2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Lazy Reconfiguration Forest (LRF) - An Approach for Motion Planning with Multiple Tasks in Dynamic Environments

Russell Gayle*
Department of Computer Science
University of North Carolina
Chapel Hill, North Carolina, USA
rgayle@cs.unc.edu

Kristopher R. Klingler† and Patrick G. Xavier†
Robotics and Intelligent Systems
Sandia National Laboratories
Albuquerque, New Mexico, USA
{krkling, pgxavie}@sandia.gov

Abstract— **We present a novel algorithm for robot motion planning in dynamic environments. Our approach extends Rapidly-exploring Random Trees (RRTs) in several ways. We assume the need to simultaneously plan and maintain paths for multiple tasks with respect to the current state of a moving robot in a dynamic environment. Our algorithm dynamically maintains a forest of trees by splitting, growing and merging them on the fly to adapt to moving obstacles and robot motion. In order to minimize tree maintenance, we only validate the task paths, rather than the entire forest. The root of the inhabited tree moves with the robot. Dynamic re-planning is integrated with tree and forest maintenance. Coupling the robot motion with the planner enables us to support multiple tasks, for example providing an "escape" path while moving to a goal. The robot is free to move along whichever task path it chooses.**

We highlight the work by showing fast results in simulated environments with moving obstacles.

I. Introduction

In many real world situations, a robot attempting to plan must cope with incomplete information about the current and future state of its environment. The lack of information can include how objects move, which objects move, or even the environment itself. Additionally, robots are susceptible to errors in sensing or localization which may cause them to deviate from a given path or discover that they are no longer on it. We desire a framework for robot path planning and execution designed to address these issues. Such a framework would broaden system capability and add a layer of fault tolerance to complex robot motion. Unfortunately, traditional motion planning algorithms for complex robots are often unsuitable for dynamic environments.

Standard probabilistic approaches such as Probabilistic Roadmaps (PRMs) or Rapidly-exploring Random Trees (RRTs) have been extremely successful for planning in high dimensional and complex configuration spaces. However, large portions of the explored space representation can become invalidated when obstacles move or information about the environment is updated [1, 2]. This results in expensive

* This work was supported by a High Performance Computational Science Graduate Fellowship funded by the United States Department of Energy and administered by the Krell Foundation.

† This work was supported by the United States Department of Energy under Contract DE-ACO4-94AL85000. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy.

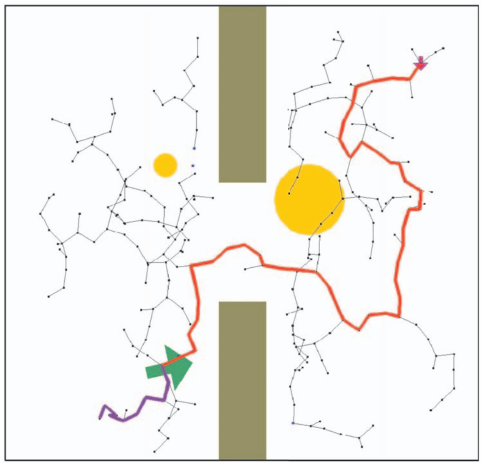
Fig. 1. A three degree of freedom robot planning in a dynamic environment. The circles are moving obstacles, while the rectangles are stationary. Segments represent forest edges, and are projected to the (x,y) plane. Thicker segments represent different robot behavioral choices or goals, and thinner segments are portions of the forest. We interactively maintain these *task paths* at run-time.

updates to rebuild the graph or tree. Several extensions to these algorithms have been made which help considerably, but often at the cost of additional assumptions such as having complete knowledge of obstacle motions.

In the traditional paradigm for using motion planning in robotics, the planner is given a problem and tries to plan a solution. The planner then hands over any found solution path to the robot to execute, and the planner discards its computations. In the emerging paradigm, as exemplified by [3, 4], the motion planner stays active and has persistent internal representations. Such a planner can interact with the robot and update internal representations continuously to enable it to revise the current plan or plan for a new task quickly. We suggest that the motion planner should be capable of planning and updating multiple paths simultaneously to enable the robot to have task options it can switch to quickly.

We introduce a novel framework for motion planning in dynamic environments that follows this emerging paradigm. Our framework supports the coordination of robot motion and sensing with path planning. At the core of our approach is a forest of RRTs. The tree that contains a node representing the robot's current state is called the *inhabited tree*. To support a robot capable of choosing among multiple tasks, our planner attempts to create and update a *task path* for achieving each task it has been given. When the robot chooses a task, the planner directs it along the corresponding task path, which we call the *active path*. As the robot moves and/or a change in the environment is noticed, the planner reconfigures the task paths, inhabited tree, and forest as necessary. Incremental growth and pruning occur as an online procedure while the robot is moving. As the planner is informed of robot motion, it shifts the inhabited tree's root to keep it associated with the robot's current state.

