Path Modification for Motion Planning

Mario E. Consuegra, Phillip Coleman, Lydia Tapia, Nancy Amato
Parasol Lab
Department of Computer Science
Texas A&M University College Station, TX 77843-3112

Abstract

In this paper we explore the use of path modification methods to help solve motion planning problems. We define path modification as the process of modifying the order and position of the different configurations in a path with the purpose of improving it. These path modification methods are used as a complementary piece to aid the algorithms used to plan the motion of an object. For a certain motion planning problem an approximate path can be built by using different possible motion planning methods as Probabilistic Roadmap Methods (PRM) or User Aided Motion Planning Methods. But this approximate path returned by the motion planners might still have room for optimization. The path modification methods explored in this paper aspire to aid in the optimization of these paths. We define the different steps present in the process of path modification, and we investigate different implementations of these steps.

1 Introduction

Motion planning is the process of finding a path for an agent to move in space from one position or configuration to another. It has many applications in industry, robotics, medicine, and pharmacology[5].

Currently there are several promising motion planning methods being studied. These methods include Probabilistic Roadmap Methods[5], Obstacle-Based Probabilistic Roadmap Methods[2], and Haptic Aided Motion Planning Methods[6]. All of these methods work on finding a path from an initial given configuration of an agent to a given goal configuration in an environment where obstacles might be present.

Even though these motion planning methods have been used successfully, the inherent difficulty of the task of planning the motion of an object causes these planners to produce paths that still have room for optimization[4]. The path modifying methods can take these paths produced by the planners, and modify them in order to produce an improved version of it. The path modifying methods might receive paths that have configurations that are not valid or that collide with an obstacle, in such case an algorithm can be applied to the path to eliminate such configurations and create new ones that are valid and collision-free. The paths received from the planners might not have configurations in collision, but might follow an irregular trajectory that could be improved. In that case a path modification method can be applied to the path to improve it. We explore path modification methods as a useful tool that can assist the process of planning the motion of an object.

We developed a generic algorithm that can be extended by specific path modifying methods to improve paths. This generic algorithm provides a base for these specific path modifying methods by handling the generic parts of the process. We also developed a particular path modifying algorithm, the Random Walk Method. This method extends our generic algorithm and applies a particular path modifying criteria to the paths entered. The work on the Random Walk Method serves as an example of how we can integrate a path modifying method with our generic algorithm.

One particular approach followed for planning the motion of an object is to produce paths with the help of user input. In our team we have done work on this approach by doing research on producing paths with haptic hints[7]. For some of the experiments that we performed in this paper we have used a haptic interface to produce paths which we then modify with our path modifying algorithms.

With this project we want to provide a strong base for future research on path modification methods. The generic algorithm will provide base functionality for different specialized path modifying methods. In the future we will do research on other path modification methods besides the Random Walk Method, and our work on the generic algorithm will save us time and resources in the task as we integrate these methods with our generic algorithm. Path modifying algorithms look like a promising technique that could have a positive impact on the field of motion planning.

1.1 Related Work:

The following are two main researh areas on motion planning that have field for applications of our path modification methods

Probabilistic Roadmap Methods. The problem of moving an agent through multidimensional space (the movers problem) has been proved to be SPACE-Hard[4]. One approach to solve this problem is through the use of sampling-based planners[5]. One of the sampling-based planners that demonstrated great potential is the probabilistic roadmap planner (PRM)[5]. PRMs generate nodes randomly and uniformly in C-Space to create a roadmap. Each of these nodes corresponds to a particular free configuration of the moving agent. These nodes are connected to each other if a local planer can find a path between the different configurations. Then a query is submitted to the planner. A query is the submission of two configurations to the planner (a start configuration and a goal configuration) and then requesting it to find a path from the start to the goal configuration. The planner connects the start and goal configurations to the roadmap, and returns a path between the two. One variation of the PRMs are the Obstacle-Based Probabilistic Random Methods (OBPRMs)[2], which instead of generating the configurations uniformly through C-Space, generate a denser population of nodes in near constraints regions. This method works better when working in environments that have numerous obstacles and constraints.

Haptic Aided Motion Planning. The use of human input for motion planning has also shown good potential[7]. Work has been done with the use of haptic interfaces that allow humans to create a path in a virtual environment. These work takes advantage of the human visual capability[3]. In this approach, a human operates a robot to generate a series of configurations that represent a path through the environment. These configurations are then passed to the planner.

2 Methods:

2.1 Generic Path Modifying Algorithm

A generic path modifying algorithm was developed and implemented in this project. This generic algorithm had to provide enough re-usability so that it could be used for the application of different path modification methods. In the future, we will study multiple path modifying methods and we will use our generic algorithm to handle all the functionlity that is common between all these.

