CS3105 Report - AI Robotics

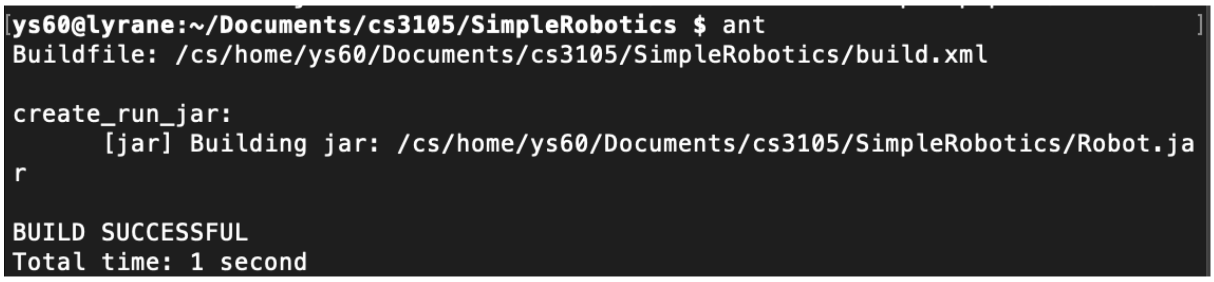
matriculation number: 160021429

1. Overview

Basically, the main aim of this practical is to improve the robot to make a fractional progress on both free space and a potential field with several obstacles. For part 1, the robot should be able to travel the free space by using the fractional progress. And for the part 2, the robot should be able to handle obstacles by combining winding and unwinding.

1 - 1. How to compile

Basically, to compile this project, you need “ant”, which is a java build tool. Apparently, ant is installed in the lab machine, thus, you would be able to compile and build jar file on the lab machine. Otherwise, you have to install ant builder. Once you type “ant”, the ant builder will build the “Robot.jar” file. It is executable jar file, so you could run the jar file either by using “java -jar Robot.jar” command or double-click the jar file.



```
ys60@lyrane:~/Documents/cs3105/SimpleRobotics $ ant
Buildfile: /cs/home/ys60/Documents/cs3105/SimpleRobotics/build.xml

create_run_jar:
      [jar] Building jar: /cs/home/ys60/Documents/cs3105/SimpleRobotics/Robot.ja
r

BUILD SUCCESSFUL
Total time: 1 second
```

2. Free Space Travel (Part 1)

The main topic of the part 1 is a free space travel with a fractional progress planner. Apparently, a pure definition of the “free space travel” is that the obstacle potential is below zero. So, once we implement the fractional progress properly, the robot should be able to travel the free space and go to the estimated goal properly. To achieve this, we were asked to make a new version of evaluateSamplePointsArc(). So, I wrote a method called “evaluateSamplePointsArcForFractionalProgress()”, which is in the PotentialFieldsRobot class.

According to “note.pdf” file on the studres, the fractional progress uses the fraction “f/(p+f)” or “p/(p+f)” where p is the past up to the end of the candidate next transition and f is the estimated future. So, the concept of the fractional progress is a way using the sample point by using the past cost and the estimated future cost to evaluate the efficiency of the move.

When I tried to implement the fractional progress, I tried both “f/(p+f)” and “p/(p+f)” to check if there is any minimal differences. Fortunately, they worked exactly same. So, I just used “f/(p+f)” one, so that I could use the minIndex() method to make the maximum fractional progress.

2 - 1. Move parameters

To implement the fractional progress, I used “the length of the first arc” as f and “sum of the length of second and third arcs and the obstacle potential” as p. Therefore, the move parameters of the fractional progress planner is the length of the arcs and the obstacle potentials of the sample points.

2 - 2. Motion mechanisms

Basically, fraction that used for the fractional progress FP is “FP = Estimated Future Cost / (Estimated Future Cost + Past Cost)”. And, we could consider the past cost as “Past *Cost* = Length of the First Arc (Between Current Pos and Candidate Move)”. Also, we could consider the estimated future cost as “Estimated Future Cost = Length of the Second Arc (Between candidate move and edge of sensor range closest to goal, which is a known future cost) + Length of the Third Arc (Between Edge of Sensor range and goal, which is an unknown future cost) + Obstacle Potential (result of a function where OP starts at 0 an increases exponentially as distance to an obstacle increases)”.

2 - 2. How to use Fractional Progress planner?

To use the fractional progress planner, you need to click “Arc” button for the planner, and then click “Fractional Progress” button, which is located at the rightmost part of the GUI. The fractional progress only works with the Arc planner, so it would not work if you press the Euclidean button for the Euclidean planner mode.

2 - 3. Comparing the FP planner with others

To test the efficiency of the robot for free space travel, I tested with starting point at (0, 0), and goal at (500, 500) with no obstacles on the field. As you could see in <figure 1> in the appendix, the total turns of the paths, which is the value of “path smoothness”, of the fractional progress planner was 0.28461871803020916, and the value of travelled distance was 650.

However, with the greedy arc planner, the path smoothness rating was 0.2704113303115399, which is lower than the non-greedy planner, however, the total distance was 670, which is larger than the non-greedy planner’s value. You could see this in <figure 2> in the appendix.

Moreover, as you could see below, the greedy euclidean planner had same path smoothness value and travelled distance value (see <figure 3>).

However, if there is at least one obstacle on the field, then the non-greedy arc planner (fractional progress planner) makes the best move, whose value of path smoothness rating and distance travelled are the smallest (<figure 4> <figure 5> <figure 6> in appendix).

So, it is clear that the greedy arc based planner is the worst one for the free space travel. There would be some exceptional cases, however, it is possible to say that the greedy euclidean based planner works better than the non-greedy arc based planner, in general.

3. Obstacle Navigation (Part 2)

Apparently, the fractional progress could handle normal obstacles, as one of the factors of the fraction that is used for the fractional progress is the obstacle potential. However, the potential fields robot with the non-greedy arc based planner that I implemented in the part 1 had some problem if there is a c curve in the field. Apparently, this is a local minimum problem that is one of the most important problem that the potential field algorithm has.

