Replanning with RRTs

Dave Ferguson, Nidhi Kalra and Anthony Stentz
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania
{dif,nkalra,tony}@cmu.edu

***Abstract*— We present a replanning algorithm for repairing Rapidly-exploring Random Trees when changes are made to the configuration space. Instead of abandoning the current RRT, our algorithm efficiently removes just the newly-invalid parts and maintains the rest. It then grows the resulting tree until a new solution is found. We use this algorithm to create a probabilistic analog to the widely-used D* family of deterministic algorithms, and demonstrate its effectiveness in a multirobot planning domain.**

I. Introduction

In the field of robotic path planning, probabilistic sampling-based approaches have recently become very popular. In particular, Rapidly-exploring Random Trees (RRTs) have been shown to be effective for solving single-shot path planning problems in complex configuration spaces [1], [2]. By combining random sampling of the configuration space with biased sampling around the goal configuration, RRTs efficiently provide solutions to problems involving vast, high-dimensional configuration spaces that would be intractable using deterministic approaches. As a result, they have been widely-used in robotics [3], [4], [5], [6]. See Fig. 1 for an example of an RRT constructed for a two-dimensional path planning problem.

It is often the case in real world scenarios that the initial information available concerning the environment is incomplete or that the environment itself is dynamic. In these cases, the initial solution may become invalid as new information is received, for example, through an onboard sensor. When this occurs, typically the current RRT is abandoned and a new RRT is grown from scratch. This can be a very time-consuming operation, particularly if the planning problem is complex. On the other hand, in the deterministic planning community there exist replanning algorithms that are able to efficiently repair the previous solution when such changes occur without needing to replan from scratch.

In this paper, we present a replanning algorithm for repairing RRTs when new information concerning the configuration space is received. Instead of abandoning the current RRT entirely, our approach mimics deterministic replanning algorithms by efficiently removing just the newly-invalid parts and maintaining the rest. It then grows the remaining tree until a new solution is found. The resulting algorithm, Dynamic Rapidly-exploring Random Trees, is a probabilistic analog to the widely-used D* family of deterministic replanning algorithms [7], [8].

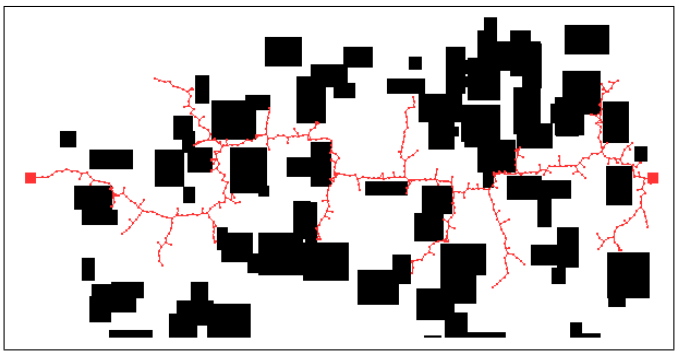


Fig. 1. An RRT created for planning a path from one end to the other of a binary environment. Obstacles are shown in black. The start and goal locations are shown as large red squares, with nodes of the tree small squares.

We begin by describing the basic RRT algorithm and a number of extensions that have been made to this algorithm to facilitate both initial planning and replanning. We then introduce Dynamic Rapidly-exploring Random Trees and how they can be used for mobile robot navigation in unknown or dynamic environments. We go on to present a number of results from a multirobot constrained exploration domain and conclude with discussions and extensions.

II. Rapidly-exploring Random Trees

The essential RRT algorithm is outlined in Fig. 2. Beginning with the initial robot configuration as the root node, it incrementally grows a tree until the tree reaches the goal configuration. To grow the tree, (lines 3-8 in function *GrowRRT*), first a target configuration q_{target} is randomly selected from the configuration space using the function *ChooseTarget*. Then, a *NearestNeighbor* function selects the node $q_{nearest}$ in the tree closest to q_{target}. Finally, a new node q_{new} is created in an *Extend* function by growing the tree some distance from $q_{nearest}$ towards q_{target}. If extending the tree towards q_{target} requires growing through an obstacle, no extension occurs. This process is repeated until the tree grows to within some user-defined threshold of the goal (line 3). A very nice property that follows from this method of construction is that the tree growth is strongly biased towards unexplored areas of the configuration space. Consequently, exploration occurs very quickly.

