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

Multipartite RRTs for Rapid Replanning in Dynamic Environments

Matt Zucker[1] James Kuffner[1] Michael Branicky[2]

[1]*The Robotics Institute*
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA, 15213, USA
{mzucker,kuffner}@cs.cmu.edu

[2]*EECS Department*
Case Western Reserve University
10900 Euclid Ave., Glennan 517B
Cleveland, OH 44106-7221, USA
mb@case.edu

***Abstract*— The Rapidly-exploring Random Tree (RRT) algorithm has found widespread use in the field of robot motion planning because it provides a single-shot, probabilistically complete planning method which generalizes well to a variety of problem domains. We present the Multipartite RRT (MP-RRT), an RRT variant which supports planning in unknown or dynamic environments. By purposefully biasing the sampling distribution and re-using branches from previous planning iterations, MP-RRT combines the strengths of existing adaptations of RRT for dynamic motion planning. Experimental results show MP-RRT to be very effective for planning in dynamic environments with unknown moving obstacles, replanning in high-dimensional configuration spaces, and replanning for systems with spacetime constraints.**

I. Introduction

Motion planning in dynamic or uncertain environments is an important problem in the fields of manipulation and mobile robot navigation. Because of the need for highly responsive algorithms, prior research on dynamic planning has focused on re-using information from previous queries across a series of planning iterations.

Dynamic programming based approaches such as D* [1] and D*-Lite [2] are dynamic extensions of the A* algorithm which perform minimal modifications to a previous search tree in order to maintain an optimal path. However, these algorithms are typically limited to low-dimensional search spaces. Recently, probabilistic approaches have gained popularity due to their ability to trade optimality for fast runtimes when searching high-dimensional configuration spaces. In the context of dynamic environments, examples include dynamic roadmap approaches [3], [4] that maintain a connected graph in the free configuration space of the robot ($\mathcal{C}_{free}$) which is pruned or validated against obstacle location updates. Other examples include the RAMP algorithm [5] which maintains sets of free paths using genetic algorithms, and the decomposition-based strategy of [6] which uses workspace based-heuristics to guide the search of $\mathcal{C}_{free}$. Despite the broad array of approaches to dynamic motion planning, there is still no generally applicable solution to the problem. Drawbacks of current approaches include difficulty planning in high-dimensional spaces, planning with nonholonomic or differential constraints, planning among moving obstacles, and efficiently handling changes to the initial and goal states of a query.

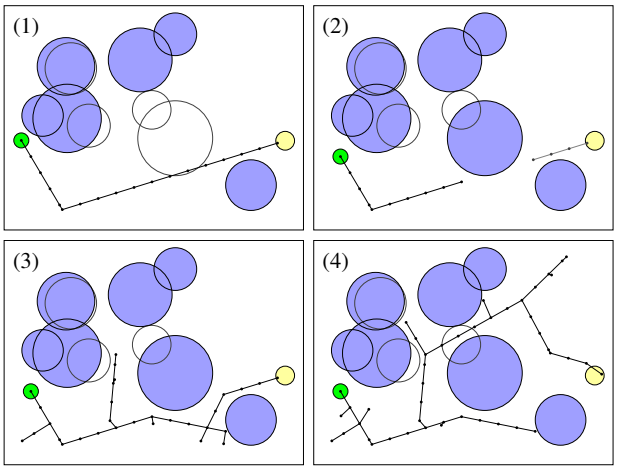

Fig. 1. Re-planning for a translating point robot R among unknown obstacles. 1) The previous plan. R is green circle, left, known obstacles are solid blue, unknown (occluded) obstacles are gray outlines, goal region is yellow circle, right. 2) R steps along path, observing a new obstacle. MP-RRT removes invalid path segments, saving disconnected subtree for later re-use (gray, right). 3) After a small amount of expansion, MP-RRT re-uses the disconnected subtree from (2). 4) Tree produced by DRRT after deleting both the invalid nodes and their descendents in (2).

In this paper, we present a variant of the Rapidly-exploring Random Tree (RRT) algorithm [7], a probabilistic motion planner which has found widespread use in the field of robot motion planning. Our Multipartite RRT (MP-RRT) algorithm leverages the strengths of several existing dynamic RRT variants, along with an opportunistic strategy for reusing information from previous queries. We present experimental results that show MP-RRT to be a responsive planning algorithm which is well-suited for searching high-dimensional configuration spaces among unknown or moving obstacles, and which overcomes some drawbacks of previous approaches.

