A SIMULATED MOTION PLANNING ALGORITHM IN 2D AND 3D ENVIRONMENTS USING HILL CLIMBING

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

[1]Associate Professor, Computer Science/Information Technology Departments, LIU, Bekaa-Lebanon

[2]Instructor, Computer Science/Information Technology Departments, LIU, Bekaa-Lebanon,

ABSTRACT

This paper describes a computer simulated artificial intelligence (AI) agent moving in 2D and 3D environments. In the presented algorithm, the agent can take two operating modes: Manual Mode and Map or Autopilot mode. The user can control the agent fully in a manual mode by moving it in all possible directions depending on the environment. Obstacles are sensed by the agent from a certain distance and are avoided accordingly. Another important mode is the Map mode. In this mode the user create a custom map where initial position and a goal position are set. The user is able also to assign sudden and predefined obstacles. By finding the shortest path, the agent moves to the goal position avoiding any obstacles on its path.

The paper documents a set of algorithms 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. These obstacles are avoided also in manual mode and the agent moves automatically to a safe location. The implementation is based on the Hill Climbing algorithm (descent version), where the agent finds its path to the global minimum (target goal). The Map generation algorithm, which is used to assign costs to every location in the map, avoids a lot of the limitations of Hill Climbing.

KEYWORDS

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

1. INTRODUCTION

A variety of applications of AI, robotics and virtual reality especially applications for navigating agents in 2D and 3D environments, are based on agent motion planning algorithms. Partially observable and non-observable environment present always a challenge to any artificially intelligent agent due to the very high level of uncertainty. This case can be described 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 (plane drones) where the environment becomes more complex, obstacles are more unpredictable, and agent's motion becomes more difficult.

A crucial and necessary condition is that the data taken from sensors should be precise in order for the agent to respond in a well-timed manner tackling any contingencies or sudden obstacles during the 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

DOI : 10.5121/ijaia.2014.5103

real agents. The algorithms presented in this paper needs to be thoroughly tested on a real airplane drone in a real world scenario. This article describes the design, implementation and simulation of the algorithms. Using these algorithms an agent can find its way to a target location automatically and without human intervention in a complex, continuous and partially observable 2D and 3D environments. In such environments, the agent encounters many static or moving obstacles on its way like when a plane faces 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 both in 2D and 3D environments. In 3D environment the agent is simulated as a plane in a virtual city with virtual obstacles modeled as buildings that may suddenly appear.

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. The Piano Mover's is a classical motion planning problem known in the literature [1].

One of the fundamental problems in robotics is a motion planning. 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 demand solutions for other problems such as controlling motion in real time, adequate sensing and task planning.

The problem of motion planning can be identified 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

The probabilistic roadmap methods (PRMs) is one of the most significant categories of motion planning approaches studied in the literature [3, 4, 5, 6, 7, 8]

A roadmap can be defined as a union between many one dimensional curves. All start points and goal points in C_{free} are 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.

The probabilistic roadmap methods utilize randomizations comprehensively to build roadmaps in C Spaces. With no calculations needed, heuristic functions can be used for sampling all C obstacles and C spaces. The method of probabilistic roadmap planner involves two important phases: a phase for construction or learning and a phase for querying. Robots with many degree of freedom utilizes PRM planner. The construction phase consists of collision-free configuration as nodes or vertices in a graph and of collision-free paths as edges. The roadmap is built by repeating the following steps:

- Select a certain arbitrary configuration of the robot, check the collision degree and repeat this step until the arbitrary configuration is without collisions.

- Try to join the previous configuration to the roadmap using a quick local planner.

To discover a path in the query stage, the idea is to connect original and target configurations to the roadmap and to search the roadmap for a arrangement of local paths linking these vertices. The path is thus obtained by a Dijkstra's shortest path query [2, 3, 9]

2.3 ROBOT'S (AGENT'S) WORKSPACE (ENVIRONMENT)

In motion planning, a robot is defined as an object or as versatile, multipurpose or adaptable 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].
Robots can travel in a countless number of environments comprising from 2D, 3D to even N-dimensional. A robotic agent can be symbolized as a point in space having translational coordinates (x, y, z). This is an abstracted form that aims to solve the problem. Generally speaking in 3D environments, it is a frequent 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 vital to intelligent and rational agents. By planning, robots can attain their goals autonomously and with great degree of flexibility through continuously constructing their well-defined sequences of actions. Planning as a sub discipline in the artificial intelligence field has been an area of research for over three decades. The difference between planning and problem solving has been obscure and hard to define in all the history of study in this domain, [1, 2].
In artificial intelligence, the term planning takes a more discrete representation 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 categories of discrete representations can still be demonstrated using continuous spaces but it appears more suitable to describe them as finite and predetermined 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 paper'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 many divers interactive applications. Games for X-Box 360, for mobile phones and for windows operating systems can be built easily with XNA Game Studio [28].

