Military Technical College
Kobry El-kobbah,
Cairo, Egypt

5th International Conference
on Electrical Engineering
ICEENG 2006

Binary Space Partitioning algorithm for the navigator agent in the Virtual Environment

Khaled Elmenshawy, Ismail Abd-Elghafar , Ali Ali Fahmy

ABSTRACT

Virtual environment can be brought to life by adding walking characters (navigator agent) to enhance the navigation process. The navigator agent helps the visitor to find specific locations in the virtual environment. It is specialized in determining appropriate free and safe paths through the virtual environment, and can provide guidance to the user that tries to follow such paths. Path finding depends on a number of waypoints to be set out in the virtual environment. The navigator agent receives an initiation message from the headquarter agent depends on the reaction of the watching agent and sends a request to the navigation module to plan the shortest and safe path. This paper discusses the development of a real-time navigation algorithm for the navigator agent as a guide tour through the virtual environment. The algorithm is handled by two channels. The first is the 3D Graph, which contains all nodes and path structure information. The second is the Motion Planning channel, which calculates the fastest route through the path structure from the current position to a destination position. This algorithm is called Binary Space Partitioning Algorithm (BSP). BSP divides the 2D map of the Virtual Environment into two configurations or more depending on the start and destination locations, after that it generates two trees, one from the initial side (T_{start}) and the second from the destination side (T_{goal}). The navigator agent uses join algorithm to connect these trees together. A path is found when the two trees can be connected. Finally, the navigator agent provides two choices for the user, the first one it draws a generated path on the scene, then the user follow this path or, the second choice is the navigator agent calls the animation algorithm to move on the path as a tour guide for the user. The time has been taken to explore the virtual environment by using the Binary Space Partitioning algorithm is decreased approximately by the half compared by the traditional motion-planning algorithms.

Key words

Virtual Environment, navigator agent, watching agent, headquarter agent.

1. Introduction

The proposed model of the embodied agents in the synthetic Virtual Environment consists of five agents. The first agent is *a watching-agent*. This agent works as interface agent. It observes the interaction between the user and the VE from the sideline learns from it and interacts with the controller agents through *headquarter-agent* so we called it a Watching-agent. The second agent is a headquarter-agent, it works as a supervisor agent, and it is a mediator agent between the higher level agent (*watching-agent)* and the lower level of the working agents. It

stores the dynamically information about the *VE*. It sends this information to the working agents group. The third agent is the *position-tracker agent*. It is used to determine the position of the navigator agents in the synthetic VE. It stores the status of each break-point, edge and path in the *VE*. The fourth agent is the *object-identifier-agent*. It is used to keep track of the changes of the dynamic objects in the *VE*. Finally, the fifth agent is the navigator-agent. It is the moving agent that helps the visitors of the *VE* to find their interested way. In this paper, new real-time navigation algorithm is developed for the navigator agent. The navigator agent can Sense, interact the objects that make up the spatial layout of the world and Communicate with any other agents situated in the VE.

Intelligent navigator agents are added to the Virtual Environment Browser to enhance the navigation process. Autonomy is necessary to enable the agents to explore the *VE* in the absence of continuous instructions by the user. The agent is able to engage in complex communication with other agents, including people, in order to obtain information or enlist their help in accomplishing its goals [5].

2. Related Works

The motion planning problem has been extensively studied in the past two decades. A good survey of motion planning algorithms is in [6]. In the recent years a new path-planning scheme called random sampling scheme for path planning was proposed. A special version of planner with this random sampling scheme is called the probabilistic roadmap method [7]. In this method, a significant amount of time is spent in preprocessing the connectivity information of the configuration space such that it can answer path-planning queries afterward in a short amount of time. The *sampling* approaches introduced in [7] consist in first sampling the configuration space with collision-free configurations and trying to link them by collision-free paths computed with the steering method. This so-called *learning* phase builds a graph *(roadma*p) whose connected components tend to capture the connectivity of the topological space. A given problem is solved in a so-called *query* phase: both starting and goal configurations are added as new waypoints of the roadmaps; then the existence of a collision-free local path between and the existing roadmap waypoints is checked. Finally the search is per-formed by a graph search algorithm. Any existing path in the graph corresponds to an admissible motion. This type of planner is good for applications where static environments can be assumed and several planning queries are needed. The Probabilistic Road Map method (PRM) is suited for very complicated environments. Unfortunately, the roadmap produced by this method can be rather wild, leading to ugly paths that require a lot of time-consuming smoothing to be useful for gaming applications. PRM algorithm is evaluated by using a dynamic roadmaps for on-line motion planning in changing environments. When changes are detected in the workspace, the validity state of affected edges and waypoints of a pre-computed roadmap are updated accordingly. a randomized path planner is used as a part of simulated environment for providing high level software control of humanoid robots.[8] Another approach is applied by using a hybrid Artificial Potential field to resolve the two component problems of getting agent to its goal and avoiding any obstacles in the process.[11]Un-fortunately, most randomized or optimization driven path planning algorithms can be expensive in particular environments, and may even fail to reach the goal state. The problem of Motion Planning for various types of moving agents was widely investigated by many researchers despite the success of the Probabilistic Road Map framework, work has mostly concentrated on static environments. [1], [2], [9]

3. Behaviors of the Navigator agent

The navigator agent receives an initiation massage from the headquarter agent and sends a request to navigation module as shown in fig.1. The navigation module is responsible for planning the shortest and safe path to the navigator agent from the start to the goal position according to user's choice. After the navigation module completes its plan, it sends a safe path to the navigator agent to follow it. The navigator agent model composed of capabilities, plans, databases, and events. Agents can address other agents and post events to them thus modeling inter-agent communication.