II. Background

The RRT algorithm attempts to find a continuous path through $\mathcal{C}_{free}$ from the initial state of the system q_{init} to

some state $q_{goal} \in G$, a set of valid goal states. Paths indexed by time will be referred to as *trajectories*. The original RRT algorithm begins by initializing the search tree T to its root q_{init}. Then the following steps are repeated until a valid path is found, or the progress reaches another termination criterion (e.g., a specified time or memory limit is exceeded or the algorithm has reached a maximum number of iterations): A configuration $q_{new} \in \mathcal{C}$ is selected at random. Then its nearest neighbor in the tree $q_{nearest}$ is selected according to a scalar metric $\rho(q_1, q_2) \mapsto \Re$ defined on $\mathcal{C}$. Next, an edge e is generated which extends from $q_{nearest}$ towards q_{new}; if e lies in $\mathcal{C}_{free}$, then the terminal point of the edge (often q_{new} itself) is inserted into T as a child of $q_{nearest}$.

Assuming uniform sampling of $\mathcal{C}$, the RRT algorithm is biased to quickly construct paths into unoccupied Voronoi regions of the vertices of T leading to a rapid exploration of $\mathcal{C}_{free}$ [8]. One simple enhancement to this algorithm entails adding a "goal bias" to the search by choosing some goal configuration $q_{goal} \in G$ as the sampled state q_{new} with small probability p_{goal} for each iteration. In practice, this yields much faster convergence than simply waiting for a random sampler to select states in G.

A. The dynamic planning problem

Dynamic planning is similar to static motion planning, with the modification that planning steps will be alternated with system and world update steps. Applications for dynamic planning include navigation or manipulation planning in a world with unknown or moving obstacles. Between planning iterations, several possible events may have occurred simultaneously: 1) The robot may have moved, and in doing so may have discovered new obstacles to avoid. 2) The goal may have moved. 3) Previously sensed obstacles may have moved independently of any action by the robot.

More formally, there will be multiple planning iterations to find a path through $\mathcal{C}_{free}$ from q_{init} to $q_{goal} \in G$; however we do not require that the state q_{init} and the sets $\mathcal{C}_{free}$ and G remain fixed between iterations.

One way to approach the dynamic planning problem is to simply re-run the original RRT algorithm for every planning iteration; however, the world typically exhibits temporal coherence which this naïve approach fails to exploit. While all the obstacles and the goal region may have moved, it is quite possible that the changes are small enough that the plan from the previous iteration is still valid. In this case, the extra computation of constructing a new plan from scratch is inefficient.

The intuition that information from previous planning iterations should be preserved led to the development of three modifications to the RRT algorithm: ERRT [9], DRRT [10], and RRF [11]. In ERRT, when a successful plan is generated, the states along the path from q_{init} to q_{goal} are inserted into a *waypoint cache* which is then used to bias random selection for the next planning iteration. At the start of the next iteration, the previous RRT is discarded, but q_{new} will be selected from the waypoint cache with probability $p_{waypoint}$. In practice, ERRT performs much better than naïve iterated RRT in the face of small changes to $\mathcal{C}_{free}$ and G because states which were used in a previously successful plan are likely to be good intermediate states in the current plan.

DRRT takes a different approach to exploiting temporal coherence; whereas ERRT biases selection towards states from the previous path, DRRT attempts to reuse branches of the previous RRT which lie in $\mathcal{C}_{free}$ and remain connected to the root. The implementation of DRRT imposes a substantial overhead at the start of planning: before the previous RRT can be reused, it must be validated against any updates to $\mathcal{C}_{free}$. In practice, though, time spent on validation (i.e., collision checking) enables the reuse of very large subtrees, up to the size of the entire previous RRT.

Growing multiple trees backwards from a set of goal configurations has been shown to be efficient in the context of one-shot planning for redundant manipulators [12]. Reconfigurable Random Forests (RRF) build a forest of disconnected RRTs for the purposes of multiple-query planning, with the aim of incrementally building a data structure that eventually captures the entire topology of $\mathcal{C}_{free}$ [11]. Our MP-RRT algorithm also uses a forest of disconnected RRTs for multiple-query planning. However, we decided to instead focus the MP-RRT algorithm on a single start-goal pair, not only because it yields a more responsive planner, but because a tree structure that spans multiply-connected regions of $\mathcal{C}_{free}$ tends to yield significantly suboptimal paths which are in general not homotopic to the optimal solution.

B. Point-to-point connections

Consider a scenario involving a mobile robot with a limited sensing horizon, and in which G does not change substantially between planning iterations. In this case, it is advantageous for DRRT to plan in reverse, building a tree rooted at $q_{goal} \in G$ which connects to the current configuration of the robot $q_{current}$. In such a scenario, modifications to $\mathcal{C}_{free}$ will typically take place in the vicinity of the robot configuration; therefore, a tree rooted at the goal state will be able to reuse much more information from previous planning iterations than one rooted at the current robot configuration.

