

# A MAPPING SIMULATOR

Catalin-Antonio Dedu<sup>1</sup>  
Costin-Anton Boiangiu<sup>2</sup>  
Ion Bucur<sup>3</sup>

## ABSTRACT

*This paper approaches the domain of mapping simulations and presents the result of subsequent research in path finding algorithms. The testing phase and the obtained results were carried by the means of an elaborate OpenGL application.*

**Keywords:** mapping simulator, path finding, iRobot Roomba, OpenGL

## 1. INTRODUCTION

Almost everyone heard of iRobot Roomba which is an automated vacuum cleaner that cleans one's room using a "smart" algorithm. This paper consist of an analysis of Roomba's mapping algorithm and a possible alternative solution to the robot's mapping algorithm, that will result in a less expensive and more energy-efficient operation mode.

## 2. A CLOSER LOOK

First of all we should consider the system's resources for this scenario: the electrical energy necessary to charge the robot's batteries, the percent of room coverage and nevertheless the time in which the robot cleans the room. Also it is important to know that the algorithm is a company property and not public knowledge, but even so the official site presents meaningful schematics of its algorithmic path finding: the robot explores the room more or less randomly and acquire input data by the means of proximity and infrared sensors creating a complete mesh which covers a large part of the room's floor. However this approach is not optimal in terms of consumed energy and floor coverage.

---

<sup>1</sup> Department of Computer Science and Engineering, Faculty of Automatic Control and Computers Science, University "Politehnica" of Bucharest, Splaiul Independenței 313, Bucharest, 060042, Romania, catalin.dedu@cti.pub.ro

<sup>2</sup> Department of Computer Science and Engineering, Faculty of Automatic Control and Computers Science, University "Politehnica" of Bucharest, Splaiul Independenței 313, Bucharest, 060042, Romania, costin.boiangiu@cs.pub.ro

<sup>3</sup> Department of Computer Science and Engineering, Faculty of Automatic Control and Computers Science, University "Politehnica" of Bucharest, Splaiul Independenței 313, Bucharest, 060042, Romania, ion.bucur@cs.pub.ro

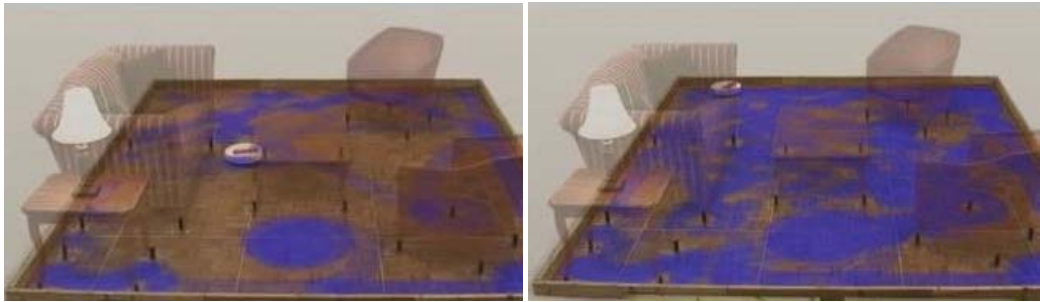


Figure 1. Roomba room coverage

Without considering some features of the robot like avoiding stairs and returning to debris areas, the robot would be more efficient if it would explore the room using a modified A\* algorithm.

### 3. AN OPTIMIZED SOLUTION

It is necessary that we find an optimum algorithm which will guarantee the solution. Classic algorithms like Dijkstra have the tendency to overload the processor with unnecessary calculations. For this reason and after experimenting with different algorithms and heuristics it became clear that the exploration algorithm should be an A\*.

Let us consider for a moment that our room is a map of equally sized squares (the size of a square being the size of the robot), and we want to find the optimum road between two squares on this map, the first being the starting point (S) the second being the point of arrival (G). Travelling from S to G, the Robot should travel through all squares.

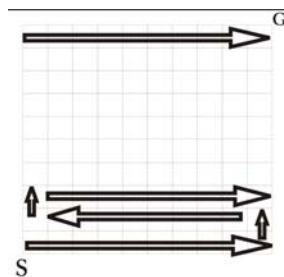


Figure 2. A\* application

As shown in **Figure 2** the exploration algorithm will be applied in every row of the logical map.

Another important aspect of our research is collision detection. Our robot must “sense” objects and avoid them. In the virtual OpenGL test environment it was considered the logical map mentioned above and a series of objects bigger or equal in size with a logical square. In order to have a more meaningful representation of the logical map it was created a terrain filled with crates (obstacles), so that the visual representation should be easy to follow.