We call our planning approach a Lazy Reconfiguration Forest (LRF) because it lazily maintains the forest by growing and pruning it. The growing only occurs incrementally when task paths need to be (re-)discovered. The pruning only removes the invalidated nodes and edges rather than entire subtrees. Our preliminary results in combining lazy pruning and incremental growth have been promising in both performance and functionality. The work reported here builds on both [3] and [4].

The remainder of the paper is organized as follows. Section II describes some related work in the areas of motion planning with an emphasis on dynamic environments. Section III gives an overview of our LRF approach and the key ideas behind it. Sections IV and V provide details about the framework and planning algorithm. Sections VI and VII present our preliminary results and conclude by suggesting future work.

II. Related Work

The generic motion planning problem requires a robot to move within some environment from a start state to a goal state. This problem has been heavily researched and has a rich history of results. For a review of the field and coverage of many theoretical and practical methods, see [5–7].

Deterministic motion planning methods are considered practical only for low-degree-of-freedom systems and simple environments. Randomized methods offer probabilistic completeness while allowing practical performance for a wider range of applications. The most popular of these are Probabilistic Roadmaps (PRMs) and Rapidly-exploring Random Trees (RRTs) [1, 2]. The former places random samples throughout the robot configuration space C and searches for nearby neighbors to create a graph or roadmap. After connecting the start and goal configurations to this roadmap, a path can be found using any shortest path graph algorithm. The RRT method grows a tree in configuration space and is described in more detail in Section III-B.

In many real-world situations a robot must operate in an environment with moving or otherwise changing obstacles. Such dynamic elements greatly increase the difficulty of motion planning. In fact, motion planning for a single disc with bounded velocity among rotating obstacles is PSPACE-hard [8]. However, there have been many attempts to provide practical methods to cope with changing environments. For example, the D* deterministic planning algorithm repairs previous solutions instead of re-planning from scratch [9, 10].

There have been two types of approaches for adapting randomized planners to dynamic environments. The first type includes both PRMs and RRTs that reuse previously computed information to aid in finding a new path [3, 4, 11–13]. The most relevant to our work are [3] and [4]. The Dynamic Rapidly-exploring Random Tree (DRRT) proposed by [4] trims away subtrees of nodes found to be invalidated by an obstacle and causes regrowth using the trimmed tree if the solution path is invalidated. Discussion of the Reconfigurable Random Forest of [3] is integrated into Section III-C.

The second type of approach integrates obstacle motion directly into the planning process. Some variations plan directly in the $C \times time$ space. For example, the method of [14] generates a roadmap in this space. Other approaches additionally model differential constraints placed on the robot [15]. From the dynamics information, motion bounds can be computed from trajectory information to remove the chance of being in a state where a collision is definitely going to occur [16]. By storing the state of the entire environment at each configuration, planning can allow for deforming robots in deforming environments [17]. Other approaches attempt to leverage ideas from both deterministic and randomized algorithms to create a hybrid planner [18, 19].

III. Overview of Approach

We now provide an overview of the Lazy Reconfigurable Forest approach with its underlying ideas, beginning with an introduction of additional definitions and notation.

A. Notation and Definitions

For each motion planning task, we consider the robot, R, to be either a rigid body or a set of rigid bodies connected in some fashion by joints. We call the set of all possible configurations the configuration space, C, where each configuration $q \in C$ describes the position and orientation of the robot or its components. We consider the robot as a time-varying body, whose configuration at time t is $R(t)$. We assume that a metric ρ is defined on C.

The robot R exists in a 2D or 3D workspace along with a set of n obstacles $O = \{o_1, o_2, \ldots, o_n\}$. Each obstacle can vary over time, and its configuration at time t is described by $o_i(t)$. The motion of each obstacle need not be known *a priori*. We define the space $C_{free}(t)$ to be the set of all non-colliding configurations at time t. It should be noted that an explicit representation of C_{free} is not practical, but by using collision detection algorithms we can determine if a configuration q is in C_{free}.

Problem Formulation: Given an initial configuration q_{start} and a final configuration q_{goal}, we wish to find a sequence of robot configurations $Q = \{R(t_0), R(t_1), \ldots, R(t_n)\}$ such that the robot $R(t_i) \in C_{free}(t_i)$ for all t_i where $R(t_0)$ is q_{start} and $R(t_n)$ is q_{goal}.