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 guide

found in [29] is used to build a virtual 3D city and a flying airplane, on which the algorithm is implemented.

Direct X is recommended in order to have the optimal performance in the 3D Simulator [30].
C sharp (C#) is an object oriented platform-independent language developed by Microsoft (.Net) following the tradition of Java [13, 14, 31]. The C# language is intended to be a simple, modern, general-purpose, object-oriented programming language. It present a robust typing, functional, class-based programming, which offers the programmer myriad of rigid and easy programming capabilities. C# helps programmers to write programs using parallel threads, called asynchronies methods, and this enable us to fully control the dataflow needed to govern effectively the agent both in 2D and 3D environments.

2.6 PREVIOUS PLANNING AND OBSTACLE AVOIDANCE ALGORITHMS

Many planning algorithms are based on the principle of optimality or on the norm of dynamic programming. This provides an understanding for computation needs by reducing considerable computation efforts in such algorithms. The well-known Dijkstra's algorithm for finding single source shortest paths in a graph is a special form of dynamic programming. Nonholonomic planning and kinodynamic planning are based on the idea of dynamic programming limited only to problem with low degree freedom [1, 7].

Kinematics, which is a discipline that explains the motion of bodies or points, is studied through difficult models in steering methods of agents in motion planning problems [15, 16].

The nonholonomic planning has been a great research field in agent motion planning. Most recent approaches rely on the existence of steering methods that can be used in combination of holonomic motion planning techniques [17, 18].

Kinodynamic planning is one of the major PSPACE hard problems. It is as difficult to solve as the mover problem presented in the literature. The most effective method in kinodynamic planning is the randomized approach. The finest two practices used in randomized kinodynamic planning are the randomized potential field and probabilistic roadmaps. Probabilistic roadmaps answer the problem of defining a path between an initial configuration of the agent and a target configuration while escaping any collisions. They take random samples from the configuration space of the robot, testing them for whether they are in the free space, and use a local planner to attempt to connect these configurations to other nearby configurations. When initial and target configurations are set precisely, a graph search algorithm can be applied to the resulting graph to determine a path between the initial and target configurations. On the other hand in a randomized potential field approach a heuristic function is defined on the configuration space that attempt to steer the robot depending on a heuristic function toward a goal state through gradient descent if the search becomes trapped in a local minimum different solutions are present An simple one is the use of random walks [1, 4, 7, 19, 20].

2.7 LOCAL SEARCH ALGORITHM: HILL CLIMBING

The hill-climbing search algorithm is simply a local greedy search algorithm that continually moves in the direction of increasing value—that is, uphill or decreasing value that is downhill. The algorithm terminates in a goal when it reaches a global maximum (no neighbor has higher values) or global minimum (no neighbors have lower values) depending on the problem solved. The algorithm does not maintain a search tree, so its space complexity is considerably low compared to other search algorithms [11, 12].

Hill Climbing has many shortcomings. It is incomplete because it can get stuck on a plateau (flat area of the state-space landscape) or on a local minima or maxima (a peak that is higher than each of its neighboring states but lower than the global maximum or vice versa). Several improvements were proposed in [11, 12].

The algorithm proposed in this paper is a combination solution to previous problems caused by previous planning algorithms mentioned above. Our map generation algorithm guarantees that the agent will always find a global minimum also solves Hill climbing shortcomings. Our approach will be presented in the following sections.

3. MOTION PLANNING ALGORITHM

As stated previously, the algorithm has two central modes: **Manual and Map mode**. In manual mode, the user can control the agent by moving it in all possible and allowed directions depending on the environment. In this mode there is a special feature called **Contingency Mode** that enables the agent to sense the obstacle from a predefined distance and evade it safely. The Second mode is the map mode. In this mode the user assigns a starting point, target point and any number of obstacles required. Now in order to find the shortest path to a goal position, the agent then must apply the algorithm presented later, moving and changing directions automatically when encountering all kinds of obstacles [21, 22, 23, 24].

The implementation of algorithm passed through three essential stages:

1. The Design of the algorithm stage was the first stage where the algorithm was designed to meet as much as possible the following requirements: optimality, completeness, and acceptable time and space complexities:

    a. An Optimal algorithm must always find the shortest path with the lowermost cost throughout the search process.

    b. A complete algorithm always finds a solution (or a path) when it is available so it must return the path if it exists or returns nothing when it does not find one.

    c. Satisfactory time and space complexities for any algorithm are essential and desirable, because designing an optimal and complete algorithm is unusable if it has slow running time complexity and takes significant amount of memory.

