

CS5011- Artificial Intelligence Principles

AI in Action

Practical 2 –Report

Student 150021237

Date 26.10.2015

Table of Contents

Introduction	2
Extent of this Submission.....	2
Run instructions	2
Starting the Application	2
Keyboard Shortcuts.....	2
Design.....	3
Robot Implementation.....	3
Sensors, Heading and Sample Points	3
Differential Drive	3
Distance Metrics	4
Distance Metrics	4
Part 1: Free Space Travel	4
Brief Behaviour Description	4
A_FreeSpaceBotWithAngle / A: Free Space (Angle)	4
C_FreeSpaceBotWithSamplePoints / C:Free Space (Samples)	5
Results	5
Part 2 – Obstacle Navigation.....	5
Brief Behaviour Description	5
B_ObstaclePanicBotWithAngle / B: Dodger (Angle)	5
D_AvoidObstacleWithSamplePoints / D:Potential Fields	5
E_FractionalProgressWithSamplePoints / E:Fractional Progress	6
F_PelletBot / F: Pellet Bot	6
Results	6
Conclusion.....	7
Global Map vs Local Sensing	7
Bibliography	7
List of Figures	7
Appendix	8

Introduction

This report is part of the practical 2 assignment for the CS5011 module at the University of St Andrews. The provided Java source code includes the JTS Topology Suite for processing geometry [1]. Besides that, everything has been written by the submitting student. This project takes not advantage from the available Easygui toolkit and the potential fields/rapidly exploring random trees examples provided by the lecturer. An own 2D robot simulation system according to the specifications of the practical has been developed in Java 1.8. The application has been successfully tested on a Linux lab machine.

Extent of this Submission

This submission consists of the following elements.

- This report
- Runnable jar file Robot.jar
- Several test maps

The following robot implementations are provided and analysed.

- A_FreeSpaceBotWithAngle
- B_ObstaclePanicBotWithAngle
- C_FreeSpaceBotWithSamplePoints
- D_AvoidObstacleWithSamplePoints
- E_FractionalProgressWithSamplePoints
- F_PelletBot

The following features are notable about the developed application.

- Place the robot and adjust the heading by holding your mouse click
- Place obstacles and the goal
- Place circular moving obstacles
- Show the number of steps and total distance travelled
- Start, stop and reset the robot and moving obstacles
- Show all the paths travelled
- Save and load maps
- Dynamically choose the agent implementation
- Dynamically choose the distance metric
- Increase and decrease the robot size
- Clear the screen

Run instructions

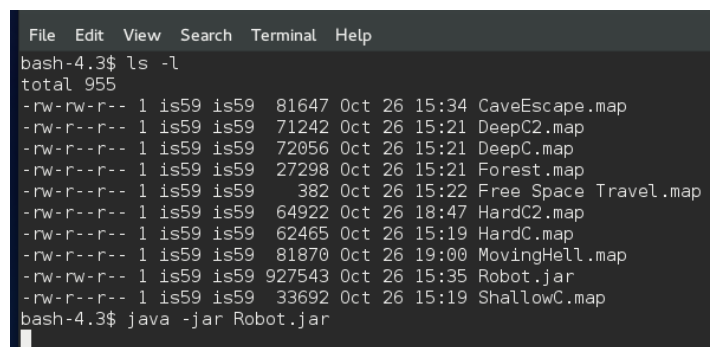
Starting the Application

The application can be run with the following command on a Linux lab machine or any similar environment with a JRE supporting Java 1.8.