Events	Events are those things that an agent responds to. They arise internally to an agent as reasoning progresses, as a result in a change in the agent's beliefs, or on receipt of a communication from another agent.
Plan	A plan is a specification of a sequence of actions to undertake in response to an event.
Capability	Capabilities are sets of plans, events, and databases that are functionality grouped to provide a specific capability to an agent.
Database	The Database is the implementation of the beliefs of the agent.

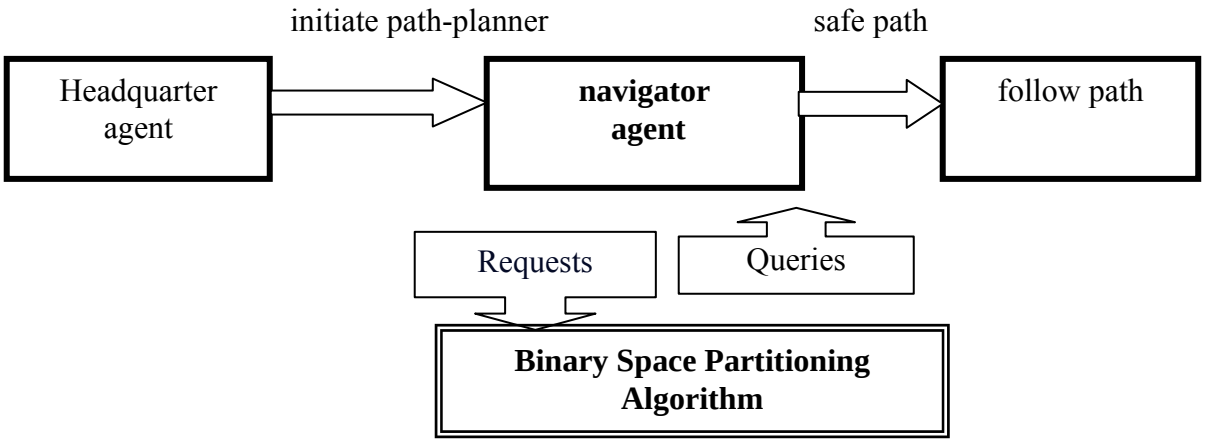


Fig.1. the navigator agent calls the navigation module to activate path-planner

The navigator-agent has its own capabilities to perform its tasks. It executes different plans according to different situations as shown in fig.2.

```
#handles event Headquarter_Request
#handles event Modification_2Dworld
#handles event surface_contacts
#uses plan RestrictedArea
#uses plan Free_Space
#uses plan MoveToNextBreak-point
#uses plan Building_Navigation_Map
#uses plan Tracing_Agents
```

#uses plan Surface_following
#private data Board Messages_Controller()

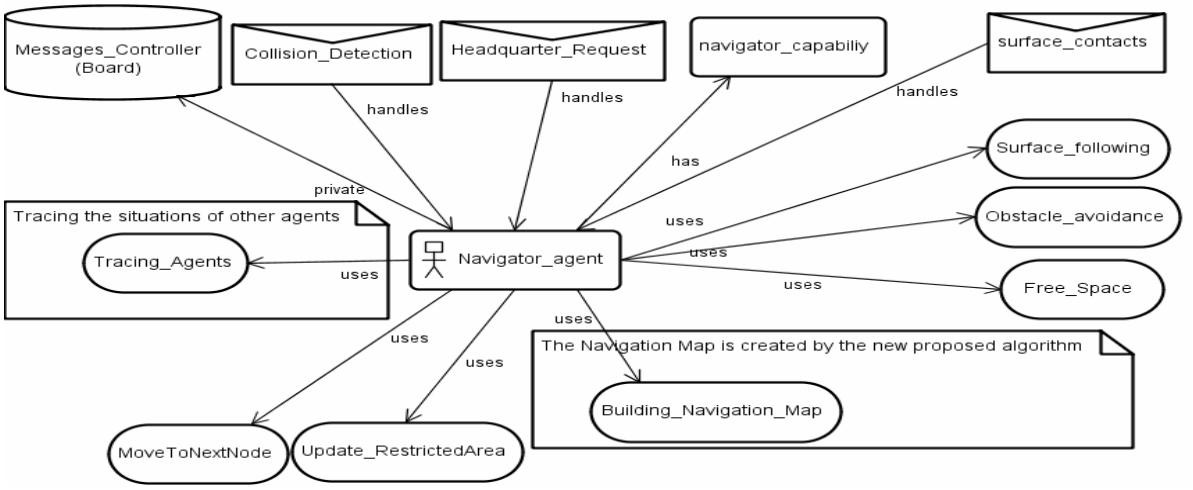


Fig.2. The Architecture of the Navigator Agent's role

Navigation module plays an important role in formulating proper paths to reach a destination point. Often the agent must deviate from its intended course to negotiate obstacles that are in its path. After the maneuver has been made, the agent may find that it has strayed in a direction farther away from it goal. In this case the agent must take corrective actions to move towards its goal. A fitness function may be used to determine how well the agent is meeting its goal of navigating to an assigned waypoint. The Navigator-Agent communicates with the headquarter-agent by using KQML languages. Table (1) represents different messages transferred between the navigator agent and the headquarter-agent, which define the agent's actions in the ***VE***:

Table (1) messages transfer between the headquarter-agent and the navigator

Message	Action
RegisterAgent Message	Registers a new navigator agent in the ***VE.***
RegisterTimeEventMessage	Registers any Time which can be used to start/Stop agent-specific animations.
MovePlanMessage	Moves the agent between different waypoints and edges after using MoveToNextBreak-point plan.
SetRotationMessage	Changes the agent's orientation in the ***VE*** as defined in the message parameters.
StartAnimation Message	Starts an agent specific animation.
ObstacleDetection	Starts an agent specific obstacle response plan.

