Appear in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2002

An Incremental Learning Approach to Motion Planning with Roadmap Management

Tsai-Yen Li
Computer Science Department
National Chengchi University,
Taipei, Taiwan, R.O.C.
li@nccu.edu.tw

Yang-Chuan Shie
Computer Science Department
National Chengchi University,
Taipei, Taiwan, R.O.C.
g8909@cs.nccu.edu.tw

Abstract

Traditional approaches to the motion-planning problem can be classified into single-query and multiple-query problems with the tradeoffs on run-time computation cost and adaptability to environment changes. In this paper, we propose a novel approach to the problem that can learn incrementally on every planning query and effectively manage the learned roadmap as the process goes on. This planner is based on previous work on probabilistic roadmaps and uses a data structure called Reconfigurable Random Forest (RRF), which extends the Rapidly-exploring Random Tree (RRT) structure proposed in the literature. The planner can account for environmental changes while keeping the size of the roadmap small. The planner removes invalid nodes in the roadmap as the obstacle configurations change. It also uses a tree-pruning algorithm to trim RRF into a more concise representation. Our experiments show that the planner is flexible and efficient.

1. Introduction

The motion-planning problem has been well studied in the last three decades. The basic problem, called the *find-path problem* or the *piano-mover's problem*, is about finding a collision-free path for a robot moving in a workspace cluttered with obstacles[17]. The developed techniques for solving this problem has been shown to be well applicable to many domains other than robotics such as computer animation[9], assembly maintainability[7], intelligent navigation interfaces[14], and drug designs. According to [2], most path planners consist of two phases: *preprocessing* and *query* phases. In the preprocessing phase, the planning problem is converted into abstract data structures such as graphs that will be searched later for a feasible path in the query phase. The percentage of running times for the two phases might vary greatly for different planners. For example, in the Randomized Path Planner (RPP)[3], most time is spent in the query phase while in the Probabilistic Roadmap Method (PRM) planner[8], most time is spent in building a roadmap in the preprocessing phase.

Depending on how a planner is used, one can classify the planning problems into two categories: *single-query* and *multiple-query* problems. In the single-query problem, one does not assume anything about previous queries, and the planner always starts to answer every query from scratch. On the other hand, in multiple-query problems, one usually assumes that the environment does not change often and multiple queries will be issued for the same environment. In this case, the planner can afford to spend more time on preprocessing such that the queries afterward can be answered more quickly. Choosing an appropriate planner for a given problem remains a state of art requiring human judgment.

In this paper we propose a unified path-planning approach that can be used in an either single-query or multiple-query problem. The planner is well suited for a single-query problem, and it learns the given environment incrementally as the planner is called multiple times. The planner is as efficient as other single-query planners and the performance gets improved when the learning process goes on. A data structure, called *Rapid-exploring Random Tree (RRT)*, has been shown to be an effective roadmap representation[11][12]. Our planner extends this structure to a more flexible one, called *Reconfigurable Random Forest (RRF)*. This data structure allows us to modify the roadmap by removing invalid nodes as the obstacle configurations change at run time. In addition, the planner is designed to periodically trim unnecessary nodes in RRF in order to keep the roadmap slim.

The rest of the paper is organized as follows. We will first review related work in the next section. We will then review the RRT structure, the RRT-Connect algorithm, and present our unified approach with the RRF data structure in Section 3. In the fourth section, we will extend the planner to maintain a concise roadmap in a changeable environments. Experimental settings, results, and analysis will be given in Section 5. Finally, we will conclude our work in the last section.

2. Related Work

One can obtain an introduction to the general motion-planning problem or a survey of approaches in [13].

Generally speaking, early research focuses on developing theoretical foundation and complete solutions for the problem[17]. However, under the curse of dimensionality, this type of solution deems to be impractical for problems involving high dimensional spaces. In the last decade, several researches start to look for practical solutions that can be applied to wider arrange of applications despite they usually lack completeness[3][8].

