MSRS: Critique on its usability via a path planning algorithm implementation

George Markou and Ioannis Refanidis

Department of Applied Informatics, University of Macedonia
Thessaloniki, Greece

{gmarkou, yrefanid}@uom.gr

Abstract In recent years an expanding number of robotics software platforms have emerged, with Microsoft expressing its interest in the field by releasing its own in 2006. This fact has created a highly competitive environment, as the majority of the products are mostly incompatible to each other, with every platform trying to establish itself as the field's standard. Thus, the question that arises is whether a platform is suited for educational purposes or creating a complete robotics intelligence package. This paper provides a study on the learnability, usability and features of Microsoft Robotics Studio, by creating and integrating into it a version of the Lifelong Planning A* algorithm (LPA*) algorithm.

1 Introduction

In the last few years there has been an increasing interest in the unification of artificial intelligence and robotics platforms. This has led to the creation and use of an expanding number of robotics software platforms, with a significant amount of undergraduate classes making use of the new technologies by creating rather advanced robotics projects within one or two semester courses [1, 23, 26]. In 2006 Microsoft entered the robotics field with its own robotics platform, named Microsoft Robotics Studio, competing against already widespread platforms such as the Player Project.

In this paper we implement a path planning algorithm in a simulated robotics environment, of which will be able to change its topology and the number of obstacles it contains during the agent's movement in it. The robotics platform that will be used is Microsoft's Robotics Studio, due to the fact that its introduction has caused extensive discussion and controversy as to whether or not it is suited for academic research, or educational and industrial purposes [3, 24, 25]. We will address this mixture of skepticism and enthusiasm by giving Microsoft's Robotics

Studio's features, ease of use and learnability a thorough critique, through the implementation of the aforementioned algorithm.

Due to the nature of the simulated environment in which the agent will move, the path planning algorithm that we will implement will have to be able to create a new plan or adapt an existing one every time the environment's topology changes. Koenig et al. [7] suggested that in systems where an agent has to constantly adapt its plans due to changes in its knowledge of the world, an incremental search method could be very beneficial as it can solve problems potentially faster than solving each search problem from scratch. They combined such a method with a heuristic one, which finds shortest paths for path-planning problems faster than uninformed search methods. This led to the creation of the algorithm we will implement, Lifelong Planning A* (LPA*) [8], which produces a plan, having a quality which remains consistently as good as one achieved by planning from scratch.

The remainder of the paper is organized as follows: In Section 2 we review works related to our own research, while in Section 3 we compare Microsoft's Robotics Studio to other prominent robotics platforms. Section 4 focuses on the theoretical aspects of the LPA* algorithm. In Section 5 we discuss the domain that was created in Robotics Studio, both in regard to the simulated maze and to the robot that was used. Section 6 presents the experiments we implemented, and Section 7 concludes the paper and poses directions for future work.

2 Related Work

In order for Microsoft Robotics Studio (MSRS) to become the standard robotics development platform, it has to achieve mainly two different goals: First, partnerships within the robotic industry, as well as with the academia. Secondly, the program itself needs to be able to offer advantages in comparison with other platforms. The first goal has been fulfilled to a point, as several companies, universities and research institutes opted to support and use MSRS, such as Kuka, Robosoft, fischertechnik and Parallax, Inc. [13]. Additionally, it is available currently for free download and use to anyone using it for noncommercial purposes.

As to the second goal, in [6] the author concluded that MSRS offers a wide range of technological solutions to problems common in the robotic field, by providing features such as visual programming or its combined system of concurrency control with efficient distributed message passing. However, he admits that there are still evident limitations to the program, like its integration with low level processors. The former opinion is shared by Tsai et al in [23] who used MSRS in an effort to design a service oriented computer course for high schools. They concluded that there are several disadvantages in the structure of the program, mainly that the visual programming language that is used in MSRS requires detailed knowledge of an imperative programming language, and that the loop structures which are used in it are implemented by "Goto", instead of by structure construct.

Also, they pointed out that some of the service oriented features that Microsoft had promised to provide were not available.