`Java -jar Robot.jar`

Keyboard Shortcuts

For a faster user experience, several keyboard shortcuts have been implemented.



```
File Edit View Search Terminal Help
bash-4.3$ ls -l
total 955
-rw-rw-r-- 1 is59 is59 81647 Oct 26 15:34 CaveEscape.map
-rw-r--r-- 1 is59 is59 71242 Oct 26 15:21 DeepC2.map
-rw-r--r-- 1 is59 is59 72056 Oct 26 15:21 DeepC.map
-rw-r--r-- 1 is59 is59 27298 Oct 26 15:21 Forest.map
-rw-r--r-- 1 is59 is59 382 Oct 26 15:22 Free Space Travel.map
-rw-r--r-- 1 is59 is59 64922 Oct 26 18:47 HardC2.map
-rw-r--r-- 1 is59 is59 62465 Oct 26 15:19 HardC.map
-rw-r--r-- 1 is59 is59 81870 Oct 26 19:00 MovingHell.map
-rw-rw-r-- 1 is59 is59 927543 Oct 26 15:35 Robot.jar
-rw-r--r-- 1 is59 is59 33692 Oct 26 15:19 ShallowC.map
bash-4.3$ java -jar Robot.jar
```

Figure 1 - Make sure the maps are in the same folder as the jar file when running the Application.

Key	Action
r	Place robot
g	Place goal
o	Place obstacle
m	Place moving obstacle
s	Start/Stop robot
d	Reset robot to starting position
c	Clear

Figure 2 - Keyboard Shortcuts for the developed application

Design

The implementation of this practical follows general software development principles and makes use of several object-oriented programming (OOP) patterns like the model view controller (MVC), the strategy pattern and the singleton pattern among others. The design is very easy to extend and maintain. It was designed with the purpose to construct a basis which can be used by future students for their potential field research as individual agent behaviour can be easily added. I decline the submitted version to be used in future classes or any other work though I'd be happy to arrange it for this specific purpose.

Robot Implementation

The robot has been implemented to match the specifications of the practical.

Sensors, Heading and Sample Points

The robot model is implemented by the *model.Robot* class. The robot has a centre coordinate, as well as, a front coordinate. Based on those, the sensor rays and sample points are evenly distributed on the front facing side of the robot. If a sensor ray is intersecting an obstacle, it will shorten its length in respect to the intersection point and change its colour from green to red. The sample points are uniformly arranged around the robot. The following elements are the most important information provided to the behaviour classes for processing the world.

- Robot centre and goal coordinates
- Current left and right wheel speed
- Current angle with respect to the x-axes
- Amount of steps taken and distance travelled
- The length of every sensor ray and its coordinates
- The coordinates of the sample points

The behaviour classes are only allowed to manipulate the robots left and right wheel speed to interact with the world. The wheel speed is limited by an upper and lower boundary (V_MIN and V_MAX).

Differential Drive

The robot uses a differential drive to move. It can be changed instantaneously. If the speed is too small or too big it will change it to the according limit. The robot movement is calculated in the *nextStep* function of the *model.robot* class. It calculates the angle and distance of the new position from the left and right wheel speed by using the following formulas [1]. The sample points are positioned uniform in front of the robot and are not dependent on the actual reach of the robot.

$$\omega = \frac{(v_{Left} - v_{Right})}{\text{wheel distance}}$$

Figure 4 - Calculates the angle to the destination based on the left and right wheel speed.

$$d = \frac{(v_{Left} + v_{Right})}{2} * t$$

Figure 3 - Calculates the distance to the destination based on the average speed and the time t.

Distance Metrics

The distance metric is used to evaluate every sample point during every step, so that, the best sample point may be chosen by the implemented behaviour. The following metrics have been implemented in the *bots_Behaviour* class. This class is an abstract class which every behaviour must extend.

Every metric needs 2 coordinates. The first one is the sample point coordinate and the second one is the goal coordinate. The coordinate, in this case, is equivalent to the 2-dimensional position vector.

- The linear distance metric is the Euclidean distance metric.

$$linear(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$$

Figure 5 - Euclidean distance used for the linear distance metric.

- The quadratic distance metric is the squared Euclidean distance metric.

$$quadratic(p, q) = (p_1 - q_1)^2 + (p_2 - q_2)^2$$

Figure 6 - Euclidean distance used for the linear distance metric.