The randomized planners are the popular approaches along this direction. The early RPP planner is a typical single-query planner utilizing artificial potential fields as search heuristics. In contrast, the PRM planner is a typical multiple-query planner that uses a great portion of time to construct a representative roadmap for later queries. For this type of method, the way that the sampled configurations are selected greatly affects the planning results. Variations of sampling strategies have been proposed for a generic or a specific problem[1][5]. The work in [15] uses visibility information to produce a smaller roadmap. In recent years, one form of randomized roadmap called RRT has been shown to be effective in solving several difficult problems by being able to explore the freespace evenly[11][12]. A single-query path planner, called RRT-Connect, using this data structure to perform bi-directional search has also been developed[10].

Several planners in the literature took a learning approach[4][6][8][15]. In one of the early papers proposing the idea of probabilistic roadmap, roadmap was used as a way to learn the freespace[15]. They observed sharp learning curves when the roadmap got denser. However, the number of sampled configurations for an acceptable success rate remains an empirical setting. In [6], a planer called ERPP uses the local minima learned in RPP-based planner to build a roadmap for a static environment. The work in [4] uses a genetic algorithm to evolve critical configurations, called landmarks, for freespace connectivity.

3. Building incremental roadmap

The basic path-planning problem is to find a collision-free path for a robot amongst obstacles in a given environment. The set of all possible configurations q for the robot define the so-called *Configuration Space* (*C-space* for short), denoted by C. Let C_{free} denote the open subset of collision-free configurations in C. The path-planning task is to find a continuous curve in C_{free} connecting an initial configuration, q_i, and a goal configuration, q_g.

3.1. The RRT-Connect planning algorithm

The Rapidly-exploring Random Tree (RRT) was introduced in [11] as an efficient data structure to explore C_{free}. Its main difference from traditional probabilistic roadmaps is on that RRT grows outward from a tree although configurations are sampled randomly in the freespace. As depicted in Figure 1, the growing process starts by selecting a random configuration, q_{rand}, as the growing direction. The nearest configuration, q_{near}, in the current RRT to q_{rand} is determined, and a new configuration, q_{new}, that is ε-distance away from q_{near}, is computed and added into the RRT. This process is called EXTEND. In [10], an efficient single-query planning algorithm, called RRT-Connect, uses RRT as the main data structure to connect the given initial and goal configurations (q_i and q_g). Two RRT's, rooted at q_i and q_g, respectively, are used to connect to each other. At each step of the growing process, a random configuration q_{rand} is sampled in the freespace. One RRT uses the EXTEND procedure to add to itself a new configuration, q_{new}, while the other RRT uses another procedure called CONNECT to grow (EXTEND) toward q_{new} as much as possible. If CONNECT can bring the RRT to reach q_{new}, then the two RRT's have been successfully connected and a feasible path is returned. Otherwise, the two RRT's swap to allow them to grow in the other direction.

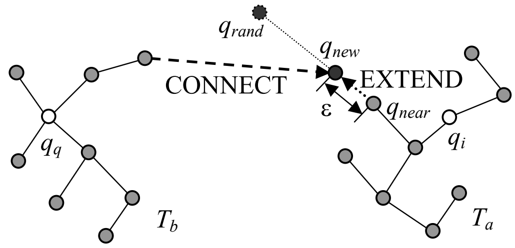


Figure 1: Two RRT's use EXTEND and CONNECT to merge into one tree

3.2. The incremental learning algorithm: RRF_CONNECT

Since the RRT-Connect planner is a single-query planner, it always starts from scratch for applications requiring multiple queries,. However, we think the freespace explored in previous planning queries could be very useful in the ones that follow. Therefore, we extend the RRT-Connect algorithm to take advantage of the previous learned knowledge about the freespace to save time in future queries. Since previously learned RRT's are kept for future uses, the data structure becomes a forest consisting of multiple RRTs. We called this forest, Reconfigurable Random Forest (RRF). It is reconfigurable because the trees in the forest can be merged, split, or pruned in the planning process.