3 - 1. Local minimum problem

Fractional Progress navigation techniques are vulnerable to the local minimum problem because such obstacles create areas of the problem space, that are not the goal, which have higher Fractional Progress values than any of the others around them. Therefore, once the goal based agent has entered these areas, Fractional Progress only navigation will not allow them to leave the local minimum. In this practical, this manifested as the robot being stuck in a loop. Basically, the robot is unable to leave the local minimum because the fractional progress inside the C shape is higher than that of the entrance/exit.

3 - 2. BUG algorithm to solve local minimum problem

To overcome this problem, I used the BUG algorithm. Basically, if the obstacle potential of the robot's coordinate is greater than the precomputed threshold, then the robot will activate the bug mode, which makes the robot runs like a BUG algorithm.

Taking inspiration from the provided POTBUG paper, where an algorithm that combines Fractional Progress move evaluation is combined with BUG-like obstacle navigation behaviour in order to navigate complex and difficult pathways. These include tunnels and concave obstacles. BUG-like techniques are necessary because these obstacles create local minimum problem, which means that raw Fractional Progress and Steepest Gradient Descent navigation cannot overcome them.

3 - 3. Design and Implementation of BUG algorithm

The bug mode takes a great deal of inspiration from the PotBug paper, but has been adapted to the needs of the practical. Basically, the bug mode is triggered when the robot approaches an obstacle at a distance close enough that the obstacle potential at the robot's position is higher than a set threshold, which I set with (robot_radius / 4).

The robot then follows a new algorithm for bug mode, until the bug mode is disengaged when the Robot obstacle potential falls below the "threshold / 4", when the robot is once again in the free space.

First, the program checks to see if Bug Mode is required. When the robot enters Bug mode, its position is recorded as a variable IntPoint called 'startBugPosition', and the aim of this variable is to record the position of the robot when it gets into the bug mode. Then, the program checks to see if Bug mode has been activated. If not, then the robot simply chooses the move which maximises the Functional Progress.

If the bug mode has been activated, then the robot splits its available moves into 3 groups: unviable, unwinds and winds. If the move is moving to an area of low obstacle potential and is closer to the goal (by Euclidean distance) than the 'Hit Point', the move is classed as an unwind. If the move has an obstacle potential less than the threshold but cannot be classed as an unwind, it is classed as a wind. Apparently, all other moves are unviable, thus the program will ignore all those unviable points.

After that, the bug mode algorithm will select the most preferred move. First, if the unwind is available, the point with the largest improvement in the fractional progress will be selected as a preferred move. If the unwind is not available, however, the winding is available, then the winding point with the highest obstacle potential (which must still be below the threshold potential) will be selected. If both winding and unwinding are not available, then the bug mode algorithm will just select the point that makes the largest improvement in the fractional progress.

After select the most preferred move, the program will check if it is possible to leave the bug mode. If it is possible, then the program will run with the normal fractional progress planner that was used in the part 1. Otherwise, the program will keep use bug algorithm until the robot gets out from the c curve.

3 - 4. Winding and Unwinding?

The winding and unwinding are things that are used for improving the fractional progress algorithm to handle local minimum problems. Winding is to do with obstacle navigation when the robot is being forced to turn because the robot would otherwise get too close to the obstacle. According to the lecturer, unwinding is to do with undoing some of the winding when there is the opportunity to do so without increasing the obstacle potential excessively, such as going round corners. Once sufficient unwinding has been done, the robot will have got rid of the forced turns, will have got round the obstacle corners, and be ok to head for the goal. Winding and unwinding determine which sample points are selected from for fractional progress not the other way round.

Basically, this is why my bug mode algorithm checks if the unwinding is available first. Because, if there is any chance to undo the winding, we should unwind the robot to get out from the c curve.

3 - 5. Move parameters

Basically, the key parameter that I added for the part 2 is the threshold, which is used to check whether the program should enter to the BUG mode or not. Moreover, the program checks if it is okay to leave the BUG mode by comparing the obstacle potential of the current coordinate with the threshold. And the program calculates the threshold by using the robot radius, thus, it is possible to say that the key move parameter that is used to implement the BUG mode is the “robot radius”.

As the fractional progress method that is used in the part 2 is an improved version of the fractional progress method in part 1, thus, the length of the arcs and the obstacle potentials of the sample points are also the move parameters.

3 - 6. Testings

I tested my improved version of non-greedy arc based planner with the hard course, that contains several obstacles and a c curve, which invokes the local minimum problem. So, if the bug mode works properly, then the potential fields robot should be able to get to the goal with reasonable length of travelled distance.

As you could see in <figure 7>, the robot could get to the goal even there is a c curve on the field.

To check if the small size of robot works properly with the small sensor range, I setted the robot size to 31, and robot sensor range to 62, which is double of 31. As you could see in <figure 8>, it works properly.

Next, I tested if the planner works properly with robot size = 30 and sensor range = 60. As you could see in <figure 9>, it works, however, not really efficient. I think it is basically because that my threshold value is calculated from the robot size, thus, if the both robot size and the sensor range is small enough, then my BUG algorithm does not work efficiently.

Furthermore, I tested my program by generating a new c curve, which is more closed. So, what I have done is make add a 5 point shape at (100, 100), (100, -300), (-400, -300), (-400, 400), (400, 400) with a starting point at (-100, 100), and a goal at (300, -100). As you could see in <figure 10>, my robot could handle it.

I also tested my program with some other c curves, and the program worked properly (see <figure 11>. However, I found that it my program does not work if the severe c curves when the robot radius is smaller than 30 and sensor range is smaller than 60.

So, according to the tests, my program works properly with all c curves in most cases. However, the planner is not able to handle shallow c curve when the robot radius is smaller than 25 and sensor range is smaller than 55, and the planner is not able to handle severe c curve when the robot radius is smaller than 30 and sensor radius range is smaller than 60.

4. Overview Question

4 - 1. Local minimum problem

Question 1:

What is/are the likely reason(s) as to why the set book by Russell and Norvig says potential fields suffer from the Local Minimum problem?

Answer:

Apparently, the local minimum problem issue is one of the most significant challenges for the potential field algorithm. The local minimum problem, which is also called as local minima problem, occurs when all artificial forces (attractive and repelling) cancel each other out such as in situations when obstacles are closely spaced. As I mentioned in the section for part 2, the robot is unable to leave the c curve when the fractional progress inside the c curve is higher than that of entrance/exit.