This generic algorithm takes care of implementing all the generic parts of the process of modifying a path, and leaves enough flexibility in several key sections of it. These key sections where generic functionality is allowed comes from the following:

What determines when a node in a path is in an invalid location and requires to be moved to a valid location? The fact is that is that this criteria might vary from user to user. So to allow for flexibility in this section, the generic algorithm provided for a generic implementation of this section.

In the process of modifying a path different parts of the path might need different modification strategies. See in Figure 1 how a path goes through different regions of the environment, and each of these regions of the environment has different characteristics. Taking this into account, the generic algorithm will divide the path into a series of segments. It will allow the user to define how to divide the segments and tell when a segment is invalid and when a segment is valid to then be able to apply different modifying methods to each segment.

After a path has been modified then it is necessary to evaluate it to determine if an acceptable path has been produced or if it is necessary to continue working on it. At this point the generic algorithm once again offers a generic implementation of the section so that the user can define how the resulting path is evaluated.

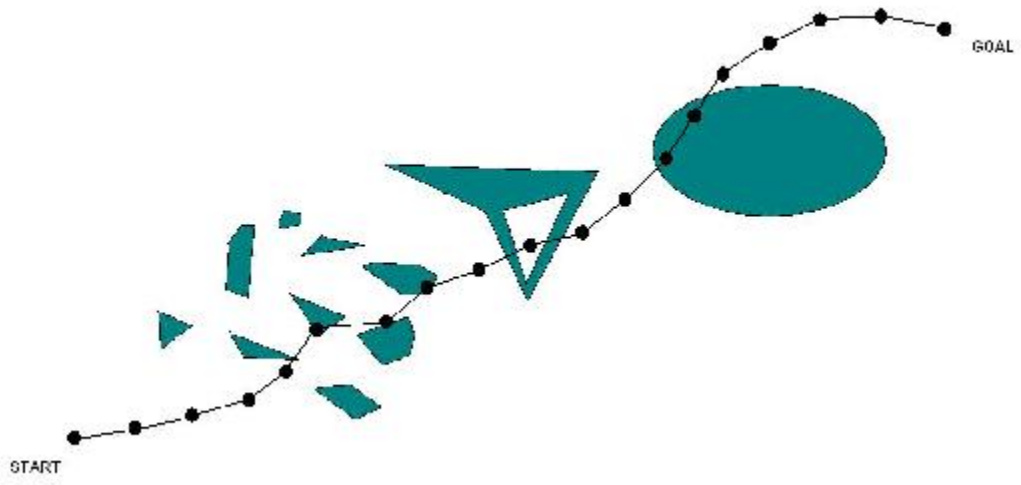

Figure 1: Example of a path that goes through different regions of an environment

The generic algorithm was divided in several sections. Each one of these sections is related to the main steps that we want to keep generic during the process of modifying a path.

The steps in which the algorithm was divided are:

Node Evaluation: In this step we iterate over all the nodes in the path and evaluate their validity. In our implementation we define the validity of the node in terms of its collision status and its connectivity.

Segmentation of Path: After each node is evaluated it is then pushed into a segment. A valid node will probably be pushed into a valid segment and an invalid node into an invalid segment. Some algorithms might push some valid nodes into invalid segments and vice-versa depending on the specific actions that are going to be applied to the segments.

Segment Evaluation: After the path has been partitioned into segments, we need to evaluate each of these segments and determine what type of modification we want to apply to it.

Segment Modification: We apply the modifications to each one of the segments and replace the old segments by the newly produced ones.

Path Evaluation: Then, we evaluate the path to determine our grade of success. At this stage we can decide if the new path is acceptable or if we need to apply a different path modifying method.

2.2 Generic Algorithm Description

Below we present a high level description of the generic algorithm. The Segment Construction section of this algorithm is presented in Algorithm 2.1

Algorithm 2.1 Segment Construction Section

```
1: For each node in path
2:         isValid = EvaluateNode( node )
3:         see if can store( node as isValid into currentInvalidSegment)
4:         see if can store( node as isValid into currentValidSegment)
5:         if(currentValidSegment isDoneGathering)
6:                 push segment into list of all segments and start a new validSegment
7:         if(currentInvalidSegment isDoneGathering)
8:                 push segment into list of all segments and start a new invalidSegment
```

After we divide the path into segments we proceed to modify them. The Segment Modification section is presented in Algorithm 2.2.

