Improved Performance of Asymptotically Optimal Rapidly Exploring Random Trees

Boardman, Beth Leigh
Harden, Troy Anthony
Martinez, Sonia

Journal of Dynamic Systems, Measurement and Control. Received May 18, 2017;
Accepted manuscript posted July 24, 2018. doi:10.1115/1.4040970
Copyright (c) 2018 by ASME

Improved Performance of Asymptotically Optimal Rapidly-Exploring Random Trees

Beth Boardman[*]
R & D Engineer
Applied Engineering and Technology
Los Alamos National Laboratory
Los Alamos, New Mexico 87545
Email: bboardman@lanl.gov

Troy Harden
R & D Engineer
Applied Engineering and Technology
Los Alamos National Laboratory
Los Alamos, New Mexico 87545
Email: harden@lanl.gov

Sonia Martínez
Professor
Department of Mechanical Engineering
University of California
San Diego, California 92093-0411
Email: soniamd@ucsd.edu

Three algorithms that improve the performance of the asymptotically-optimal Rapidly-exploring Random Tree (RRT) are presented in this paper. First, we introduce the Goal Tree (GT) algorithm for motion planning in dynamic environments where unexpected obstacles appear sporadically. The GT reuses the previous RRT* by pruning the affected area and then extending the tree by drawing samples from a shadow set. The shadow is the subset of the free configuration space containing all configurations that have geodesics ending at the goal and are in conflict with the new obstacle. Smaller, well defined, sampling regions are considered for Euclidean metric spaces and Dubins' vehicles. Next, the Focused-Refinement (FR) algorithm, which samples with some probability around the first path found by an RRT*, is defined. The third improvement is the Grandparent-Connection (GP) algorithm, which attempts to connect an added vertex directly to its grandparent vertex instead of parent. The GT and GP algorithms are both proven to be asymptotically optimal. Finally, the three algorithms are simulated and compared for a Euclidean metric robot, a Dubins' vehicle, and a seven degree-of-freedom manipulator.*

1 Introduction

Sampling-based motion planning algorithms quickly connect samples from the free configuration space in order to find a collision-free path from an initial configuration to a goal configuration. Improving the performance of sampling-based algorithms, in convergence speed and its adaptation to moving obstacles, is key in the development of real-time motion planning algorithms. Motivated by this, we present three different variants to the asymptotically optimal Rapidly-exploring Random Tree (RRT*) algorithm and evaluate their potential gains on several examples.

Rapidly-exploring Dense Tree algorithms (RDTs, also known as RRTs) [1] and Sampling-Based Roadmaps (SBRs, including Probabilistic Roadmaps (PRMs) [2]) are sampling-based motion planners which are resolution or probabilistically complete, and find a feasible path to the goal without the explicit modeling of the configuration space. As opposed to SBRs, RDTs do not require pre-processing and can find a path relatively quickly. The path produced by these planners can be very jagged, resulting in unnecessary motion that increases the execution time. Consequently, this motivated research into obtaining better paths from these planners.

One way to improve paths is to apply a post-processing algorithm. In [3], one such algorithm limits the allowable deviation from the original path and results in a new path with fewer nodes. A divide and conquer method is used in [4] to shorten a given path by connecting the first and last nodes in the path directly. If not successful, the path list is bisected until the connection is successful. For a predetermined number of times, the post-processing algorithm in [5] randomly selects two points from the path list and attempts to replace the segment between them with a straight line.

A subsequent effort focuses on obtaining paths that guarantee asymptotic optimality with probability one. At the core of this line of work are the PRM* and RRT* [6]. The RRT* can handle any-time applications [7] and manipulators [8]. The Ball-tree algorithm, [9], improves the performance of the RRT and RRT* by using volumes of free-space instead of points as the vertices of the tree. The RRT$^{\#}$ [10] is another planner that returns an optimal path by maintaining a

[*]Address all correspondence to this author.

graph and a spanning subtree. The RRT$^{\#}$ separates the exploration and exploitation tasks so they can be run in parallel to improve performance. The Fast Marching Tree (FMT*) [11], performs a "lazy" dynamic programming recursion on samples from the configuration space to produce a tree of paths. A key result from [11] is the algorithm convergence rate.

The following papers also study the effects of exploitation versus exploration on the RRT*. Akgun et al. [12] uses local biasing to choose the sampling point based upon the current best path to the goal. The RRT*-Smart in [13] finds an initial path to the goal, then it optimizes it using first a smoothing technique, and then it further shapes it by biasing sampling to balls around the nodes in the optimized path. While these two papers share the same idea of exploitation of a given path to the goal, the method focuses on a single path only, which results, at most, in a locally-optimal path.

The idea of re-adapting motion plans when finding new unexpected obstacles has been exploited significantly in the literature. The discrete-time D*, and D* lite algorithms [14], [15] re-adapt A* algorithms to find the optimal path in a discretized space. However, these algorithms become intractable as the dimensionality increases, while they have a limited ability to handle differential constraints.

The sampling-based algorithms in [16], [17] [18], [19], and [20] all extend the RRT algorithm to deal with dynamic environments. The Dynamic Rapidly-exploring Random Tree (DRRT) [16] roots the tree at the goal and trims branches in the tree that are obstructed by the new obstacle. The trimming is done by removing nodes that are within a region that contains the obstacle and whose edge is in conflict. The descendants of the affected nodes are also removed so that only one tree is maintained. The remaining paths in the tree still lead to the goal but are not optimal.