- The arc distance metric is based on the relation between the Euclidean distance from the starting point s and the Euclidean distance to the goal.

$$d_1 = linear(s, p)$$

$$d_2 = linear(p, q)$$

$$d_3 = linear(s, q)$$

$$arc(p, q, s) = \left(1 - \frac{d_1}{d_1 + d_2}\right) * d_3$$

Figure 8 – Arc distance metric based on the relation between the distance to the start point, as well as, to the goal.

$$arc(p, q, s) = \frac{d_2 * d_3}{d_1 + d_2}$$

Figure 7 - Short version of Figure 8.

Distance Metrics

Several maps have been created to test the different robot behaviours. The maps are located in the jar file itself and can be created with the Save Button in the graphical user interface (GUI). The maps are loaded when the application starts. A saved map will not show up until the application is restarted. The following maps are provided:

- FreeSpaceTravel
- Forest
- ShallowC
- HardC
- DeepC1
- DeepC2
- MovingHell
- CaveEscape

Part 1: Free Space Travel

This part analyses the free space travel implementations. This comprises the behaviours A and C. A is implemented with inverse kinematics whereas C can use the 3 different distance metrics to choose one of its samples points.

Brief Behaviour Description

A_FreeSpaceBotWithAngle / A: Free Space (Angle)

A behaviour which is based on inverse kinematics. It calculates the maximum turning angle to align the robot and its front with the goal. It then calculates the number of steps by multiplying the Euclidean distance to the goal with the golden ratio and then divides this distance by the current average speed of both wheels. The result is a nice smooth movement towards the goal.

C_FreeSpaceBotWithSamplePoints / C:Free Space (Samples)

This behaviour uses the sample points which are evenly distributed in front of the robot. It evaluates every sample point by using one of the 3 distance metrics elaborated in chapter *Distance Metrics*. The behaviour chooses the sample point with the smallest distance.

Results

Every configuration was able to reach the goal. The results show how different a smooth curve can be. Considering that A is actually inverse kinematics I would recommend the linear version of C if sharp turns are permitted. If sharp turns are a problem one might choose the arc metric to reach the goal.

Behaviour	Steps
A	583
C-linear	402
C-quadratic	402
C-arc	652

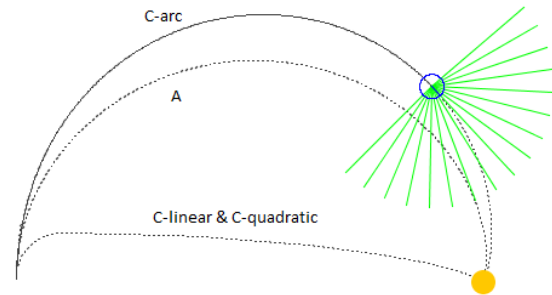


Figure 9 - Demonstration of free travel results.

Part 2 – Obstacle Navigation

This part analyses robot behaviours which are able to dodge obstacles. This comprises the behaviours B, D, E and F whereas B is implemented with inverse kinematics and all the others using sample points which can be chosen by using one of the three distance metrics.

Brief Behaviour Description

B_ObstaclePanicBotWithAngle / B: Dodger (Angle)

This robot is based on A. It still uses inverse kinematics. It calculates the maximum turning angle and calculates the new wheel speeds to turn towards the angle. This version uses a greedy approach to turn. The path travelled will, therefore, be not smooth at all. To avoid obstacles, it checks if there are any obstacles in a radius of 50 degrees in front of it. If there are obstacles found in that sector, it will panic and forget the goal. The new goal will be to get away from the goal as fast as possible.