For systems with momentum or nonholonomic constraints, backward planning introduces an additional complication: whereas ERRT and simple RRT only require that we generate an edge from $q_{nearest}$ towards q_{new}, planning backwards requires the ability to generate an edge exactly connecting some node in the tree to $q_{current}$.

When planning under differential constraints (i.e., generating trajectories for a nonholonomic robot), point-to-point connections may require solving a boundary value problem to generate the correct controls for a trajectory segment exactly interpolating the endpoints. If no inverse dynamics model for the system can be computed, the aforementioned algorithms are difficult to use; even if such a model can be computed, doing so may impose significant overhead on the planning model.

Like DRRT in the reverse planning scenario, our MP-RRT algorithm requires the ability to make point-to-point

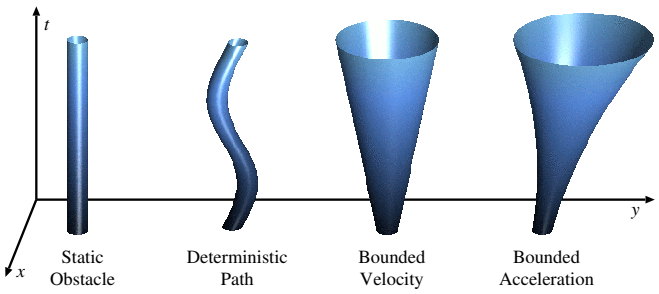

Fig. 2. Obstacles as spacetime volumes

connections; however, unlike the algorithms above, we anticipate that such connections may require significantly more computation than normal RRT extensions, and therefore our algorithm attempts to make point-to-point connections in an opportunistic and principled fashion. We believe guiding the planning process to be "choosy" about point-to-point connections is a novel contribution that makes MP-RRT more appropriate for certain classes of planning problems than approaches such as probabilistic roadmaps (PRM) or RRF.

C. Moving obstacles

Consider the scenario in which the obstacles themselves are moving through the workspace. One approach to solving the planning problem is to generate paths in $\mathcal{CT} = \mathcal{C} \times [0, \infty)$, where the additional dimension indexes time [13]. Then, a workspace collision checker can use the time component $t \in \mathcal{T}$ of a state to check against predetermined trajectories or worst-case obstacle motion assumptions (see figure 2). This effectively represents obstacles as spacetime volumes in the workspace. In this formulation, updates between planning iterations may alter not only estimates of the current position of an obstacle, but estimates of its future trajectory as well.

Although convenient, the $\mathcal{CT}$ formulation places some interesting restrictions on the class of planning algorithms which can be used. In particular, multi-tree approaches such as Bidirectional RRT [7] or augmentation via local RRTs [14] may not be practical to implement because the precise time to reach the goal may be undefined. In this case, G will span a large (possibly infinite) interval on the t axis of $\mathcal{CT}$. Given that there is no single goal state, we are faced with the difficult question of where to root a second tree. PRM approaches typically cannot be applied in a straightforward manner because the PRM spans $\mathcal{C}$ and not $\mathcal{CT}$. Some approaches [15] solve the problem with respect to static obstacles in $\mathcal{C}$ only, before incorporating dynamic obstacle constraints in $\mathcal{CT}$. DRRT may suffer particularly in the $\mathcal{CT}$ formulation because it cannot plan backwards from the goal. If sensor updates happen in the vicinity of the robot, the valid subtrees reused between subsequent plans will be relatively small.

III. The MP-RRT Algorithm

We present the MP-RRT algorithm, an RRT variant which is well-suited for dynamic planning problems for mobile robots and manipulators in environments with unknown or dynamic obstacles. Conceptually, MP-RRT combines the strengths of biasing the sampling distribution towards previously useful states as in ERRT and analytically computing which segments of previous RRTs can be re-used as in DRRT.

At the start of a given planning iteration, if there is no tree T from a previous iteration, or if the intital state or goal region have changed substantially, MP-RRT builds an RRT from scratch. Otherwise, the MP-RRT runs the PruneAndPrepend procedure (see accompanying pseudocode for the MpRrtSearch and PruneAndPrepend procedures). During planning, MP-RRT maintains a forest F of disconnected subtrees which lie in $\mathcal{C}_{free}$, but which are not connected to the root node q_{root} of T. In PruneAndPrepend, any nodes and edges of T and F which are no longer valid are deleted, and any disconnected subtrees which are created as a result are placed into F.

Procedure MpRrtSearch(q_{init})
Performs the MP-RRT algorithm.