In this practical, a local minimum problem was occurred when we tried to run our basic fractional progress method with the c curve. Thus, we were asked to improve this problem by combining the winding and unwinding in part 2.

4 - 2. Further development

Question 2 :

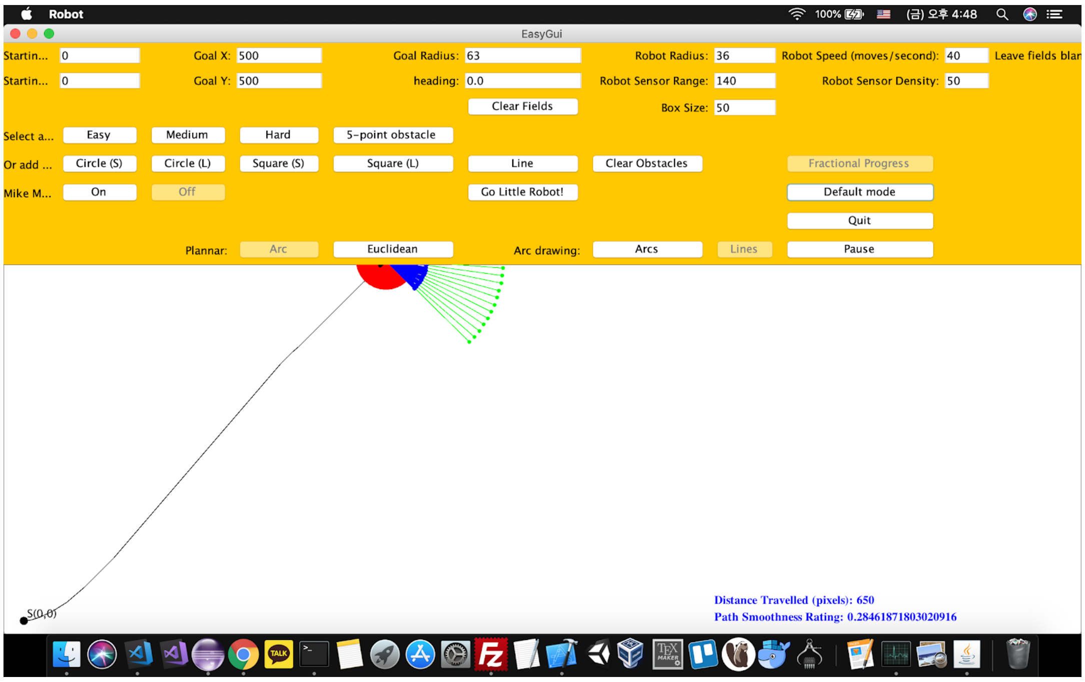
How would you develop your system further to deal with any problems you found with your final optimal system?

Answer:

Apparently, my robot does not work properly when the both robot radius and robot sensor range are extremely small. For instance, if you set the robot radius as 10 and robot sensor range as 30, then the robot will get stuck in the c curve. Therefore, it would be nice to figure out that problem and fix that error.

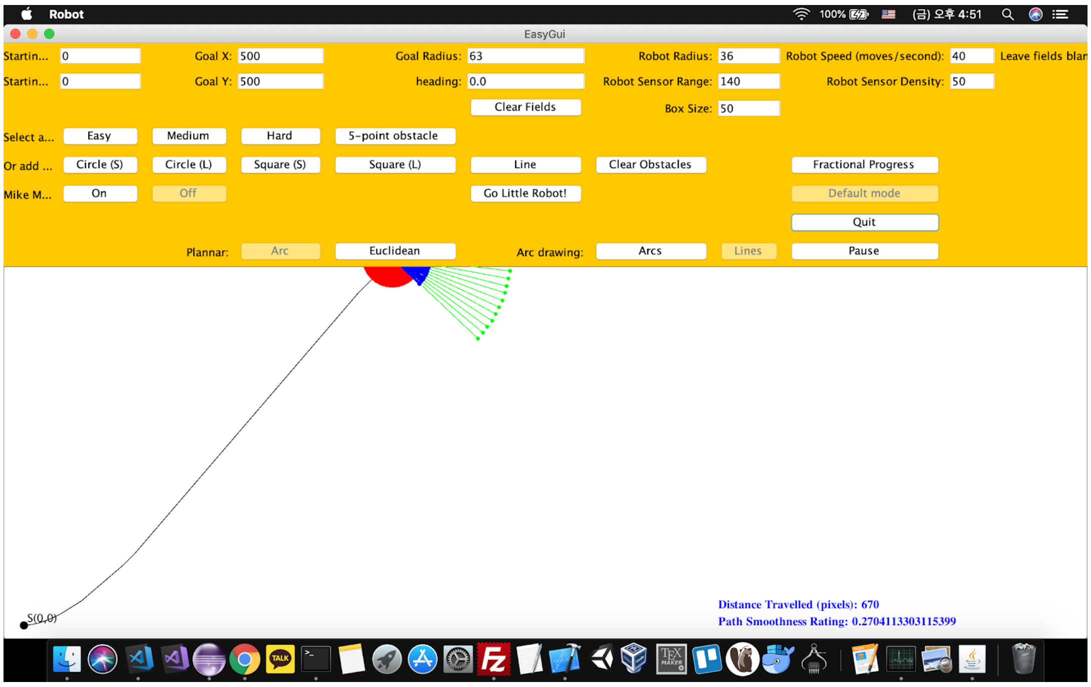
Moreover, most people found that the out of memory exception occurs when the you make your robot have a long journey. Apparently, the problem is because that the invoked methods are generating too many instances before the JVM calls the garbage collection method, so that the heap memory is out of use. Henceforth, it might be worth to find the way to handle this memory issue, so that we could use our simulator with some longer journey.