D_AvoidObstacleWithSamplePoints / D:Potential Fields

This behaviour will use the sample points in front of the robot to search for the best place to go. The potential field comprises of the goal potential and the obstacle potential. The goal potential is subtracted from the potential field and the obstacle potential is added to the potential field. The behaviour then chooses the sample with the lowest potential field value. The formulas used for those potential fields are similar exponential functions. The main difference is that the goal potential needs to be effective on a great distance while the obstacle potential will be summed up for every intersection found. This means if the robot has more sensors, it can have more intersections and, therefore, a stronger obstacle potential.

$$obstaclePotential(d) = \frac{e^{2.5}}{distance - 4}$$

Figure 11 - Formula for calculating the obstacle potential of a specific sample point based on its distance d to the goal.

$$goalPotential(d) = \frac{e^{5.8}}{distance - 6} + 10$$

Figure 11 – Formula for calculating the goal potential of a specific sample point based on its distance d to the goal.

The numbers for the potentials have been set manually. Their result has been satisfactory so far.

E_FractionalProgressWithSamplePoints / E:Fractional Progress

Fractional progress has been used to improve the shortcomings of the normal potential fields behaviour. The past cost p , the future cost f and the linear distance d between the start and the goal positions are calculated. The following formula is thereafter used to adjust the distance measure to the goal. The past cost is always the linear Euclidean distance. The future cost on the other hand is evaluated by using the chosen distance metric.

$$d' = \left(1 - \frac{p}{p + f}\right) * d$$

Figure 12 – The past cost p , future cost c and the linear distance from start to goal d are used to calculate the adjusted distance to the goal.

F_PelletBot / F: Pellet Bot

This implementation is an attempt to create a robot which can overcome very deep c shaped obstacles or local minima in general. The robot places a no-collide obstacle in the world after a certain amount of steps. Even though, the robot can not collide with the obstacle, it does think so as the sensor ray's can intersect with those pellets. This leads to obstacles inside the local minima which force the robot to move somewhere else. The strategy is able to proceed out of local minima but can also make the robot crash faster.

Results

The tested behaviours behaved sometimes quite similar. This was expected. The PelletBot (F) will behave exactly like the Fractional Progress Version (E) if it never gets close to its own path. The Pellet Bot (F) never got stuck and the linear version of it never crashed as well. The pellets forced the robot to constantly adapt to new routes. If the robot does not manage to escape from the local minima eventually the „noise“ from the pellets will become so strong that it forces it to run into the wall.

Bot/Map	Forest	ShallowC	HardC	DeepC1	DeepC2	MovingHell	CaveEscape
B	693	1183	stuck	stuck	stuck	crash	crash
D-linear	760	1384	stuck	stuck	stuck	846	stuck
D-quadratic	1074	1377	stuck	stuck	stuck	850	stuck
D-arc	806	1476	4028	stuck	crash	843	stuck
E-linear	793	1381	stuck	stuck	stuck	846	stuck
E-quadratic	1074	1381	stuck	stuck	stuck	850	stuck
E-arc	821	1473	4033	stuck	crash	848	stuck
F-linear	793	1381	2833	2169	1353	846	1981
F-quadratic	1444	1381	5162	1674	crash	850	2042
F-arc	821	1473	4033	>20'000 ¹	crash	848	crash

Figure 13 - Number of steps needed by each bot to reach the goal.

¹ The F-arc behaviour managed to escape the shallow C but continued to go towards the goal in a huge C shape. The simulation had been stopped before it reached the goal.

Conclusion

The pellet strategy as implemented in the F version has been superior against all the other implementations. It seems a good strategy to get out of minima efficiently. The arc metric offers sometimes a good escape from local minima. However, the path of the implemented arc metric is most of the times larger and, therefore, inefficient.

The final ranking is as follows.

1. F-linear (PelletBot)
2. F-quadratic (PelletBot)
3. F-arc (PelletBot)

Global Map vs Local Sensing

A simultaneous local and global state estimation combines the best things from two worlds. Both systems seem to have advantages and I don't believe that either one is ideal.

Bibliography

- [1] «JTS Topology Suite,» [Online]. Available: <http://tsusiatsoftware.net/jts/main.html>. [Zugriff am 26 10 2015].
- [2] Dudek, «CS W4733 NOTES - Differential Drive Robots,» 26 10 2015. [Online]. Available: <http://chess.eecs.berkeley.edu/eecs149/documentation/differentialDrive.pdf>.