Others, however, are far more positive towards MSRS. Workman and Elzer in [26] used the program in an upper-level undergraduate robotics elective in order to document its usefulness in such an academic environment. They found that MSRS provided a great link between the language syntax already known to students and unfamiliar robotics semantics and highly recommended its use, adding that they were quite satisfied with the available features of the program and the support it provided for different hardware. Tick in [22] goes even further to suggest that the introduction of MSRS in the robotic market shows the future direction for programming for Autonomous Mobile Research Robots and could possibly determine the evolution of these systems as its own features will force other platforms to develop their competitive products so as to offer similar capabilities.

In conclusion, based on the related bibliography up-to-date it still remains unclear whether MSRS will evolve to be the industry's standard, as other Microsoft's programs have achieved in the past. On the other hand, it is quite definitive that it has a lot of useful features to offer, especially in the educational field, as well as that it is already at least a simple starting point for anyone who wants to become involved with a field as complex as robotics.

3 Robotics Platforms Overview

Before we present the domain we created in MSRS we briefly discuss the similarities and differences of it in comparison to some of the most prevalent robotics platforms. Although MSRS is available as a free download for researchers or hobbyists, it is not open source, and it is also not free of charge if intended for commercial use, whereas several platforms like the Player Project are both. Moreover, MSRS is the only platform in our comparison that can only be used in one operating system, while most are compatible with at least two, typically both Windows and Linux operating systems. The Player Project and the Orocos Project do not natively support Windows, but the former can run on Linux, Solaris, Mac OSX and *BSD, whereas the latter is aimed at Linux systems, but has also been ported to Mac OSX. One other major difference of Microsoft's robotics platform in contrast to its antagonists is that it does not provide a complete robotics intelligence system so that the robots it supports can be made autonomous, but relies on the programmers to implement such behaviours.

Its advantages over the competition, however, are also significant. It is one of the few major robotics platforms - along with Gostai's and Cyberbotics' collaborative platform Urbi for Webots - to provide a visual programming environment, and its architecture is based on distributed services, with these services being able to be constructed in reusable blocks. Furthermore, the platform enjoys the financial and technological support of one of the largest corporations in the world. In

Table 1 there is a comparison of some of the available characteristics of six of the most widely used robotics platforms today.

Table 1. Features of several of the most prominent robotics platforms [2, 4, 20].

	MSRS 1.5	MobileRobots	Skilligent	Orocos	Player Project	Urbi/ Webots
Open Source	No	No	No	Yes	Yes	Parts of Urbi
Free of Charge	Express Edition	No	No	Yes	Yes	No
Windows/ Linux	Yes/ No	Yes/ Yes	Yes/ Yes	No/ Yes	No/ Yes	Yes/ Yes
Other OS	No	No	No	No	Yes	Yes
Distributed Services	Yes	No	Yes	No	Limited	Yes
Drag-and-Drop IDE	Yes	No	No	No	No	Yes
Object Recognition	No	No	Yes	No	No	No
Localization	No	Yes	Yes	No	No	No
Learning/ Social Interaction	No	No	Yes	No	No	No
Simulation Environment	Yes	Yes	No	No	Yes	Yes
Reusable Service Blocks	Yes	Yes	Yes	Yes	No	Yes
Real-Time	No	No	No	Yes	No	No

4 Lifelong Planning A*

It is very common for artificial intelligence systems to try and solve path-planning problems in one shot, without considering that the domain in which they operate might change, thus forcing them to adapt the plan that they have already calculated. Solving the new path-planning problem independently might suffice if the domain is sufficiently small and the changes in it are infrequent, but this is not usually the case.

Koenig et al in [8] developed the Lifelong Planning algorithm to be able to repeatedly find a shortest path between two given vertexes faster than executing a complete recalculation of it, in cases where this would be considered a waste of computational resources and time. It combines properties of a heuristic algorithm, namely A* [5], and an incremental one, DynamicSWSF-FP [16]. The first search LPA* executes is identical to a search by a version of A* that breaks ties in favour of vertices with smaller g-values. The rest of its searches, which take place when a change in the domain happens, however, are significantly faster. This is achieved by using techniques which allow the algorithm to recognize the parts of the search tree which remain unchanged in the new one.