5. Appendix


Startin... 0
Startin... 0
Goal X: 500
Goal Y: 500
Goal Radius: 63
heading: 0.0
Robot Radius: 36
Robot Sensor Range: 140
Box Size: 50
Robot Speed (moves/second): 40
Robot Sensor Density: 50
Leave fields blan

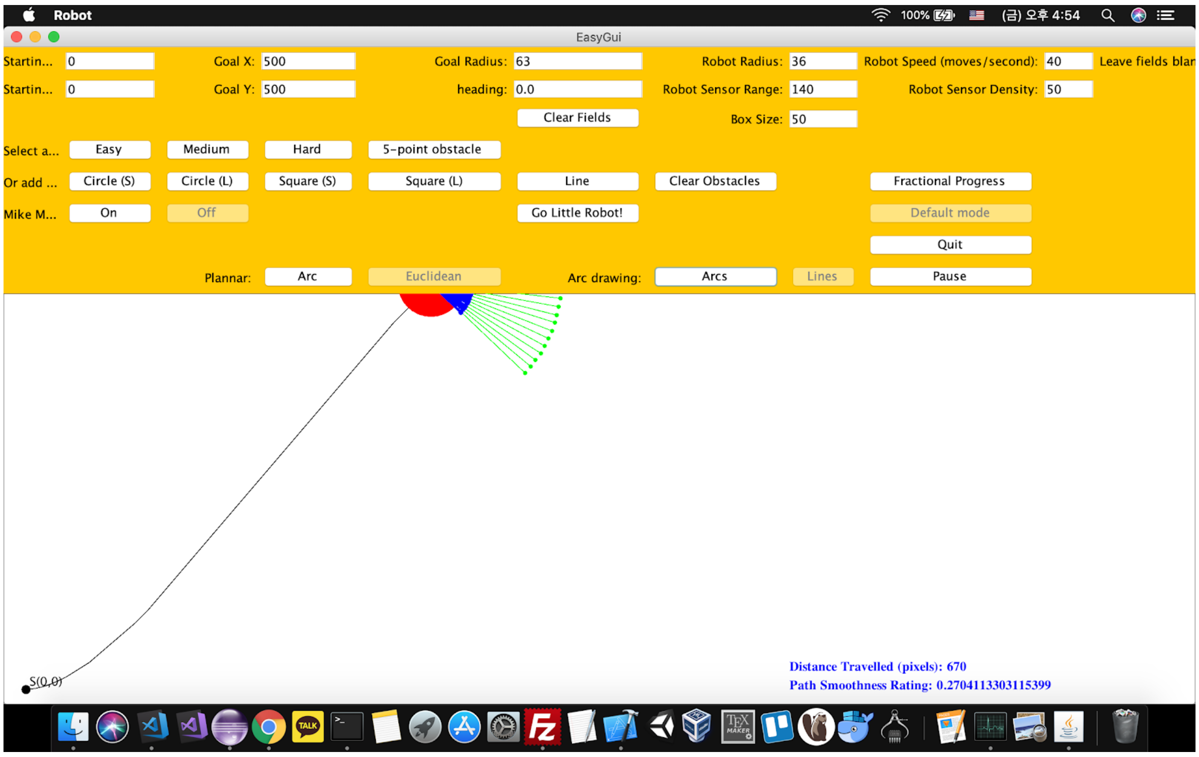
Distance Travelled (pixels): 650
Path Smoothness Rating: 0.28461871803020916

<figure 1>


Startin... 0
Startin... 0
Goal X: 500
Goal Y: 500
Goal Radius: 63
heading: 0.0
Robot Radius: 36
Robot Sensor Range: 140
Box Size: 50
Robot Speed (moves/second): 40
Robot Sensor Density: 50
Leave fields blan

Distance Travelled (pixels): 670
Path Smoothness Rating: 0.2704113303115399

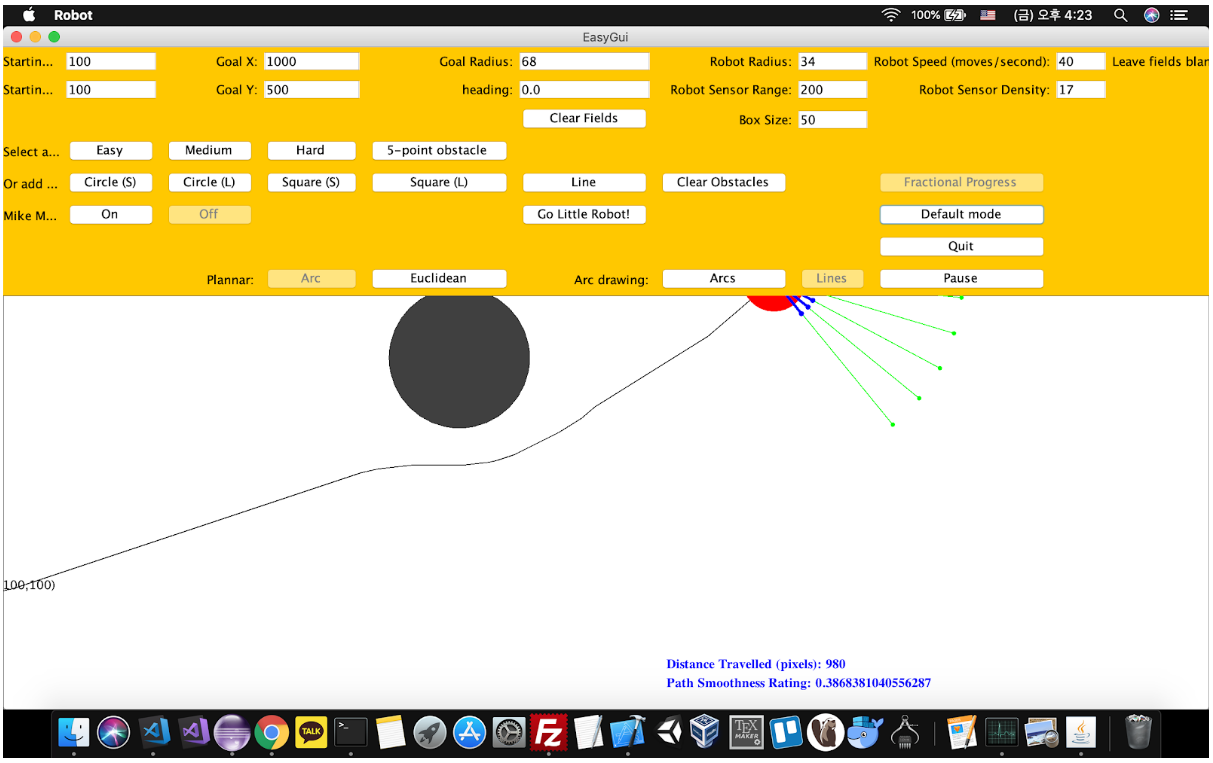
<figure 2>



Startin... 0
Startin... 0
Goal X: 500
Goal Y: 500
Goal Radius: 63
heading: 0.0
Robot Radius: 36
Robot Sensor Range: 140
Box Size: 50
Robot Speed (moves/second): 40
Robot Sensor Density: 50
Leave fields blan
Select a...
Or add ...
Mike M...
Plannar:
Arc drawing:

Distance Travelled (pixels): 670
Path Smoothness Rating: 0.2704113303115399

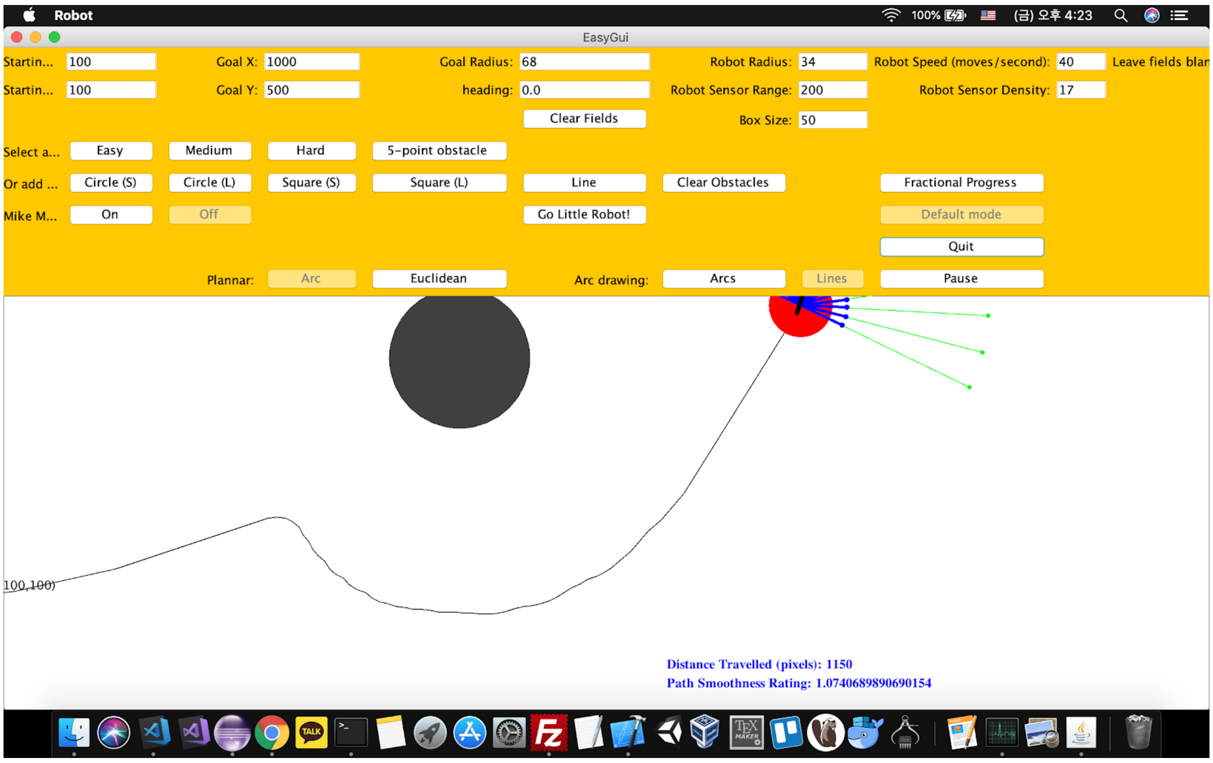
<figure 3>



Startin... 100
Startin... 100
Goal X: 1000
Goal Y: 500
Goal Radius: 68
heading: 0.0
Robot Radius: 34
Robot Sensor Range: 200
Box Size: 50
Robot Speed (moves/second): 40
Robot Sensor Density: 17
Leave fields blan
Select a...
Or add ...
Mike M...
Plannar:
Arc drawing:

Distance Travelled (pixels): 980
Path Smoothness Rating: 0.3868381040556287

<figure 4>



Startin... 100 Goal X: 1000 Goal Radius: 68 Robot Radius: 34 Robot Speed (moves/second): 40 Leave fields blan

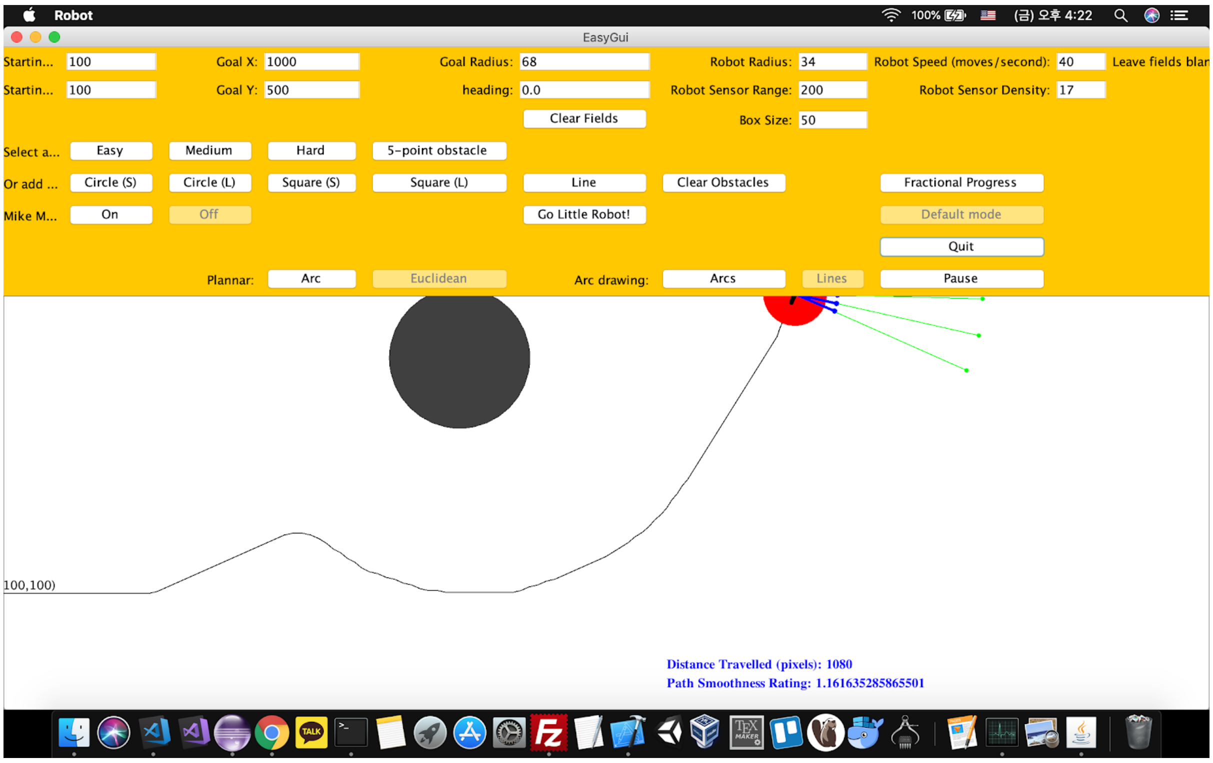
Startin... 100 Goal Y: 500 heading: 0.0 Robot Sensor Range: 200 Robot Sensor Density: 17