2. The second stage was the choice of programming language for implementing the designed algorithm. The preferred implementation language chosen for this project was an Object Oriented Programing language that is fully supported like C#. This stage is important for the next stage, which is the simulation, because it will use the implementation of the algorithm and apply it on the agent.

3. The third stage was the simulation of the algorithm on a computer using 3D graphics engine. A 3D virtual city was created modeling a 3D environment and a virtual aircraft modeling a flying agent. The manual mode and the map mode, both with obstacle avoidance, were implemented and simulated using the algorithms that we will discuss later.

4. IMPLEMENTATION AND TESTING

This section covers the design, implementation and simulation of the Map mode algorithm using virtual agents in 2D and 3D environments.

4.1 DESCRIPTION OF THE ENVIRONMENT

Many properties characterize an environment in artificial intelligence (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 know the locations with obstacles or the safe locations unless the obstacles are in near proximity (in sensors range).

- **Single Agent Environment:** characterizes an environment with only one agent. There are no other agents that can decrease or increase the performance measure of the agent.

- **Stochastic Environment:** is when 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 there is a continuous change in the environment while the agent is thinking or acting. In this project, obstacles can appear arbitrarily and unexpectedly.

- **Continuous Environment:** is where there are an unknown number of clearly defined percepts and actions. 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. The precise cost of each location is the distance of this location to the target position. Consequently, the cost of the Goal position is equal to 0. The designed agent in 2D mode is only allowed to move in four directions (up, down, right and left). Each neighbor of the goal location is then checked:

- If it is not an obstacle.
- If it is not an initial position or target position.
- If it is inside the boundary of the environment.
- If it is not visited.

If all these checks are achieved, then each neighbor's cost will be equal to one since they are only one step away from the goal. After that, incrementally all these neighbors are visited and all their possible neighbors' cost will be equal to two. The map generation follows this way by having at

each step the cost of every neighbor is equal to the cost of the position that is currently visited plus 1.

In Figure 1, the pseudo code of The Map Generation is presented.

```
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. Pseudo Code of Map Generation

Figure 2 shows the map resulting from the execution of the map generation mode, where the yellow position is used to specify the starting position and the green position is used to specify the goal position.

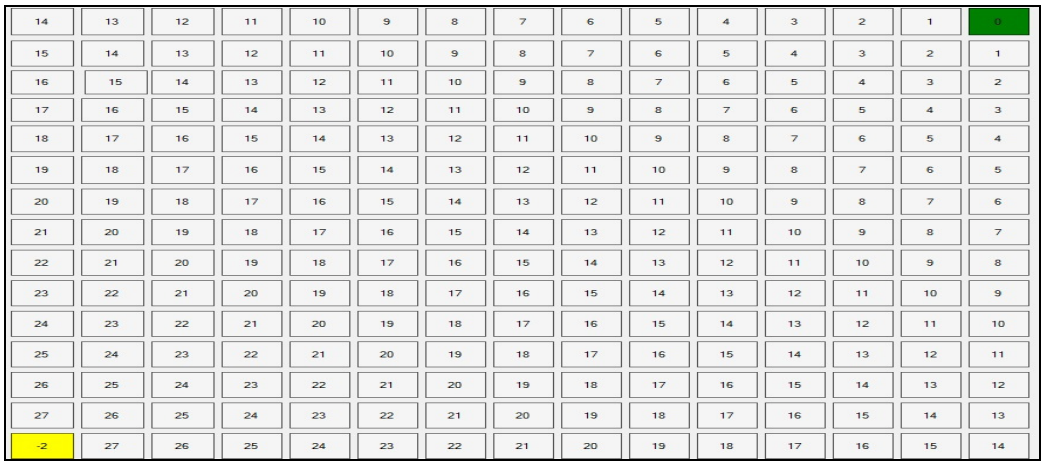

Figure 2. Map Generation

4.2.2 FINDING THE SHORTEST PATH ALGORITHM IN MAP MODE:

Finding the shortest path becomes simple after the map is generated. 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).

- Hill Climbing does not maintain a search tree, so it saves a lot of memory space. Only current state and value of its objective function are stored.

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

4.2.3 APPLYING HILL CLIMBING

In this situation the search will start from the starting location until finding the target. The lowest point with the lowest cost is the target location with a cost of 0 so the agent descends from its current 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. Figure 3 presents the shortest path algorithm pseudo code used to find the goal.

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

Figure 3. Shortest Path Algorithm

Generally speaking, Hill Climbing Algorithm is not complete and not optimal because of several problems explained in [11, 12]. The map generation algorithm does not allow the appearance of local minima or plateaus thus making Hill Climbing complete in this case giving a solution whenever a route exist. Given the limitations of the movements allowed for the agent in 2D in the current design (up, down, left and right), the algorithm always discover the best path with the lowest cost and effort.

A path does not exist if the initial position is enclosed with obstacles. As a result the algorithm has low space complexity of O (n) since there is no storage of paths, and there is only one loop that is repeated till the goal 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. A 2

dimensional matrix is used to represent the coordinates of each position. The 2D environment consists of 15 x 15 positions, so the array of positions is defined as: a [15, 15].

The user, in the map based mode, sets a starting location, target location and obstacles on the map, and the agent must move automatically from its location (starting) to the target location 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 location and all its neighbors, a FIFO queue stores the visited positions. This idea looks like 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. Figure 4 shows the implementation of the algorithm.