Properties of A* are used to focus the search on parts of the tree that are more likely to be part of the shortest path and determine which start distances should not be computed at all, while DynamicSWSF-FP is used in order to decide whether certain distances remain the same and should not be recomputed. The combination of these techniques can be very efficient in reducing the necessary time to recalculate a new path if the differences between the old and the new domain are not significant, and the changes were close to the goal. Finally, it is noteworthy that our implementation does not follow the original LPA* algorithm. Instead we opted to implement the backwards version presented in [9] which continuously calculates a new shortest path from the goal vertex to the agent's current position, and not, as it originally was, from the start vertex to the goal.

5 Maze Domain

The entire simulation domain was created using Microsoft Robotics Studio 1.5 Refresh, which was the current version of the program when we started working on this paper. Subsequently, as Microsoft released a new version of the platform Microsoft Robotics Developer Studio (MRDS) 2008 we migrated our project to the newest version of the program. The platform allows the creation of new user-defined entities, which can be associated with a mesh, making the entity appear more realistic. As a three-dimensional mesh can be created and imported into the MSRS' simulations environment from most 3D graphical editing programs [15], the resulting simulation can reflect almost any real situation.

Although creating a particularly realistic environment is not suited for a novice user as it can be a very complex procedure, several lifelike environments exist as built-in samples in Microsoft's Visual Simulation Environment in MRDS 2008. They have been developed by SimplySim, a French company that provides professional quality real time 3D simulations, and depict environments ranging from urban sceneries and apartments to a forest [19].

The environment for our experiments is a much simpler one, based on the "MazeSimulator" project, a program which allows users to create labyrinths based on a bitmap image. It was created by Trevor Taylor [21], who in turn used elements from previous work done by Ben Axelrod. The maze environment we simulated is explained in further detail in Section 5.1.

5.1 Simulated Maze

We created a gridworld of size 7×7, containing nodes which can randomly alternate their status between blocked and unblocked. This scenario is an abstraction of the Robocup Rescue Simulator Competition [18], where the roads in a city being hit by an earthquake change their status from free to blocked due to collapsing

buildings. The maze is safely explorable, that is the robot can safely reach the goal node from any node of the domain.

In order to create the obstacles in the maze, and make the environment dynamic, we opted for a solution that removed the obstacles from the simulation, updated their mass appropriately, and then re-inserted them in the Simulation Engine. This implementation, though not the obvious approach, was the simplest possible since the platform does not provide through its libraries a method of dynamically changing the mass of an object. The resulting simulated environment is shown in Fig. 1.

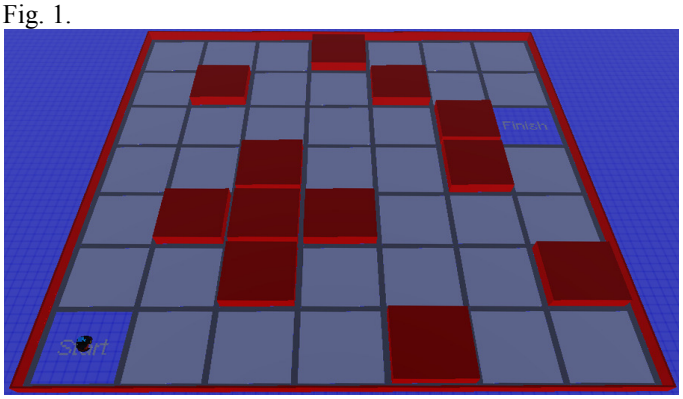

Fig. 1. Initial state of the simulated domain.