Algorithm 2.2 Segment Modification Section

```
1: For each segment in the listOfAllSegments
2:         Evaluate( segment )
3:         Now depending on the evaluation given to the segment
4:         Modify( segment )
```

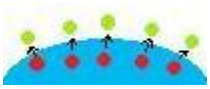

Figure 2: Segment Modification section: Each configuration in the segment is modified.

Algorithm 2.3 Path Evaluation Section

```
1: newPath = append all modified segments
2: Evaluate( newPath )
3: if (newPath is accepted) then return newPath
4: else choose a different path modification strategy and apply
```

After we modify the segments we proceed to evaluate the resulting path. The Path Evaluation section is presented in Algorithm 2.3.

Figure 3: Path Evaluation section: The resulting path is evaluated.

2.3 The Random Walk Method: An example of an implementation of the generic algorithm

After having finished defining and implementing the generic algotihm, the next step in our project was to use it to implement a particular mode of path modification. So we developed the Random Walk method by defining each one of the generic sections of the generic algorithm.

Description: The Random Walk method defines nodes as invalid if they are colliding with an object. It defines a valid segment as a segment that contains only a series of valid nodes in the path. It defines an invalid segment as a series of invalid configurations in the path. However, the first and last configurations of an invalid segments will be valid since we will store the last valid configuration from the path that occured before the invalid segment, and the first valid configuration that occurred after the invalid segment. The implementation of the Path Segmentation section takes care of building the segments in this way.

After having divided the path into segments, we leave the valid segments as they are and apply our particular segment modification method to the invalid segments. This method takes each one of the invalid nodes in the segment and applies Algorithm 2.4:

Algorithm 2.4 Random Walk Segment Modification section

1: Generate d number of random directions.
2: For each direction D of the d direction
3: do a random walk (Starting from the original node, walk in that direction for a constant maximun of m steps with a constant stepsize s)
4: For each on of these random walks
5: if a free configuration is found (see Figure 4)
6: then check if it can be connected to the last node of the current segment of modified nodes (see Figure 5)
7: if it can be connected then add it to the segment of modified nodes, and quit walking that direction
8: else drop the node and continue walking
9: if after m steps no new node was added to the the segment of modified nodes
10: then quit walking that direction

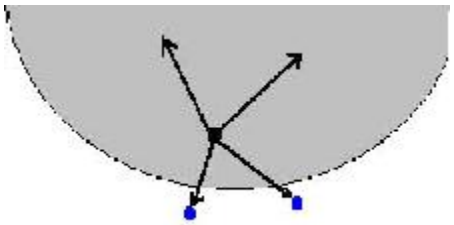

Figure 4: Here an example of the Random Walk Method. Here four random directions are taken. Two find free configurations.

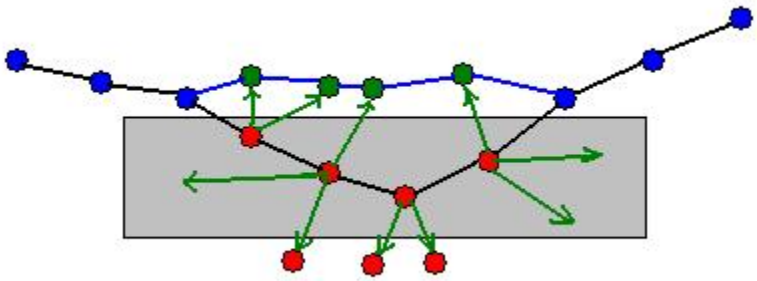

Figure 5: In the Random Walk Method new free configurations are not added to the new path if they cannot be connected to it.

3 Experiments:

3.1 Environments Tested

We used three environment with different degrees of difficulty to test our random walk method. Narrow, Walls, and Hook-Rigid.

Narrow: This is the simplest of the three environments tested. This environment has two walls close to each other leaving only a very narrow passage between them. Located between these two plates we have our moving agent, in this case a small box. The query to solve is to pull the agent out from its initial position at the center of the narrow areas to a location outside the space between the two walls. The space contained between the two walls gives a lot of room for the agent to operate, making this the simplest of the three environments.

