DESIGN, IMPLEMENT AND SIMULATE AN AGENT MOTION PLANNING ALGORITHM IN 2D AND 3D ENVIRONMENTS

Haissam El-Aawar[1] and Hussein Bakri[2]

[1]Associate Professor, Computer Science/Information Technology Departments, LIU, Bekaa-Lebanon
e-mail: haisso@yahoo.com., haissam.aawar@liu.edu.lb.
[2]Instructor, Computer Science/Information Technology Departments, LIU, Bekaa-Lebanon,
e-mail: hussein.bakri@liu.edu.lb.

ABSTRACT

This article presents a computer simulated artificial intelligence (AI) agent that is able to move and interact in 2D and 3D environments. The agent has two operating modes: Manual Mode and Map or Autopilot mode. In the Manual mode the user has full control over the agent and can move it in all possible directions depending on the environment. In addition to that, the designed agent avoids hitting any obstacle by sensing them from a certain distance. The second and most important mode is the Map mode, in which the user can create a custom map, assign a starting and target location, and add predefined and sudden obstacles. The agent will then move to the target location by finding the shortest path avoiding any collision with any obstacle during the agent's journey.

The article suggests as a solution, an algorithm that can help the agent to find the shortest path to a predefined target location in a complex 3D environment, such as cities and mountains, avoiding all predefined and sudden obstacles. It also avoids these obstacles during manual control and moves the agent to a safe location automatically.

KEYWORDS

Motion Planning Algorithm, Artificial Intelligence, Real Time Motion, Automatic Control, Collision Avoidance.

1. INTRODUCTION

An agent motion planning algorithm plays an important role in many applications of AI, robotics and virtual reality especially in navigating agents in 2D and 3D environments. Designing an artificially intelligent agent is a complex task especially in a non-observable or partially observable environment where the level of uncertainty is very high. The best way to describe this case is through the following scenario: imagine yourself in a dark room, where you can't see anything and you need to find your way out of the room, following the shortest path to the door and avoiding all objects in the room. Imagine all that and add to it an extra 3rd dimension, like the case of flying agents (planes) where the environment becomes more complex, obstacles are more unpredictable, and agent's motion becomes more difficult.

David C. Wyld et al. (Eds) : CST, ITCS, JSE, SIP, ARIA, DMS – 2014
pp. 323–337, 2014. © CS & IT-CSCP 2014 DOI : 10.5121/csit.2014.4130

Complex and precise data is required from sensors of the plane in order for it to respond in almost real time manner to sudden obstacles or contingencies faced during flight. Although this work is completely simulated on a computer, we suggest that the algorithm of Map Mode and obstacle avoidance can be used in any real environments and on any real agents. It should be noted that the algorithm still needs to be tested on a real airplane drone in a real world environment.

This work describes the design, implementation and simulation of an algorithm that can help an agent to find its way to a target location automatically and without human intervention in a complex, continuous and partially observable 2D and 3D environments. The agent in these environments may face many static and/or moving obstacles, for example a plane may face buildings, birds or other planes. The algorithm should avoid these obstacles without abandoning its pre- mentioned goals. The algorithm is implemented using C# object oriented programming language and using multiple data structures like arrays, queues and trees. Finally, the algorithm is simulated on a computer in 2D and 3D environments (precisely in 3D Virtual City) which contains virtual real-time obstacles.

2. RELATED RESEARCH

In this section we provide a brief overview of some of prior works related to path planning in AI.

2.1 Motion Planning

Motion planning is a term used in robotics. Motion planning algorithms are used in many fields, including bioinformatics, character animation, video game AI, computer-aided design and computer-aided manufacturing (CAD/CAM), architectural design, industrial automation, robotic surgery, and single and multiple robot navigation in both two and three dimensions. A classical version of motion planning is sometimes referred to as the Piano Mover's Problem [1].

Motion planning is a fundamental problem in robotics. It may be stated as finding a path for a robot or agent, such that the robot or agent may move along this path from its starting position to a desired goal position without colliding with any static obstacles or other robots or agents in the environment. This problem interacts with other important problems, such as real-time motion control, sensing and task planning.

The basic motion planning problem is stated as: Given a start pose of the robot, a desired goal pose, a geometric description of the robot and a geometric description of the world, the objective is to find a path that moves the robot gradually from start to goal while never touching any obstacle [1, 2].

2.2 Probabilistic Roadmap Methods

One of the most important classes of motion planning methods addressed in the literature [3, 4, 5, 6, 7, 8] is the probabilistic roadmap methods (PRMs).

A roadmap, RM, is a union of one-dimensional curves such that for all start and goal points in C_{free} that can be connected by a path. In order to compute collision-free paths for robots of virtually any type moving among stationary obstacles (static workspaces) the Probabilistic Roadmap Method/Planner (PRM) is used.

PRM uses randomization extensively to construct roadmaps in C spaces. Heuristics functions are used without calculations for sampling C obstacles and C Spaces.

PRM planner can be applied to robots with many degrees of freedom (dof). The method consists of two phases: a learning (construction) phase and a query phase.