Box Size: 50

Distance Travelled (pixels): 1150

Path Smoothness Rating: 1.0740689890690154

<figure 5>



Startin... 100 Goal X: 1000 Goal Radius: 68 Robot Radius: 34 Robot Speed (moves/second): 40 Leave fields blan

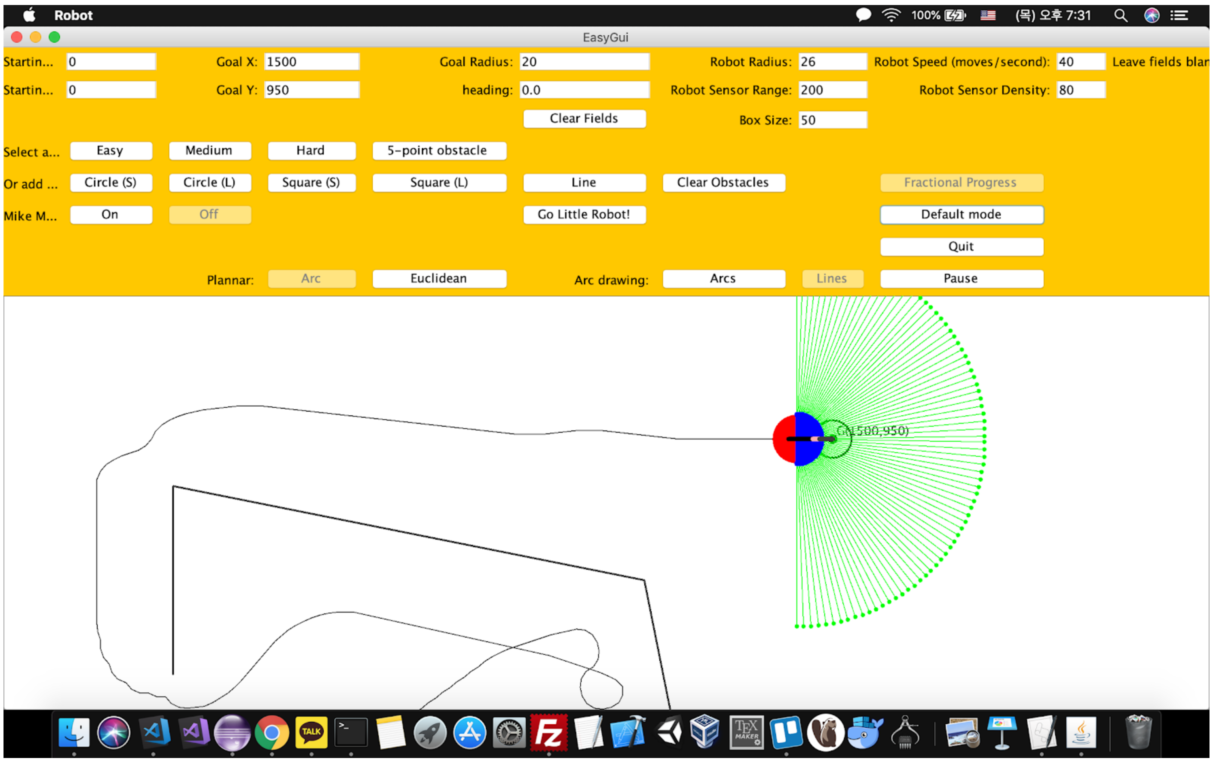
Startin... 100 Goal Y: 500 heading: 0.0 Robot Sensor Range: 200 Robot Sensor Density: 17