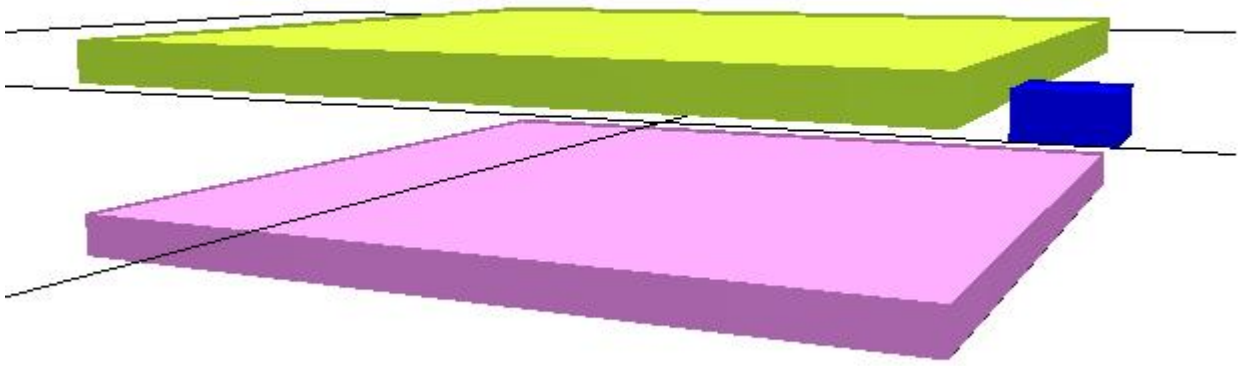

Figure 6: Narrow Environment. Here the moving agent (the blue box) is outside the narrow area

Walls: This environment has a series of walls, each one of them has one window. This series of walls forms a series of rooms that have access to one another through small windows. The moving object, in this case a small cube, needs to move through all the rooms going through the windows. The main difficulty in this task is that there is only one port of communications between every pair of adjacent walls, and the agent must go necessarily through these ports.

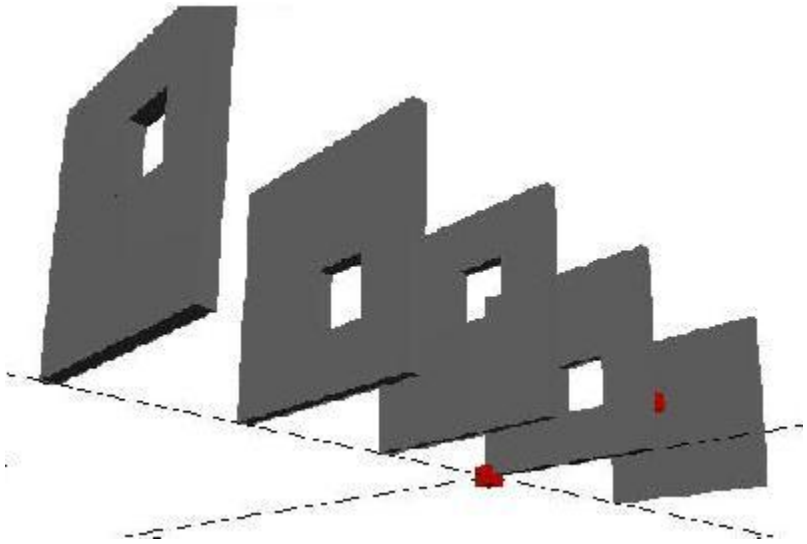

Figure 7: Walls Environment. The walls form a series of rooms that comunicate through windows. The agent (in this case a cube) needs to move from the first to the last room.

Hook-Rigid: Of the three environments this is the hardest one. Here we have two walls with a long but thin window in the middle. We have a Hook-Rigid as the moving agent and we need to move accross each one of the walls. This operation will require a large, complex series of translation and rotations because the agent has a very complex structure.

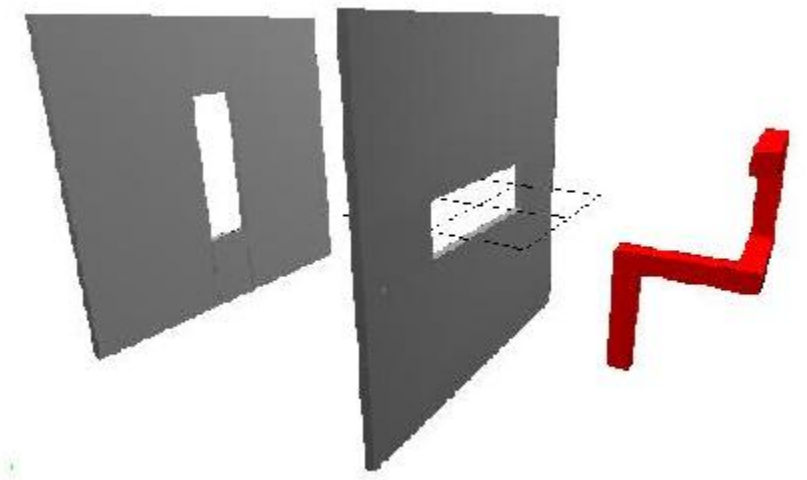

Figure 8: The agent (in this case a Hook-Rigid) needs to pass through the two windows through a series of controlled translations and rotations

3.2 Environment: Narrow

In this environment we need to modify a path that tries to move the agent out of the narrow space between the walls through a straihgt line. See in figures 9a) through 9c) how the path collides with one of the two walls and makes the agent go through it.