In the learning phase, a roadmap is a graph where the nodes are built of collision-free configurations and where the edges are built of collision-free paths by repeating the two following steps:

- Choose a certain random configuration of the robot, check the collision rate and repeat this step until the random configuration is free of collisions.
- Try to connect the former configuration to the roadmap using a fast local planner.

To find a path in the query phase, the idea is to connect initial and goal configurations to the roadmap and to search the roadmap for a sequence of local paths linking these nodes. The path is thus obtained by a Dijkstra's shortest path query [2, 3, 9]

2.3 Robot's (Agent's) Workspace (Environment)

A robot is defined in motion planning problems as an object or as versatile mechanical device which can move, rotate and translate. It can be polymorphic i.e. taking several forms like rigid object or a manipulator arm, a wheeled or legged vehicle, a free-flying platform (a plane) or a combination of these or a more complex form like or a humanoid form – equipped with actuators and sensors under the control of the computing system [4]. Furthermore, a robot is a reprogrammable, multi-functional manipulator designed to perform a variety of tasks through variable programmed motions, such as moving material, parts, tools, or specialized devices [10].
In motion planning algorithms, robots can move in a myriad of environments consisting from 2D, 3D to even N-dimensional. As an abstracted version that aims to solve the problem, a robot can be represented as a point in space taking translational coordinates (x, y, z). Normally in 3D workspace, it is a recurring theme to use six parameters: (x, y, z) for locating the position of the robot and (α, β, γ) for its rotation at every point [2].

2.4 AI Planning

Planning is essential to intelligent and rational agents, in the sense of achieving their autonomy and flexibility through the construction of sequences of actions to achieve their goals. Planning as a sub discipline in the artificial intelligence field has been an area of research for over three decades. Throughout the history of research in this domain, the distinction between planning and problem solving has been indefinable [1, 2].

In AI, the term planning takes a more discrete flavor than a continuous one. Instead of moving a piano through a continuous space, problems solved tend to have a discrete nature like solving a Rubik's cube puzzle or sliding a tile puzzle, or building a stack of blocks. These types of discrete models can still be modeled using continuous spaces but it seems more convenient to define them as finite set of actions applied to a discrete set of states. Many decision-theoretic ideas have recently been incorporated into the AI planning problem, to model uncertainties, adversarial scenarios, and optimization [1, 2, 11, 12].

2.5 Computational Environments

The article's implementation and simulation are executed on Microsoft Visual Studio 2010 (using C# language). The 3D simulation is implemented on Windows XNA Game Studio 4.0, which

uses the XNA framework. Microsoft XNA Game Studio is used to build interactive games on windows based computers, X-box 360 and windows mobile [22].

The **XNA framework** helps making games faster. Typically XNA framework is written in C#, and uses DirectX and Direct3D which is designed to virtualize 3D hardware interfaces for windows platforms, so instead of worrying about how to get the computer to render a 3D model, user may take more focus view on the problem and gets the 3D data onto screen. An XNA tutorial [23] is used to create a flight simulator and produce a flying aircraft in a true 3D city which is used to apply the algorithm.

Direct X is recommended in order to have the optimal performance in the 3D Simulator [24].
C sharp (C#) was developed by Microsoft (.Net) as Common Language Infrastructure (CLI) to be a platform-independent language in the tradition of Java [13, 14, 25]. The C# language is intended to be a simple, modern, general-purpose, object-oriented programming language. It offers a strong typing, functional, class-based programming, which makes programming much easier and rigid. C# helps programmers to initiate parallel threads, called asynchronies methods, and this brews for us a full control of the dataflow to govern the AI agent.

3. MOTION PLANNING ALGORITHM

As mentioned before, the algorithm has two main modes: **Manual and Map mode**. In manual mode the user has full control over the agent and can move it in any possible direction. In this mode there is a special feature called **Contingency Mode**. This mode allows the agent to sense the obstacle from a certain distance and avoid it by moving to a safe position away from it. The Second mode is the map mode. In this mode the user assigns a starting point, target point and a number of obstacles. The agent then must apply the algorithm in order to find the shortest path to the target and move to it avoiding all kinds of obstacles [15, 16, 17, 18].

The development of algorithm passed through three main phases:

1. The Design of the algorithm phase was the first phase where the algorithm was designed to meet as much as possible the following requirements: optimality, completeness, and acceptable time and space complexities.
    a. Optimality means that the algorithm must always find the lowest cost path or the shortest path during the search process.
    b. Completeness means that the algorithm must always find a solution when a solution exists so it must return the path if it exists or null if it does not.
    c. Acceptable time and space complexities are very important, because creating an optimal and complete algorithm is useless if it has slow running time and consumes considerable amount of memory.
2. The implementation of the algorithm using a programming language was the second phase. The preferred algorithm implementation language chosen was an OOP language (even it is preferable to be an open source or fully supported language like Java, VB, or C#). This phase is important for the next phase which is the simulation, because it will use the implementation of the algorithm and apply it on the agent.
3. The simulation of the algorithm on a computer using 3D graphics engine was the third phase were a virtual city was created representing the environment and a virtual plane representing the agent. The manual mode with obstacle avoidance and the map mode were implemented and simulated using the algorithm that we will discuss in this paper.

