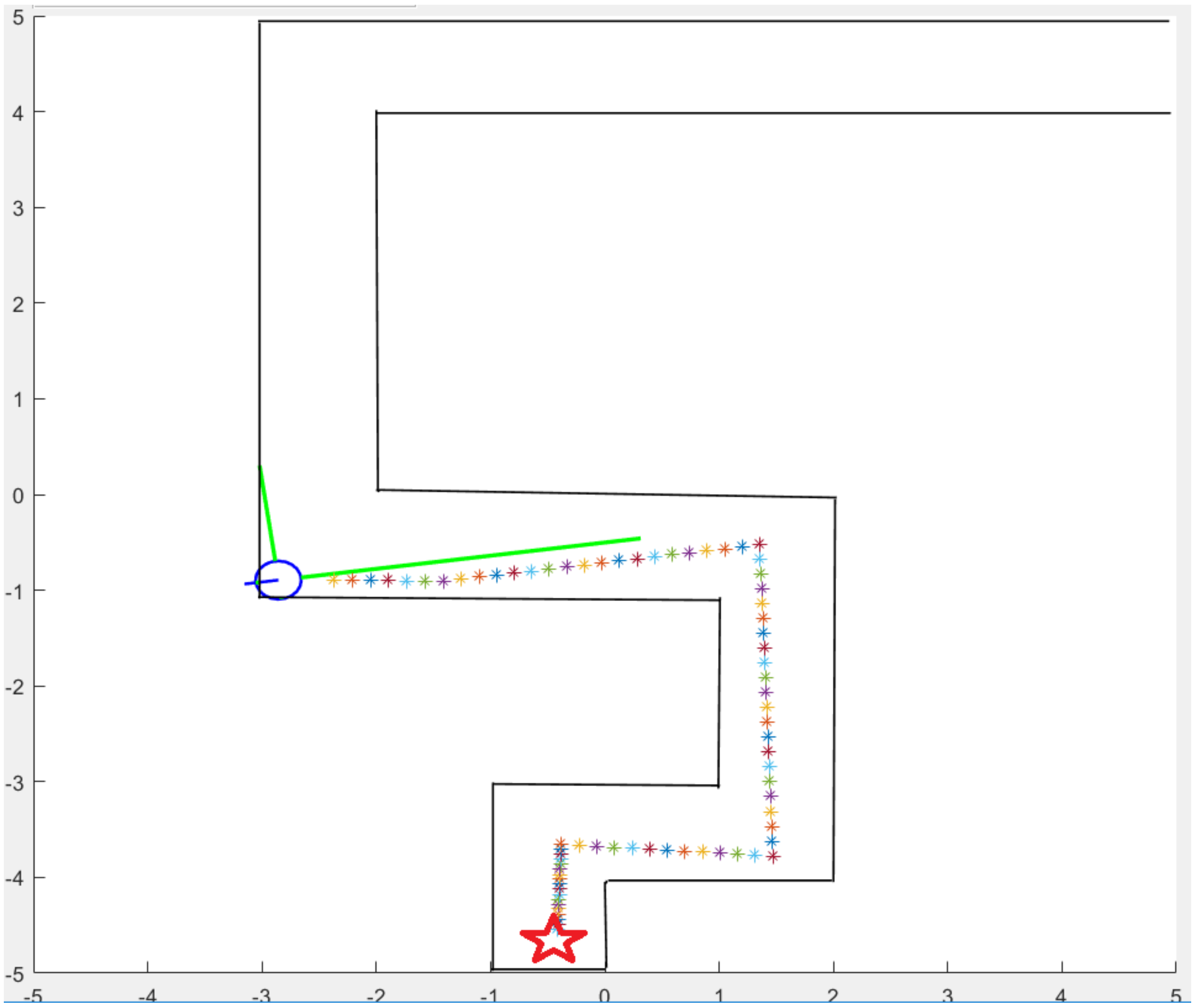


However, here is another simulation where the robot didn't end up exiting the map, and ended up going from Room 004 to Path C, to Room 002, then Path B, and ending up hugging a wall, which caused MATLAB to return this error:

```
Error in
matlab.graphics.internal.figfile.FigFile/read>@(hObject,eventdata)SimulatorGU
I('push_auto_start_Callback',hObject,eventdata,guidata(hObject))
Error while evaluating UIControl Callback
```

Among other many error messages.