5.2 Robot

Microsoft Robotics Studio 1.5 Refresh supported - with built-in services - a wide variety of robots, ranging from simple and affordable hobbyist robots, such as the iRobot Create, to sophisticated humanoid robots capable of performing fighting and acrobatics, like the Kondo KHR-1. The list also includes the Lego Mindstorms NXT, MobileRobots' Pioneer 3DX, the Boe-Bot Robot from Parallax and fischertechnik's ROBO Interface [14]. All the aforementioned robots are also supported in MRDS 2008, with the exception of the Parallax Boe-Bot. The robot used in our experiments was a Pioneer 3DX, with a mounted sick laser range finder on top of it, as at the time it was one of the most widely used in various MSRS' tutorials and projects .

We have defined the movement of the robot to consist of three parts. First, the robot moves in a straight line for a distance equal to the length of a node. Then, it decides, based on the plan created from LPA*, whether or not is required to make a turn, and finally it executes the turn, rotating in angles which are multiple of 90 degrees. Using the laser range finder, the robot builds a tri-color map of the environment, in which white color symbolizes free space that the robot has explored. Black color is drawn on the points on the map that the laser hit an obstacle, and the rest of the map – the part of the environment that the robot has not explored, is shown in grey color. Each time the robot moves through a specific location, the

part of the map that corresponds to that region will be overwritten by the new data that the robot collects. In essence, we build a simple occupancy grid map, with each cell of it containing a value that represents the possibility that it is occupied.

6 Experiments

We created three different experiments, all with the same initial maze settings, but each one changing in a different way after the robot had reached a certain node of the domain. In two of the experiments the changes were known beforehand, whereas in the last one they were random. In each one, however, the changes were minimal, blocking / unblocking a maximum of two nodes.

We implemented LPA* in MSRS without having to study the program in great depth or learn a new programming language, since the support for multiple languages gave us the opportunity to work in one related to our previous knowledge, in our case C#. An inexperienced user however, has the option to use a graphical "drag-and-drop" programming language provided by Microsoft, which is designed on a dataflow-based model. Microsoft's Visual Programming Language (VPL) allows users to create their program by simply "orchestrating activities", that is, connecting them to other activity blocks. An activity is a block with inputs and outputs that can be represent pre-built services, data-flow control, a function, or even a composition of multiple activities.

Initially, it was our intention to make use of the visual programming environment that Microsoft developed to implement our project, so as to additionally document the strengths and weakness of the new programming language as well as MSRS. However, the task proved to be extremely difficult, if not impossible, due to obvious deficiencies of VPL: First of all its diagrams tend to become exceedingly large as the program's complexity increases. Moreover, VPL has limited support for arbitrary user-defined data types and does not support a generic object which, naturally, is an important restriction to a programmer's tools. What is more, the only type of control flow and collection of items that have built-in support in VPL are "if statements" and lists respectively; that is, recursion and arrays are not natively supported at the moment.

Thus, expert programmers will likely prefer to write in an imperative programming language, although they can still find VPL useful as a tool, especially if they are not familiar with MSRS' environment, as it can easily be used for creating the skeleton of a basic program by wiring activities to each other and automatically generating the consequent C# code through it. The opinion we formed through our experience though, is that VPL is best suited for novice users who only have a basic understanding of programming concepts such as variables, and might enjoy the easiness of not writing any code.

One element of the platform that is especially helpful to the programming process is the Concurrency and Coordination Runtime (CCR), a programming

model that facilitates the development of programs that handle asynchronous behavior. Instead of writing complex multithreaded code to coordinate the available sensors and motors functioning at the same time on a robot, the CCR handles the required messaging and orchestration efficiently as its function is to "manage asynchronous operations, exploit parallel hardware and deal with concurrency and partial failure" [12]. Furthermore, it has been proven to be not only useful as a part of MSRS, but in non-robotics development processes [11, 17].

As aforementioned, we implemented LPA* so that it can work backwards. The reason behind this choice was that in this way we were able to calculate a new shortest path for the part of the maze we were interested in, i.e., from the goal node to the robot. Had we used the original version of LPA*, the algorithm would calculate an entirely new shortest path from the original node to the finish. The algorithm was applied successfully into the rest of the MSRS domain we created and performed as one would expect having read the theoretical properties of LPA* in [7, 8, 9].