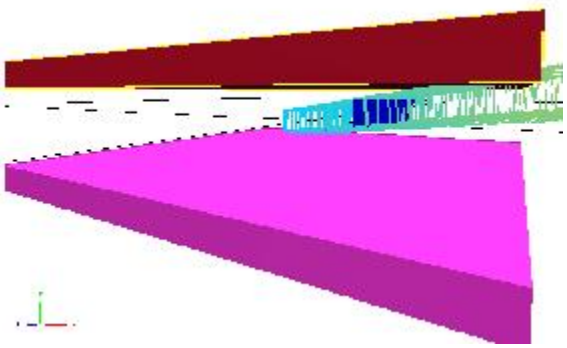

Figure 9 a): Original path in the Narrow Environment. The path that the box is going to follow is colliding with one of the walls.

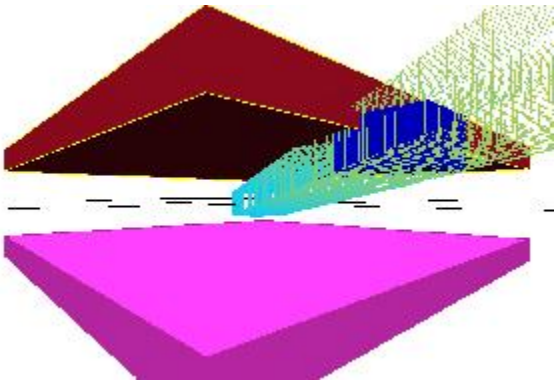

Figure 9 b): Original path in the Narrow Environment. The box collides with the wall

Figure 9 c): Original path in the Narrow Environment. The box is colliding with the wall (viewed from a different angle).

Path Name	Directions	Max. Steps	Step Size	Path Length	Nodes	Collisions	Invalid Edges
Original Path	N/A	N/A	N/A	22.5	502	76	78
Modified Path 1	5	5	0.1	22.99	427	0	2
Modified Path 2	10	10	0.1	39.56	449	0	5

Table 1: Experiments carried on environment Narrow

In Table 1 we can see the statistics of two experiments carried to solve the query. The parameters used for the modifications are:

-Directions: Number of random directions produced for the random walk process

-Max. Steps: Maximum number of steps to take if no free configurations are found on a direction

-Step Size: Size of each the steps taken during the random walk process.

Oberve in both experients that the number of nodes in the modified paths is lower than in the original, and the length of the modified paths is increased in both experiments. This growth in path size happens because of the order in which nodes are generated by the Random Walk Method. Some nodes that are close to the goal might be generated before nodes that are farther from the goal. This causes the agent to move back and forth. In Experiment 1 the increase in path size is not very noticeable. It goes from 22.5 to 22.9, and produces a path with 0 collisions and only 2 invalid edges. In Eperiment 2 we increase the value of some parameters from Experiment 1 and we obtain a larger path size. It goes from 22.5 to 39.56. This increase in size is excesive. We want to produce the shortest acceptable path that we can. In Experiment 2 we produce more random directions than in Experiment 1 and this produces many more unordered configurations than in Experiment 1, which causes the agent to do a lot of back-and-forth moves. Below we can see images of a the path resulting from modifying the original path in Experiment 1.

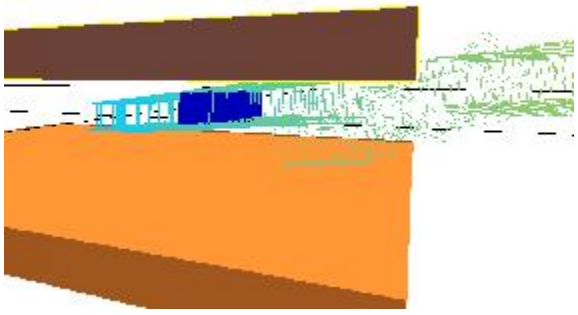

Figure 10 a): Modified path in the Narrow Environment. Observe how new configurations have been created avoiding the obstacle in the modified path

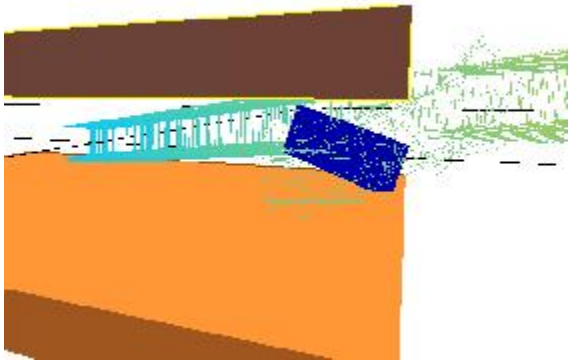

Figure 10 b): Modified path in the Narrow Environment. The agent deviates from the old trajectory to take the new path