4. IMPLEMENTATION AND TESTING

This section covers the Map mode algorithm's design and its implementation. It also covers the implementation of the environment (i.e. Simulator) on which the algorithm was tested and simulated using virtual agents in 2D and 3D environments.

4.1 Description of the Environment

In AI, the environment is classified based on many properties (single vs multi-agent, stochastic vs deterministic, discrete vs continuous…). In this case the agent is moving in a partially observable, single-agent, stochastic and continuous environment which is considered one of the hardest environments in AI.

- **Partially Observable Environment:** the agent's sensors have no access to all states of the environment. In this project, the agent can't determine which location has an obstacle and which does not unless the obstacles are in near proximity (in sensors range).
- **Single Agent Environment:** there is only one agent in the environment, and there are no other objects that can decrease or increase the agent's performance measure
- **Stochastic Environment:** the next state of the environment is not determined by the current state and/or the actions executed by the agent [11].
- **Dynamic Environment:** is when the environment is changing continuously while the agent is thinking or acting. In this project, obstacles can appear randomly and suddenly.
- **Continuous Environment:** the number of clearly defined percepts and actions is unknown. The speed and location of the agent sweeps through a range of continuous values and do so smoothly over time [11].

4.2 The Map Mode Algorithm's Design in 2D Environment

In map mode, the agent must move automatically and autonomously from its current position to a predefined target location on a map following the shortest path possible and avoiding predefined or sudden obstacles on the map.

4.2.1 Map Generation

The search process starts from the target location and ends when the starting location is found. Each location will have a specific cost which is equal to the distance to the target location. Therefore, the cost of the target (Goal) location is equal to 0.

Since the agent can move in four directions in 2D environment (up, down, left and right) then each neighbor of the target is checked:

- If it is not an obstacle.
- If it is not a starting location or target location.
- If it is inside the boundary of the environment.
- If it is not visited.

If all these conditions are satisfied, then the neighbor's cost will be equal to 1 since they are only one step away from the target. After that, all these neighbors are visited and all their possible neighbors' cost will be equal to 2. The map generation continues in this manner where the cost of every neighbor is equal to the cost of the location that is currently visited plus 1.
The Map Generation pseudo code is shown in Figure 1.

```
Set search node to target node.
Set the TempCost to 1.
 L1:
 Check an unchecked adjacent node If (not the start node And inside environment boundary
And not Obstacle)
{
 Mark the node as checked.
Set its weight to TempCost.
}

If all adjacent nodes are checked then
{
Increment TempCost.
Set search node to a sub node.
If all sub nodes are checked then return
}

repeat L1.
}
```

Figure 1. Map Generation Pseudo Code

After the Map Generation algorithm is executed the map is shown in Figure 2.

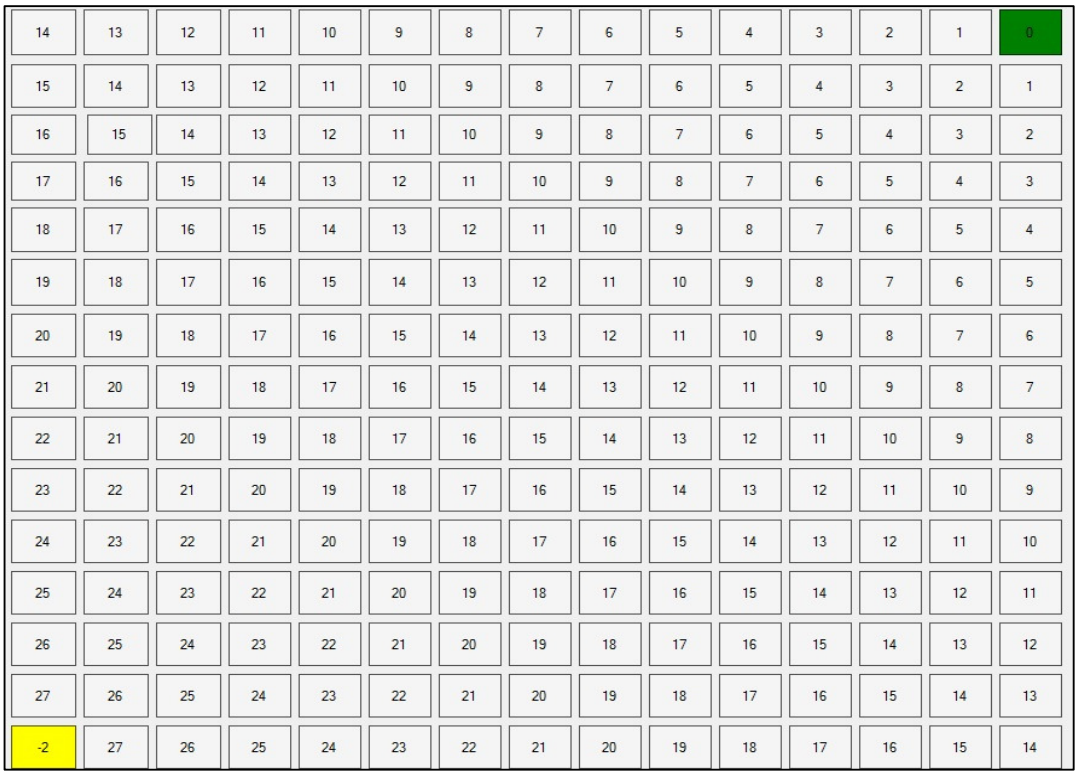

Figure 2. Map Generation

Where the yellow location is the starting location and the green location is the target location.