The simulation environment was aesthetically appealing and served our functional needs. Based on the robot's interaction with it and in particular while the robot followed the course through the maze depicted in Fig. 2 (b), the laser range finder built the occupancy grid map that is shown in Fig. 2 (a).

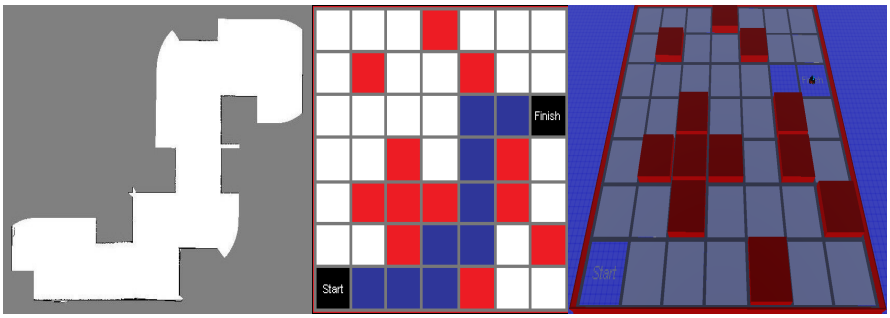

Fig. 2.a (Left) Occupancy grid map of the maze's final state. **2.**b (Center) Ground plan of the maze. The robot's course is shown in blue. **2.**c (Right) Final state of the simulated domain.

7 Conclusions and Future Work

In this paper we used Microsoft Robotics Studio to implement a realistic environment in which an agent follows a shortest path course from a given node to a goal one using the LPA* algorithm. It was our intention to critique whether or not MSRS is a suitable program for use in academic, educational, or even industrial environments. Our experience indicates that while it may be fairly time-consuming for a person to familiarize himself with the program, the process is made significantly easier by the facts that multiple programming languages are

supported and by some of the platform's features, such as the Concurrency and Coordination Runtime. In that sense, our findings are in accordance with those from Workman and Elzer [26] mentioned in Section 2.

Moreover, it is not necessary to delve into all of the aspects of the program in order to create a simple functional program, especially if the project's basis is formed through orchestrating pre-built services in VPL. Such knowledge may be needed though if more sophisticated programs are to be implemented. Our critique of Microsoft's Visual Language is partly different from that of Tsai et al [23] as we concluded that VPL requires only minimal knowledge of an imperative programming language, is not as strong as one, and if anything its use is greater for a novice programmer than for an expert. Such a programmer will probably find it easier to use one of the multiple imperative programming languages that are supported by the platform.

In general, it was evident that the program has extensive features and capabilities that could potentially establish it as the field's standard, especially considering the vast support a company like Microsoft can provide for it. Initially, while we were implementing our experiments in MSRS 1.5 Refresh we encountered several minor or major difficulties, with the most important being the program's unexpected termination depending on the machine it was executed on. However, in general these problems can be attributed to the relatively small life cycle of the product, as after the migration of our project to MRDS 2008 most of them, including the termination of the program, seemed to have been resolved. Such problems could possibly discourage some researchers or educators from relying solely on MSRS for their needs, and as such it is a matter of utter importance for Microsoft to keep improving the platform as it did with MRDS 2008, so that it can become fully stable and functional.

Future work can focus on inducing more than one changes to the maze domain, and coordinating the MSRS services that are involved in the program so that they communicate with each other every time such a change occurs. Finally, in order to evaluate the ease of use and learnability of the platform in an academic environment in a more efficient, semi-quantitative way and in greater detail, we could base our assessment on an experiment along the following lines: Develop a structured questionnaire and ask two different groups of students to fill them out after each of them has implemented a similar robotics project in MSRS and another robotic platform such as the ones mentioned in Section 3, in order to establish the advantages and disadvantages of each one as accurately as possible.