#### 4. HEURISTICS USAGE

By tuning the terrain matrix of the map we can establish the resolution of the map. Terrain and obstacles configuration can be set by editing a text file which will be presented further on. The bigger the resolution will be the better resemblance with the real room scenario will be also. And now we have a viable testing environment for our algorithm.



Figure 3. OpenGL testing environment

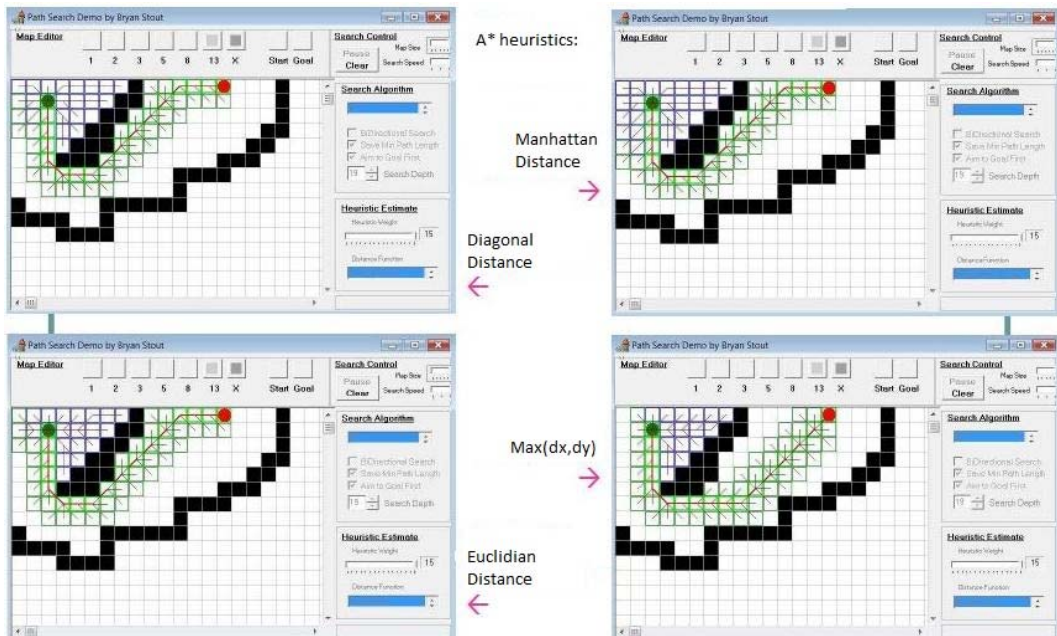


Figure 4. A\* heuristics

Furthermore there are many ways that the robot can go around the obstacle. It must be taken under consideration the optimal way to do that.

We can see from the picture above that using Manhattan heuristics the distance between destination and source is of 15 squares similar to the one using diagonal heuristics and maximal heuristics. The question is: which one to use?

The maximal heuristics is the easiest to understand and it requires less processing power. For the robot to find its destination, a certain sequence of squares must be explored. Each square along that path has a minimal cost. Any other square which deviates from the path has a higher cost. The bigger the deviation will be, the bigger the cost will be. For big deviations alternative routes should be considered.

The algorithm's complexity is linear only if the map does not have too many obstacles. Starting from a certain point beyond the complexity becomes exponential, and certain squares near obstacles will be more than once visited. This is the case for the next test scenario.

## 5. TESTING SCENARIOS

Let us consider the next object matrix where each dot represents a free square and each "X" character represents a square with an object on it. The point of passing between the 2 large areas will be visited the most. And will be the "cleanest". This translates in more energy and time consumed. Nevertheless, this exploration algorithm is more efficient than the one implemented on Roomba. Moreover if we consider the areas with no obstacles the squares are visited only once.