```
Array Initializers: all elements in the array = 800
Assign the target and starting locations: a[tx,ty]=0  and a[stx,sty]=-2 then
Put the target location in FIFO queue Q
While the queue is not empty {
String xy = Q.get() //Ex: xy = 3, 9
Int x = xy.x
Int y = xy.y
Int c = a[x, y]
If (left , right , up and down neighbors =800 And neighbor inside boundary){
a[neighborX, niehborY] = c +1
Q.Put(neighbor)
}}
```

Figure 4. Used code for Algorithm

where, tx and ty are coordinates of goal position.
stx and sty are coordinates of a starting location. The obstacles locations have cost = 1000.

4.3.2 FINDING SHORTEST PATH IMPLEMENTATION

As stated previously, a simple Hill Climbing algorithm is used to find the shortest path and to simulate the movement of the agent. The objective is to reach the global minimum with the cost of 0 which is the target point. So, we need to choose the neighbor with the lowest cost. Figure 5 presents the algorithm implementation pseudo code.

```
integer x = stx ;  integer y=sty
While(x is not equal to stx And y is not equal to sty)
{
Find Low Cost Neighbor ( Check left, right , up and down neighbor)
If (Low Cost = 800 or 1000) { return no route }
Else
{
Assign Low Cost to x: x=MinNieghbor.X  // MinNiehbor is neighbor with lowest Min Cost
Assign Low Cost to y: y=MinNieghbor.Y
Mark  MinNieghbor
}
}
```

Figure.5.Algorithm implementation Pseudo code

As shown in the pseudo code in Figure 5, the best position that is chosen next is always the position with the lowest cost. There are two methods to detect that there is no path that exists:

1) The best neighbor is not discovered yet (cost = 800) meaning that the goal or agent location is enclosed with obstacles

2) The best neighbor has a cost of 1000 meaning the initial position is enclosed directly with obstacles (see Figure 6).

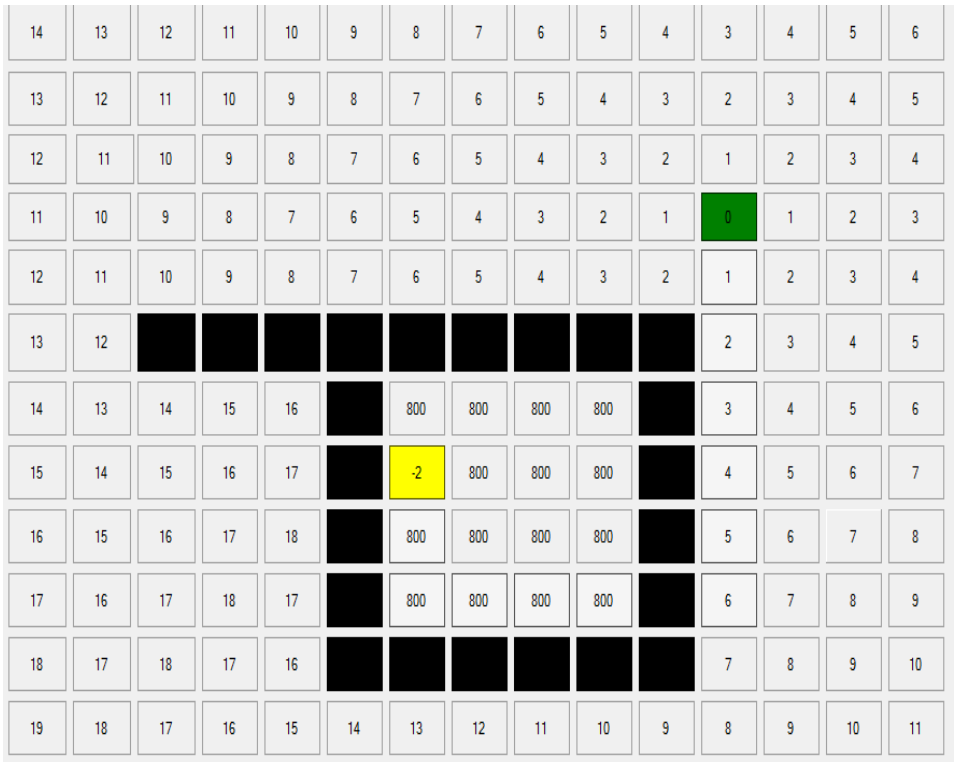

Figure 6. Example of Blocked path

It should be noted that all the numbers used in this paper like 1000 and 800 and all other numbers (in the array per example) will have to be augmented dynamically or altered depending the size of environment.
A windows form comprising a Map of clickable 15 x 15 buttons is used to simulate the Map Mode and the Manual Mode. Each button on the map represents a location from [0, 0] till [14, 14] ([Row, Column]).

The user first assigns a starting location, which will color the pressed button with yellow. Then, he assigns a goal location which will color the next pressed button with green. After that, the control of the agent movement, moving it up, down, left, and right is done through the W, S, A, D keys respectively. Obstacles can be added easily on the map by clicking on any desired position which will be then colored with black. Whenever the user wants, he can press the generate map and the find path buttons. The first button "Generate" (see figure 7) will put a number on each button to represents the current cost of position represented by this button. The "find path" button will show 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 continuously recalculates the shortest path and selects effectively the new path.

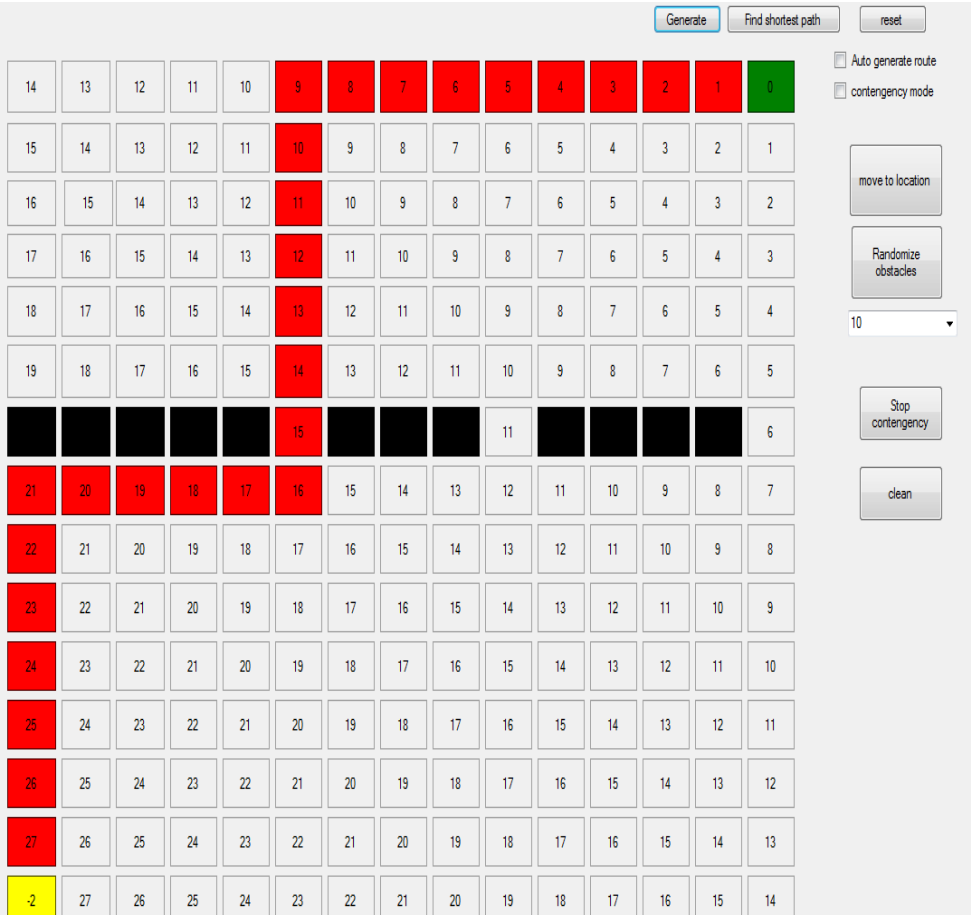

Figure 7. Path Finding

The 2D simulator contains different buttons, such as:

- "Auto Generate route", which is used in direct driving mode so that whenever the agent is moved manually to a new location the shortest path is generated automatically.
- "Find Shortest path", which generates the path.
- "Move to location", which starts moving on the marked path.
- "Randomize Obstacles", which creates a number of random obstacles on the map.

At each position the agent recalculates the shortest path in case a new obstacle(s) appear on the map. So if the user places an obstacle on the path that the agent is moving on, the agent will directly generate a new shortest route avoiding all the obstacles.
The auto-moving agent is shown in Figure 8.

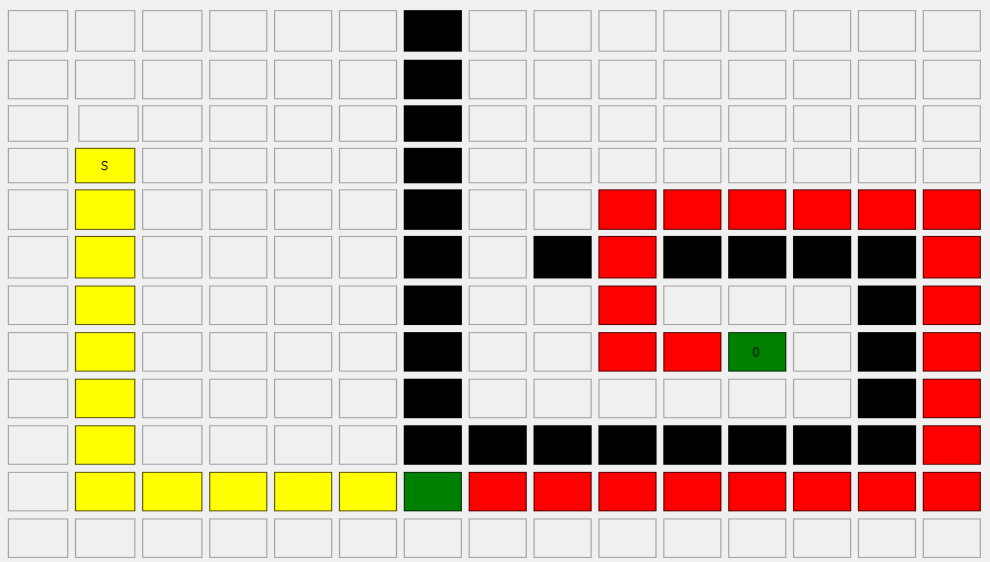

Figure 8. Auto Moving Agent

The shortest path is generated respecting only the allowed movements (UP, DOWN, LEFT and RIGHT) other movements can be added (by some alterations in the algorithm) for allowing moving diagonally or in any other direction.

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.

A flying agent in a city containing buildings of different sizes and shapes needs to find the shortest path to a target location. It is common sense that if the agent changes its height it will reach the goal location faster than moving on any 2D routes (roads) in a certain city. But since our environment is partially observable and dynamic, consequently, altering the height is not always an optimal solution (suppose the obstacle is a high mountain or a high building). The algorithm first of all finds a shortest path on the same height like in the 2D mode. When target is not found (no route returned because the agent or the target are surrounded with obstacles) on such height the program will not return that no path exists. It will return instead 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 9).

Set search node to starting node.
L1:
Test all neighbor node If (not the start node And inside environment boundary){
If neighbor node is target then return target is found.
Else
If there is blocked path then
remove all the obstacles on the map
Invoke Generate Map and find shortest path
Else
Set a search node to a sub node that contains the lowest cost.}
Mark the visited search node.
Repeat L1

Figure 9. Algorithm Development

The main 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 situation where the obstacle is a building, when the agent senses that an obstacle exists, it will raise its height till the agent is higher than the maximum height of the building.

What are the modifications required to be done in the three main methods (Generate, find and move) of the map mode?

- No change in the generate map method.
- Minor changes need to be done in Finding the shortest path method when no path exists on the same height, since this method detects whether a path is found or not.
- Moving the agent on marked path should be done as in 2D by applying the find shortest path algorithm. When no path exists on the same height due to obstacles, the agent while moving on the shortest path, evade them by flying over them.

4.4.1 THE 3D SIMULATOR

The simulator must have the following characteristics:

- The simulation should be done in two environments: 2D and 3D.
- The agent position and information should be observable by the user