4.2.2 Finding the Shortest Path Algorithm in Map Mode:

After the map is generated, finding the shortest path becomes simple. Therefore, a simple Hill Climbing algorithm is used to find the shortest path.

The Hill Climbing algorithm is used because:

- It is implemented as a loop that continually moves in the direction of the "best" neighbour with increasing value that is, uphill [1], it can also be implemented in the opposite way i.e. in the direction of the neighbour with decreasing value that is downhill
- It terminates when it reaches a "peak" where no neighbour has a higher value (Climbing to the top) or where no neighbour has a lower value (descending the Hill to the bottom).
- The algorithm does not maintain a search tree, so only the current state and the value of its objective function are stored saving by that a lot of space.

Hill Climbing "does not look ahead beyond the immediate neighbours of the current state" this is why it is a greedy local search algorithm [11].

4.2.3 Applying Hill Climbing

In this situation the search will start from the starting location until finding the target. Since the target has cost = 0 then it will be needed to get down from a required location on the hill to find the global minimum, which is the lowest point in the hill. The algorithm should always choose the "best neighbor", because the cost represents the distance to the goal. Note: all obstacles have cost = 1000, Unvisited node = 800 and starting node cost = -2. Finding the shortest path algorithm pseudo code is shown in Figure 3.

```
Set search node to starting node.
L1:
Check all adjacent node If (not the start node and inside environment boundary){
If adjacent node is target then return target is found.
else
If there is no possible route exists then return no route exist.
else
Set search node to a sub node that has the lowest cost.
Mark the visited search node.
Repeat L1
```

Figure 3. Shortest Path Algorithm

Normally, Hill Climbing Algorithm is not complete and not optimal because of several limitations explained in [11, 12] and may other textbooks. The enhancement done here that make it complete in this case is the way the map is generated initially that does not allow the appearances of local minima or plateaus.

Giving the constraints that we set for the movements allowed of the agent in 2D, the algorithm finds the best path with lowest cost and effort. The altered algorithm is also complete since it always gives a solution whether a route exists or not. A route does not exist if the target or starting locations are surrounded with obstacles. This algorithm has also a low memory (space) complexity of (O (n)), since there are no routes or nodes that are stored, and there is only one loop that is iterating till the target is found.

4.3 The Algorithm's Implementation in 2D Environment

The 2D Environment is considered easier to deal with since there is no height (z-axis). The coordinates of the locations on the map are defined in terms of x-axis and y-axis coordinates. For that, we are using a 2 dimensional matrix to define the coordinates of each location. The 2D environment consists of 15 x 15 locations, so the array of locations is defined as: a [15, 15].
In the map based mode, the user sets a starting point, target point and obstacles on the map, and the agent must move automatically from its location (starting point) to the target point avoiding obstacles (predefined and sudden) applying the shortest path algorithm.

4.3.1 Map Generation Implementation

Based on the algorithm design, the cost of every location must be stored on the map and since we are using an array to store these locations, then the value of each member of this array will be equal to the cost; if the target is at x=1 and y=1 then a[1][1]=0. In order to visit every position and all its neighbors, a FIFO queue is needed to store the visited locations. This idea resembles the method used in Breadth first search in trees where all nodes are visited level by level, where each level contains nodes of equal costs in order to find the goal node which is in this case the starting node. The implementation of the algorithm is shown in Figure 4.

```
Initial state: all elements in the array = 800
After assigning the target and starting point a[tx,ty]=0  and a[stx,sty]=-2
Put the target point in FIFO queue Q
While the queue is not empty {
String xy = Q.get() //Ex: xy = 2,8
Int x = xy.x
Int y = xy.y
Int c = a[x,y]
If (left , right , up and down neighbors =800 & neighbor inside boundary){
a[neighborX,niehborY] = c +1
Q.Put(neighbor)
}}
```

Figure 4. Used code for Algorithm

where, tx and ty are coordinates of target location.
stx and sty are coordinates of starting location. Obstacles locations have cost = 1000.