List of Figures

Figure 1 - Make sure the maps are in the same folder as the jar file when running the Application.	2
Figure 2 - Keyboard Shortcuts for the developed application	3
Figure 3 - Calculates the distance to the destination based on the average speed and the time t.	3
Figure 4 - Calculates the angle to the destination based on the left and right wheel speed.	3
Figure 5 - Euclidean distance used for the linear distance metric.	4
Figure 6 - Euclidean distance used for the linear distance metric.	4
Figure 7 - Short version of Figure 8.	4
Figure 8 – Arc distance metric based on the relation between the distance to the start point, as well as, to the goal..	4
Figure 9 - Demonstration of free travel results.	5
Figure 11 – Formula for calculating the goal potential of a specific sample point based on its distance d to the goal...	5
Figure 11 - Formula for calculating the obstacle potential of a specific sample point based on its distance d to the goal.	5
Figure 12 – The past cost p, future cost c and the linear distance from start to goal d are used to calculate the adjusted distance to the goal.....	6
Figure 13 - Number of steps needed by each bot to reach the goal.	6
Figure 14 - Different Fractional Progress Paths based on the distance metric. The loop was caused by the quadratic distance metric which makes the robots react to the obstacles faster.....	8
Figure 15 - Greate example how the PalletBot is able to overcome local minima.....	8
Figure 16 - Another example how the PalletBot escapes from strong local minima.	9
Figure 17 - Fractional Progress with quadratic distance measure fails to escape the "cave".	9
Figure 18 - The PalletBot exploring the cave until he finds the exit.	10
Figure 19 - Even though all of the robots have no path prediction, they do very well in dodging moving obstacles. ..	10

Appendix

Some images to visualise the robot's behaviour and the obstacles to overcome.

E:Fractional Progress, Map: Forest.map Steps: 821 Length: 821.0
ROBOT IS IN GOAL!