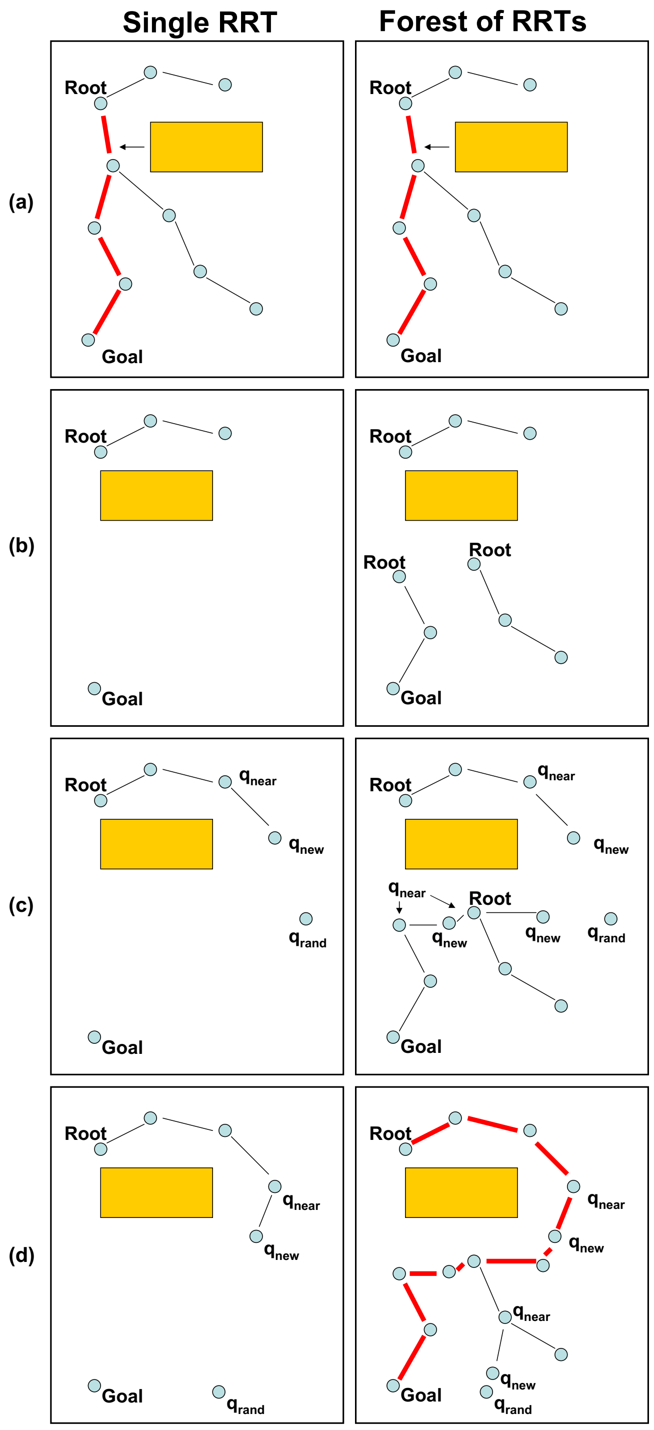

Fig. 2. Single RRT (left) vs. Forest of RRTs (right) in dynamic environments. (a) An obstacle moves left toward the initial tree. (b) Portions of the (inhabited) tree become invalid due to the obstacle. These portions are removed. In the single-RRT case (left), this results in over-pruning, but in the forest-of-RRTs case (right) sub-trees not invalidated by the obstacle become new trees in the forest. (c) The structures are incrementally grown toward a sample configuration. In the forest of RRTs (right), two trees are merged, eliminating one root. (d) The forest structure (right) reaches the goal node in fewer growth steps than the single tree approach (left).

```
$\mathcal{T}$.root($q_{start}$);
repeat
    $q_{rand}$ = RandomSample();
    $q_{near}$ = NearestNeighbor($q_{rand}$, $\mathcal{T}$);
    $q_{new}$ = Extend($q_{near}$, $q_{rand}$);
    if Connect($\mathcal{T}$, $q_{near}$, $q_{new}$) then
        $\mathcal{T}$.addNode($q_{new}$);
        $\mathcal{T}$.addLink($q_{near}$, $q_{new}$);
    end
until $q_{new} \in \mathcal{T} \wedge$ Connect($\mathcal{T}$, $q_{new}$, $q_{goal}$) ;
```

Algorithm 1: Psuedo-code for a Rapidly-exploring Random Tree (RRT) construction algorithm.

B. Rapidly-exploring Random Trees

Generally, RRTs work by growing a search tree in the configuration space until the goal configuration is reached. Algorithm 1 is an outline for RRT construction. Each step in the algorithm begins by selecting a random configuration, q_{rand}. Next, the nearest neighbor to q_{rand} in the tree, q_{near}, is found. The algorithm then attempts to grow the tree from q_{near} toward q_{rand} growing up to a fixed distance and creates a node q_{new}. If the link between q_{near} and q_{new} is valid, then both the link and the node are added to the tree.

There are several motivating factors for using RRTs in planning. One factor is that rapid exploration of C-space occurs naturally. The greater the fraction of C_{free} that is unexplored, the greater that uniform random sampling biases the tree exploration toward its unexplored portions. Additionally, standard RRTs are less susceptible to narrow passage problems than probabilistic roadmap (PRM) planners, since the likelihood of generating a sample that will cause growth through a passage is higher for RRTs than PRMs.