- Obstacles should be shown dynamically on the application's main interface.
- The 3D simulator should be able to display the agent's position and movement according to the algorithm. It also must provide a clear 3D representation for the obstacles and the agent.

The 3D simulator provides two modes: Map Mode and Manual Mode:

- **In the manual mode:**
    - The virtual plane in the 3D simulator should have a smooth navigation (aerodynamically) where the application should have all the capabilities of direct

or manual flying mode (fly forward, to the left, to the right, upward and downward. Landing feature is optional.

- The application should supply the user with the ability to switch from manual mode to other modes and vice versa.

- The agent must avoid collision with obstacles at a certain distance and move away to a certain safe position.

- **In the map mode:**

  - The map must be clear and represent the environment correctly. It should also represent the correct coordinates of obstacles and agent's position.

  - The user must be able to assign on the map, initial starting position, target position and a number of obstacles on the map.

  - The agent must move automatically to target position avoiding all obstacles on the map and according to the previously mentioned shortest path algorithm without any lag or poor performance.

  - The application must return no route in 2D simulator if no route exist, while in 3D the application warn the user that no route exist on this height and asks the user if he wants to change the plane's height.

  - The user can add sudden obstacles during the automatic movement of the agent in both environments or random obstacles can appear.

In order to test the algorithm, a virtual but faithful representation of environment, obstacles and the plane is needed. Since the algorithm was implemented using C#, we had to use a 3D engine which is based on this language. XNA Game Studio 4.0 in Microsoft Visual Studio 2010 fits the criteria well. A 3D virtual City was built and virtual airplane drone can now navigate in it (see Figure 10).

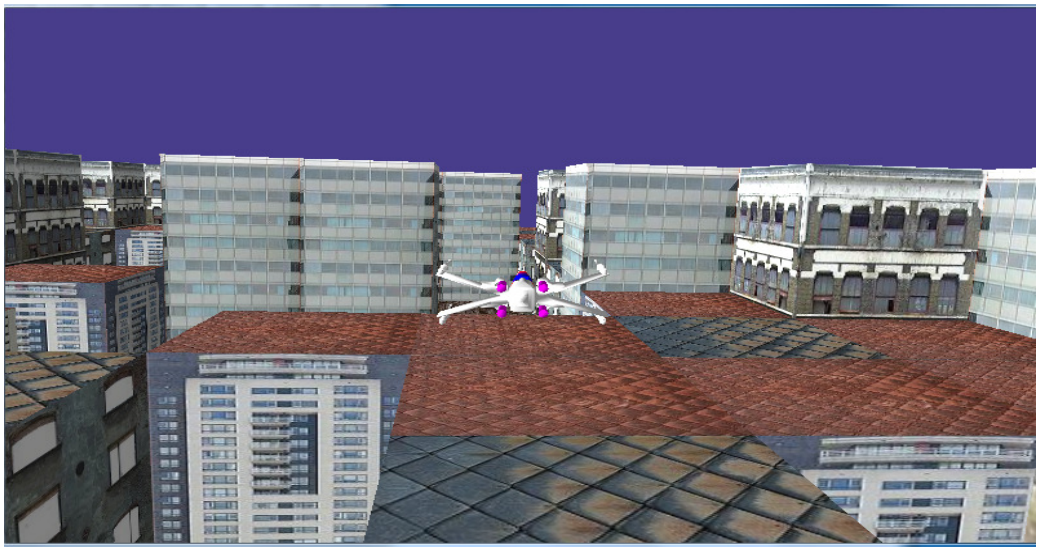

Figure 10. 3D screenshot

Figure 10 shows a complex environment where buildings (obstacles) of different sizes can be created.

A windows form, which contains a map that work on the same concept as the 2D simulator (buttons, colors and labels) was created in order to apply the map mode algorithm, to add obstacles and to retrieve the agent's location. The map form, loads first and through it, the user can start the 3D simulator. The user can assign an initial position and a goal position and add obstacles at will on the map. The obstacles will appear 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 that counts the steps taken by the agent since the start of the simulator. The random map button adds random buildings in random positions. Map mode is started whenever the user presses its button, before that, the user should assign an initial and a target position. Consequently the flying agent start moving according to the red route viewed on the map showing step by step the current position 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). Stop Watch returns the running time in milliseconds of any procedure [25, 26, 27].