- Test case #2:

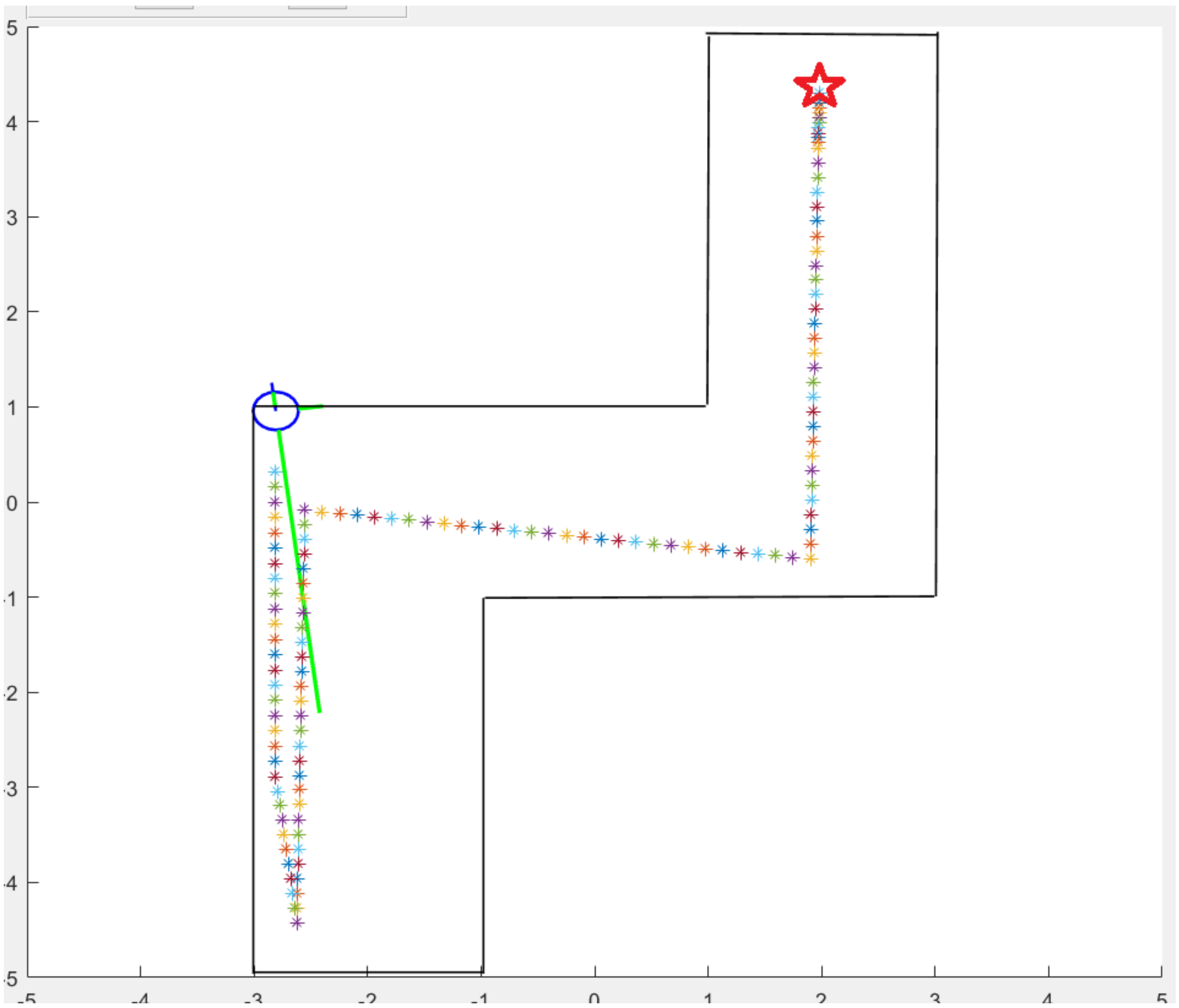


In this test case, we attempted to make the robot move through a much narrower path. We have tried many times to get it to the end of the map, however, it always ended up hugging a wall due to the difficulty of starting in a perfectly perpendicular orientation, and returning an error similar test case #1:

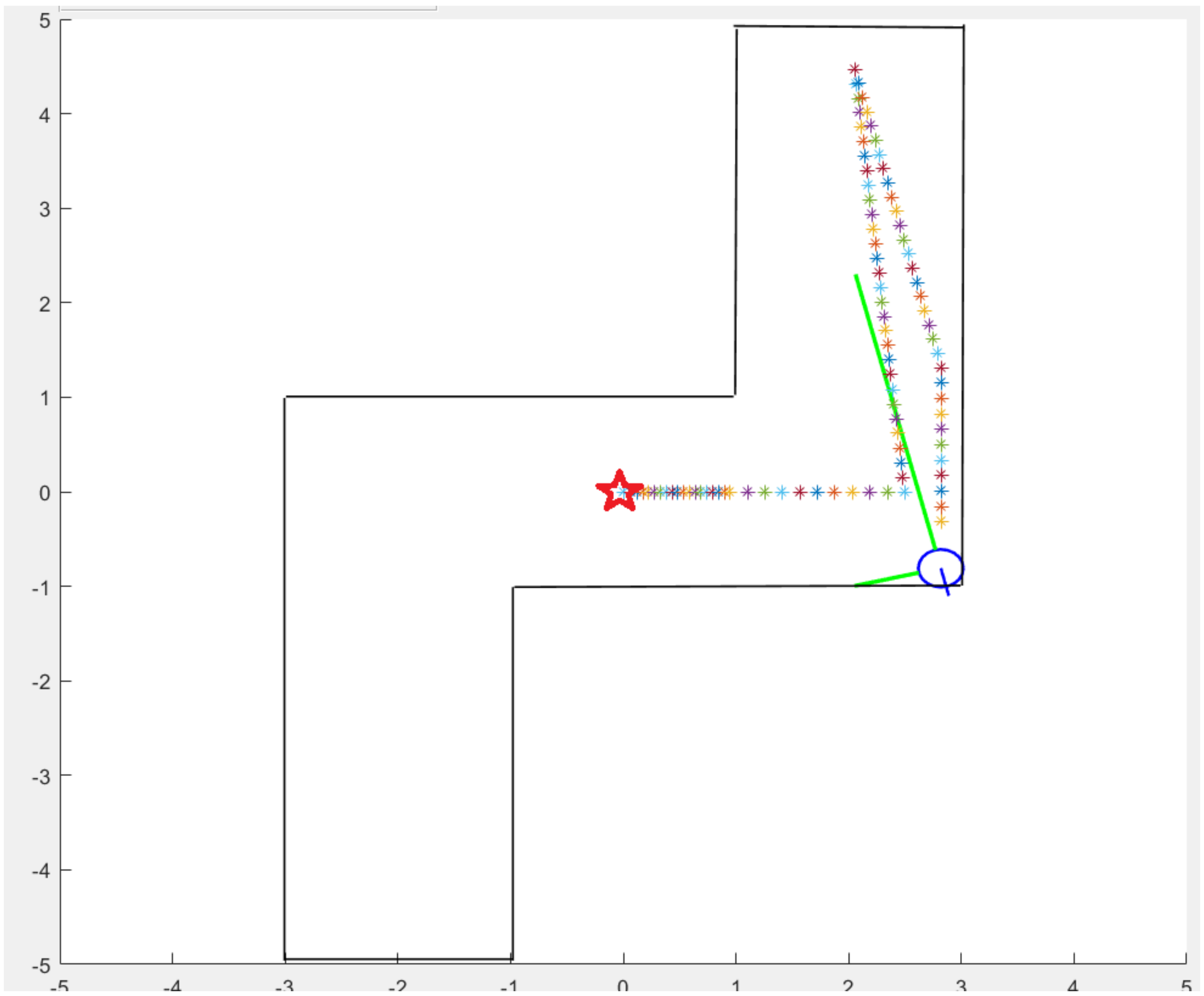
```
Error in
matlab.graphics.internal.figfile.FigFile/read>@(hObject,eventdata)SimulatorGUI('push_auto_start_Callback',hObject,eventdata,guidata(hObject))
Error while evaluating UIControl Callback
```