The bi-directional RRT variation grows RRTs rooted at the start and goal toward each other until they can be connected [2]. The basic RRT algorithm has been extended in other ways to improve performance. See, for example [20, 21].

C. Reconfigurable Forests

Forest-based planners maintain multiple trees instead of just a single tree [3, 22]. These planners plant several tree roots throughout C_{free} and grow an RRT from each one, thus generalizing bi-directional RRT. When trees are sufficiently close, they can be merged to create a larger tree.

Li and Shie proposed the Reconfigurable Random Forest (RRF), which supports dynamic environments and is more suitable for multiple-query motion planning. When an obstacle moves, invalid edges are removed and the forest is regrown to fill empty portions [3]. The RRF also includes a forest pruning algorithm designed to reduce the number of nodes in the trees and results in a forest that concisely represents C_{free}.

Our framework combines ideas from RRF and DRRT. We propose a Lazy Reconfiguration Forest (LRF) with the goal of improving efficiency of forests in dynamic environments while maintaining beneficial properties of RRF. By taking a lazy evaluation approach to updating tree-validity, we seek to minimize unnecessary computation. The LRF only checks links along the task paths, similarly to how DRRT focuses on links in the path to the goal, but unlike RRF, which checks moving obstacles against all tree edges. However, removing links in the LRF spawns new trees, as RRF does, instead of destroying entire subtrees as DRRT does.

As we illustrate in Figure 2, DRRT performance can suffer due to over-pruning (Fig. 2b, left) and lengthy path rediscovery (Fig. 2d, left). Over-pruning occurs when large portions of the tree, or equivalently explored regions of C_{free}, are removed due to obstacle motion. This results in more construction steps for path rediscovery since more growth is necessary. LRF seeks to overcome these problems (Fig. 2, right) while maintaining efficiency.

Algorithms that are particularly important in the LRF framework are described in Section IV.

D. Robot Motion and Multiple Tasks

For future applications, we need a general framework for planning that is compatible with a robot that can reason about its state and the environment, request path planning and plan maintenance for multiple tasks, select among multiple tasks, and choose to switch tasks on the fly. Such a robot will require task paths corresponding to alternative tasks that can be switched to, enabling it to detour immediately from the originally-planned or active path as needed. This could include moving toward the goal when no explicit path exists, moving away from an obstacle that moves too close, or moving to a known safe configuration. The LRF framework supports this need by continuously updating the inhabited tree. This update includes setting the current robot state (configuration) followed by growing and merging trees. Updating obtains and re-acquires task path planning solutions as necessary. If the robot discovers its state is disconnected from the inhabited tree, it has the option to create a new tree and grow to reconnect.

Allowing the robot to move somewhat freely along the forest forces a more coupled relationship between robot motion and the planner. The LRF framework proposes several interface conditions:

- The robot must provide the planner with regular, frequent updates about what it knows about the environment and what tasks it wishes to accomplish.
- The planner must provide the robot the next task path segment to execute, with sufficient lookahead agreed upon with the robot.
- When the robot is moving, the root must shift to the next anticipated node that the robot will arrive at on the current active task path. Note that the anticipated state is what we really mean by *current state* used elsewhere in the paper to simplify the discussion.
- Aside from the start state, obstacles, and environment, a path planning task for a given robot may be specified by a goal state predicate or by a weight function. These weights can be directly evaluated or accumulated along a path to decide which path best satisfies a given task.

```
for Each path $p_i$ do
    for Each link $e_j$ along $p_i$ do
        if Not Connect($e_j$.Tree, $e_j$.Begin, $e_j$.End) then
            RemoveLink($e_j$);
        end
    end
    for Each node $n_j$ along $p_i$ do
        if $n_j$ is invalid then
            RemoveNode($n_j$);
        end
    end
end
```

Algorithm 2: The PrunePaths algorithm.

The tree structure of the LRF is exploited in order to maintain efficient evaluation of the task metrics. We emphasize that since RRT is good for exploration in general, it is an ideal candidate for planning multiple paths from a single start point for state-predicates or state-cost task metrics.

We further describe how the LRF framework is designed to support a robot architecture that integrates task planning and task selection with sensing and motion in more detail in Section V.

IV. Lazy Reconfiguration Forests

We now describe a reconfiguration forest that is able to adapt at run-time through maintenance of the existing trees and by adding new growth.

A. Reconfiguration Forest Basics

We call our planning structure a *reconfiguration forest* since its forest structure can frequently change at runtime. We assume that each node in a tree knows its parent and its children. Two basic operations to reconfigure trees in the forest are *tree-split* and *tree-merge*. The tree-split is used when pruning the forest, and the tree-merge is used when growing it.