WaitAnimation Message	Stops an agent specific animation for certain period of time.
GoalRequired Message	Agent accomplishes its tasks.
LocalMapBuilderMessage	Agent rebuild its local navigation map according to the changes occurred in the ***VE***.
LocalMapRequestMessage	To be sent when the agent builds its local navigation map
PerceptionRequestMessage	Notifies the headquarter-agent to send the locations of other navigator agents or the moving obstacles when it is equipped a break-point or the edge.

Surface Following plan

The ability to follow the surface during the navigation process is a requirement for all Virtual Environment Browsers as it is needed to accomplish a basic 'Walk' type of navigation. If an agent is to simulate this type of navigation where the surface is a variable one, `some kind of a Constrained Surface Movement will be required.[11] Once a path has been found between the desired start and destination configurations, the path is rendered for the user and also provides an automatic navigation function that propels the avatar along that path.

Collision Detection Plan

Collision Detection is another plan that is required in agent navigation. In [30] the authors describe a collision detection algorithm, which has been built into Virtual Reality Modeling Language Browsers. (VRML)

Object/Object collision detection plan

```
For each object/object pair
        Draw the smallest non oriented box to surround each object in the pair
        For each member of the pair
            If any vertex of this object's box
                    Then collision is true
            Endif
        Next member
    Next Pair
```

Agent/Obstacle collision detection plan

```
For each agent in the VE
        Draw the smallest non oriented box around the agent
        For objects within range of the agent
        Draw the smallest non oriented box around the object
        If any vertex of this object's box is within the bounds of the agent's
        box or If any vertex of the agent's box is within the bounds of this
        object's box Then collision is true
```

Endif
Next object
Next agent

3.1 The navigation path planner

Any path-planning algorithm must be based on a representation of the free space of the environment's model. In order to navigate through the *VE*, the navigator agent must use the navigation map that is created after excluded the static obstacle regions. The navigation map would be changed if any changes occur in the VE. The navigation map is normally represented as a graph in which the waypoints correspond to placements of the agent and the edges represent collision-free paths between these placements and can be written as G = (*V*,*E*), fig.3. illustrates an example of the 2D-Environment consists of 2500 obstacles.

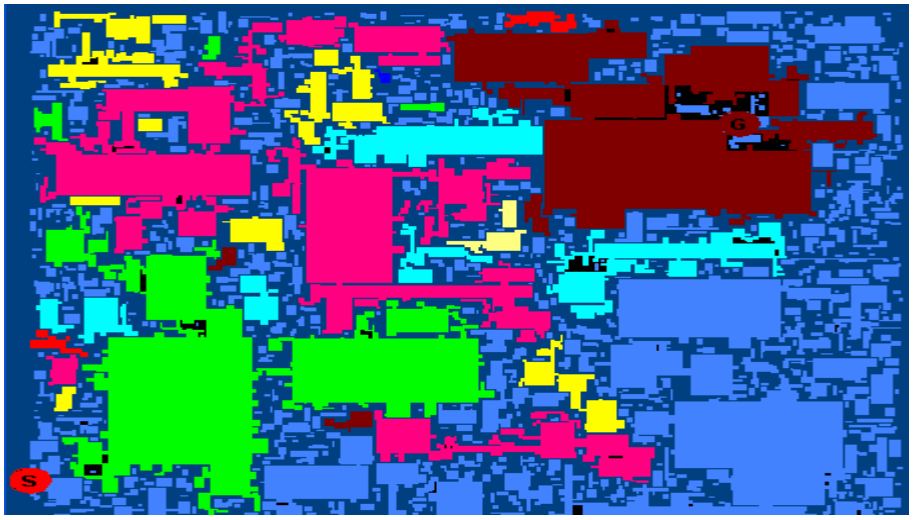


Fig.3 an example of 2D-Environment consists of 2500 obstacles

4. Building the navigational map

The following steps are used to generate the navigational map from the 2D environment as shown in fig.4.

- The 2D map of the VE is divided into two configurations or more depending on the Start and destination locations by using horizontal or vertical straight lines. The steps are described in fig.4.
- Applying the Bi-Directional Single Query Probabilistic Navigation map. It uses the two input query configurations to explore little space (bi-directional) it explores the navigator agent's free space by building a navigation map made of two trees rooted at the start and Goal query configurations. We grow two trees from Start and destination, respectively. We choose the waypoints that are most likely to see a large portion of the free space.
- Applying Join algorithm to connect the two navigation maps together. A path is found when the two maps can be connected.
- Applying smoothing algorithm to improve a generated path.
- Follow the generated path.