4.3.2 Finding Shortest Path Implementation

As mentioned before, a simple Hill Climbing algorithm can be used to find the shortest path and simulating the movement of the agent. In this case, we want to reach a global minimum which is zero (target point). So, we need to choose the neighbor with the lowest cost. The algorithm implementation pseudo code is shown in Figure 5.

```
int x = stx  int y=sty
While(x!=stx && y!=sty)
{
Find Min Cost Neighbor ( Check left, right , up and down neighbor)
If(Min Cost = 800 or 1000) { return no route }
Else
{
x=MinNieghbor.X  // MinNiehbor is neighbor with lowest Min Cost
y=MinNieghbor.Y
Mark  MinNieghbor }}
```

Figure 5. Algorithm Pseudo code

As shown in the code in figure 5, the best location that is selected next is the one with the lowest cost. There are two ways to detect that there is no route that exists: the first way is when the best neighbor is undiscovered (cost = 800) which means that the target or agent is surrounded with obstacles. The second way is when the best neighbor has cost = 1000 which means that the starting location is surrounded directly with obstacles (see Figure 6).

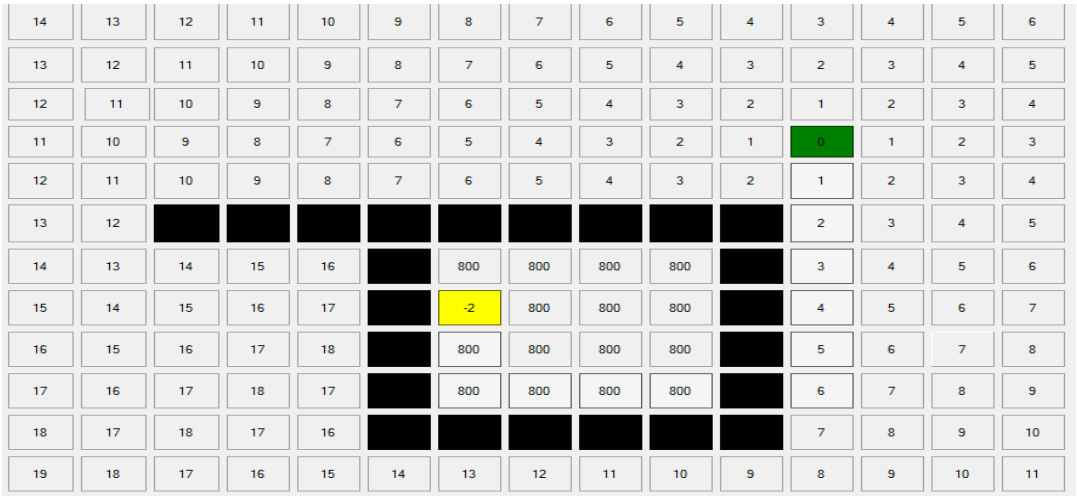

Figure 6. Blocked path example

It should be mentioned that 1000 and 800 and all other numbers used here can be increased or changed depending on how much the environment is big.

In order to simulate the Map Mode and the Direct Drive Mode, a form containing a Map of clickable 15 x 15 buttons is used. Each button on the map represents a location from [0, 0] till [14, 14] ([Row, Column]).

The user first assigns starting location which will color the pressed button with yellow. Then, the user presses another button which will be the target location, and will be colored green. After that, the user can control the agent freely, and move it up, down, left, and right using the W, S, A, D keyboard keys respectively. The user is also able to add obstacles on the map by pressing a location which will be then marked with black. Whenever the user wants, he can press the generate map and the find path buttons. The first will change the text of each button to a number that represents the current cost of location represented by this button. The find path button will trace the best or shortest path in red color. Figure 7 shows the starting location (yellow), target location (green), sudden obstacles (black), shortest path (red). In this image the agent was moving already toward the target and sudden obstacles (in black) were put on the map. The algorithm recalculates the shortest path and chooses adequately the new path.