Figure 2 shows the RRF_CONNECT planning algorithm. The algorithm assumes a global data structure called *forest* to store the list of currently maintained trees. A main subprocedure used in RRF_CONNECT is called MERGE_RRTs. This procedure tries to connect each tree in the forest, except for the currently considered tree T_A, to the designated new configuration, q_{new}, via the CONNECT procedure. The tree is merged with T_A if the connection is

```
MERGE_RRTs(T_A, q_new)
for each T in forest
    if (T ≠ T_A)
        if (CONNECT(T , q_new) = Reached)
            REVERSE_PARENT(T , q_new);
            forest.remove(T);
return;
```

```
RRF_CONNECT(q_i , q_g , K)
T_i.init(q_i); T_g.init(q_g);
forest.add(T_i); forest.add(T_g);
MERGE_RRTs(T_g , q_g);
MERGE_RRTs(T_i , q_i);
if (T_i.tree_id = T_g.tree_id)
    return PATH(q_i, q_g);
for k =1 to K do
    q_rand ← RANDOM_CONF( );
    if (EXTEND (T_i , q_rand) ≠ Trapped)
        MERGE_RRTs (T_i , T_i.q_new);
        if (T_i.tree_id = T_g.tree_id)
            return PATH(q_i, q_g);
    SWAP(T_i , T_g);
return Failure;
```

Figure 2: The RRF_ CONNECT algorithm

successful. In the RRF_CONNECT algorithm, after the trees rooted at q_i and q_g are initialized, we first call the MERGE_RRTs procedure to see if we can connect the two configurations to the forest without adding additional configurations. If this is not successful, a randomly sampled configuration, q_{rand}, will be selected to extend T_i, and MERGE_RRT will then be called again. This process will repeat until the T_i and T_g are merged (success) or a predefined maximal number of sample configurations is reached (failure).

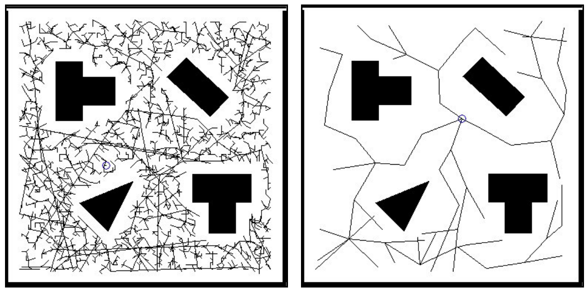
4. Roadmap management

As the learning process goes on, the RRF structure might need to be updated for a few reasons. First, if the obstacle configurations are changeable at run time, then there should be a way to invalidate certain portion of the forest. Second, there should be a way to trim unnecessary nodes as the forest grows. Keeping a tidy roadmap not only save space but also the time required to search for a path.

4.1. Extension to consider environment changes

The assumption of static environment restricts the application domain of roadmap-based methods. In general, when the environment changes, the roadmap needs to be reconstructed. However, there exist applications where obstacles in the environments need to be moved but not constantly or frequently. For these scenarios, a major portion of the roadmap might still remain valid and useful for future queries. All we have to do is remove invalid nodes after the obstacle changes and reconstruct the RRF structure.

We use the following process to reconstruct RRF after the



(a) (b)

Figure 3. An example of tree pruning from (a) to (b)

obstacle configuration changes. First, we compute the candidate nodes whose configurations fall inside the bounding box of the obstacle's new configuration. Second, we perform collision checks on these nodes to find the list of invalid ones to be removed. Third, for each invalid node, we check if their children are also in the invalid list. If not, then the subtree rooted at each of these child nodes will be trimmed off and becomes a new tree in RRF.

In the above update process, the first step may contain example-specific procedure to compute the bounding box of obstacles in C-space. The second step is the most time-consuming one since it involves collision detections to find invalid nodes. However, for time-critical applications, we think this step could be totally skipped without sacrificing the correctness of the planning result. First, these nodes are selected under a necessary condition. The list contains a conservative list of candidate nodes. Second, if the obstacle is currently moving, its bounding box could contain nodes that will become invalid sooner or later. Third, since the RRT structure tends to grow the tree toward unexplored area, the extra space cleaned up due to the imprecise update can be filled up quickly in the future learning process. Examples of this update process will be given in the next section.

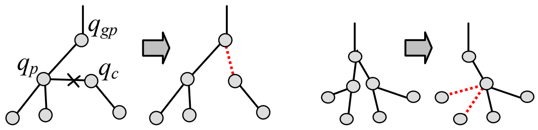
4.2. Forest pruning

As the learning process goes on, the number of nodes added into RRF increases significantly. Although the more nodes in a roadmap, the better they can capture the overall structure of freespace. However, as the number of nodes increases to some degree, the performance of the planner might be worsen due to the large roadmap size. As a result, it is desirable to prune RRF to make it a more concise representation. In Figure 3, we show an example of pruning an RRF from a dense roadmap (2675 nodes, Figure 3(a)) to a tidier one (70 nodes, Figure 3(b)).