Data: T: the previous search tree, if any
F: the previous forest of disconnected subtrees
q_{init}: the initial state
Result: a boolean value indicating plan success

```
if Empty(T) then
    q_root = q_init;
    Insert(q_root, T);
else
    PruneAndPrepend(T, F, q_init);
    if TreeHasGoal(T) then
        return true ;
    end
end
while search time/space remaining do
    q_new = SelectSample(F);
    q_nearest = NearestNeighbor(q_new, T);
    if q_new ∈ F then
        b_connect = Connect(q_nearest, q_new);
        if b_connect and TreeHasGoal(T) then
            return true ;
        end
    else
        b_extend = Extend(q_nearest, q_new);
        if b_extend and IsGoal(q_new) then
            return true ;
        end
    end
end
return false ;
```

Unlike DRRT, MP-RRT anticipates significant updates to both the initial robot configuration and goal region G. After pruning, if the initial state q_{init} of the search is no longer the root of the tree T, the ReRoot procedure attempts to make a point-to-point connection between the new q_{init} and the old tree T. If such an attempt fails, the old tree is placed into F

Procedure PRUNEANDPREPEND(T, F, q_{init})
Used at start of MP-RRT search query.

Data: T: the original search tree
F: the forest of disconnected subtrees
q_{init}: the new initial state
Result: The search tree T is valid and the forest F contains any surviving subtrees of the original search tree.

```
for each node q ∈ T, F do
    if not NODEVALID(q) then
        KILLNODE (q);
    else if not ACTIONVALID(q) then
        SPLITEDGE (q);
    end
end
if not EMPTY(T) and q_root ≠ q_init then
    if not REROOT(T, q_init) then
        place old T in F;
        initialize T to have q_root = q_init ;
    end
end
```

Procedure SELECTSAMPLE(F)
Used to generate samples for MP-RRT.

Data: F: the forest of disconnected subtrees
Result: q_{new}: the selected sample state

```
p = RANDOM(0, 1);
if p < p_goal then
    q_new = q_goal ∈ G;
else if p < (p_goal + p_forest) and not EMPTY(F)
then
    q_new = q ∈ SUBTREEROOTS(F) ;
else
    q_new = RANDOMSTATE() ;
end
return q_new ;
```

and T is re-initialized to contain the single node q_{init}. After pruning the tree and prepending the new root, pre-processing for MP-RRT is complete. At this point, T may contain some node in G, in which case the MPRRTSEARCH function will return successfully.

If T does not contain a goal state after pre-processing, the inner search loop is run similarly to the original RRT algorithm, but with one key modification: the SELECTSAMPLE function is implemented to return the root of some disconnected subtree in F with some probability p_{forest}. If q_{new} lies in F, then the MP-RRT algorithm attempts to make a point-to-point connection between q_{new} and its nearest neighbor $q_{nearest}$; otherwise it extends an edge from $q_{nearest}$ towards q_{new} as in the original RRT algorithm.

With a few key changes, the MP-RRT algorithm can be transformed into existing algorithms. If disconnected subtrees are simply deleted instead of being inserted into F, MP-RRT algorithm becomes DRRT. MP-RRT can be turned into ERRT by invalidating all edges in T at the start of each planning iteration, invalidating all nodes except those on a previously successful path, and changing the CONNECT to an EXTEND. Finally, the main difference between MP-RRT and RRF is that RRF attempts to CONNECT a newly inserted node $q_{new} \in T$ to every disconnected component of F after each successful EXTEND.

A. Parameter selection

The MP-RRT introduces a forest bias p_{forest} which balances the tradeoff between the high cost of collision checking and the performance benefit of integrating disconnected subtrees. The forest bias essentially controls the number of new nodes to be generated in between connection attempts. Increasing the forest bias too much results in a large number of failed connections (each one representing a NN query and an edge collision check), whereas decreasing it too much results in a failure to exploit the potential connections to subtrees. For tuning the forest bias, one useful statistic to watch is the ratio of failed connections to successful connections. In our test cases we observed ratios between 2:1 and 3:1 still yielding significant performance gains over DRRT.

B. Heuristics and implementation details

One useful heuristic in any RRT implemention is to always select a goal state as the first random sample. An intuitive next step for MP-RRT is to select as the second random sample the root of a disconnected subtree which contains a goal state (if such a subtree exists). As in [7], we use the RRT-Connect heuristic to repeat extensions towards the randomly sampled state q_{new}. When extending an edge from $q_{nearest}$ towards q_{new}, we repeat the call to EXTEND as long as the terminal state of the new edge decreases the metric to q_{new}. This provides the ability to grow very long branches of the tree while still allowing intermediate points to be selected as nearest neighbors for expansion.