References

1. Blank D, Kumar D, Marshall J & Meeden L (2007) Advanced robotics projects for undergraduate students. AAAI Spring Symposium: Robots and Robot Venues: Resources for AI Education: 10-15

2. Bruyninckx H (2001) Open robot control software: the OROCOS project. Proceedings of the IEEE International Conference on Robotics and Automation (3): 2523-2528
3. Bruyninckx H (2007) Microsoft Robotics Studio: Expected impact, Challenges & Alternatives. Panel presentation at the IEEE International Conference on Robotics and Automation
4. Gerkey B (2005) The Player Robot Device Interface - Player utilities. http://playerstage.sourceforge.net/doc/Player-2.0.0/player/group__utils.html. Accessed 15 January 2009
5. 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 & Cybernetics, 4 (2): 100–107
6. Jackson J (2007) Microsoft Robotics Studio: A Technical Introduction. IEEE Robotics & Automation Magazine 14 (4): 82-87
7. Koenig S, Likhachev M & Furcy D (2004) Lifelong Planning A*. Artificial Intelligence, 155 (1-2): 93-146
8. Koenig S, Likhachev M, Liu Y & Furcy D (2004) Incremental Heuristic Search in Artificial Intelligence. AI Magazine, 25 (2): 99-112
9. Likhachev M & Koenig S (2005) A Generalized Framework for Lifelong Planning A*. Proceedings of the International Conference on Automated Planning and Scheduling: 99-108
10. Michael N, Fink J & Kumar V (2008) Experimental Testbed for Large Multirobot Teams. IEEE Robots and Automation Magazine, 15 (1): 53-61
11. Microsoft Corporation (2008) Microsoft CCR and DSS Toolkit 2008: Tyco Case Study. http://go.microsoft.com/fwlink/?LinkId=130995. Accessed 13 January 2009
12. Microsoft Corporation (2008) Microsoft Robotics Developer Studio: CCR Introduction. http://msdn.microsoft.com/en-us/library/bb648752.aspx. Accessed 12 January 2009
13. Microsoft Corporation (2008) Microsoft Robotics Studio Partners. http://msdn.micrsoft.com/en-us/robotics/bb383566.aspx. Accessed 15 October 2008
14. Morgan S (2008) Programming Microsoft Robotics Studio, Microsoft Press
15. Morgan S (2008) Robotics: Simulating the World with Microsoft Robotics Studio. http://msdn.microsoft.com/en-us/magazine/cc546547.aspx. Accessed 13 January 2009
16. Ramalingam G & Reps T (1996) An incremental algorithm for a generalization of the shortest-path problem. Journal of Algorithms, 21:267-305
17. Richter J (2006) Concurrent Affairs: Concurrency and Coordination Runtime. http://msdn.microsoft.com/en-us/magazine/cc163556.aspx. Accessed 13 January 2009
18. RoboCupRescue (2006) Rescue Simulation Leagues. http://www.robocuprescue.org/simleagues.html. Accessed 25 October 2008
19. SimplySim (2008) Generic Environment. http://www.simplysim.net/index.php?scr=scrAccueil&idcategorie=1. Accessed 12 January 2009
20. Somby M (2008) Software Platforms for Service Robotics http://linuxdevices.com/articles/AT9631072539.html. Accessed 18 October 2008
21. Taylor T (2008) MSRS Maze Simulator. http://www.soft-tech.com.au/MSRS/MazeSimulator/MazeSimulator.htm. Accessed 23 September 2008
22. Tick J (2006) Convergence of Programming Development Tools for Autonomous Mobile Research Robots. Proceedings of the Serbian – Hungarian Joint Symposium on Intelligent Systems: 375-382
23. Tsai W T, Chen Y, Sun X, et al. (2007) Designing a Service-Oriented Computing Course for High Schools. Proceedings of the IEEE International Conference on e-Business Engineering: 686-693
24. Turner D (2006) Microsoft Moves into Robotics. http://www.technologyreview.com/computing/17419/page2/. Accessed 21 October 2008
25. Ulanoff L (2006) Rivals Skeptical of Microsoft's New Robot Software. http://www.pcmag.com/article2/0,1895,1979617,00.asp. Accessed 21 October 2008
26. Workman K & Elzer S (2009) Utilizing Microsoft robotics studio in undergraduate robotics. Journal of Computing Sciences in Colleges 24 (3): 65-71