The PRUNE_TREE algorithm that is used to prune the trees in an RRF is listed in Figure 4. A tree is considered too dense *vertically* or *horizontally* if there exists a node too close to its grandparent node or to its sibling nodes, respectively. In this algorithm, we traverse the given tree in post-order, where a node is examined after its subtrees

```
PRUNE_TREE(q_p)
if q_p is NOT ROOT
  for each child q_c of q_p
    if DIST(q_p.parent , q_c) < MinVMergeD
       V_MERGE(q_p.parent, q_p, q_c);
for each child q_c of q_p
  PRUNE_TREE(q_c);
if q_p.nb_children >= 2
  for each pair (q_c1, q_c2) in (q_p.child, q_p.child)
    if DIST(q_c1, q_c2) < MinHMergeD
       H_MERGE(q_c1, q_c2);
```

Figure 4: The PRUNE_TREE algorithm prunes a given tree to a more concise representation



(a) (b)

Figure 5: (a) Vertical and (b) horizontal merges

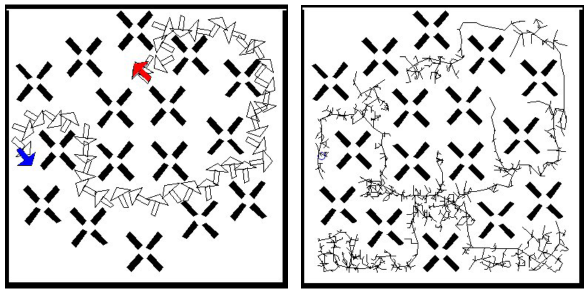
are traversed. When a tree is traversed, we remove nodes that are considered *redundant* to the structure of the tree. When examining a node (q_p), we first check if the distance between each of its child nodes (q_c) and its parent node (q_{gp}), in some user-specified metric, is less than some limit, *MinVMergeD*, and there exists a collision-free straight-line path between them. If the above conditions are met, we perform a *vertical merge* (V_MERGE) operation that makes the q_c node connect to the q_{gp} node directly instead of to the q_p node, as shown in Figure 5(a). Since the q_p node must have children to satisfy the above conditions, it must be an interior node in a tree. However, if the q_p node becomes a leaf node after its children are all relinked to its parent, then it is deleted from the tree. Second, we check if the distance between any ordered pair of child nodes (q_{c1} and q_{c2}) is less than some limit, *MinHMergeD*, and all of q_{c1}'s child nodes, if any, can be moved to q_{c2} with collision-free links. If so, we perform a *horizontal merge* (*H_MERGE*) operation to move the links, and q_{c1} is deleted from the tree, as shown in Figure 5(b).

5. Experiments

The aforementioned planner has been fully implemented in Java. The planning times reported in this paper were collected from experiments running on a regular PC with a K6-3 400 MHz processor. The size of the C-space (x, y, θ) for all examples shown in this paper is 128x128x100. The roadmaps depicted in this section are actually 2D projections of 3D C-space into the 2D workspace.

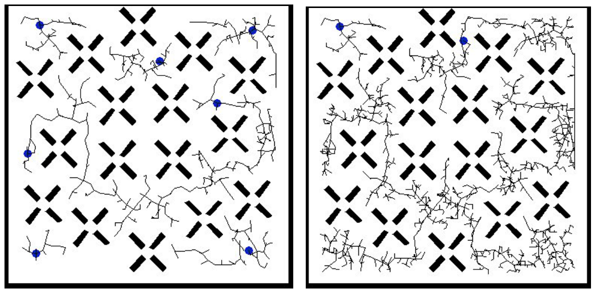
5.1. Experiments for static environments

Among several tested examples, a basic path-planning example with an arrow-shaped robot in static environments is shown in Figure 6. The number of sampled nodes and the planning time are 699 and 0.2 sec. The example assumes a clean start with no pre-built roadmap. In our experiments, we continued to issue a great number of random planning queries for the same static environment to see how the later one can take advantage of the roadmap learned earlier. Two snapshots of the incremental roadmap construction process are shown in Figure 7. The roots of the trees in RRF are depicted with solid dots. The RRF in Figure 7(a) contains seven trees. As the number of planning queries increase, the number of nodes in RRF increases to 2359 and the number of trees reduces to two only, as shown in Figure 7(c). The planning times for the example will be reported in a later subsection.