Keeping track of failed extensions while connecting to disconnected subtrees can also increase performance. When the RRT algorithm randomly selects some node $q \in F$, the NEARESTNEIGHBOR function should return the nearest neighbor from which an extension has not previously failed. In practice, we observed that this can modestly increase performance of MP-RRT.

It is also useful to enforce an upper bound on the size of F, because given a large number of forest nodes, simply validating the RRT from the previous planning iteration may take far longer than constructing a brand-new plan from scratch. A useful strategy for bounding the number of forest nodes is to tag each node in F with the number of planning iterations during which it has been disconnected from the main RRT. If the age of the node is greater than some threshold, the node should be pruned. A more agressive bound is implemented by simply deleting all subtrees in F immediately before the PRUNEANDPREPEND step. In this case, F contains only

those subtrees which were disconnected due to pruning at the start of the current iteration.

IV. Experiments

A. Experimental setup

Experiments were conducted comparing MP-RRT against three other RRT variants in configuration spaces of two, three, and four dimensions. In each case, the workspace was a 2D plane populated with circular obstacles of varying sizes. For the two- and three-dimensional experimental setups, the simulated robot was given limited sensing capabilities—any obstacles which were not visible to the robot based on a simple occlusion model were not considered by the collision checker until the robot was subsequently able to observe them. The robot was modeled as a disc-shaped translating robot moving in $(x, y) \in \Re^2$ for the 2D case, and a rectangular robot whose configuration is given by $(x, y, \theta) \in SE(2)$ for the 3D case.

The final experimental setup was a space-time planner where the state of the robot is given by $(x, y, \theta, t) \in SE(2) \times \Re$, with t indexing time. As opposed to the 2D and 3D $\mathcal{C}$-space examples, the 4D example used a global sensing model where the number of obstacles, their positions, and velocities were known, but the obstacles themselves moved unpredictibly. In between each planning interval, the obstacles moved at constant speeds, but could perturb the direction of their 2D velocities according to a roughly Gaussian distribution. The robot was constrained to have limited velocity in the (x, y) and θ directions, but unlimited acceleration.

In all experimental setups, (x, y) sampling was performed uniformly in a bounding rectangle outside of which the robot was not allowed to travel. In the three- and four-dimensional setups, θ sampling was uniform in the interval $[0, 2\pi)$, and for the space-time experimental setup, we used an adaptive approach to sampling in t which sampled uniformly in the interval $[t_0, 2t_{min}(x, y, \theta)]$, where t_0 is the time of the initial state for the planer, and $t_{min}(x, y, \theta)$ computes the minimum possible time to reach the randomly sampled (x, y, θ) line in $\mathcal{CT}$ from the initial state, given the velocity bounds, and neglecting obstacles. The $\mathcal{CT}$ formulation of the final experimental setup facilitates two natural criteria for pruning not present in the 2D and 3D setups: if the t component of a configuration in T or F precedes the time of q_{init}, or if the configuration cannot be reached from q_{init} without violating the speed constraints, then the node is deleted.

The goal region was chosen as any states whose (x, y) components were inside a circle of given radius centered on a fixed (x_{goal}, y_{goal}) location in the workspace. In the 2D setup the goal region is a simple circle; in the 3D version, a cylinder swept along the θ axis; and in the 4D version, a hypercylinder swept along the θ and t axes. For each experiment and for each algorithm tested, a single trial proceeded as follows:

1) Generate a random distribution of obstacles and seed the initial and goal locations randomly on opposite sides of the bounding rectangle.

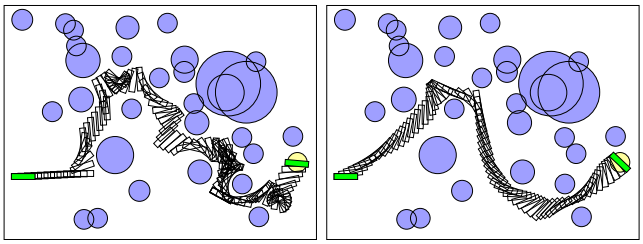

Fig. 3. Effect of incremental path smoothing in 3D $\mathcal{C}$-space experiment. *Left:* without greedy smoothing heuristic *Right:* with smoothing.

2) Plan a path from the start to the goal given current knowledge about the environment (with no more than 100 samples per planning iteration and no more than 5000 nodes total in the final RRT).
3) Step the robot (if possible) along the unique path in the RRT which ends at the node which is closest to the goal region.
4) Repeat steps 2-3 above until the robot has reached the goal region, the robot has collided with an obstacle (spacetime setup only), or the tree is full and no path to the goal region has been found.