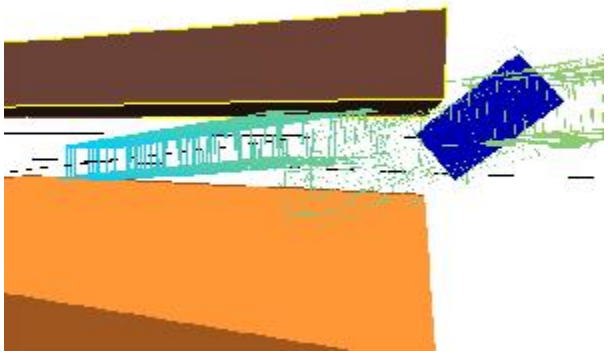

Figure 10 c): Modified path in the Narrow Environment. The agent is able to avoid the obstacle and resume through the old collision free segment of the path

3.3 Environment: Walls

In this environment, to solve our query we need to modify a path that tries to move the agent from the first room to the last room following a stright line that goes close to the the windows. Obrserve in the image below how the original path makes the agent go through the obstacle.

Path Name	Directions	Max. Steps	Step Size	Path Length	Nodes	Collisions	Invalid Edges
Original Path	N/A	N/A	N/A	16	1002	162	167
Modified Path 1	6	5	0.1	261	1461	0	6
Modified Path 2	10	10	0.1	728	1997	0	6

Table 2: Experiments carried on environment Walls

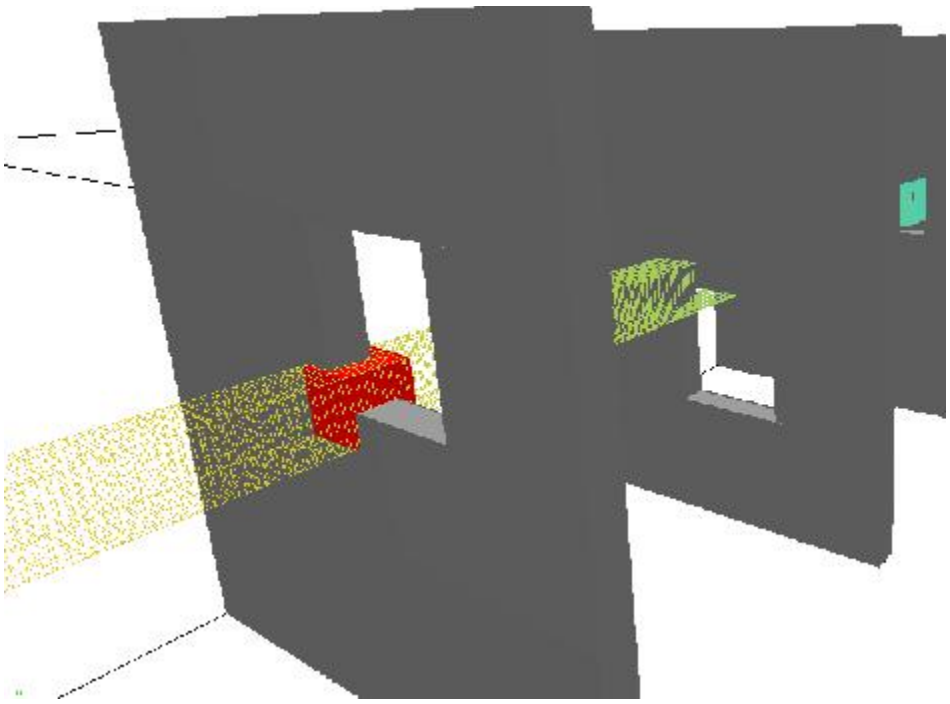

Figure 11: Original path in the Walls Environment. Agent goes through the wall.

We carried experiments to solve the query, and Table 2 holds the results. Oberve in Experiment 2 that by increasing the parameters from Experiment 1 we obtained similar results in the number of invalid edges while the path length increased almost three times. We always prefer to produce the shortest valid path so the parameters used in Experiment 1 seem like the most optimal of the two to solve this query. In Figure 12 we can see the modified path produced in Experiment 1. Overall the Random Walk method performed fairly well in this environment by producing valid configurations located inside the windows, therefore opening a collision-free passage for the agent to move from one room to another.