Several extensions to this basic algorithm tend to improve the speed of the search. Firstly, the basic algorithm uniformly

In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May, 2006.

```
InitRRT()
 1  T.add(q_start);

GrowRRT()
 2  q_new = q_start;
 3  while (Distance(q_new, q_goal) > distance-threshold)
 4    q_target = ChooseTarget();
 5    q_nearest = NearestNeighbor(q_target);
 6    q_new = Extend(q_nearest, q_target);
 7    if (q_new ≠null)
 8      T.add(q_new);

ChooseTarget()
 9  p = RandomReal([0.0, 1.0]);
10  i = RandomInt([1, num-waypoints]);
11  if (p < goal-sampling-prob)
12    return q_goal;
13  else if (p < goal-sampling-prob + waypoint-prob)
14    return WaypointCache[i];
15  else
16    return RandomNode();
```

Fig. 2. **The (Extended) RRT Algorithm.**

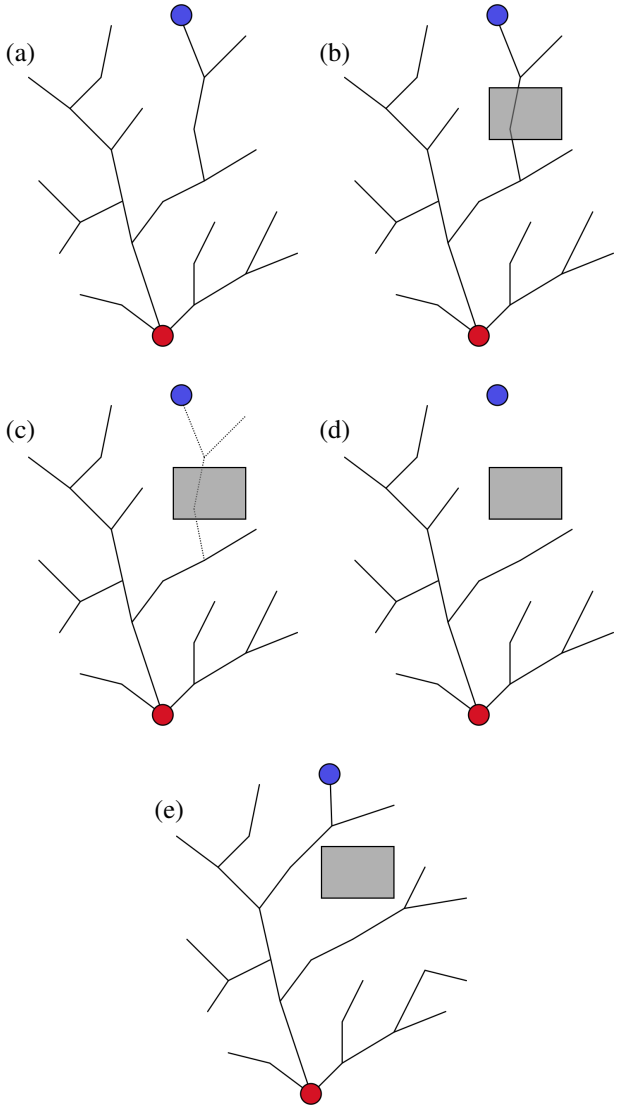


Fig. 3. Replanning with RRTs. (a) An initial RRT generated from a start position (red) to a goal position (blue). (b) A new obstacle is added to the configuration space. (c) Parts of the previous tree that are invalidated by the new obstacle are marked. (d) The tree is trimmed: invalid parts are removed. (e) The trimmed tree is grown until a new solution is generated.

grows the tree by always selecting n_{target} randomly. However, we can increase efficiency by biasing the search towards the goal: in the *ChooseTarget* function, we let q_{target} be the goal with probability p and choose it randomly with probability $1-p$. As p increases, the RRT behaves increasingly like best-first search.