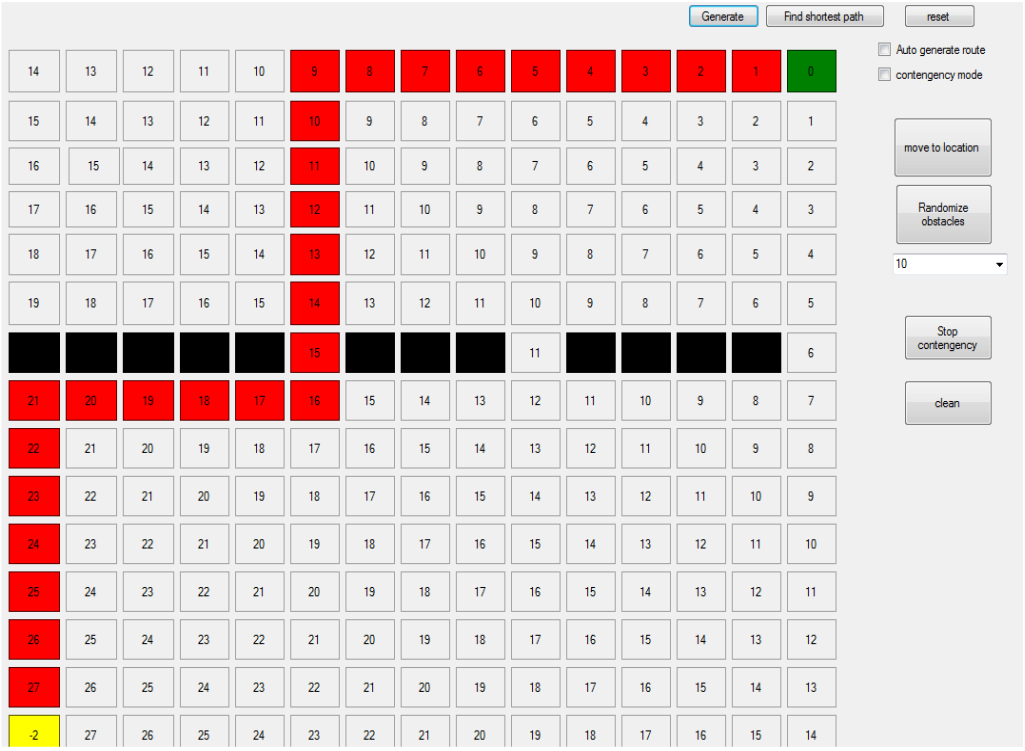

Figure 7. Path Finding

4.4 The Map Mode Algorithm in 3D Environment

In the 3D environment there is an extra dimension (Z-axis or height) which makes the movement of the agent more complicated. Consider that the agent is flying in a city which contains buildings of different heights and shapes. This agent needs to find the shortest path to a destination. It is logical that if the agent changes its height while moving, it will probably reach the target location faster than moving on the path of the roads of the city. But since our environment is partially observable and could change at any moment, therefore, changing the height is not always an optimal solution (like if the obstacle was a tall mountain or building). So first, the algorithm tries to find a shortest path on the same height like in 2D mode, but the difference in that if no route was found (obstacles are surrounding completely the agent or the target) the application will not return that no route exists. It will return that no route exists at the current height of the agent and offers the user the option to choose to move the agent following the shortest path considering that there are no obstacles on that height (see Figure 8).

Set search node to starting node.
L1:
Check all adjacent node If (not the start node And inside environment boundary){
If adjacent node is target then return target is found.
Else
If there is no possible route exists then
remove all obstacles on the map
Call Generate Map and find shortest path
else
Set search node to a sub node that has lowest cost.}
Mark the visited search node.
Repeat L1

Figure 8. Algorithm Development

The question now is how the agent will avoid the obstacles?
The answer is simple: as long as the agent is sensing that an obstacle exists in front of it at a certain distance, it goes up. Consider the case where the obstacle is a building, when the agent senses that an obstacle exists, it will increase its height till the agent is higher than the maximum height of the building.

What are the changes in the three main methods (Generate, find and move) in the map mode?

- The generate map method remains the same.
- Finding the shortest path has some changes when no route exists on the same height, since this method detects whether a route is found or not.
- Moving the agent as in 2D depends on marked route made by the find shortest path algorithm. When there is no route on the same height. If the agent faces an obstacle while moving on the shortest path it will avoid it by flying over it.

4.4.1 The 3D Simulator

In order to test the algorithm, we needed a virtual but realistic environment, obstacles and a virtual agent to represent a plane. Since the algorithm was implemented using C#, we had to use a 3D engine which is based on this language. The best choice was XNA Game Studio 4.0 in Microsoft Visual Studio 2010. A 3D City was built and a plane can now move in it (see Figure 9).

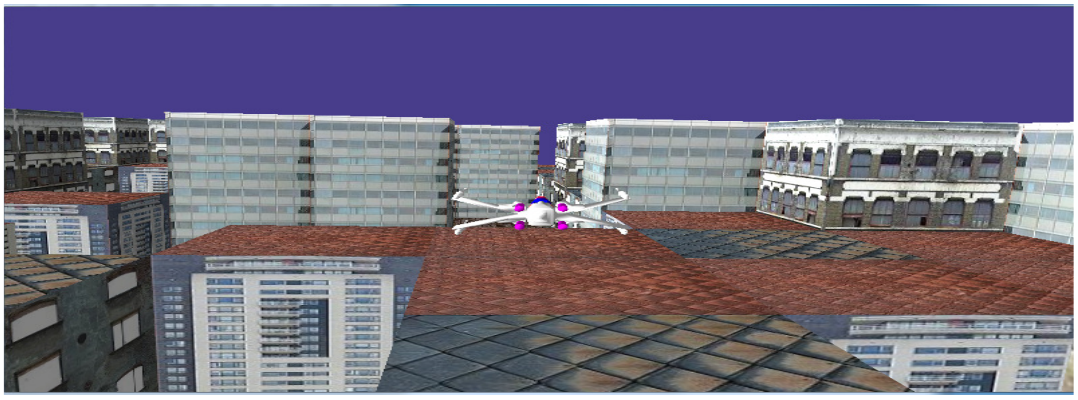

Figure 9. 3D screenshot

Figure 9 shows a complex environment where buildings (obstacles) of different sizes can be created.

In order to apply the map mode algorithm, to add obstacles and to retrieve the agent's location, a windows form which contains a map that work on the same concept as the 2D simulator (buttons, colors and labels) was created. The map form, loads first and through it, the user can start the 3D simulator. The user can assign a starting location and a target location and add obstacles wherever he wants on the map. The obstacles will appears suddenly as buildings in the simulator.