Our tree-split implementation takes either a single link or a single node and breaks the tree at this point. In the case of a single link, the result is two trees, whereas a split at a node will result in all of the links incident to that node being removed along with the node. Thus, the resulting number of new trees will be equal to the number of children of the node before its removal.

Our tree-merge implementation takes two nodes in two different trees and adds a link between them. One of these nodes will be the child of the other in the resulting tree. If the node that will be the child is not the root of its current tree, root shifts are repeated down the path to that node until it becomes the root. (Shifting the root to one of its children simply reverses their parent-child relationship.) That node is then made a child of the other linked-to node, which is made its parent. The root of the tree of the node made the parent is the root of the resulting tree. If the two nodes linked are sufficiently close, then they are merged to become one node.

```
$q_{rand}$ = RandomSample();
for Each Tree $\mathcal{T}_i$ do
    $q_{near}$ = NearestNeighbor($q_{rand}$, $\mathcal{T}_i$);
    $q_{new}$ = Extend($q_{near}$, $q_{rand}$);
    if Connect($\mathcal{T}_i$, $q_{near}$, $q_{new}$) then
        $\mathcal{T}_i$.addNode($q_{new}$);
        $\mathcal{T}_i$.addLink($q_{near}$, $q_{new}$);
    end
    for Each Tree $T_j$, such that $i \neq j$ do
        $q_{near,j}$ = NearestNeighbor($q_{new}$, $\mathcal{T}_j$);
        if Connect($\mathcal{T}_j$, $q_{near,j}$, $q_{new}$) then
            MergeTrees($T_i$, $T_j$);
        end
    end
end
```

Algorithm 3: The general LRFGrow algorithm.

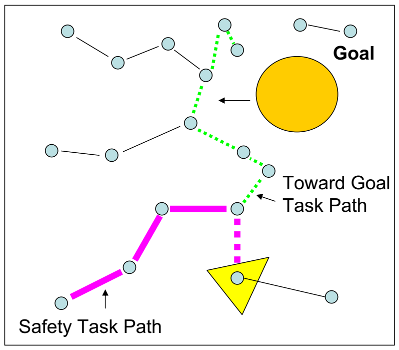

Fig. 3. Multiple Task Paths - Our planner maintains several task paths. Each path allows the robot to have a plan for a different task. The thick line represents a safety task path which is overall farthest away from the obstacle. The dotted line is a task path which would move the robot to the node nearest to the goal, even though no path to the goal exists. The two task paths overlap on the thick dotted line.

B. Forest Maintenance

The maintenance steps allow our forest to automatically adjust to moving obstacles. This involves a sequence of pruning and growing steps. Pseudo-code for these algorithms is given in Algorithm 2 and Algorithm 3, respectively. The pruning steps are necessary to remove portions of the forest discovered to be invalid. The growth steps are necessary to repair task paths and explore the configuration space.

C. Forest Pruning

LRF takes advantage of a convenient property of a tree for pruning. The convenient property is that a path to the root from a node can be found by following the chain of parent pointers in reverse. The pruning algorithm simply walks the current task paths from leaf to root and checks whether each link and node are valid. Invalid links and nodes are removed by using the tree-split operation (see Algorithm 2). Pruning in this manner localizes the work to the subset of the forest that is most important to the task solutions.

Note that this method is dependent upon a path being defined from the root of the inhabited tree to some other node. Therefore, we periodically move the root when the robot is moving. This is further discussed in section IV-E.

D. Forest Growth

Forest growth generalizes bi-directional RRT by attempting to grow multiple pairs of trees into each other. First, a random sample q_{rand} is generated. For each tree T_i in the forest we try to extend T_i to q_{rand} via some node $q_{new,i}$. We check trees T_j where $i \neq j$ to see whether any node in T_j is within the merging threshold distance of $q_{new,i}$. If some node is close enough, then trees T_i and T_j are merged (see Algorithm 3).

Although the runtime of the growth algorithm is theoretically quadratic in the number of trees, in practice the number of trees has remained relatively small once the initial forest is built. We note that there are other growth techniques that the robot can choose to use. These options include extending a single random tree toward q_{rand}, extending some specified tree toward q_{rand}, or growing trees directly toward each other. Other techniques can be created by combining some of these growth methods with a biased random sample generator.

We have experimented with several of these behaviors, but found that it was often sufficient to perform the growth strategy in which each tree grows toward a single random configuration.

E. Moving the Root

One feature in the LRF approach is that the node in the inhabited tree corresponding to the robot's current state is made the root of the inhabited tree. Currently, this is done for conceptual reasons and to simplify implementation. Conceptually, moving the root of the inhabited tree with the current robot state ensures that the state is associated with a node having a unique property within the tree. In implementation, it enables us to reduce various sub-algorithms largely to following parent pointers to the root of a tree, as in Section IV-C. If the planner's knowledge of robot state is updated continuously, then the root can be moved incrementally, requiring no more than a single shift per update.