(a) (b)

Figure 6. A basic example: (a) found paths and (b) the generated roadmaps.



(a) (b)

Figure 7. Snapshots of a growing RRF

5.2. Example for environments with obstacle configuration changes

We have implemented the algorithm in Section 4.1 to allow configuration changes of environmental obstacles. An example illustrating the idea of reconfigurable forest is shown in Figure 8. The example starts with no pre-built roadmap (Figure 8(a)) and after 1000 planning queries, the RRF ends up with a dense roadmap consisting of a single RRT (Figure 8(b)). Then, we moved the obstacles on the lower-right corner to the center of the workspace. The roadmap is updated with the principles described in Section 4.1, and the update process takes only about 50 ms. About 300 invalid nodes are detected and removed from the RRF, resulting in 22 new trees as shown in Fig-

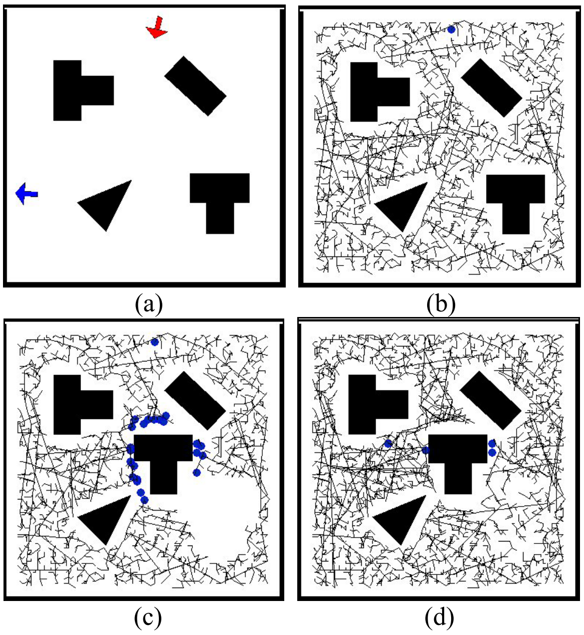


Figure 8. An example of RRF in a changeable environment

ure 8(c). The roots of these trees, surrounding the moved obstacle, indicate where the forest is split. After another 500 random planning queries, the empty area that was originally occupied by the obstacle is quickly and evenly filled with new nodes, as shown in Figure 8(d). In this aspect, the RRT structure, compared to other roadmap representations, demonstrates its strength in exploring unvisited area and therefore is more appropriate for managing roadmaps in such a dynamic scenario.

5.3. Experiments on forest pruning

We use the example shown in Figure 9 to illustrate the effects of the forest pruning process. The RRF roadmap shown in Figure 9(a) contains 5974 nodes in total. According to the PRUNE_TREE algorithm in Section 4.2, two types of merges might be applied to RRF to reduce its size. To observe the effect of each type of operation, we only perform the vertical-merge operation that attempts to reduce the hierarchy of RRF by removing interior nodes. After 741ms of computation, we can reduce the number of nodes to 3412 as shown in Figure 9(b). This operation has flattened the RRF, but it also results in trees that are too broad horizontally. By applying the horizontal-merge operation, which takes 291ms, we obtain an RRF consisting of 518 nodes only as shown in Figure 9(c). If we apply the vertical and horizontal merges simultaneously as in the PRUNE_TREE algorithm, the computation time is only about 330ms. Like most algorithms for smoothing paths generated by a path planner, the PRUNE_TREE procedure does not guarantee to result in an optimal solution at once. Instead, the procedure can be applied to an RRF iteratively possibly until the size cannot be further reduced.