Also among other errors.

- Test case #3



Just like in Test case #2, the slightest difference from a perfectly perpendicular start will cause the robot end up Hugging a wall and return an error.



This is another simulation with the robot starting in the *default* position, with its sides almost perfectly parallel to the walls of the maze, and yet, the robot ended up hugging the wall and returning an error.

6.0 Solving the Wall Hugging Problem

- Theory:

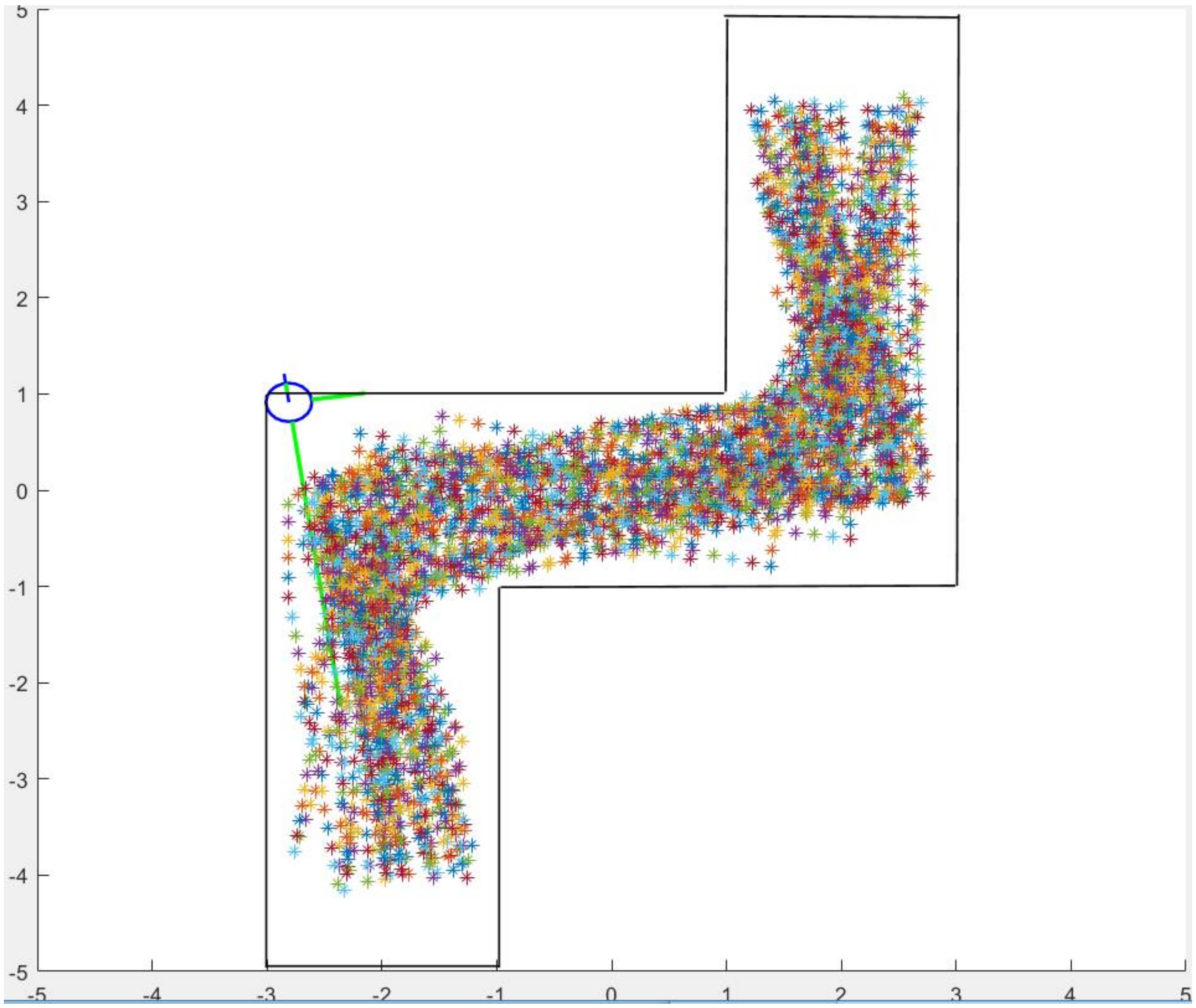
  We initially set the distance for the robot to stop before turning at d < 0.5, because we wanted to test our code on a much narrower path (see test case #2). However, we then realized as we were doing the test cases in 5.0 that it has prevented our robot from functioning as it should, always ending up hugging a wall, and returning an error. Therefore, we have decided to change back the distance to d < 1.0, and re-run our tests.