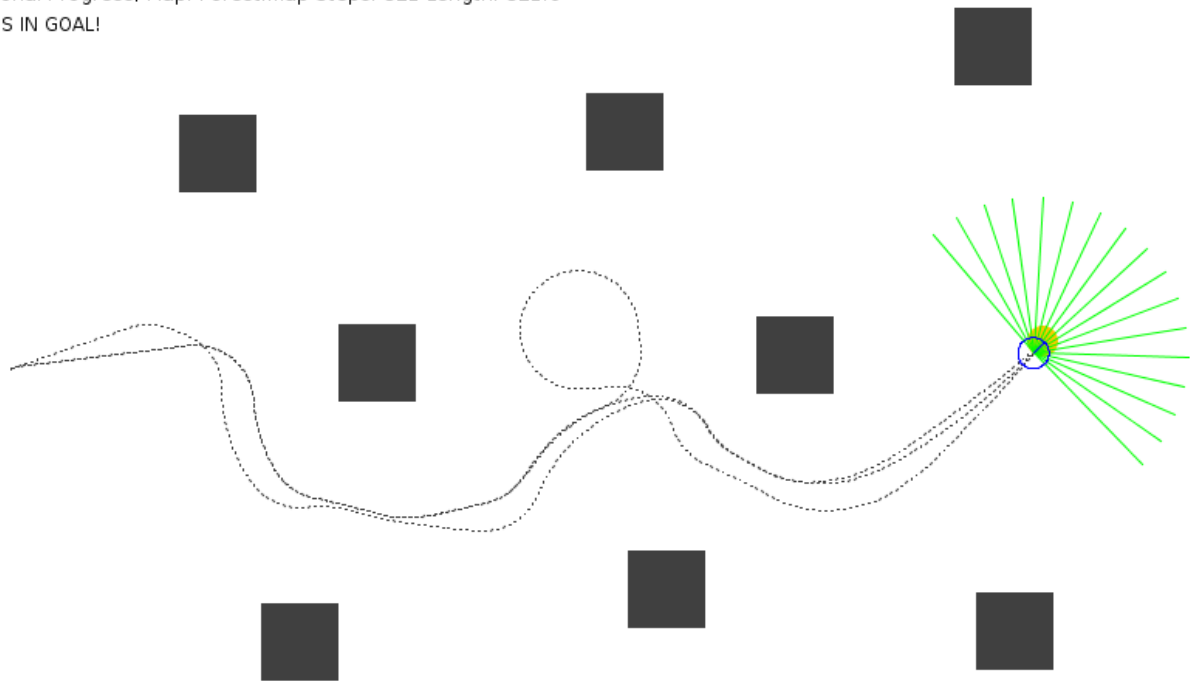


Figure 14 - Different Fractional Progress Paths based on the distance metric. The loop was caused by the quadratic distance metric which makes the robots react to the obstacles faster.