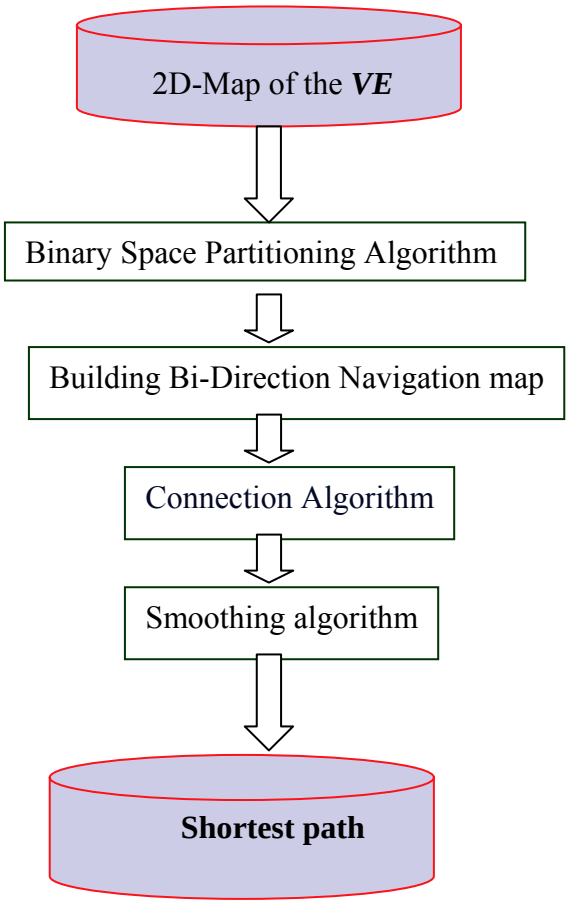
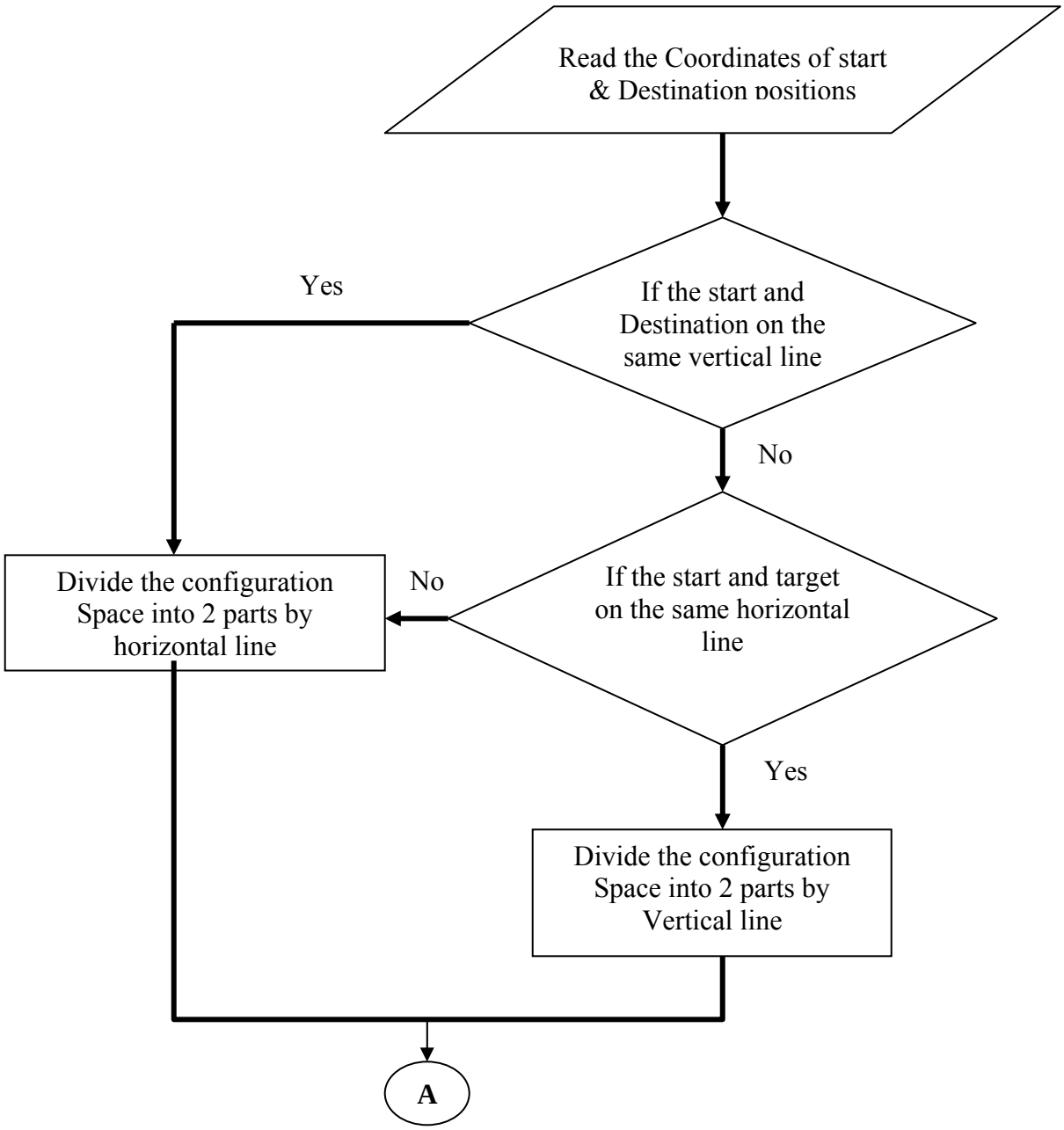


Fig.4. steps for developing the Navigation path

The navigation map is used to allow the agent to navigate through any large environment. The environment is specified in terms of geometric primitives and their connectivity. The agent is allowed to navigate in the environment in two modes:

- ***The Global mode***: In this case, the user determines the *start* and *goal* locations in the environment. The algorithm automatically computes a safe collision free path that satisfies the motion constraints.
- ***The Local mode***: The user is allowed to explore the environment in a driving mode, where the agent responds to transnational and rotational inputs from the user. The algorithm automatically performs collision checks with the environment and constrains the agent to terrain surfaces in the environment.