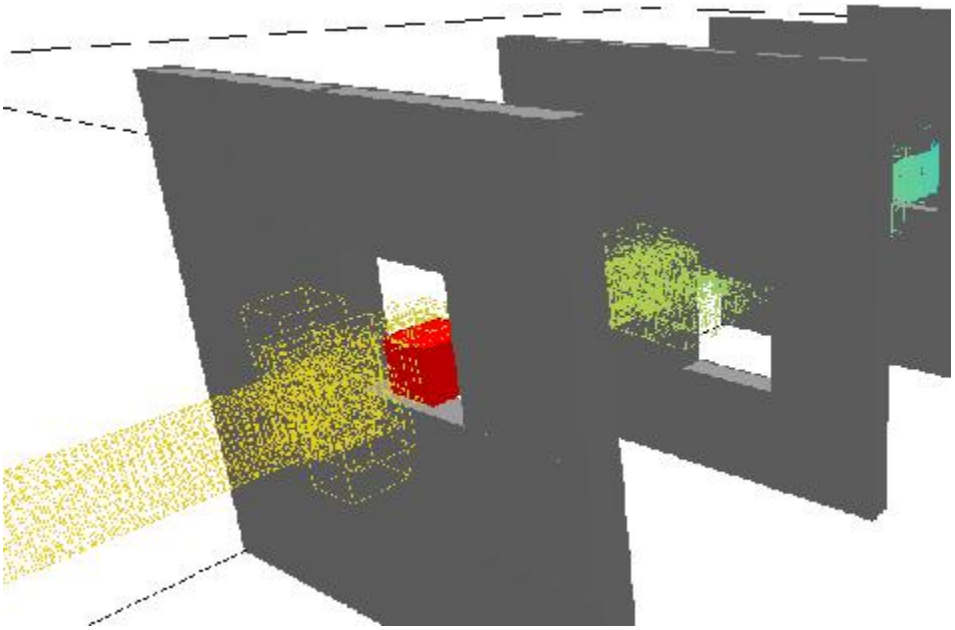

Figure 12: Modified path in the Walls Environment. Observe how new free configurations have been created inside the windows. Now the agent can avoid the wall and pass through the windows

3.4 Environment: Hook-Rigid

In this environment we need to modify a path that tries to move the agent through the two thin windows located in the walls. This is a very hard task that requires an accurate series of translations and rotations of the agent (Hook-Rigid). We carried two experiments to solve the query and Table 3 holds the results.

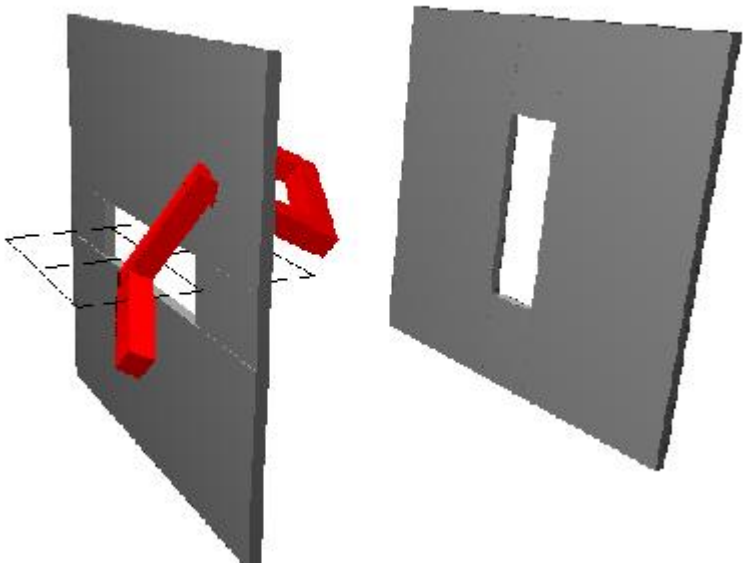

Figure 13: The original path in the Hook-Rigid Environment has many configurations in collision, as this one here where the agent is divided in two by the obstacle

Path Name	Directions	Max. Steps	Step Size	Path Length	Nodes	Collisions	Invalid Edges
Original Path	not applicable	not applicable	not applicable	1475	347	49	292
Mod Path 1 (*)	10	25	1	1450	322	0	246
Mod Path 2 (*)	20	25	1	1446	324	0	247

Table 3: Experiments carried on environment Hook-Rigid(*) The path produced failed to solve the query

These two experiments returned bad results as none of them was able to solve the query successfully. Apparently the degree of difficulty of this task presents too much of a challenge for the Random Walk method. Observe in the image below (Figure 14), which shows the modified path, how the path is cut into three connected components with no communication with one another. This is caused by the low probability of producing free configurations located within the thin windows. Also, the original path created through the haptic device produced many configurations in parts of the path that were not on the difficult regions (the difficult regions are the passages located in the middle of the walls) while producing a very small number of configurations near the difficult regions. This causes the Random Walk Method to fail to produce enough free configurations in the difficult regions.