In many domains where plans are executed by robots, planning and execution are interleaved: the robot creates a tree, executes the returned path for some number of steps or until it is no longer valid, then grows a new tree, and so on until the goal is reached. The Execution-extended RRT (ERRT) algorithm is a further extension that is useful in these scenarios because it reuses information from previous planning episodes when generating new trees [3]. After a plan is returned, some of its nodes are stored in a waypoint cache. In future searches, the node q_{target} is set to one of these nodes with some probability (lines 11, 14-15), thus biasing the growth towards previously successful solutions. However, even the ERRT algorithm rebuilds an RRT from scratch every time new information invalidates the current solution, regardless of how much of the solution is affected. This usually results in far more work than is necessary for generating a new solution.

On the other hand, researchers have developed methods of reusing as much previous computation as possible when performing multiple-query path planning with Probabilistic Roadmaps (PRMs) [9], [10]. In particular, the Reconfigurable Random Forest (RRF) approach of Li and Shie creates a roadmap of the environment using several different RRTs, each rooted at different locations [9]. Periodically, the individual RRTs are checked to see if they can be connected together, as in the RRT-Connect algorithm [11]. When changes are made to the configuration space, the newly-invalid edges in the forest are removed and new trees are formed from the branches that were connected to these edges. This approach effectively updates roadmaps of the environment and is useful for multiple-query path planning in changing environments.

In the following section we describe a related extension of the RRT algorithm that allows for efficient repair of the tree when changes are made to the configuration space. As with RRFs, this approach prunes sections of the tree that are no longer valid. However, it maintains only a single tree rather than an entire forest and thus it is particularly suited to single-shot path planning problems. The resulting approach is simple to implement and can be much more efficient than replanning

```
RegrowRRT()
 1  TrimRRT();
 2  GrowRRT();

TrimRRT()
 3  S = ∅; i = 1;
 4  while (i < T.size())
 5    q_i = T.node(i); q_p = Parent(q_i);
 6    if (q_p.flag = INVALID)
 7      q_i.flag = INVALID;
 8    if (q_i.flag ≠ INVALID)
 9      S = S ∪ {q_i};
10    i = i + 1;
11  T = CreateTreeFromNodes(S);

InvalidateNodes(obstacle)
12  E = FindAffectedEdges(obstacle);
13  for each edge e ∈ E
14    q_e = ChildEndpointNode(e);
15    q_e.flag = INVALID;
```

Fig. 4. **The Dynamic RRT Algorithm.**

from scratch when small changes are made to the configuration space. Further, as we will show, it can be used to develop an incremental replanning algorithm for mobile robot navigation.

III. Dynamic Rapidly-exploring Random Trees

Deterministic replanning algorithms such as D* efficiently repair previous solutions when changes occur in the environment [7], [8]. They do this by determining which parts of the solution are still valid and which parts need to be recomputed. We can use this same basic idea to improve the efficiency of replanning with probabilistic algorithms such as RRTs.

The general process is illustrated in Fig. 3. We begin with an RRT generated from an initial configuration to a goal configuration (Fig. 3(a)). When changes occur to the configuration space (e.g. through receiving new information), we mark all the parts of the RRT that are invalidated by these changes (Fig. 3(b) and (c)). We then trim the tree to remove all these invalid parts (Fig. 3(d)). At this point, all the nodes and edges remaining in the tree are guaranteed to be valid, but the tree may no longer reach the goal. Finally, we grow the tree out until the goal is reached once more (Fig. 3(e)).