V. Robot Motion and Tasks

While standard methods for obtaining robot motion from a planned path should apply, the LRF framework was aimed at solving an additional set of tasks. The robot is allowed to adjust its active task path based upon what it believes it should be doing next. To accommodate this behavior, a coupled approach between planning and motion is used in conjunction with multiple definable robot tasks. While these concepts could be used independently of LRFs, the integrated LRF framework provides a relatively simple and effective solution.

A. Coupling Planning and Motion

When the robot's motion and planning are coupled, the robot and planner must work together to complete a task.

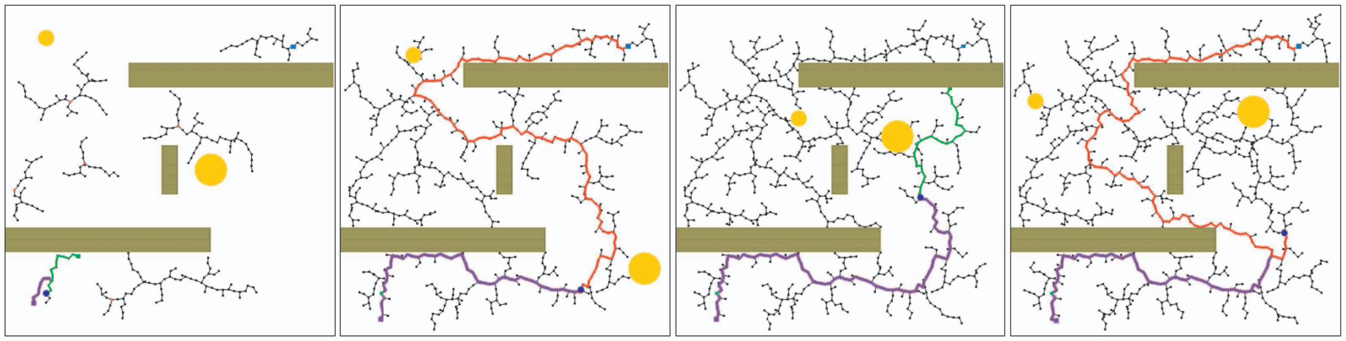

Fig. 4. Snapshots showing a robot moving in a dynamic environment and the LRF it is using. Among the task paths shown, the robot is mostly focusing on the task to move to the goal at upper right. The third frame shows a state where that path has just been invalidated by a moving obstacle and is being re-planned.

```
for Each $i = 0$ to $k$ do
    $q_{rand}$ = RandomSample();
    Forest.addRoot($q_{rand}$);
end
while Robot has not reached $q_{goal}$ do
    Query robot about environment and tasks;
    PrunePaths();           //see Algorithm 2
    if Paths to solve tasks do not exist then
        LRFGrow();          //see Algorithm 3
    end
    Update task paths;
    Send robot next waypoint for the active task;
end
```

Algorithm 4: Top-level algorithm followed by the planner in the LRF framework: initialize the LRF structure and incrementally grow, prune and maintain it according to the robot motion, task path plan requests, and updated obstacle information.

This requires an interface much like a feedback loop between the planner and the robot. The robot specifies its current or lookahead location, what it senses in the environment, and its current task to the planner. Through growth and pruning steps, the planner attempts to adjust to changes in the environment or task, and reports an updated command to the robot. By construction, the robot is associated with the tree it is traversing in the forest, the inhabited tree.

B. Multiple Robot Tasks

The overall performance and behavior of the planning is closely related to the tasks being planned for the robot. We generalize a *task path* to be a path on the robot's tree which optimizes or satisfies a task metric function. A task metric function is a scalar function, $\mathcal{B} : C(t) \to \mathbb{R}$ that is used to assign weights to a node at time t. The task path associated with a given task is determined by either finding the required value among the nodes, the lowest value among the nodes, or the lowest cost accumulated on each path.

In our current, preliminary implementation, we have defined three tasks: going to the goal, going toward the goal, and moving into a safe configuration. The first two utilize the same task metric function, our configuration space metric, ρ. For each node q, we set the task weight to be $\mathcal{B}(q) = \rho(q, q_{goal})$. The going to the goal task path is set only when the metric value reaches 0, signifying that the goal has been found. If no path to the goal exists, the going-toward-the-goal task path is set to find to the lowest value for that metric.

For determining the safe configuration task path, we associate each node with a base cost $\mathcal{B}(q) = \frac{1}{d^k}$ (for example, $k = 1$ or 2) where d is the minimum distance from the obstacles to the robot in the node's configuration. A safe configuration task metric function should penalize paths which travel near obstacles and reward paths to nodes far from all obstacles. Our method periodically traverses the tree while computing accumulated costs from the root to each node. We then select the path from the root to a node that has the minimal accumulated cost. The accumulating function is not limited to simple sums, but could include any function that can be calculated in a very small number of tree traversals.