The approach of limiting the number of samples per iteration to a relatively small number while allowing a relatively large maxiumum tree size is an example of receding horizon control, as seen applied to dynamic planning in [16]. The motivation for this approach is that at the start of any given planning iteration, the robot typically does not have perfect knowledge of the world, so if it cannot find a solution after some small number of samples, it should step the robot in the direction of the goal and plan again after getting more information.

The four algorithms used in each experimental setup were MP-RRT, DRRT, ERRT, and a naïve iterated RRT. For all algorithms, the RRT goal bias was set at 0.05, meaning that on average, one out of every 20 samples chosen was selected from inside the goal region. The ERRT waypoint bias was set to 0.5, the MP-RRT forest bias was set to 0.1, and the maximum number of ERRT waypoints was set to 50. For any given set of trials, the initial locations of the robot, obstacles and goal remained the same regardless of which algorithm was being tested.

B. Smoothing

All experimental setups (2D, 3D, and 4D) and all algorithms were run both with and without a greedy smoothing heuristic enabled (pseudocode below). The heuristic attempts to incrementally generate shorter paths by skipping intermediate nodes, similar to the strategy described in [9]. At the end of each planning iteration in which a successful RRT is generated, the smoothing algorithm iterates back from the goal state to try to find a "shortcut" which connects directly to the initial state.

Unlike more sophisticated path smoothing approaches which can explicitly optimize for safety [17], the smoothing method described here tends to produce paths that pass closer

Procedure SMOOTHPATH(P)
Used to smooth paths to eliminate unnecessary detours

Data: P: path $(q_0, \ldots, q_N)$ computed by the RRT
Result: P': shortened path containing elements of P

```
for each node q_i ∈ q_N, ..., q_1 do
    if CONNECT(q_0, q_i) then
        break;
    end
end
P' = (q_0, q_i, q_{i+1}, ..., q_N) ;
return P';
```

to obstacles than the original path, due to the greedy shortcut approach. See figure 3 for an example of smoothed vs. unsmoothed paths.

C. Experimental results

We ran 100 randomized trials of the 2D and 3D $\mathcal{C}$-space setups, and 1000 randomized trials of the more difficult 4D setup, which exhibited more variance in per-trial scores. For each trial, we recorded: 1) whether the robot successfully reached the goal region, 2) the number of samples generated by the SELECTSAMPLE procedure, 3) the number of collision checks against edges in the RRT and 4) the number of planning iterations.

In all experimental setups, the MP-RRT algorithm generated fewer samples than the other algorithms (see Table I). With the exception of small differences in the 3D case, MP-RRT scored as well or better than the other algorithms in terms of number of successful trials and total planning duration. In the 2D and 3D cases, enabling smoothing did not significantly alter the scores of the DRRT and MP-RRT algorithms. However, the iterated RRT and ERRT approaches, benefited significantly from the ability to take shortcuts to the goal state, although neither performed as well as MP-RRT or DRRT.

The most interesting results we observed were in the 4D setup. The plots in figure 4 show sorted performance curves for sample count, edge collision check count, and planning duration for all successful trials. The x-axis of the plots index individual trials, sorted by score, and the y-axis is the score at that rank. The curves are different lengths on the x-axis because each algorithm yielded a different number of successful trials.

While MP-RRT modestly outperforms DRRT and ERRT in the no-smoothing condition, the performance gap is much larger in the smoothed condition. One interpretation of this result is that MP-RRT is able to construct much more robust plans in the face of dynamic obstacle motion than the other algorithms particularly when the robot moves in close proximity to obstacles. Figure 5 plots the number of planning iterations per successful trial in the 4D setup, both smoothed and unsmoothed. We observe that as the difficulty of the planning problem increases, MP-RRT requires fewer iterations than DRRT or ERRT to make comparable progress.

TABLE I
EXPERIMENTAL RESULTS

Setup	Algorithim	Successful	Samples	Edge CC	Duration
2D	Iter. RRT	92 / 100	202,134	283,242	20.77s
	ERRT	96 / 100	113,548	163,682	12.03s
	DRRT	99 / 100	31,821	120,107	5.90s
	MP-RRT	99 / 100	25,346	100,278	4.81s
3D	Iter. RRT	62 / 100	546,363	667,554	69.52s
	ERRT	71 / 100	437,689	527,165	53.65s
	DRRT	88 / 100	223,207	400,639	37.48s
	MP-RRT	82 / 100	238,026	416,548	39.23s
4D	Iter. RRT	706 / 1000	1,668,316	2,319,569	298.66s
	ERRT	712 / 1000	1,487,460	2,041,470	260.25s
	DRRT	783 / 1000	1,705,502	3,079,187	366.54s
	MP-RRT	825 / 1000	1,401,247	2,815,880	326.72s
4D*	Iter. RRT	35 / 1000	2,115,648	3,226,302	419.15s
	ERRT	38 / 1000	1,936,447	2,953,861	380.23s
	DRRT	180 / 1000	2,132,522	3,940,813	478.56s
	MP-RRT	509 / 1000	1,680,962	4,101,404	480.65s

*The second set of 4D results use the greedy smoothing technique described in section IV-A.