We call this approach *Dynamic Rapidly-exploring Random Trees* (DRRTs). Pseudocode for trimming and regrowing the tree is presented in Fig. 4. When an obstacle is added to the configuration space, first the edges in the current tree that intersect this obstacle are found (line 12). Each of these edges will have two endpoint nodes in the tree: one will be the parent of the other in the RRT. In other words, one of these nodes (the parent) will have added the other (the child) to the tree through an *Extend* operation. The child endpoint node of each edge is then marked as invalid (line 15).

```
Main()
 1  q_start = q_goal; q_goal = q_robot;
 2  InitRRT();
 3  GrowRRT();
 4  while (q_goal ≠ q_start)
 5    q_goal = Parent(q_goal);
 6    Move to q_goal and check for new obstacles;
 7    if any new obstacles are observed
 8      for each new obstacle o
 9        InvalidateNodes(o);
10      if solution path contains an invalid node
11        RegrowRRT();
```

Fig. 5. **Using DRRTs for Mobile Robot Navigation.**

After all the child endpoint nodes of the affected edges have been marked, the solution path is checked for invalid nodes. If any are found, the RRT needs to be regrown. This involves trimming the tree and growing the trimmed tree out to the goal (*RegrowRRT*, lines 1 - 2). Trimming the tree involves stepping through the RRT in the order in which nodes were added and marking all child nodes as invalid whose parent nodes are invalid. This effectively breaks off branches where they directly collide with new obstacles and removes all nodes on these branches.

Once the tree has been trimmed, it can be grown out to the goal. This can be performed in exactly the same manner as the basic RRT algorithm for initial construction. However, depending on how the configuration space has changed, it may be more efficient to focus the growth towards areas that have been affected, as discussed in the following section.

DRRTs can be used to provide a probabilistic analog to D* for mobile robot navigation in unknown or dynamic environments. To do this, we first reverse the direction of growth of the RRT so that it grows from the desired configuration towards the current robot configuration. This is a common modification made by deterministic replanning algorithms and allows us to reuse the previous tree when the robot configuration changes (as the robot moves through the environment) and new obstacles appear. Otherwise, the root of the tree would constantly be changing and so the entire tree would need to be regrown. Further, since observations are typically being made in the vicinity of the robot (through onboard sensors), this modification allows us to maintain more of the previous tree during repair, as only the tips of the tree will be affected by newly-observed obstacles.

Fig. 5 presents pseudocode for using the DRRT as a real-time replanning algorithm for mobile robot navigation. Note that this algorithm can be used for the navigation of either a single robot or a team of robots.

IV. Experiments and Results

The motivation behind this work was efficient multirobot path planning. In particular, we are interested in the problem of constrained exploration [12], where a team of robots is