C. Robot Integration with Planning

We use the proposed framework with robot motion to solve a planning problem with dynamic obstacles. Pseudo-code is given in Algorithm 4. As a preprocessing step, we plant a fixed number of tree roots throughout the environment. Optionally, these roots can be grown during preprocessing so that the forest is mature at start. Once we allow the robot to move, it evaluates the task path metrics and queries the next node(s) from the planner. At each step, the robot determines whether or not any additional growing is necessary. The planner grows and prunes the forest based on information from the robot. This occurs until the robot accomplishes its chosen task.

The robot is responsible for the overall performance because it determines when updates to the planner should occur. We therefore have the robot grow the forest only when no path to the goal exists or when the robot senses danger (with respect to obstacles) above a threshold. Similarly, the task paths are only updated when the forest has been changed or when the obstacles have moved too close to the safe configuration task path.

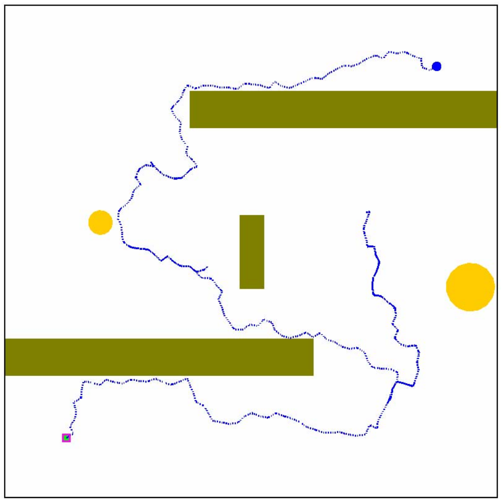

Fig. 5. Path traced by the robot in the example in Figure 4. Obstacles are shown at time of task path completion.

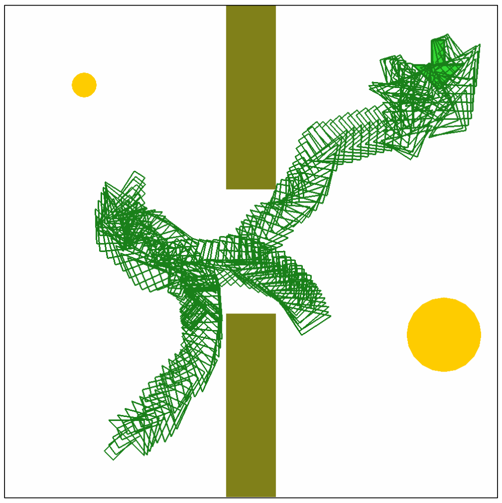

Fig. 6. Path traced by the robot in the example in Figure 1. Obstacles are shown at time of task path completion.

VI. Results and Discussion

A. Results

We have implemented the approach on a Dell M60 Mobile Workstation with a 1.7 GHz Pentium M processor and 2 GB of RAM. The targets of our research include high-degree of freedom (DOF) robots with complex configuration spaces operating in environments with numerous moving entities. However, we have been testing the preliminary implementation on 2DOF and 3DOF robots in simple simulated environments to get a feel for the algorithms and how the tree and forest structures behave.

Figures 4 and 5 show a 2DOF point robot, and Figures 1 and 6 show an arrow-shaped 3DOF robot with two translational and one rotational degree of freedom. The circles and rectangles in the figures represent obstacles. Rectangular obstacles are fixed, and the circular obstacles are allowed to move. Figure 4 shows snapshots of the 2DOF LRF as the robot moves from a start to a goal. Figure 1 is a single snapshot of the 3DOF LRF simply projected into the plane. Figures 5 and 6 show the paths executed, with obstacles after task completion.

In the example in Figure 4 the robot (blue dot) is attempting to to build and maintain paths for three tasks. The first task is to get to a goal location (blue square at upper right). The second task is to move toward the goal location. The third task is to flee from the two moving obstacles (yellow circles). Four snapshot frames are presented to give a feel for the very actively changing forest. In the example the robot and obstacles are moving quickly while the LRF is regularly reconfiguring to adapt to these changes.

Notice that in the first frame there are several trees, each with its own root (red dots). Here, the robot has found a task path to move toward the goal (green line) and a task path to flee from obstacles (purple line), but has not found a task path to go to the goal.

In the second frame, the trees have connected to each other while the robot has been traversing the forest. The connectivity has allowed for a “go to the goal” task path (red line) to be discovered. Note that the “flee from obstacle” task path (purple line) has been updated.

In the third frame we see that the obstacle has caused pruning of the task path to the goal (red line). The ‘move toward the goal” (green line) and “flee from obstacle” (purple line) task paths have also been updated.

In the final frame, growth has lazily reconfigured the tree to allow a new “move to the goal” task path (red line) to be discovered. Additionally, the “flee from obstacle” task path (purple line) has been updated.