Read the Coordinates of start & Destination positions
If the start and Destination on the same vertical line
Yes
No
If the start and target on the same horizontal line
No
Divide the configuration Space into 2 parts by horizontal line
Yes
Divide the configuration Space into 2 parts by Vertical line
A

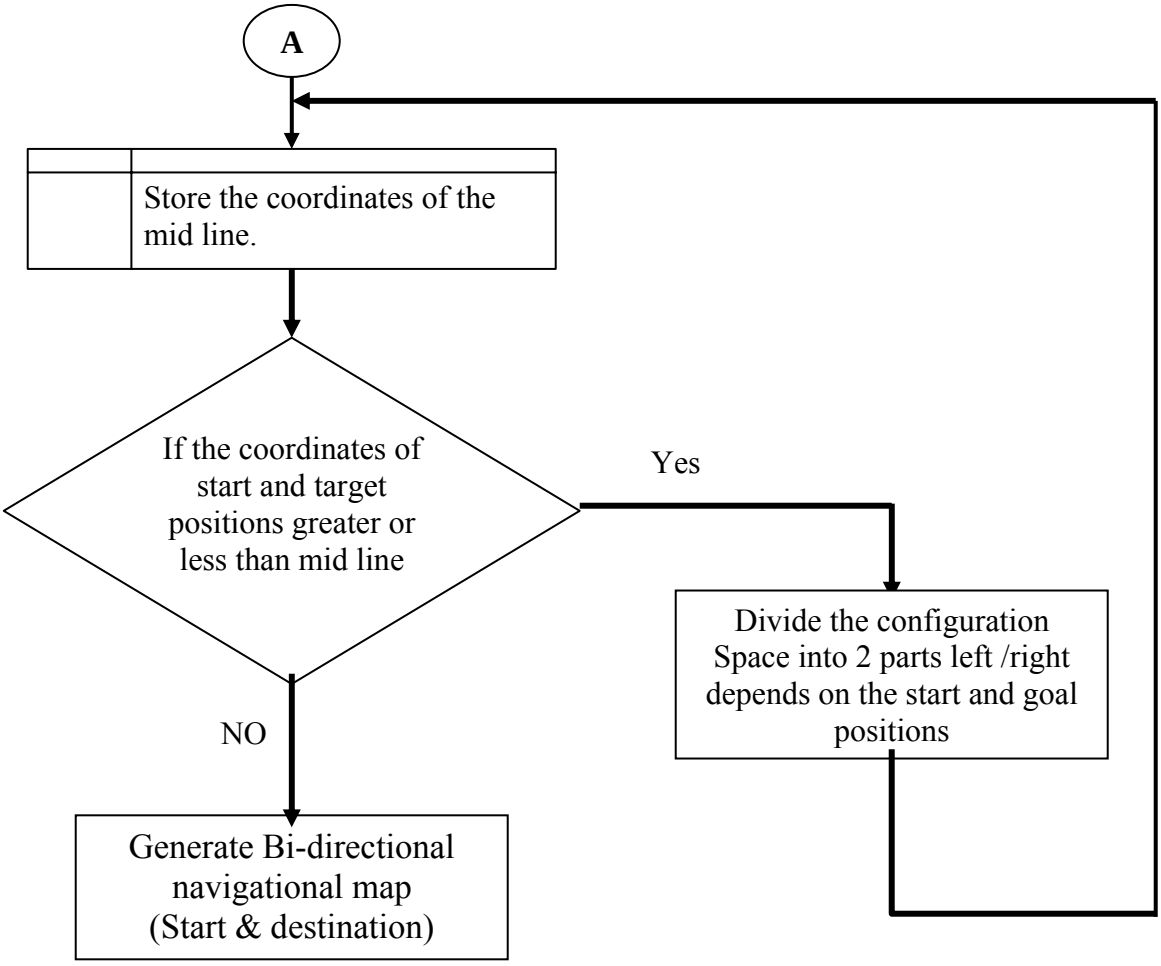


Fig.5. Binary Space Partitioning algorithm

The next section describes the preparation phase of creating the navigation map of possible motions of the agent(s) from the 2D-map of the ***VE***.

4.1 Preparation phase

During the preparation phase, a global navigation-map of the synthetic ***VE*** is created. This is randomly done by generating collision-free configurations of the agent in the scene depending on the knowledge of the ***Position-Tracker-agent***. The interested points (break-points) are connected to each other by using a local planner that enforces our constraints including collision detection using a bounding box hierarchy. We also prune the graph to reduce the redundancy of coverage or reach-ability over the entire walk-able space. The obtainable navigation map is a global data structure that is used at runtime to navigate the user through the environment.