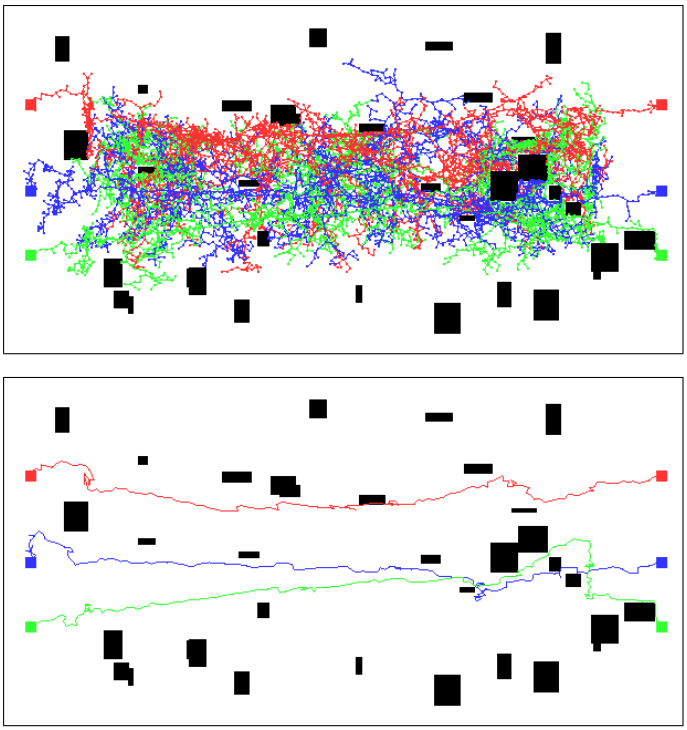


Fig. 6. Coordinated Path Planning. A path is planned for three robots from one side to the other of an environment, while maintaining line-of-sight communication constraints among the team. (top) The RRT. In order to show the 6D RRT in only two dimensions, it has been split into three 2D trees. Each 2D tree encodes all the information about a single robot from the 6D tree, including edges between parent and child nodes. The red tree corresponds to the positions of robot 1 in the 6D RRT, the blue to robot 2, and the green to robot 3. The black regions depict areas through which communication cannot be made. (bottom) The corresponding solution path for each robot (before smoothing).

tasked with exploring an environment while abiding by some constraints on their paths. For instance, imagine a known, static environment containing areas through which no communication is possible (due to eavesdropping adversaries or environmental characteristics), and all robots must remain in line-of-sight communication with the rest of the team at all times. Planning a path for the group in such a scenario becomes a joint planning problem and is exponential in the number of robots.

As the number of robots in the team increases, deterministic algorithms for planning such as A* or Dijkstra's simply cannot cope with the size of the corresponding state space. Randomized approaches such as RRTs, on the other hand, are a good choice for solving this problem since they are not crippled by its high dimensionality. For example, Fig. 6 shows an RRT and the corresponding set of paths for 3 robots moving through a 300×600 environment containing obstacles through which communication cannot be made. The size of the state space for this problem was 6×10^{15}, however the RRT was able to find a path in under a second[1]. In fact, even for large teams, RRTs are able to generate paths in a reasonable amount of time (see Fig. 8).

[1]Using a 1.5 GHz Powerbook G4.