The implementation used here had few optimizations. It used a naive nearest neighbor search as well as a simple bounding-box based collision detection scheme. Link queries were performed in a similar manner. Despite this, our algorithm ran at interactive rates. Average step times were 0.75 ms for the 2DOF case, and about 8.5 ms for the 3DOF case. The dominant difference in times between the 2 DOF and 3 DOF examples is associated with the differences in collision detection costs.

B. Discussion

There are several advantages to the proposed LRF approach. It is less likely to have over-pruning problems in dynamic environments than previous work. As a result, it has quick path rediscovery. In fact, the added cost of forest growth can be spread out over many robot steps and is made up for by quicker rediscovery time and fewer necessary re-growth steps. The multiple task paths idea allows the robot to make decisions about what to do in its environment while also being able to represent multiple tasks. While not as simple to implement as a single tree approach, the general forest structure is not difficult to implement.

There are weaknesses to the current LRF implemention as well. As with other RRT-based approaches, there is no guarantee goal configurations will be reached. Generally, such probabilistic algorithms cannot easily detect when no solution exists. The dynamicity of the environment can also inhibit finding a solution even if one exists. It would not be difficult to create a case, such as crossing a road with cars coming in each direction, where either a collision would occur or the robot would never attempt to cross, depending on its behavior choices. Also, the generated paths may be far from optimal. Some online path smoothing may help the task paths, but more fundamental problems can occur that cause inappropriately twisty, "vine-like" trees.

VII. Conclusion

We have proposed a framework for motion planning in a dynamic environment. The approach efficiently uses a forest of RRTs in order to effectively maintain the explored portions of the free configuration space. The LRF structure itself has a wide range of potential applications that require efficient planning and replanning. It can serve as a tree-based multiple-query planner for high-dimensional problems from crowd simulation to control of robotic arms. Coupled behavior between the robot and the planner along with defined task metrics allow it to make decisions about where it needs to go in the environment. It should be noted that it is not necessary to use the multiple task path ideas along with the LRF structure. Rather, LRF makes implementing multiple tasks easy and accommodates them efficiently. Similar ideas could be applied to other planning structures like PRMs, but the time to find and update task paths is likely to be greater. We have highlighted the potential of our approach with results from a preliminary implementation for robots with two and three degrees of freedom.

There are numerous directions for future work. For the LRF framework itself, an pruning algorithm that accounts for expected obstacle trajectories could potentially improve the quality of generated paths by determining areas of a path which may result in a collision in the future. Trajectory information could also be used as part of a task path metric in order to help the robot move when it needs to avoid an obstacle. Heuristics could help to generate nodes for a more balanced tree such as that described by Urmson, et al [23]. Careful re-rooting, pruning, or (re-)growing could further help to improve tree quality, including conciseness [3]. In order to generate smoother paths, a runtime look-ahead could be used to skip nodes which are close to each other. As long as obstacle motion is continuous, it is likely that certain task paths will change in a somewhat coherent manner. This could be exploited to improve the performance of task metric function and task path updates. This framework could work well within a hierarchical framework in a task that requires some global and local planning and re-planning. We are looking into adapting these ideas for kinodynamic planning solutions. Finally, we plan to deploy the algorithm on a multi-processor mobile robot.

References

[1] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, pp. 12(4):566–580, 1996.

[2] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), San Francisco, CA*, April 2000.

[3] T.-Y. Li and Y.-C. Shie, "An incremental approach to motion planning with roadmap management," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2002.

[4] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2006.

[5] J. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[6] 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, 2005.

[7] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[8] J. Reif and M. Sharir, "Motion planning in the presence of moving obstacles," *IEEE Symposium on Foundations of Compuer Science*, October 1985.

[9] A. Stentz, "The focussed D* algorithm for real-time replanning," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[10] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2002.

[11] P. Leven and S. Hutchinson, "Toward real-time path planning in changing environments," *Proceedings of the fourth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2000.

[12] M. Kallmann and M. Mataric, "Motion planning using dynamic roadmaps," *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)*, April 2004.

[13] L. Jaillet and T. Simeon, "A PRM-based motion planning for dynamically changing environments," *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[14] T. Fraichard, "Dynamic trajectory planning with dynamic constraints: A "state-timespace" approach," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, July 1993.

[15] D. Hsu, R. Kindel, J.-C.Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *International Journal of Robotics Research*, 2002.

[16] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.

[17] S. Rodriguez, J.-M. Lien, and N. M. Amato, "Planning motion in completely deformable environments," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2006.

[18] J. van den Berg and M. H. Overmars, "Roadmap-based motion planning in dynamic environments," *IEEE Transactions on Robotics*, 2005.

[19] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2006.

[20] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[21] A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle, "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain," in *Proceedings IEEE International Conference on Robotics and Automation*, 2005.

[22] E. Plaku, K. Bekris, B. Chen, A. Ladd, and L. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 597–608, 2005.

[23] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2003.