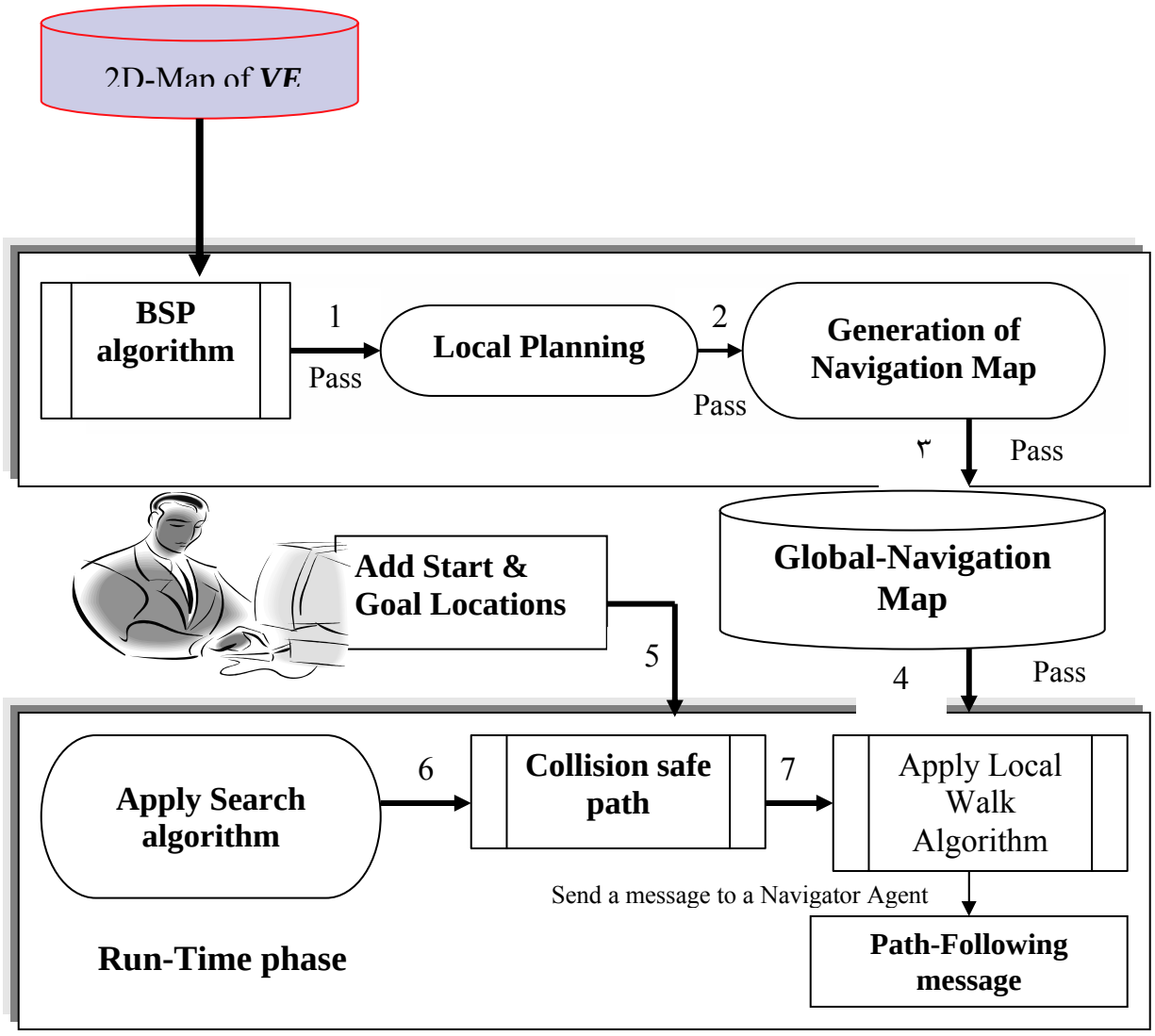


Fig.6. Phases of Generation a navigation map by using PRM

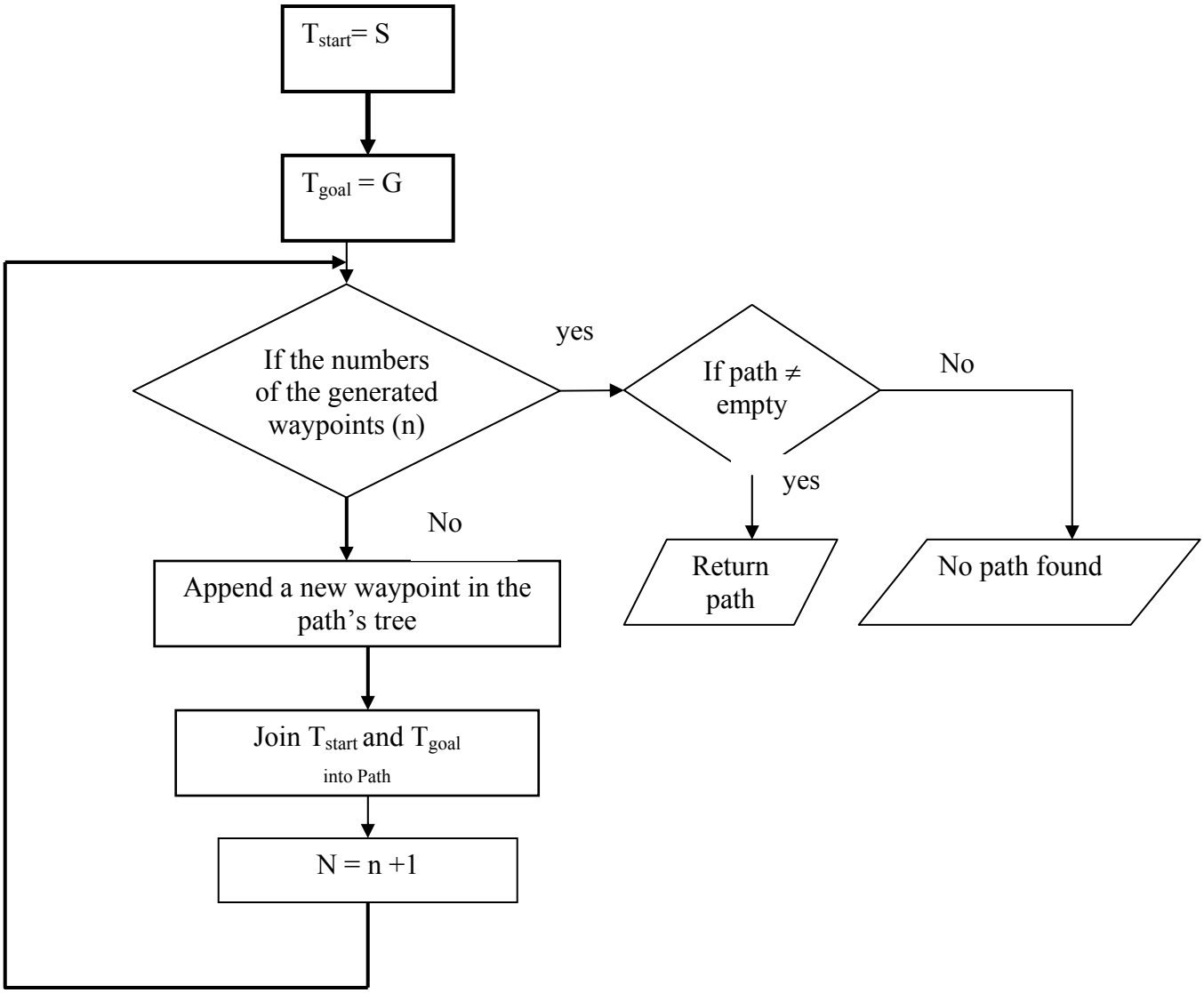
4.2 Execution phase

At execution phase, the user specifies the start and goal position. The algorithm searches the navigation map graph using any search algorithm such as Dijkstra's algorithm and performs connected component analysis to improve its performance. The user navigates along the computed path. Given the collision-free and constrained path, a few viewing options including path alignment are available to the user. When a break-point or edge has a reference counter greater than zero, it is considered not safe and thus not allowed to be traversed during the graph search procedure . The combination of these techniques enables the user to navigate automatically through the complex environment. The algorithm also allows the user to navigate any part of the environment in a driving mode. The user can also stop at any point of interest and

control his movement to inspect a particular location within the environment. Our algorithm provides the user with both local control and global navigation in a large environment.

5. Building Bi-Direction Navigation map

BSP generates two trees, T_{start} and T_{goal}, respectively rooted at the start (***S)*** and the goal location (***G***.) while the waypoints set is not empty, the Append and the join procedures are executed: APPEND which adds a break-point to one of the two trees, while JOIN tries to connect the two trees. ***BSP*** returns *failure* if it has not found a solution path. In this `case, either no solution path exists between ***S*** and ***G***, or the planner failed to find one. BSP has two parameters
n: the maximum number of waypoints that is allowed to generate,
d is a distance threshold. Two configurations are considered close to one another if their distance is less than ***d***.



The generated path consists of straight line segments; following such a path will have discontinuities at the connector break-point that cause sudden directional changes to an agent that follows the path. Next section describes the steps have been taken to resolve this problem

5.1 Discontinuity removal