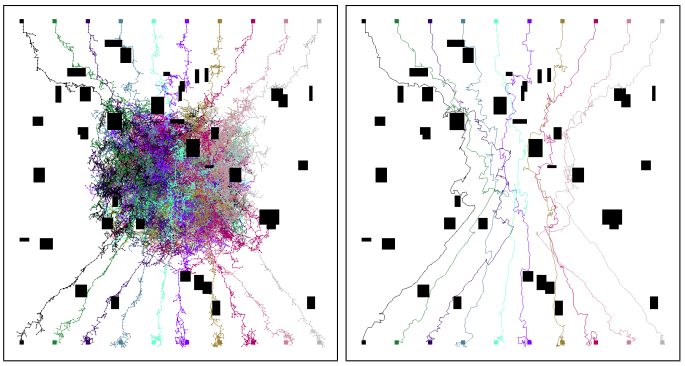


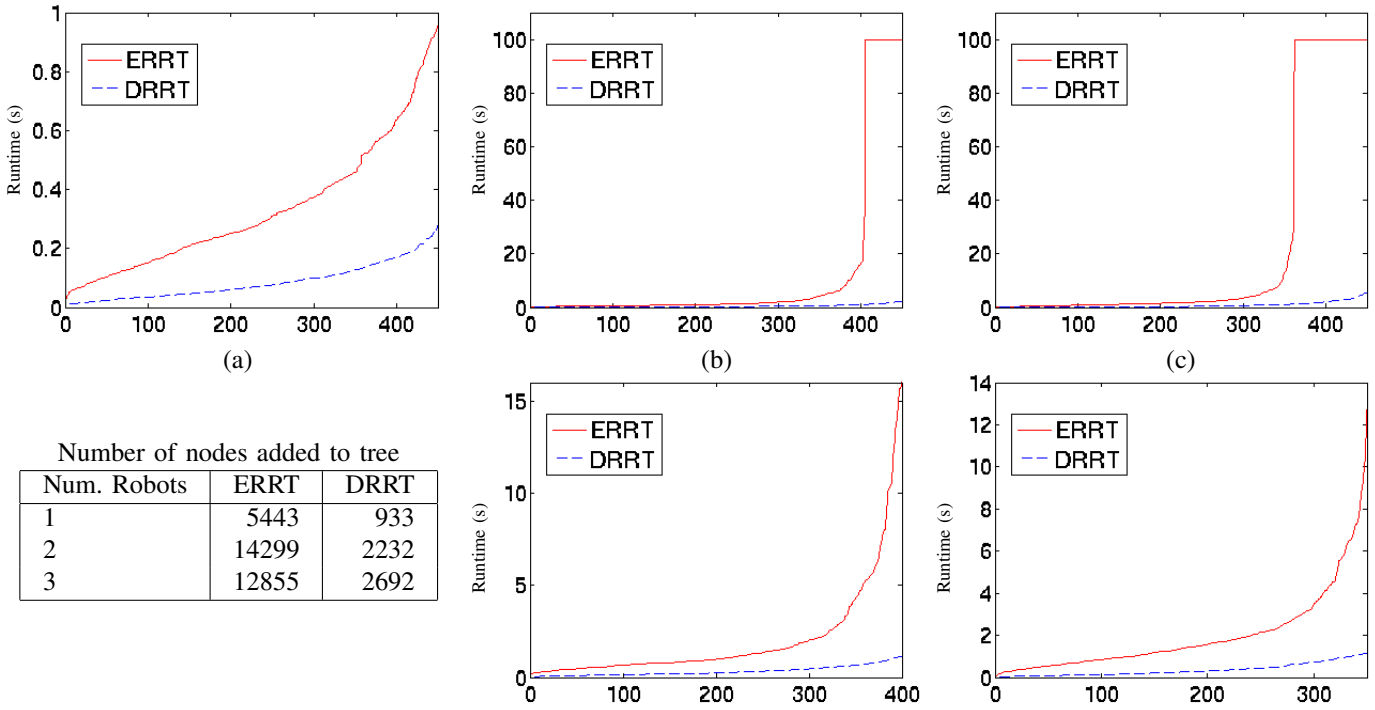
Fig. 8. Coordinated path planning with 10 robots. On the left is an example RRT and on the right are the associated robot paths (before smoothing). The state space for this problem had a size of 10^{54} and the RRT generated a path in 13 seconds. See Fig. 6 for an explanation of the domain and how the RRT is displayed.

Now, to efficiently solve such problems in unknown or dynamic environments where we may have to update the solution during execution, we need dynamic randomized approaches such as the DRRT algorithm.

To test the performance of DRRTs against current approaches, we ran a number of experiments simulating a small team of robots moving through an environment while maintaining line-of-sight communication across the team. As the robots moved through the environment, they received new information concerning the traversability of areas within some sensor field of view. If obstacles were encountered along their solution paths, a new RRT was generated that took into account the new obstacles. For comparison, this RRT was generated using both the ERRT approach and the DRRT approach presented in Fig. 5.

To efficiently determine which edges in the RRT collided with a new obstacle (for the DRRT approach), for each (x, y) position in our discretized map we kept track of the nodes in the RRT that had one of the robots located at that position. Then, when a new obstacle appeared at location (x', y'), we could quickly find any nodes that were invalidated. To find edges that were invalidated, we simply carved a 2D circle out with center (x', y') and radius equal to the maximum length of any edge in the tree, then checked if any nodes of the tree resided within this circle *and* had an edge that intersected the new obstacle. Because the edges in our tree were typically quite small (on the order of 5 map cells in distance for each robot), this was very fast. The overhead of maintaining the list of nodes for each (x, y) position and determining which are affected when new obstacles are observed are included in our runtime comparisons.