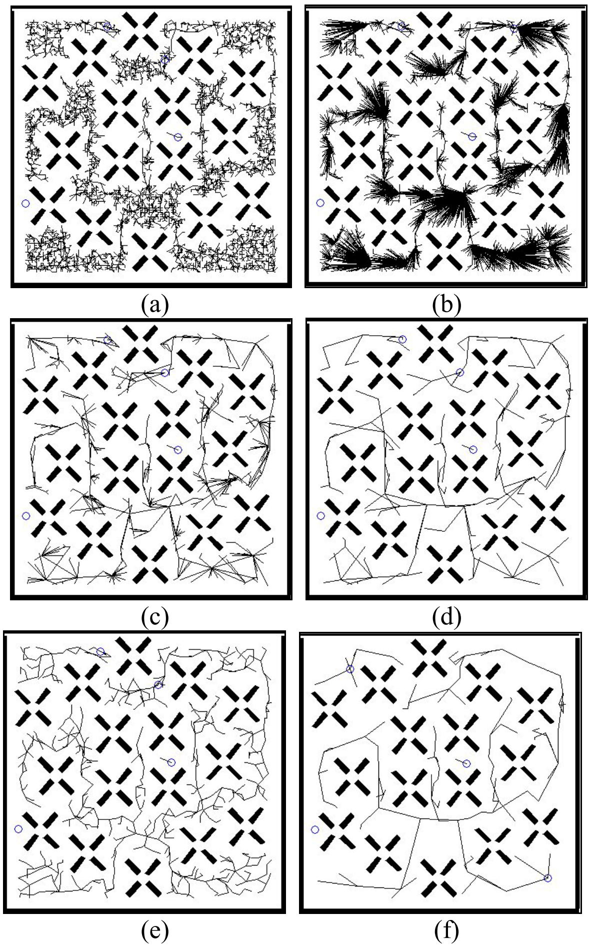


Figure 9. Experimental results on forest pruning with different parameter settings

In Figure 9(d), we show such an example consisting of 200 nodes only.

Two parameters in the PRUNE_TREE procedure can be set empirically to determine the resulting RRF: *MinVMergeD* and *MinHMergeD*. In the figures mentioned above, the MinVMergeD and MinHMergeD are four and two times of the ε-distance used in the EXTEND procedure, respectively. A smaller value for these minimal distances yields a finer but larger roadmap. In Figure 9(e), we show an example of a finer RRF consisting of 609 nodes with the minimal distances set to ε. Furthermore, the PRUNE_TREE procedure can be applied periodically to RRF during the learning process. Our experiments show that the number of nodes in RRF can often be further reduced as the process goes on. For example, a tidier representation of 122 nodes for the RRF can be obtained, as shown in Figure 9(f), after a few iterations of planning query and tree-pruning operations.

5.4. Discussion

A major advantage of the proposed learning approach is

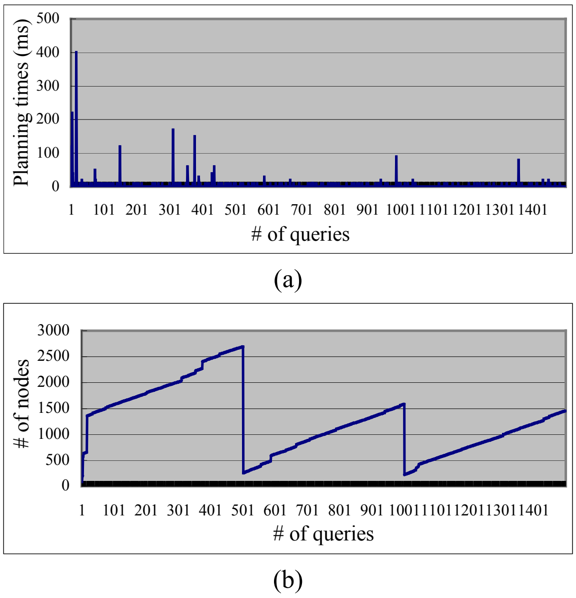


Figure 10. Planning times and accumulated number of nodes as number of queries increases

that the C-space can be learned incrementally as the process goes on. Our experiments show that, as one can expect, the planner learns more about the C-space when the query problem is difficult. However, the occasions are rather sparse. For example, as shown in Figure 10(a), the planner learns about most of the C-space in the very beginning and a few times in the middle of 1500 queries. In this experiment, the PRUNE_TREE procedure was called every 500 queries to reduce the number of nodes in RRF, and we were able to reduce it by a factor of 10, as shown in Figure 10(b).