Box Size: 50

Distance Travelled (pixels): 1080

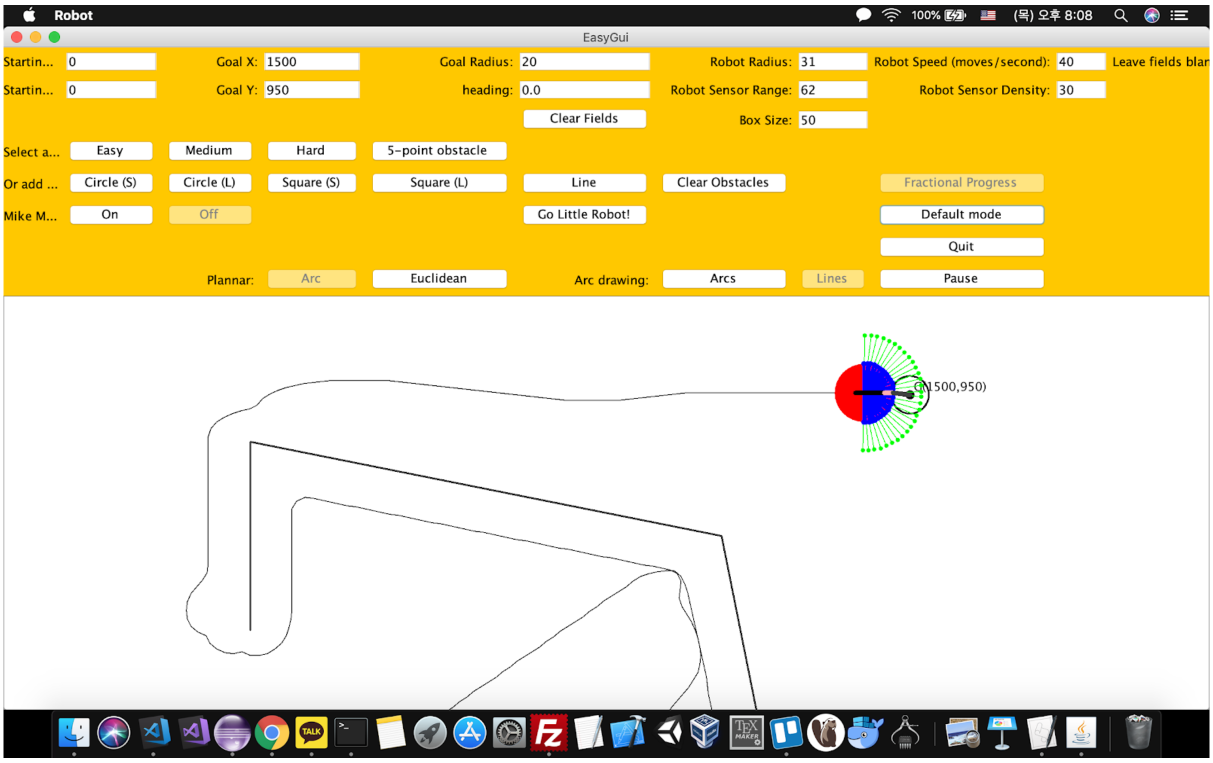
Path Smoothness Rating: 1.161635285865501

<figure 6>



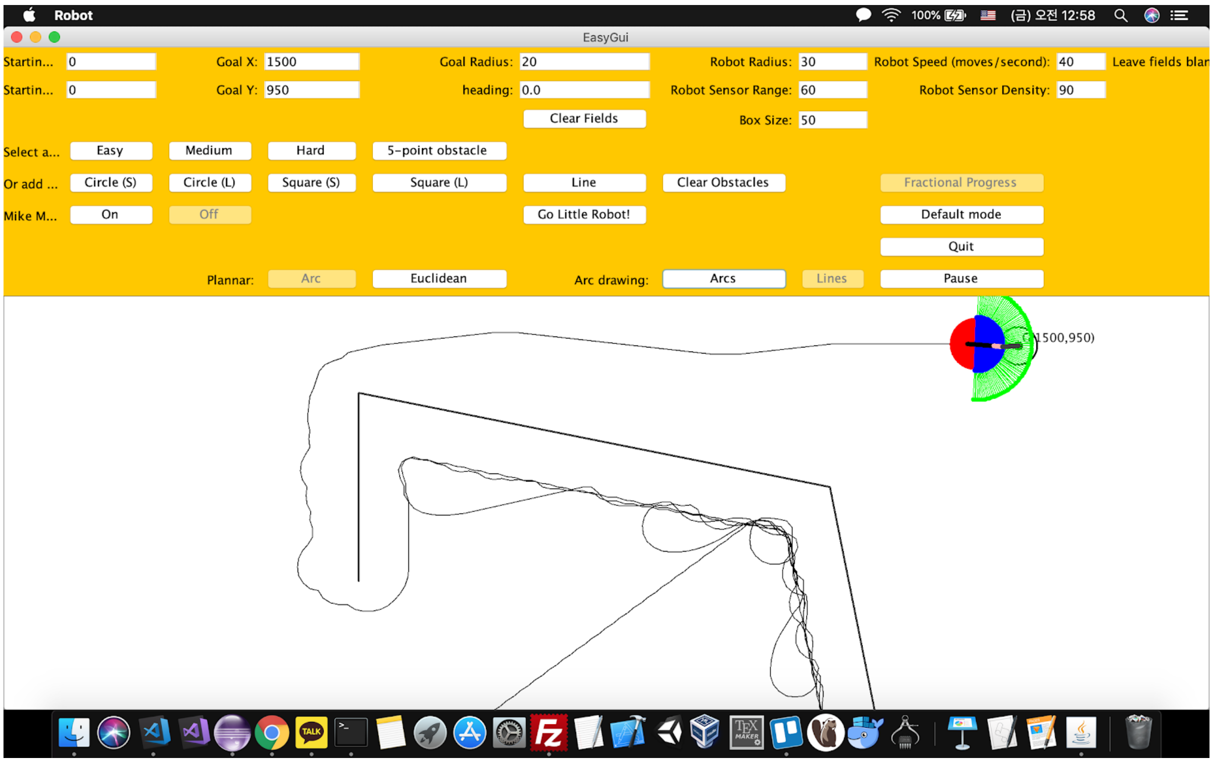
Startin... 0
Startin... 0
Goal X: 1500
Goal Y: 950
Goal Radius: 20
heading: 0.0
Robot Radius: 26
Robot Sensor Range: 200
Box Size: 50
Robot Speed (moves/second): 40
Robot Sensor Density: 80
Leave fields blan
Plannar: Arc
Arc drawing: Arcs