We also focussed re-growth of the DRRT to areas that had been affected by changes to the configuration space. Specifically, with some probability we chose a random sample point in the vicinity of the recently-affected area of the configuration space. This has the effect of focussing growth of the tree in the region just trimmed, which is also close to the current configuration of the team. In our results, this



Number of nodes added to tree

Num. Robots	ERRT	DRRT
1	5443	933
2	14299	2232
3	12855	2692

Fig. 7. Runtimes for the ERRT (red/solid) and DRRT (blue/dashed) approaches in our three constrained exploration scenarios. For each approach, the runs are ordered in terms of runtime. (a) Shown above are results for the single robot case. Shown below are the average total number of nodes added to the respective trees over the course of each traverse, for each experiment. (b) Results for the two-robot team. The bottom of the two graphs shows the same results but for only the quickest 400 runs. (c) Results for the three-robot team. The bottom graph shows the quickest 350 runs.

probability was set equal to the probability used by the ERRT approach to select a previous waypoint, which was 0.4. Each approach selected a sample very close to q_{goal} with probability 0.1. We found these parameters gave the best performance for both the ERRT and DRRT approaches.

Results for teams ranging from a single robot to three robots are shown in Fig. 7. For each case, the task was to navigate across a 300×600 environment containing randomly generated obstacles. For the single robot case, as the robot traversed the environment, new obstacles appeared at random: with each step the robot took along its path there was some chance a randomly placed obstacle would appear within the robot's sensor range (25 cells), potentially requiring that the robot repair its previous solution path. We repeated the process 5 times each for 100 different environments.

To ensure each approach was fairly compared, we had them operate over exactly the same traverses. To do this, we ran the DRRT approach first and recorded the path taken and new obstacles encountered, and then had the ERRT approach operate over the same path and obstacles. Since the agent did not move very far in between replanning episodes, this did not place the ERRT approach at a disadvantage. We also used kd-trees to efficiently compute nearest neighbors in both approaches.

For the multirobot cases, the situation was similar, except that the robots had to maintain line-of-sight communication between team members. The team started with a map specifying the areas of the environment through which communication was not possible. These areas acted as line-of-sight connectivity obstacles: if a straight-line connecting two robots passed through one of these areas, the robots were not within direct line-of-sight communication of each other. Further, as the robots moved, they observed navigation obstacles that they had to avoid but which did not affect communication. These multirobot experiments were designed to simulate the constrained exploration task, where the areas of the environment through which communication is not possible (e.g. adversary areas) are known apriori, but the nature of the environment (e.g. in terms of terrain or navigability) is not.

Navigation obstacles also appeared at random in the multi-robot experiments: for the two-robot case an obstacle would appear within each robot's sensor range with probability 0.4 and for the three-robot case an obstacle would appear in front of each robot's sensor range with probability 0.15. Again, one hundred different random environments were used with each traversed 5 times. Because of their use of randomness, the same problem may take RRTs vastly different amounts of computation depending on the random values generated. Thus, in order to see clearly the relative performance of the algorithms, we have shown the most efficient 90% of the 500 runs for each approach (and, for the same reason, we have limited the runtime of each run to 100 seconds[2]). The runtimes reported are for a 1.5 GHz Powerbook G4.

[2]We also limited the number of nodes added to any single tree to be 30000. If no solution was found within this limit we marked the run as a failure and set the time to be the default maximum of 100 seconds.

On average, DRRTs outperformed ERRTs in each case by a factor of 5, in both the average number of nodes added to the tree during each traverse and in the computation time required. The average time spent per replanning episode for the three robot team was 146 milliseconds for ERRTs and 27 milliseconds for DRRTs.

V. Discussion

We have presented Dynamic Rapidly-exploring Random Trees (DRRTs), a replanning algorithm for repairing Rapidly-exploring Random Trees when changes are made to the configuration space. Our algorithm efficiently removes just the newly-invalid parts of the tree and regrows a solution from what remains. We have demonstrated its effectiveness in a single robot navigation domain and a multirobot constrained exploration domain.