In previous subsection, we have shown the effect of the parameters, such as MinVMergeD and MinHMergeD, in the PRUNE_TREE algorithm qualitatively. However, we need to do further experiments in order to determine their effects on planning times after RRF is pruned. Intuitively speaking, less nodes means harder to merge trees but faster to search the roadmap. Similarly, it is difficult to determine an optimal frequency for pruning an RRF. It takes time to prune a tree but a smaller and better roadmap representation could save time in the long run.

6. Conclusions

In this paper, we have described an incremental learning approach to the general path-planning problem. This approach extends the RRT-Connect algorithm proposed in the literature to maintain the RRF roadmap learned in previous queries. The new planner can manage the roadmap effectively and efficiently as the environment changes and the learning process goes on. In addition, the planner can be run in an unsupervised manner since it can always maintain a concise and representative roadmap. We believe that such a path planner can be applied to a wider range of applications in robotics and other related fields.

7. References

[1] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo, "OBPRM: An Obstacle-Based PRM for 3D Workspaces," *Robotics: The Algorithmic Perspective*, pp.630-637, 1998.

[2] J. Barraquand, L. Kavraki, J.C. Latombe, T.Y. Li, and P. Raghavan, "A Random Sampling Scheme for Path Planning," *Intl. J. of Robotics Research*, 16(6), pp.759-774, Dec. 1997.

[3] J. Barraquand and J. Latombe, "Robot Motion Planning: A Distributed Representation Approach," *Intl J. of Robotics Research*, 10:628-649, 1991.

[4] P. Bessiere, J. M. Ahuactzin, E.G. Talbi, E. Mazer, "The 'Adriane's Clew' Algorithm: Global Planning with Local Methods," *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, IEEE Press*, pp.1373-1380, 1993.

[5] R. Bohlin and L. Kavraki, "Path Planning Using Lazy PRM," *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pp.521-527, 2000.

[6] S. Caselli, and M. Reggiani, "ERPP: An Experience-based Randomized Path Planner," *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pp.1002-1008, 2000.

[7] H. Chang, and T.Y. Li, "Assembly Maintainability Study with Motion Planning," *IEEE Intl. Conf. on Robotics and Automation*, pp. 1012-1019, May, 1995.

[8] L. Kavraki, P.Svestka, J. Latombe, and M. Overmars, "Probabilistic Roadmaps for Fast Path Planning in High-Dimensional Configuration Spaces," *IEEE Trans. on Robotics and Automation*, 12:566-580, 1996.

[9] Y. Koga, K. Kondo, J. Kuffner, and J.C. Latombe, "Planning Motions with Intentions," *Computer Graphics (SIGGRAPH'94)*, pp.395-408, 1994.

[10] J. Kuffner, and S. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," *Proc. of 2000 IEEE Intl. Conf. on Robotics and Automation*, May 2000.

[11] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," Iowa State University, 1998.

[12] S. M. LaValle and J. J. Kuffner. "Randomized kinodynamic planning," *Proc. of 1999 IEEE Intl. Conf. on Robotics and Automation*, 1999.

[13] J. Latombe, *Robot Motion Planning*, Kluwer, Boston, MA, 1991.

[14] T.-Y. Li, and H.-K Ting ., "An Intelligent User Interface with Motion Planning for 3D Navigation," *Proc. of the IEEE Virtual Reality 2000 Conf.*, March 2000.

[15] C. Nissoux, T. Simeon, and J.P. Laumond, "Visibility Based Probabilistic Roadmaps," *Proc. of the IEEE Intl. Conf. on Intelligent Robots and Systems*, 1999.

[16] M.H. Overmars, P. Svestka, "A Probabilistic Learning Approach to Motion Planning," *Technical Report UU-CS-1994-03*, Dept. of Computer Science, Utrecht Univ., Netherlands, Jan. 1994.

[17] J.H. Reif, "Complexity of the Mover's Problem and Generalizations," *Proc. of the 20th IEEE Symp. on Foundations of Computer Science*, pp. 421-427, 1979.