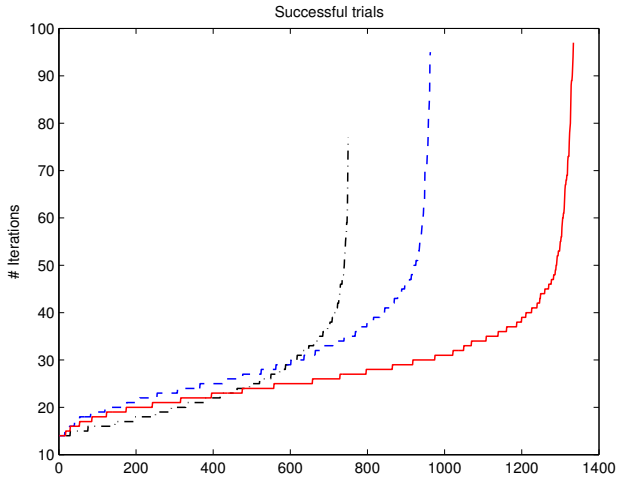

Fig. 5. Sorted performance graph for number of planning iterations per successful trial in 4D setup (both with and without smoothing heuristic). Black dash/dot line is DRRT, blue dashed line is ERRT, solid red line is MP-RRT.

V. SUMMARY AND FUTURE WORK

We have presented the MP-RRT algorithm, an RRT variant which is well-suited for dynamic planning for mobile robots and manipulators due to its novel combination of sampling distribution bias, single-query-focused tree construction, and subtree re-use. The MP-RRT overcomes some limitations of existing sampling-based dynamic planning approaches, particularly planning among moving obstacles and efficiently handling changes to both the initial and goal points of a search. Our experiments show that MP-RRT outperforms prior approaches to adapting the RRT algorithm for dynamic planning in two key areas which are directly applicable to mobile robot navigation and manipulation planning: planning among unknown or occluded obstacles, and planning around moving obstacles with unknown future trajectories.