In order to solve this problem we will replace parts of the straight line edges by circular arcs. The degree of a connector break-point is defined as the number of edges that is connected to this break-point. If a break-point has degree 1, it is an endpoint of a path segment, and no circular arc needs to be added. If a break-point has degree 2, the addition of the circular arc is straightforward. We find the centers of the two edges, and use these to create a circle arc that touches both edges A and B. Let m be the midpoint of A and let n be the midpoint of B. We replace part of the path between m and n by a circle arc that lies in the plane spanned by the two edges. This arc will have its center on the bisecting line of A and B, will touch A and B and have either *m* or *n* on its boundary, depending on which one lies closer to the break-point *v*. As the arc touches the two edges it will remove the first-order discontinuity at break-point v. Since the arc will only remove at most half of the edges A and B, we can repeat this for each break-point on the path. The obtainable path will be C^1 continuous fig.8-a. If the degree of a vertex *v* is higher than 2, we find the centers of all incoming edges fig.8-b.

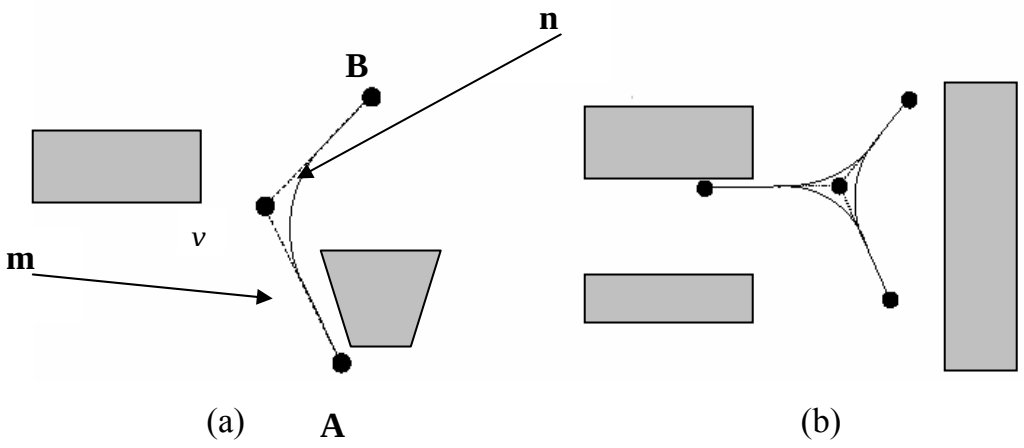


Fig.8. Circular arc between two and three edges

6. Experimental Results

Binary Space Partitioning algorithm is developed by using Java and VRML97 [4]. It is tested for different synthetic Virtual environment with different obstacles numbers and shapes. The first experiment compares the proposed BSP algorithm with three other planners: Potential field [12], visibility graph [2], and exact cell decomposition [12]. The results are shown in Table (2)

Table (2) Comparison between three different algorithms with the BSP algorithm

No of Obstacles	Roadmap (Visibility graph)	Potential Field	exact cell decomposition	Proposed Algorithm BSP
1	23.2	25.6	24.5	16.53
5	23.2	28.3	25.6	16.53
10	25.9	33.5	28	17.64
15	28.12	38	32	18.456
20	28.12	40	33.5	18.456
25	31.41	41.6	36.2	20.18
30	34.84	44.8	39.4	21.48
100	79.16	88.8	83.12	37.38

Figure (9) shows the performance of the BSP Algorithm comparatively to different planners. The fastest algorithm is *BSP*, it is approximately worth the half time of the visibility graph since the idea of the BSP algorithm is based on the same idea of the visibility graph but it uses two tree of building the navigation map in addition to set of algorithms to enhance the path quality. These performance measures were taken on a 500MHz Intel Pentium III, 256M RAM, under Win XP.