The basic DRRT approach can be thought of as a single-tree variant of the RRF approach described in Section II. However, by only maintaining a single tree, DRRTs offer a number of advantages in regards to the problem of real-time multirobot planning.

Firstly, DRRTs are simple to implement and involve very few parameters. Because there is only one tree, we need not worry about which tree to grow next, when to try to connect different trees together, how to trim the different trees, and so on. DRRTs clearly extend basic single-tree RRTs to partially-known or dynamic environments.

Secondly, when edges are invalidated in the tree, DRRTs remove the entire affected branch rather than using the branch to create a new tree. This can be beneficial in navigation scenarios as often either these branches are very small (since changes are typically taking place in the vicinity of the robot(s) and so near the leaves of the tree), or the branches extend out into areas of the configuration space already passed by the robot(s) and no longer useful for planning. Further, by using biased sampling to focus towards recently invalidated edges, parts of the pruned branches that may have been useful will be regrown quickly by the DRRT.

As we have shown, DRRTs can be used to provide a probabilistic analog of the widely-used D* family of deterministic replanning algorithms. To our knowledge, this is the first effort to incorporate the principles behind D* into probabilistic algorithms. Our results demonstrate the usefulness of this combination; consequently, we believe DRRTs are a good substitute for D* in high-dimensional problems.

There are a number of directions for future research. As discussed in Section III, for DRRTs to be effective for mobile robot navigation, we need to direct the tree growth from the goal to the position of the robot(s). However, if we want to model the complex kinematics of the robot(s), then this may not be possible. In deterministic planning, usually this is performed by combining a global backwards-searching path planner such as D* with a local forwards-searching planner that incorporates the kinematic constraints of the vehicle (e.g. as in [13]). Using DRRTs, we could address this issue by combining the global DRRT grown from the goal to the position of the robot(s) with a local tree grown outwards from the position of the robot(s) that incorporates all the kinematic constraints of the vehicle(s). Existing research in the use of bidirectional RRTs would certainly be applicable here [11].

We are also looking at how we can improve the quality of the solutions obtained using RRTs. Specifically, we are interested in how heuristics can be used to bias the growth of the tree in order to arrive at shorter or less-expensive solutions, as looked at by Urmson and Simmons [14]. Finally, we plan to implement the DRRT approach on a team of three autonomous John Deere E-gator robots that we are using for constrained exploration.

VI. Acknowledgements

This work was sponsored by the U.S. Army Research Laboratory, under contract "Robotics Collaborative Technology Alliance" (contract number DAAD19-01-2-0012). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsements of the U.S. Government. Dave Ferguson is supported in part by a National Science Foundation Graduate Research Fellowship.

References

[1] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[2] ——, "Rapidly-exploring Random Trees: Progress and prospects," *Algorithmic and Computational Robotics: New Directions*, pp. 293–308, 2001.

[3] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[4] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Motion planning for humanoid robots," in *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2003.

[5] J. Kim and J. Ostrowski, "Motion planning of aerial robots using Rapidly-exploring Random Trees with dynamic constraints," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[6] G. Oriolo, M. Vendittelli, L. Freda, and G. Troso, "The SRT Method: Randomized strategies for exploration," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[7] A. Stentz, "The Focussed D* Algorithm for Real-Time Replanning," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

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

[9] T. Li and Y. Shie, "An Incremental Learning Approach to Motion Planning with Roadmap Management," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[10] M. Kallmann and M. Matarić, "Motion Planning using Dynamic Roadmaps," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[11] J. Kuffner and S. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[12] N. Kalra, D. Ferguson, and A. Stentz, "Constrained Exploration for Studies in Multirobot Coordination," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[13] A. Stentz and M. Hebert, "A complete navigation system for goal acquisition in unknown environments," *Autonomous Robots*, vol. 2, no. 2, pp. 127–145, 1995.

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