<figure 7>

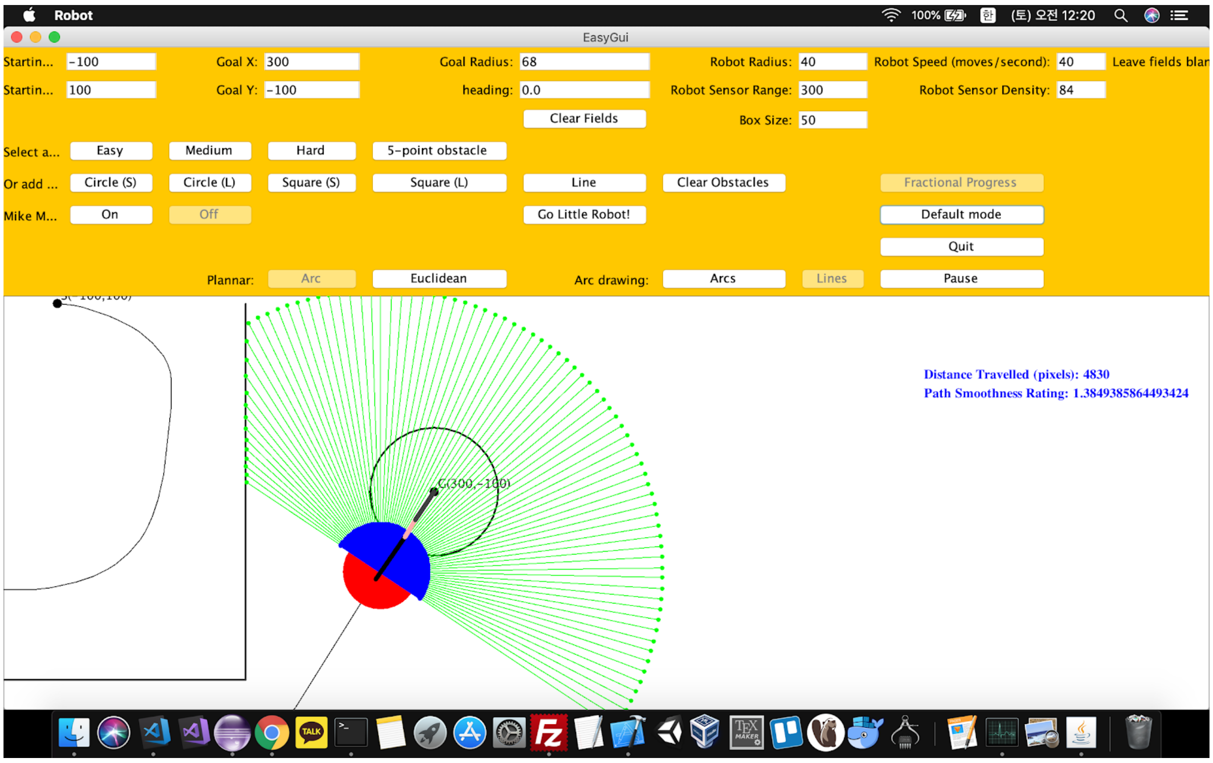


Startin... 0
Startin... 0
Goal X: 1500
Goal Y: 950
Goal Radius: 20
heading: 0.0
Robot Radius: 31
Robot Sensor Range: 62
Box Size: 50
Robot Speed (moves/second): 40
Robot Sensor Density: 30
Leave fields blan
Plannar: Arc
Arc drawing: Arcs

<figure 8>

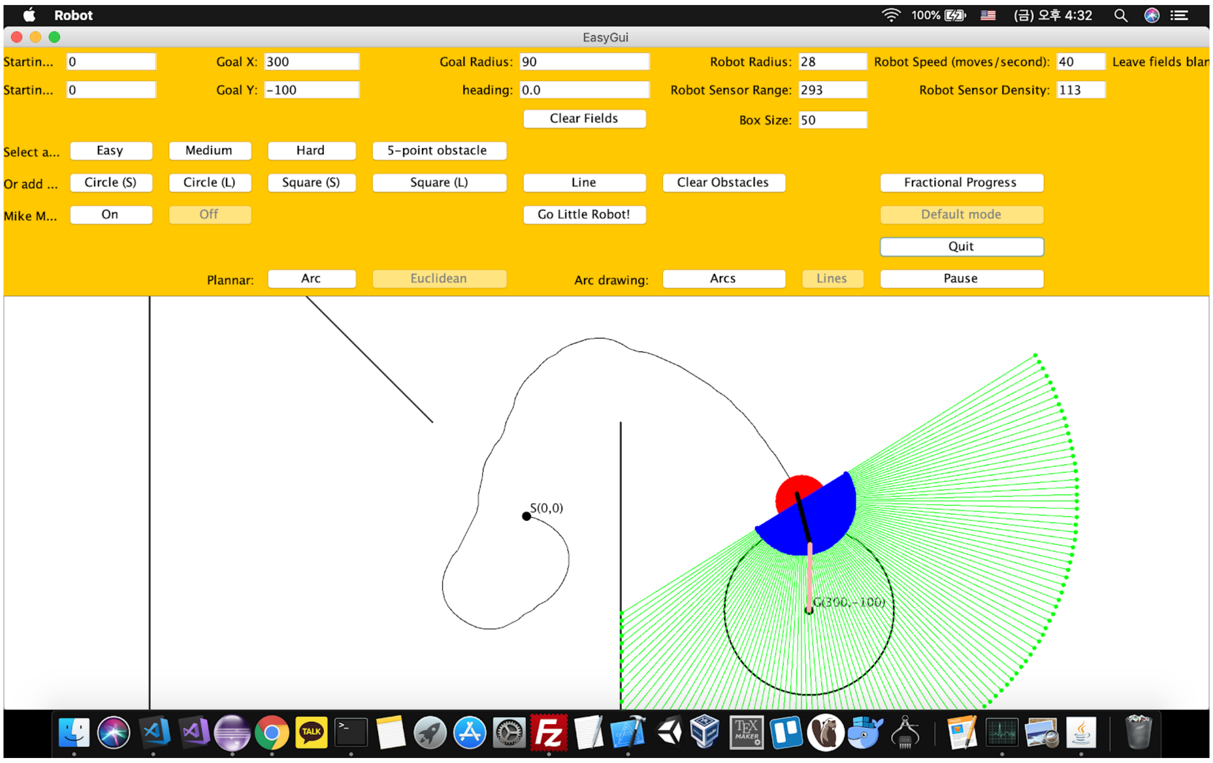

Startin... 0
Startin... 0
Goal X: 1500
Goal Y: 950
Goal Radius: 20
heading: 0.0
Robot Radius: 30
Robot Sensor Range: 60
Box Size: 50
Robot Speed (moves/second): 40
Robot Sensor Density: 90

<figure 9>


Startin... -100
Startin... 100
Goal X: 300
Goal Y: -100
Goal Radius: 68
heading: 0.0
Robot Radius: 40
Robot Sensor Range: 300
Box Size: 50
Robot Speed (moves/second): 40
Robot Sensor Density: 84

Distance Travelled (pixels): 4830
Path Smoothness Rating: 1.3849385864493424

<figure 10>



Startin... 0
Startin... 0
Goal X: 300
Goal Y: -100
Goal Radius: 90
heading: 0.0
Robot Radius: 28
Robot Sensor Range: 293
Box Size: 50
Robot Speed (moves/second): 40
Robot Sensor Density: 113
Leave fields blan

<figure 11>