Based the results gathered using this analysis method we made the following enhancements:

- Resetting the queue after every move.
- Stop searching when the starting point was found instead of discovering every point 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. Each thread must be tested to assure that no thread interfere with the other, and that threads are ended when their job finishes or when the program closes.

Additional tests were done on the manual control of the agent, turning and moving were tested in order to represent the most precise and realistic navigation that looks like that of a real plane drone.

5. REAL WORLD SCENARIO

This project's purpose is to be implemented on a real flying agent in a real environment. Therefore, the agent needs to be equipped with powerful and accurate sensors for a successful implementation of the algorithms presented in this paper. Accurate sensors that detect the distance from obstacles or even detect the sizes of obstacles can be helpful for effective

implementation. Other hardware tools are also needed for an agent in flight, such as speedometers or accelerometers, gyroscope and any useful hardware.

Instantaneously speed of the agent must be measured. Configurations of the environment with all sudden obstacles must be continuously generated. Exact position of obstacles is critical for any success navigation in a 3D environment.

6. CONCLUSION

The paper presented a description as much as possible of 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.

The problem of path planning in intelligent autonomous vehicles is still an active research topic studied by the research community for many decades. Path planning must be an integral part in the deliberation mechanisms of any intelligent autonomous vehicle [22].

In manual mode, the aim was to simulate automatic obstacle avoidance during the movement of the agent.

We used Microsoft XNA and Visual Studio, to simulate a 3D environment that resembles a real city which can be effortlessly manipulated. We chose a virtual airplane to represent the agent in this 3D environment. The movement of the plane was smooth and mimics every possible direction that a real plane can perform. The plane was able to sense all obstacles from a certain predefined distance and avoid them by moving to another safe location.

The main emphasis of the project is the map mode; the agent is required to move on a predefined map from one position to another evading all types of obstacles on the shortest path inside the boundary of the environment. The solution comprised three main algorithms: map generation, finding the shortest path and agent's navigation algorithm. First, Map Generation changes the partially observable environment to a fully observable one at certain time 'T' (since the environment is still dynamic). This is done by assigning costs to every position on the map. The implementation of this algorithm was founded on the iterative implementation of breadth first search with some alterations. Second, finding the shortest path was implemented by a local greedy 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 subsequent ideas are some of future work suggestions that need to done on the project:

- Additional optimization can be accomplished 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. Furthermore, enhancing the sensing capabilities of obstacles is crucial. Through realistic and exact

sensor's data in a real world environment, the agent can construct a knowledge base storing the properties of the obstacles (sizes, shapes, motions). This gives the agent the means to act accordingly.

- We still have to say that it is important for us to try in the future the Map Mode algorithm on a real plane. The main objective of any upcoming project will be the building of a small plane with the capability to sense obstacles through different sensors like sonars per example.

- It would be interesting to see the controlling of the plane by a smartphone or a computer via wireless radio connection or via the internet. The plane should be able to control itself autonomously in case it runs into obstacles, and should able to move on the map spontaneously (according to the algorithm presented in this paper). Other information might be indispensible to be taken into attention like the position, altitude, distance and signal strength of the plane and many others important flight parameters.

- Additional and extensive experimental and mathematical analysis on the algorithms discussed in this paper will be expanded in subsequent work.

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] J. Baillieul (1985), "Kinematic programming alternatives for redundant manipulators", in Proc. IEEE International Conference on Robotics and Automation, pp722-728.

[16] A. A. Maciejewski and C. A. Klein (1985), "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments", International Journal of Robotic Research, 4, pp109-117.

[17] Zexiang Li and J.F. Canny (Oct 31, 1992) "Nonholonomic Motion Planning", The Springer International Series in Engineering and Computer Science.

[18] Steven M. LaValle (Oct. 1998) "Rapidly-Exploring Random Trees: A New Tool for Path Planning", Technical Report No. 98-11, Dept. of Computer Science, Iowa State University.
[19] Steven M. LaValle and James J. Kuffner, Jr. (2001) "Randomized Kinodynamic Planning", The International Journal of Robotics Research, pp378-400.
[20] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock (2000) "Randomized Kinodynamic Motion Planning with Moving Obstacles", Fourth International Workshop on Algorithmic Foundations of Robotics.
[21] 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).
[22] David Sislak, Premysl Volf & Michal Pechoucek (2009) "Flight Trajectory Path Planning", Proceedings of ICAPS 2009 Scheduling and Planning Applications woRKshop (SPARK), pp76-83.
[23] 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.
[24] 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.
[25] Sedgewick, Robert (2002) "Algorithms in Java", Parts 1-4. Vol. 1. Addison-Wesley Professional.
[26] Levitin, Anany (2008) "Introduction To Design And Analysis Of Algorithms", 2/E. Pearson Education India.
[27] Goodrich, Michael T., & Roberto (2008) "Tamassia. Data structures and algorithms in Java", John Wiley & Sons.
[28] http://www.microsoft.com/en-us/download/details.aspx?id=23714
[29] http://www.riemers.net visited at 1/5/2013
[30] http://www.techopedia.com/definition/27489 /activity-diagram visited at 15/5/2013
[31] 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.