- Modified script

```
function MazeRunner(serPort)
SetDriveWheelsCreate(serPort, 0.5, 0.5)%Robot drives with a speed of
0.5m/s
while (1)
    dFront = ReadSonarMultiple(serPort, 2) %Front Sensor
    dRear = ReadSonarMultiple(serPort, 4) %Rear Sensor
    dRight = ReadSonarMultiple(serPort, 1) %Right Sensor
    dLeft = ReadSonarMultiple(serPort, 3)  %Left Sensor
    [x y th] = OverheadLocalizationCreate(serPort);
    plot(x, y, '*'); pause(0.1) %Paint path
    if   dFront < 1.0 %Stop wjhen wall is within 1 meter
        break
    end
end


if dRear > dRight && dRear > dLeft
    turnAngle(serPort, .2, 180);
elseif dRight > dRear && dRight > dLeft
    turnAngle(serPort, .2, -90);
elseif dLeft > dRear && dLeft > dRight
    turnAngle(serPort, .2, 90);
elseif dLeft == dRear
    turnAngle(serPort, .2, 90);
elseif dRight == dRear
    turnAngle(serPort, .2, -90);
else
    turnAngle(serPort, .2, 90)
end
```

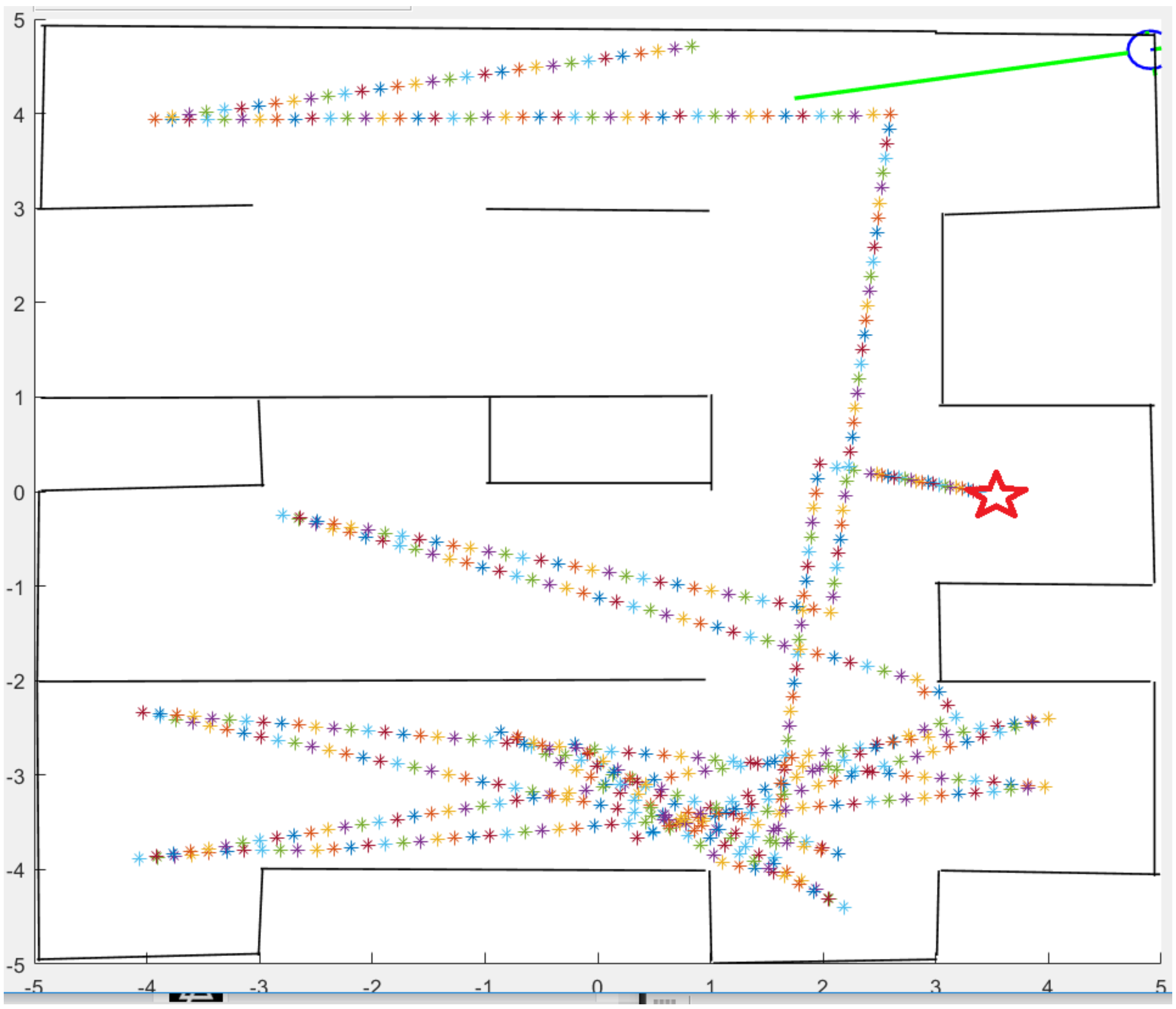
- Testing theory

Case A



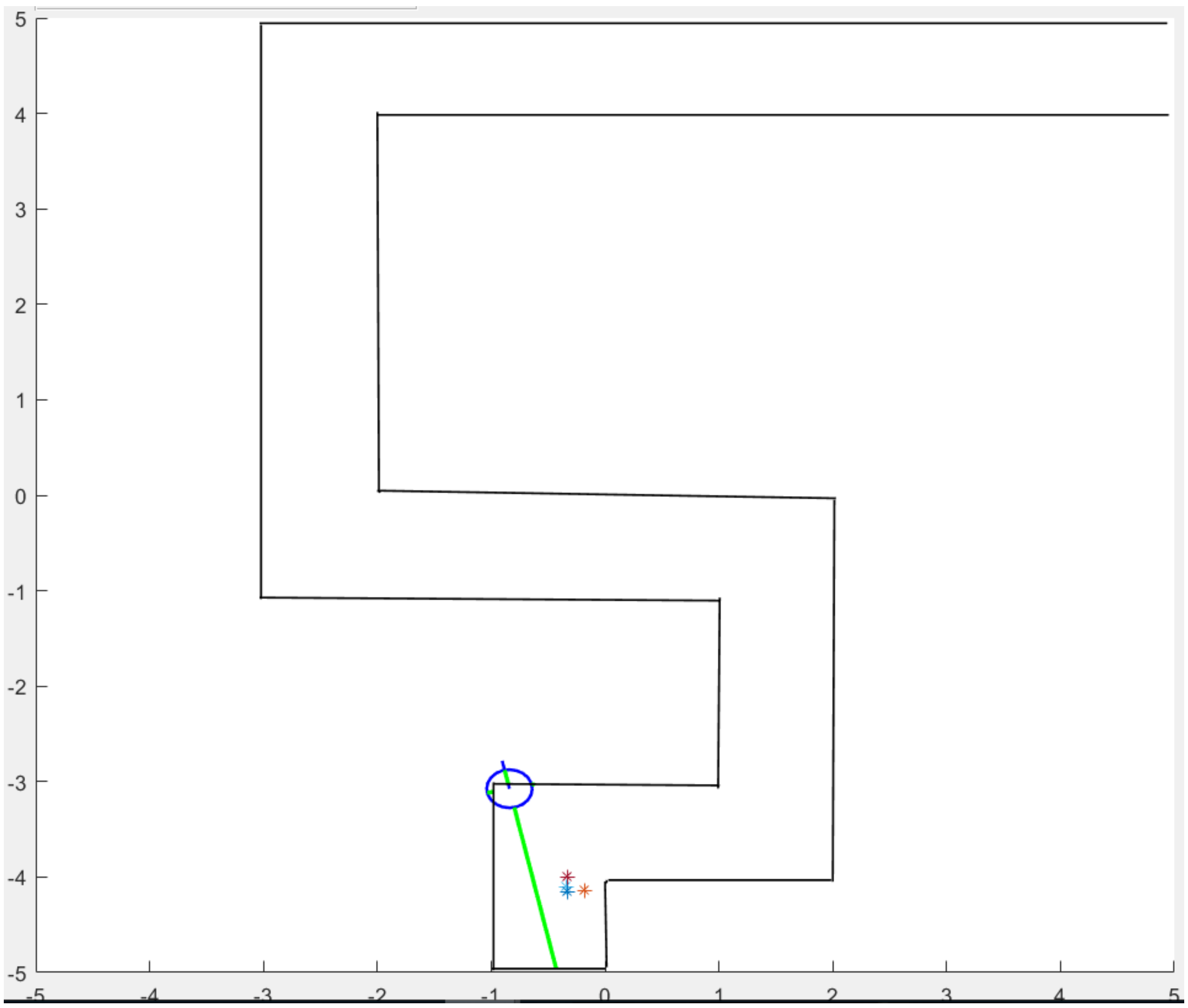