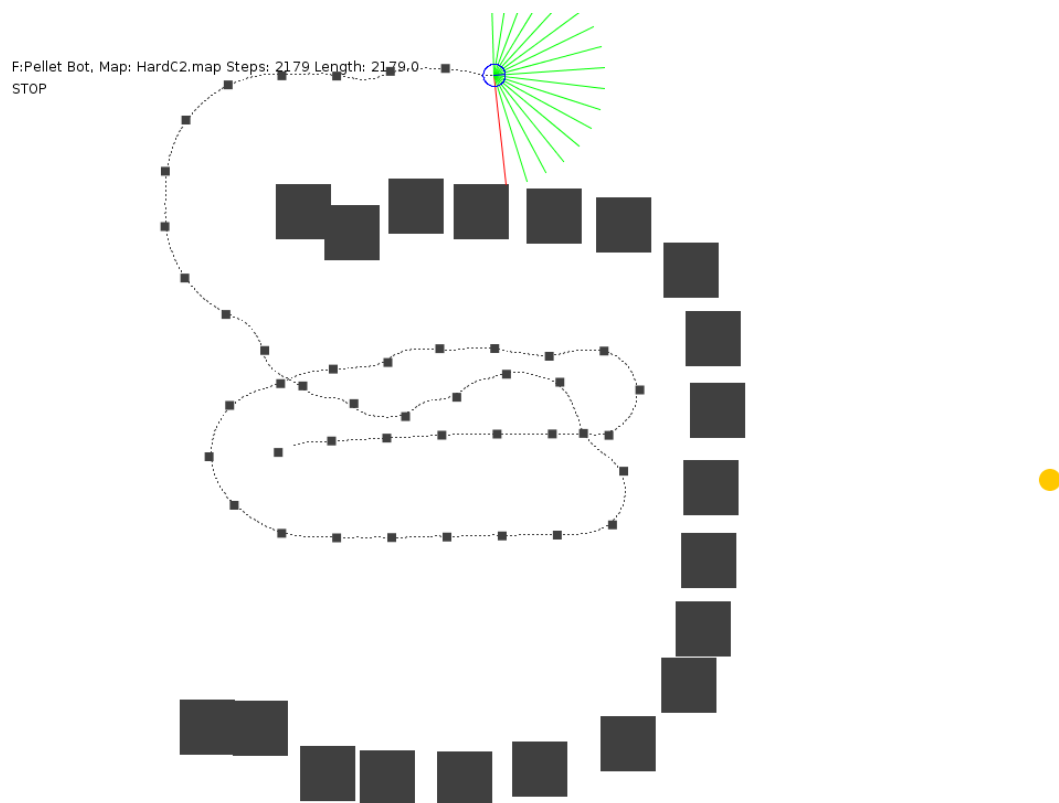


Figure 15 - Great example how the PalletBot is able to overcome local minima

F: Pellet Bot, Map: DeepC2.map Steps: 1331 Length: 1331.0
STOP

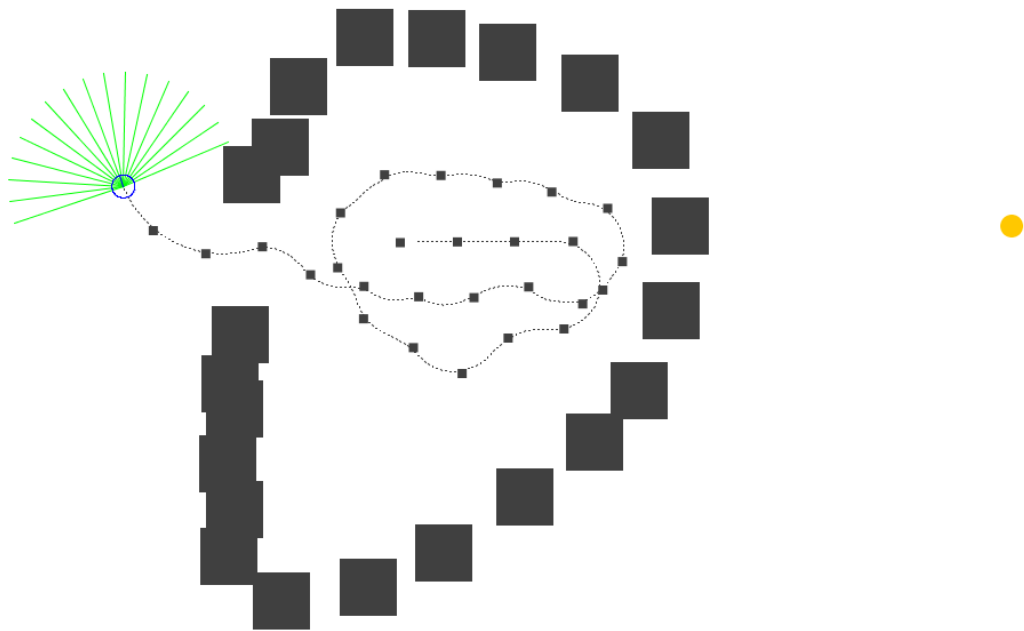


Figure 16 - Another example how the PalletBot escapes from strong local minima.

E: Fractional Progress, Map: CaveEscape.map Steps: 1994 Length: 1994.0
STOP

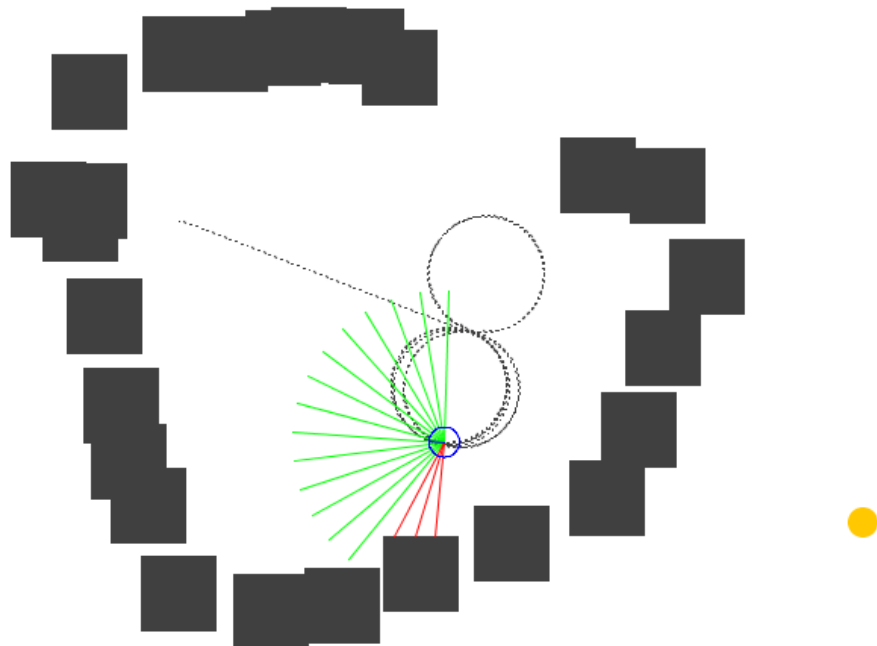


Figure 17 - Fractional Progress with quadratic distance measure fails to escape the "cave".