There are a number of interesting research questions that

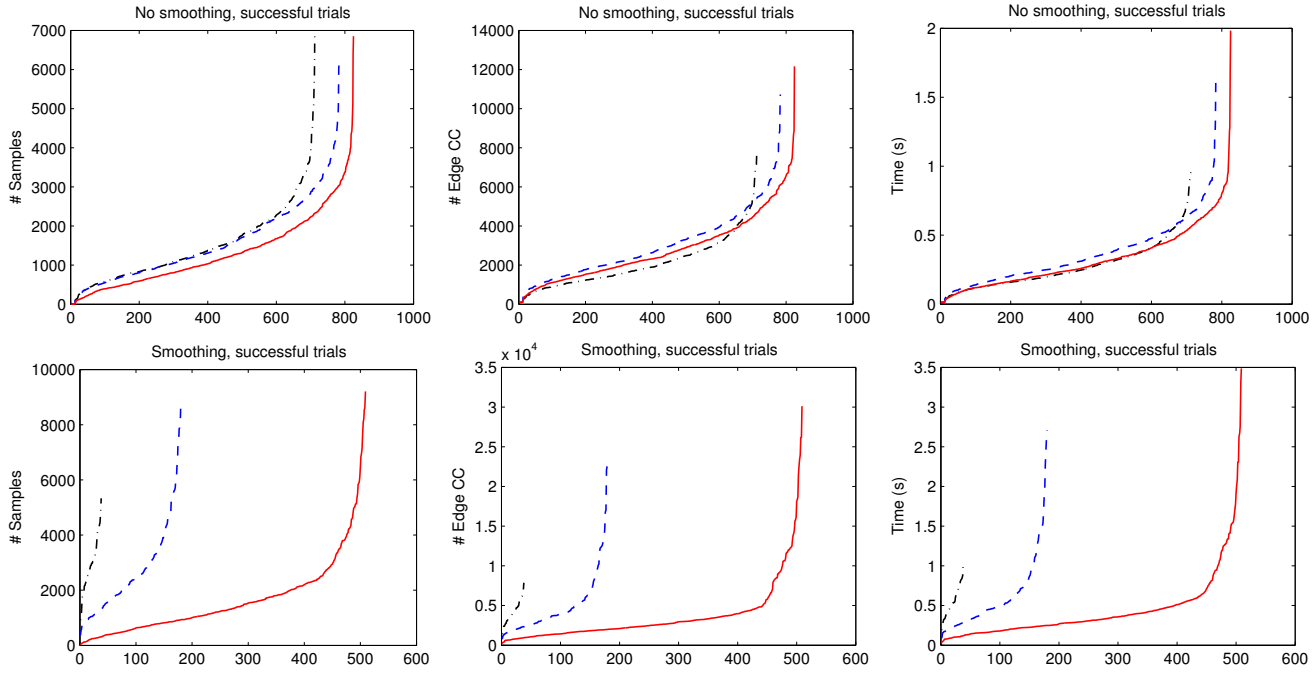

Fig. 4. Sorted performance graphs for ERRT (black dash/dot line), DRRT (blue dashed line), and MP-RRT (red line) on the 4D experimental setup. Only the successful trials from a 1000-trial run are shown. *Top row:* statistics for trials with "greedy" smoothing turned off. *Bottom row:* statistics for trials with smoothing turned on. *Left:* total number of samples generated during an experimental trial. *Center:* number of edge validation steps performed during a trial. *Right:* total planning duration in wall clock time.

remain, which form the basis of our future work. We are currently exploring deferred validation of tree and forest nodes, as in [18]. We would also like to investigate the potential benefit of splitting the tree connection problem among multiple CPUs, as in [19]. Another area of future work will be the experimental verification that MP-RRT produces more robust plans given greedy heuristics than prior approaches. Intuitively, this is supported by the notion that the forest F of disconnected subtrees serves as a store of "contingency plans" analogous to the emergency stopping paths of [20].

REFERENCES

[1] A. Stentz, "The focussed D* algorithm for real-time replanning," *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1652–1659, 1995.

[2] S. Koenig and M. Likhachev, "D* Lite," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2002.

[3] P. Leven and S. Hutchinson, "Toward real-time path planning in changing environments," *Algorithmic and Computational Robotics: New Directions: The Fourth International Workshop on the Algorithmic Foundations of Robotics*, pp. 363–376, 2000.

[4] M. Kallman and M. Mataric, "Motion planning using dynamic roadmaps," *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 5, 2004.

[5] J. Vannoy and J. Xiao, "Real-time adaptive and trajectory-optimized manipulator motion planning," in *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, Sendai, Japan, September 2004.

[6] O. Brock and L. Kavraki, "Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, vol. 2, 2001.

[7] J. Kuffner and S. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, San Francisco, CA, Apr. 2000, pp. 995–1001.

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

[9] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Proceedings of IROS-2002*, Switzerland, October 2002, an earlier version of this paper appears in the Proceedings of the RoboCup-2002 Symposium.

[10] D. Ferguson, N. Kalra, and A. T. Stentz, "Replanning with RRTs," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, May 2006.

[11] T. Li and Y. Shie, "An incremental learning approach to motion planning with roadmap management," *Proc. IEEE Int'l Conf. on Robotics and Automation*, vol. 4, 2002.

[12] Y. Hirano, K. Kitahama, and S. Yoshizawa, "Image-based object recognition and dexterous hand/arm motion planning using rrts for grasping in cluttered scene," in *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 2005, pp. 3981–3986.

[13] J. C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers, 1991.

[14] M. Strandberg, "Robot path planning: an object-oriented approach," Ph.D. dissertation, Automatic Control, Dept. of Signals, Sensors and Systems, Royal Institute of Technology (KTH), 2004.

[15] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2006, pp. 2366–2371.

[16] E. Frazzoli, "Robust Hybrid Control for Autonomous Vehicle Motion Planning," *Proceedings of the 39th IEEE Conference on Decision and Control*, vol. 1, pp. 821–826, 2000.

[17] S. Quinlan, "Real-time modification of collision-free paths," Ph.D. dissertation, Stanford University, 1994.

[18] R. Bohlin and L. Kavraki, "Path planning using lazy PRM," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, vol. 1, 2000.

[19] M. Akinc, K. Bekris, B. Chen, A. Ladd, E. Plakue, and L. Kavraki, "Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps," *The Eleventh International Symposium on Robotics Research*, 2003.

[20] V. Lumelsky and A. Shkel, "Incorporating body dynamics into the sensor-based motion planning paradigm: The maximum turn strategy," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, vol. 2, 1995.