While changing the distance did not stop the robot from getting stuck and return an error, it has certainly did improve how long the robot can keep moving in the map before getting stuck. In this particular case, it took more than 20 minutes before the robot got stuck.

Case B



Although it didn’t solve the wall hugging problem here either, an improvement can still be observed in test case #1 map here.

Case C



However, in the test case #2 map, the robot still runs into the same problems even more frequently, because the maze is just too narrow.

7.0 Conclusions

After running all tests, and observing the results, we came up with the following conclusions:

On the MATLAB script:

- The code functions without problem, however, it does not account for obstacles at angles other 0 (front), 90 (right), 180 (left), and 270 (rear).
- The distance for stopping when approaching a wall should be modified depending on the map the robot is in.
- The conditional statements in our MATLAB script covers most the situation a robot could encounter, but there may be more scenarios that can be added.

On Possible Solutions for the Wall Hugging Problem:

- The sonar sensor in this simulation only have four sides (front, rear, right, left), adding more sonar sensors at 45^o between each two sides (e.g. between front and left) will improve the robot's ability to detect obstacles. However, that may require rewriting the conditional statements, making them more complex.
- Another possible solution is to use the LIDAR detection system instead of Sonar.
- Bump sensor can be also used, but they it may give a slower performance.
- The minimum distance required to stop when approaching a wall can be written as a variable.

Other ideas:

An idea that we wanted to implement but we did not know how to code it, is to program the robot to avoid going back to areas of the map that it had already explored. This would have allowed the robot to explore the whole map before reaching its objective. Such robot would have many useful applications, such as search and rescue (e.g. a burning building), exploring hazardous areas (e.g. radioactive areas), or buildings security and securing perimeters.