F: Pellet Bot, Map: CaveEscape.map Steps: 1597 Length: 1597.0
STOP

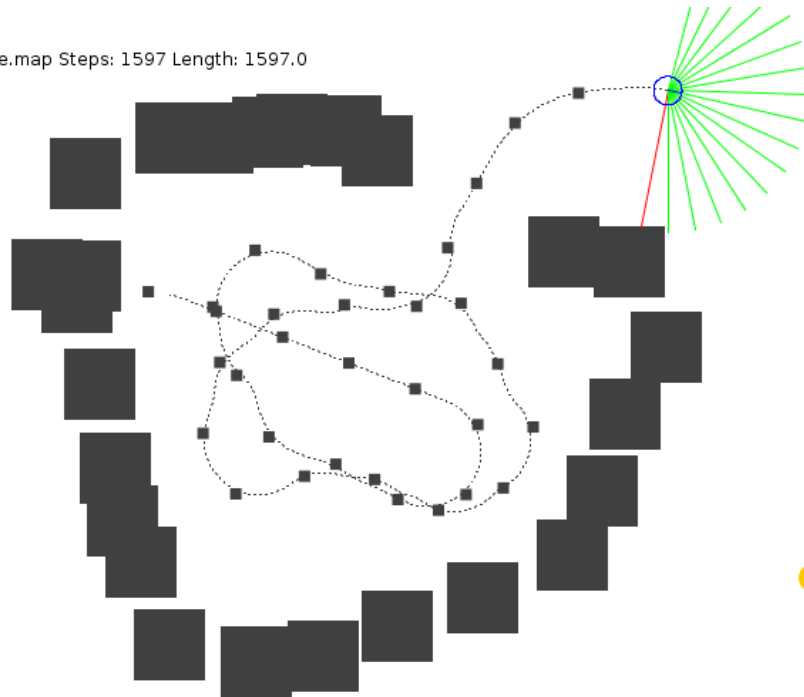


Figure 18 - The PalletBot exploring the cave until he finds the exit.

E: Fractional Progress, Map: MovingHell.map Steps: 495 Length: 495.0
STOP

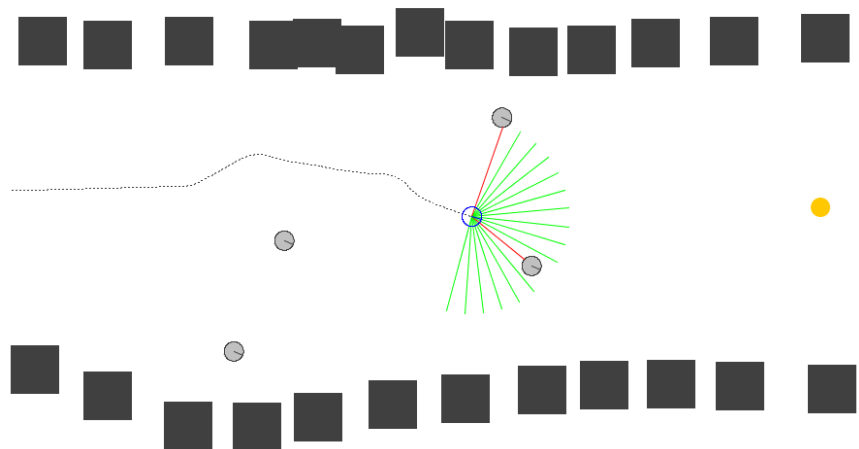


Figure 19 - Even though all of the robots have no path prediction, they do very well in dodging moving obstacles.