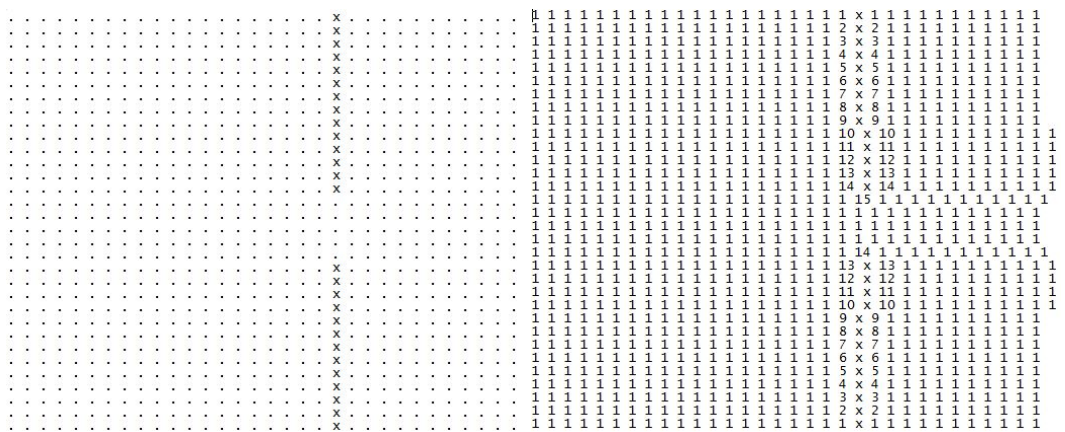


Figure 5. Logical square visit count

We can see the advantages of our algorithm. In areas where there is no obstacle every square is visited only once. The problem with every row exploration is that each time it finds an obstacle in its way the robot goes around it, thus crossing a different row, one which was visited in the past or one that will be visited in the future, therefore the squares from that row will be "double checked". But this is somehow good because considering that the robot operates in a living room and the objects are a sofa and a table, it is highly possible that debris will be found in that area. And insisting on those specific areas where the probability

of debris accumulations is high makes this algorithm particularly appropriate for this type of robot.

## 6. CONCLUSIONS

The application that simulates robotic behavior made the goal of this research, as well as the Roomba optimization example. These two cannot be one without the other in this context. Furthermore robotic simulation and optimization can be researched using tools like "Mapping Simulator". Although this application gives us an idea, there are many features that can be implemented. At its actual state, the input data consists of text files which can be edited by the tester in order to prepare the testing scenario. Implementing a graphical user interface and using a graphical engine can create a much more realistic testing environment.

## 7. ACKNOWLEDGMENTS

The work presented in this paper was funded by the Sectorial Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the financial agreement POSDRU/89/1.5/S/62557.

## 8. REFERENCES

- Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest  
OpenGL Programming Guide , Addison Westley Publishing Company  
OpenGL Cookbook – O'Reilly Media – by Dave Shreiner and Brad grantham  
The OpenGL Programing Guide – The redbook – the official guide of learning OpenGL version 2.1  
"http://nehe.gamedev.net/" - Online OpenGL tutorials, Downloaded 23<sup>rd</sup> June 2010  
"http://www.irobot.ro/" - iRobots, Roboti pentru curatenie, Downloaded 8 November 2012  
"http://en.wikipedia.org/wiki/Roomba" - Roomba, Wikipedia, the free encyclopedia, Downloaded 8 November 2012  
"http://en.wikipedia.org/wiki/A\*\_search\_algorithm" - A\*, Wikipedia, the free encyclopedia, Downloaded 8 November 2012  
Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics* SSC4 4 (2): 100–107. doi:10.1109/TSSC.1968.300136.  
Dechter, Rina; Judea Pearl (1985). "Generalized best-first search strategies and the optimality of A\*". *Journal of the ACM* 32 (3): 505–536. doi:10.1145/3828.3830.  
Koenig, Sven; Maxim Likhachev, Yaxin Liu, David Furcy (2004). "Incremental heuristic search in AI". *AI Magazine* 25 (2): 99–112.  
Pearl, Judea (1984). *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc.. ISBN 0-201-05594-5.  
Reese, Bjørn (1999). *AlphaA\*: An -admissible heuristic search algorithm*.  
Pearl, Judea (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley. ISBN 0-201-05594-5.  
Russell, S. J.; Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Upper Saddle River, N.J.: Prentice Hall. pp. 97–104. ISBN 0-13-790395-2.  
Hart, P. E.; Nilsson, N. J.; Raphael, B. (1972). "Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths"". *SIGART Newsletter* 37: 28–29.  
Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Palo Alto, California: Tioga Publishing Company. ISBN 0-935382-01-1.

Pearl, Judea (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley. ISBN 0-201-05594-5.

"What are Roomba® Virtual Walls®?". iRobot Customer Care. iRobot. Retrieved 2012-03-04.

"How can I prevent Roomba® from getting stuck?". iRobot Customer Care. iRobot. Retrieved 2012-03-04.

"Discussing and Dissecting the Roomba 780 ad Scooba 230". Roomba Community. Retrieved 2012-03-04.

"Roomba Development Tools". Archived from the original on 2006-07-11. Retrieved February 1, 2007.

"iRobot Roomba Serial Command Interface (SCI) Specification". iRobot Corporation. October 2005. Retrieved February 1, 2012.

"iRobot Corporation: Roomba Open Interface". Kibertron.org. 2005-10-24. Retrieved 2009-06-09.