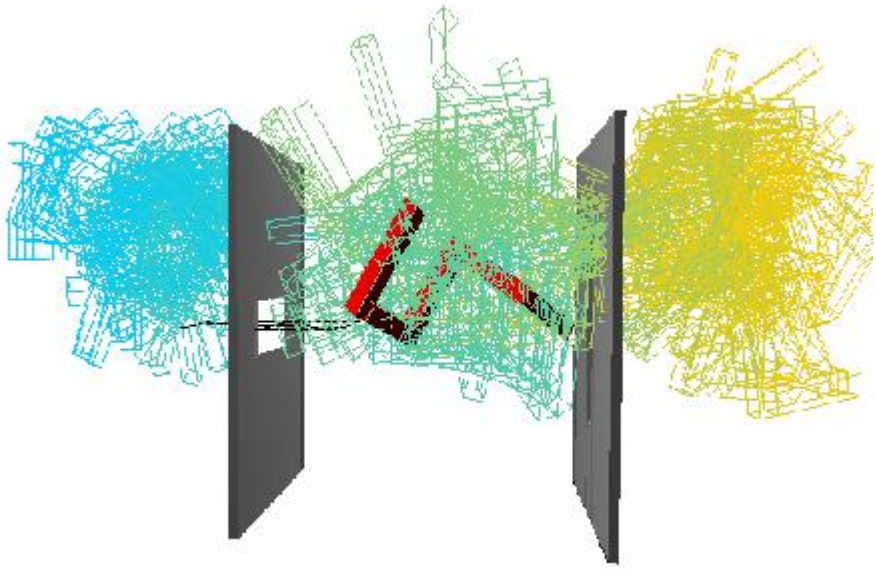

Figure 14: The resulting path in the Hook-Rigid Environemnt is cut in three connected components with no edges between one another. This problem is very challenging for the Random Walk Method

4 Future Work

To continue this research in path modification methods we should probably explore the following two tasks:

-Improve the Random Walk Method. This method could be improved in two main areas. One is to make some upgrades to it to try to tackle more difficult problems successfully. The Hook-Rigid environment showed how the current algorithm used for the Random Walk Method failed to produduce a valid path if the path to modify is not very approximate. The other area in which the Random Method could be improved is the quality of the paths it produces. Even when the algorithm produces a valid path most of the time it does a lot of back and forth movements that increase the path size unnecessarily.

-Create other path modification algorithms to construct over the generic method. It is very likely that other strategies different from the Random Walk Method could produce better results. So it would be a good idea to explore new methods.

5 Conclusions:

The generic algorithm showed to be very usable. It successfully handled the general operations of the path modification process while it left enough flexibility for the implementing Random Walk Method to add its own particula flavor of path modification. The Random Walk method performed fairly well in simple to midium difficulty problems but it showed some low performance on the harder problem. Definetely there is room for improvement on it. Overall path modification algorithms look promising, and research on the topic could provide great help in the future for the solution of motion planning problems.

References

1. L. E. Kavraki, P. Svestka, J. C. Latombe, M. H. Overmars, ***Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces.*** IEEE Trans. On Robotics and Automation.

2. N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. ***OBPRM: An obstacle-based PRM for 3D workspace.*** In Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR).

3. A. Challagalla, P. Colleman, L. Tapia, N. M. Amato. ***Haptic Aided Motion Planning.*** Parasol Lab, Department of Computer Science.

4. J. H. Reif. ***Complexity of the mover's problem and generalizations.*** In Proc. IEEE Symp. on Foundations of Computer Science, 1979.

5. H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki and S. Thrun. ***Principles of Robot Motion: Theory, Algorithms, and Implementations.*** MIT Press. Chapter 7, Boston, 2005.

6. Burchan Bayazit. ***Providing Haptic 'Hints' to Automatic Motion Planners.*** AI Robotics Seminars , Department of Computer Science, Texas AM University, Fall 1999.

7. O. Burchan Bayazit, Guang Song, Nancy M. Amato. ***Enhancing Randomized Motion Planners: Exploring with Haptic Hints***, Autonomous Robots, 10(2):163-174, 2001. Also, In Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 529-536, Apr 2000. Also, Technical Report, TR99-021, Parasol Laboratory, Department of Computer Science, Texas AM University, Oct 1999.