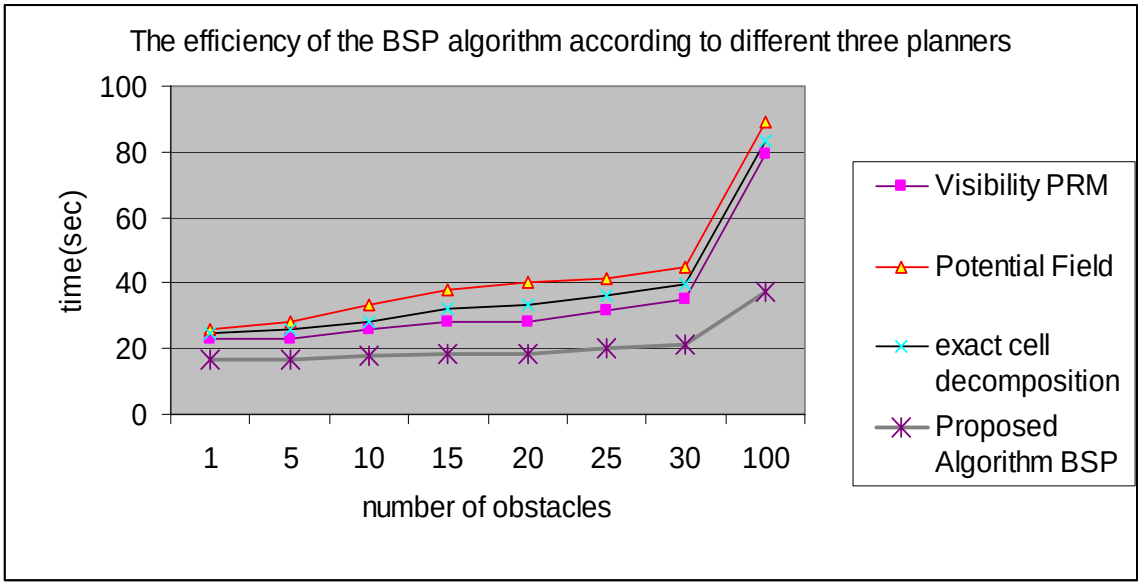


Figure (9) the performance of the binary space partitioning algorithm

7. Conclusion and future work

In this paper a new motion planning algorithm (**BSP**) is used to build the navigation map to assist the visitors of the VE. Motion Planning has been studied for several decades, and many motion planning algorithms have been presented. Therefore, research in motion planning remains one of the important fields of study in the task of building robot systems. Computation of a collision-free

path for the navigator agent in a dynamic environment is a fundamental subject in many fields. Choose the right navigation architecture:

1. Improve quality of the behaviors
2. Increase performance
3. Make it easier for Agents to integrate in the navigation system

the navigator agent observes the environment, maintain an internal representation of the world, make decision and perform tasks. the navigator agent provides two choices for the user, the first one it draws a generated path on the scene, then the user follow this path or, the second choice is the navigator agent calls the animation algorithm to move on the path as a tour guide for the user. The time has been taken to explore the VE by using the Binary Space Partitioning algorithm is decreased approximately by the half compared by the traditional motion-planning approaches. The benefit of usage the agents is to prevent the collision between the user, obstacles and other agents. For the future work we plan to study the performance of the proposed algorithm for multi-user virtual environment cluttered with moving obstacles. Developing a new algorithm for the navigator agents work in a dynamic Virtual Environment taking into account the prediction of moving obstacles.

References

[1] Ashraf Elnagar, Leena Lulu., A global path planning Java-based system for autonomous mobile robots, Elsevier, science of computer programming, (2004).

[2] D. Nieuwenhuisen, A. Kamphuis, M. Mooijekind M.H. Overmars., Automatic Construction Of Roadmaps For Path Planning In Games, (2004).

[3] Igarashi, T., Kadobayashi, R., Mase, K., and Tanaka, H, " Path drawing for 3d walkthrough". In Proc. of UIST, 173–174, (1998).

[4] Chan Su Lee and Grigore C. Burdea Virtual Reality Technology Second Edition Laboratory Manual, The State University of New Jersey U.S.A, John Wiley &Sons, (2003).

[5] J.Muller., The Design of Intelligent Agents, A Layered Approach, volume 1177 of Lecture Notes in Artificial Intelligence, Springer-Verlag, (1996).

[6] J. Latombe, Robot Motion Planning, Kluwer Academic Publishers Boston, (1991).

[7] L. Kavraki, P.Svestka, J. Latombe, and M. Overmars., Probabilistic Roadmaps for Fast Path Planning in High-Dimensional Configuration Spaces, *IEEE Transaction on Robotics and Automatio*n, 12:566-580, (1996).

[8] Léonard Jaillet & Thierry Siméon., A PRM-based Motion Planner for Dynamically Changing Environments, (2004).

[9] Marcelo Kallmann, Maja Matarić: " Motion Planning Using Dynamic Roadmaps, in proceedings of the IEEE international Conference on Robotics and Automation (ICRA), 2004.

[10] Ruth Aylett and Marc Cavazza., Intelligent Virtual Environments - A State-of-the-art Report, University of Salford, CVE, Salford; University of Teesside, School of Computing and Mathematics, (2004).

[11] Julien Burlet, Olivier Aycard and Thierry Fraichard., Robust Motion Planning using Markov Decision Processes and Quadtree Decomposition, IEEE Int. Conf. on Robotics and Automation, New Orleans, LA (US), (2004).

[12] Steven M. LaValle., Planning Algorithms, Combinatorial Motion Planning, chapter 6, University of Illinois, (2004).

[13] Tim Batchelor Hnd., ANTS: Automatic Navigation of Terrain Systems, Alumnus of Bolton Institute, (2003).

[14] Zhukov, A Iones., Building the navigational maps for intelligent agents, Elsevier Science, computers& graphics 24,pp 79-89, (2000).