During manual control the user can view the agent's position on the map (the location will be colored yellow). There are also two labels which represent the x-axis and y-axis coordinates of the plane. We added an extra label in order to count the steps that the agent has taken since the start of the simulator. The random map button adds a random building in random locations. The user can start map mode by pressing its button after assigning start and target location. After that, the plane will start moving according to the red path that can be viewed on the map plus the current location of the plane.

4.4.2 Map Generation Optimization

Since we have experimented that the map generation algorithm has the highest running time and highest space complexity, we tried to apply some improvements.

We measured empirically the running time using a diagnostic tool provided by visual studio called Stop Watch (actually it is a famous method for empirical analysis of algorithms that is based on the same concepts in different languages). This tool is able to measure the running time of a method and return the time in milliseconds [19, 20, 21].

Upon the results of using this analysis tool we made the following improvements:

- Resetting the queue after every move.
- Stop searching when the starting location was found instead of discovering every location on the map.

The second solution was very effective and had a great impact on the efficiency and execution time of the algorithm according to the distance between the agent's location and target point, where the improvement was bigger when the distance is closer, and it starts to decrease as the distance increases. The improvement in time was between 0 and 95% according to the distance.
Many tests were also performed to measure the performance and stability of the algorithm and the simulator, especially on threads. These tests had to make sure that each thread does not interfere with the other, and it's terminated when its job finishes or the application closes.

Other tests covered the manual control of the agent, where turning and moving were tested in order to represent the most accurate and realistic navigation that resembles that of a real plane.

5. REAL WORLD SCENARIO

This project's purpose is to be implemented on a real flying agent in a real environment. For a successful implementation of the algorithms presented in this paper, the agent needs to be equipped with powerful and accurate sensors and other hardware tools like speedometers or accelerometers, gyroscope and any useful hardware for an agent in flight.

The speed of the agent should be measured at each instant. Configurations of the environment with all sudden obstacles should be continuously generated. Precise location of obstacles given to the map generation algorithm is crucial for any success navigation in a 3D environment.

6. CONCLUSION

The problem of path planning in intelligent autonomous vehicles is still topical research field although it has been studied by the research community for many decades. Any intelligent autonomous vehicles have to include path planning into their deliberation mechanisms [16].
The article's aim (in addition to manual control) is to find as much as possible a complete and logical solution for automatic agent navigation in a dynamic two and three dimensional environment, where the definitions of the obstacles and restricted areas along with the agent's position are altered after each search.

In manual mode, the objective was to simulate the movement control of the agent, and automatic obstacle avoidance.

Through The Microsoft XNA and Visual Studio, we were able to create an environment that resembles a real city that can be easily manipulated by the user, and we chose the airplane to

represent the AI agent in this 3D environment. The movement of the plane was smooth and mimics every possible direction that a real plane can perform. In addition, the agent was able to sense all obstacles from a certain distance and avoid them by moving to another direction.

In the map mode which was the main focus of the project, the agent is required to move on a predefined map from one location to another avoiding all kinds of obstacles on the shortest path possible inside the boundary of the environment. The solution consisted of three main algorithms: map generation, finding the shortest path and agent's navigation algorithm. First, Map Generation was used in order to change the partially observable environment to a fully observable environment at some time 'T' (since the environment is still dynamic), by assigning costs to every location in the map. The implementation of this algorithm was based on the iterative implementation of breadth first search with some modifications. Second, finding the shortest path is the search algorithm which finds the lowest cost path to the target. The implementation of this algorithm was based on the Hill Climbing algorithm (descent version), i.e. in this case the agent is on the top of the hill and needs to find the shortest path to the bottom (Global minimum). The Map generation part avoids a lot of limitations of Hill Climbing. Third, moving the agent was a simple algorithm which consists of moving and directing the agent according to the shortest path previously generated, and must be generated with every step that the agent makes in such continuous and dynamic environment.

7. FUTURE WORK SUGGESTIONS

The following are some of future work suggestions that could be applied to the algorithm:

- More optimization can be achieved in the map generation algorithm. This part of the algorithm has the highest complexity in the project, and it must be executed at every location in order to check if the path is still the optimal one or not. In addition to that improving the way that the agent can sense obstacles is very important. Through sensor's data in a real world environment, the agent can build a knowledge base about the properties of the obstacles (sizes, shapes, motions), this allow the agent to act accordingly.
- We also hope in the future to try the Map Mode algorithm on a real plane in a real environment. The main aim of any future project will be the construction of a small plane with the ability to sense obstacles through sensors like sonars per example. This plane can be controlled by a smartphone or a computer via wireless radio connection or via the internet. The plane should have the ability to control itself in case it encounters obstacles, and should able to move on the map automatically (according to the algorithm presented in this paper). Other information might be necessary to be taken into consideration like the position, altitude, distance and signal strength of the plane and many others. More empirical and mathematical analysis on the algorithms in this paper will expanded in subsequent work.

ACKNOWLEDGEMENTS

We would like to express their special thanks of gratitude to the president of Lebanese International University HE Abdel Rahim Mourad for his continuous encouragement of research activities in the university. Secondly, we would like also to thank the LIU Bekaa campus administration for their full support during the project and last but not least special thanks go to the students Youssef Al Mouallem and Hassan Al Kerdy for their contribution and insights of the project.

REFERENCES

[1] Steven M. LaValle (2006) "Planning Algorithms", Cambridge University Press, Cambridge University Press.

[2] Antonio Benitez, Ignacio Huitzil, Daniel Vallejo, Jorge de la Calleja and Ma. Auxilio Medina (2010) "Key Elements for Motion Planning Algorithms", Universidad de las Américas – Puebla, México.

[3] Kavraki L. E., Svestka P., Latombe J.-C., Overmars M. H. (1996) "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", IEEE Transactions on Robotics and Automation, vol. 12, No 4, pp566–580, doi:10.1109/70.508439.

[4] Juan~Manuel Ahuactzin and Kamal Gupta (1997) "A motion planning based approach for inverse kinematics of redundant robots: The kinematic roadmap", IEEE International Conference on Robotics and Automation, pp3609-3614, Albuquerque.

[5] Amato N. M. & Wu Y (1996) "A randomized roadmap method for path and manipulation planning", IEEE Int. Conf. Robot. and Autom., pp113–120.

[6] Amato N., Bayazit B., Dale L., Jones C. &. Vallejo D (1998) "Choosing good distance metrics and local planer for probabilistic roadmap methods", Procc. IEEE Int. Conf. Robot. Autom. (ICRA), pp630–637.

[7] M. LaValle and J. J. Kuffner (1999) "Randomized kinodynamic planning", IEEE Int. Conf. Robot. and Autom. (ICRA), pp473–479.

[8] M. LaValle, J.H. Jakey, and L.E. Kavraki (1999) "A probabilistic roadmap approach for systems with closed kinematic chains", IEEE Int. Conf. Robot. and Autom.

[9] Geraerts, R.; Overmars, M. H. (2002) "A comparative study of probabilistic roadmap planners", Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR'02), pp43–57.

[10] C. Ray Asfahl (1985) "Robotos and Manufacturing automation", John Wiley & Sons, Inc.

[11] Russell and Norvig (1 April 2010) "Artificial Intelligence: A Modern Approach", 3rd edition by - Pearson. ISBN-10: 0132071487, ISBN-13: 978-0132071482.

[12] George F. Luger (2009) "Artificial Intelligence: Structures and Strategies for Complex Problem Solving", 6th edition, Pearson College Division.

[13] Paul Deitel and Harvey Deitel (2011) "C# 2010 for programmers", Prentice Hall, 4th edition.

[14] John Sharp (2011) "Microsoft Visual C# 2010 Step by Step", Microsoft Press.

[15] Haissam El-Aawar and Mohammad Asoud Falah (April-2009) "An Integration of a High Performance Smartphone with Robotics using the Bluetooth Technology", Communications of SIWN (ISSN 1757-4439).

[16] David Sislak, Premysl Volf & Michal Pechoucek (2009) "Flight Trajectory Path Planning", Proceedings of ICAPS 2009 Scheduling and Planning Applications woRKshop (SPARK), pp76-83.

[17] David Sislak, Premysl Volf, Stepan Kopriva & Michal Pechoucek (2012) "AgentFly: Scalable, High-Fidelity Framework for Simulation, Planning and Collision Avoidance of Multiple UAVs. In Sense and Avoid in UAS: Research and Applications", Wiley: John Wiley&Sons, Inc., pp235-264.

[18] Benitez A. & Mugarte A. (2009) "GEMPA:Graphic Environment for Motion Planning Algorithm", In Research in Computer Science, Advances in Computer Science and Engineering, Vol. 42.

[19] Sedgewick, Robert (2002) "Algorithms in Java", Parts 1-4. Vol. 1. Addison-Wesley Professional.

[20] Levitin, Anany (2008) "Introduction To Design And Analysis Of Algorithms", 2/E. Pearson Education India.

[21] Goodrich, Michael T., & Roberto (2008) "Tamassia. Data structures and algorithms in Java", John Wiley & Sons.

[22] http://www.microsoft.com/en-us/download/details.aspx?id=23714

[23] http://www.riemers.net visited at 1/5/2013

[24] http://www.techopedia.com/definition/27489 /activity-diagram visited at 15/5/2013

[25] http://msdn.microsoft.com visited at 7/5/2013.

AUTHORS

Haissam El-Aawar is an Associate Professor in the Department of Computer Science and Information Technology at the Lebanese International University where he has been a faculty member since 2009.

Haissam completed his Ph.D. and M.Sc. degrees at the State University "Lviv Polytechnic" in Ukraine. His research interests lie in the area of Artificial Intelligence, theory of complexity, microprocessors evaluation, CISC- and RISC-architectures, robotics control, mobility control and wireless communication.

Hussein Bakri is an instructor in the department of Computer Science and Information Technology at Lebanese International University. He has completed his M.Sc. degree at the University of St Andrews in United Kingdom and he is now continuing his Ph.D. His research interests lie in the area of Artificial Intelligence, software engineering, virtual worlds and virtual reality and cloud computing.