In [19], the Reconfigurable Random Forest (RRF) algorithm maintains a forest of trees from previous plannings that have been broken apart according to the new obstacle information. The RRF attempts to connect the trees as in the RRT-connect [21] making this framework good for multi-query problems. The trees are trimmed by removing all nodes from within a bounding box containing the obstacle that are determined to be in conflict with the new obstacle. The RRF also prunes its trees to maintain a manageable number of nodes to reduce searching time. The lazy reconfiguration forest (LRF) is presented in [20], and uses the idea of maintaining multiple RRT trees from the RRF but only checks for invalid edges along the task path instead of checking the entire tree.

To rebuild a tree from the initial position, way points from the previous tree are reused to increase the likelihood of a successful connection in the execution extended RRT (ERRT) [18]. The ERRT also uses an adaptive cost function that improves the generated paths. The multipartite RRT (MP-RRT) [17] combines several of the above mentioned planners and an opportunistic strategy for reusing information during replanning in a dynamic environment. However, none of these algorithms produce optimal paths. An asymptotically optimal replanning algorithm, RRT^X, was developed by the authors of [22]. The RRT^X maintains a graph and a shortest path sub-tree rooted at the goal. When an obstacle is added or removed only the effected edges are updated.

This manuscript builds on the authors' two previous papers on motion planning, [23, 24]. The first paper, [23], proposes the Goal Tree (GT) replanning algorithm to handle unexpected obstacles. The second paper, [24], proposes the Grandparent-Connection (GP) and the Focused-Refinement (FR) algorithms. Compared with previous work, this manuscript revisits the algorithms and presents an analysis of them with complete proofs and additional extended simulations. The results are shown to move us closer to real-time motion planning by finding near optimal paths in a shorter amount of time compared to the RRT*. The GT algorithm is (to the best of the authors' knowledge) one of only two sampling-based asymptotically optimal replanners.

In particular, we present a simplified version of this algorithm with respect to the version in [23] to rebuild the original tree instead building a second tree and connecting the two. We prove that the asymptotic optimality, node position, and probabilistic completeness results for the RRT* are maintained when using the GP modification. The GP is also proven to recover the optimal path to the goal in the visible set, all configuration that have geodesics to the goal that are not in conflict with any obstacle. This paper also introduces the approximation of the shadow as the collection of nodes that were removed from the tree. This approximation is used as the sample set for rebuilding the tree during the GT algorithm. The simulations have been expanded to include a seven degree-of-freedom manipulator and include results that combine the GT and GP algorithms.

The Goal Tree (GT) algorithm reuses information from a RRT* rooted at the goal configuration, $\mathscr{T}_G$, to reduce the replanning time in the presence of a new obstacle. When a previously unknown obstacle obstructs the best path, $\mathscr{T}_G$ is trimmed to reflect this information. The tree is then incrementally extended in the affected region of the configuration space. In this setting, we identify a new sampling region, strictly contained in the configuration space, such that, when used with the GT algorithm, guarantees the recovery of an asymptotically optimal path. First, a region is proven to exist, then a characterization is provided for a general robot in a d dimensional environment. By exploiting the known path types of vehicles with no differential constraints in a d dimensional configuration space and a Dubins' vehicle, alternative characterizations of the new planning region are given.

The Focused-Refinement (FR) algorithm is a modification of RRT* that reduces the computation time needed to obtain a low-cost path to the goal. This is done by exploring the environment quickly until a set of paths to the goal is found. Then, the algorithm focuses on lowering the cost of this set of paths while periodically exploring the environment. In this way, the algorithm quickly returns an asymptotically optimal path within the regions that are more intensively exploited. We present a novel way of uniformly sampling randomly within these regions that, with the right parameters, can recover the entire configuration space.

The Grandparent-Connection (GP) is a modification to the RRT* algorithm that attempts to connect the added vertex to its grandparent instead of its parent vertex. This essentially straightens the computed paths and lowers the cost.

Simulations for all three algorithms and the combination of GP with GT are given in Euclidean space, for a Dubins' vehicle, and a seven degree-of-freedom manipulator. A Dubins' vehicle running the GT algorithm is shown to improve replanning performance compared to rerunning the RRT*. The GP and FR are shown to find better cost paths more quickly than the RRT*.

This paper is organized as follows. The RRT* algorithm is reviewed in Section 2. Sections 3 and 4 detail the proposed algorithms and Section 5 analyzes the optimality of the Goal Tree and GP algorithms. Simulations test the algorithms in Section 6. Section 7 concludes the paper.

2 The Rapidly-exploring Random Tree Star Algorithm

This section briefly describes the RRT* algorithm by Karaman and Frazzoli which is theoretically analyzed in [6]. The kinodynamic RRT* is presented in [25].

The RRT*, outlined in Algorithm 1, builds a tree, $\mathscr{T}$, which is dense with probability one in the entire configuration space, X, as the number of samples, n, goes to infinity. Denote by X_{free} the free configuration space in X and X_{obs} as the obstacles space. The tree is composed of a set of vertices, $v \in \mathscr{T}.V$, and edges, $e \in \mathscr{T}.E$. Each edge is an ordered pair of vertices $e_{1,2} = (v_1, v_2)$, where v_1 is the parent and v_2 is the child. We use Cost as the notation for the cost function being minimized. Each edge added to $\mathscr{T}$ has a cost,denoted $c_{\text{edge}}(e)$, where $e \in \mathscr{T}.E$. In the original work by [6], the edge cost considered is the *cost-to-go*,the cost of moving from the parent v_1 to the child v_2. Then, the cost of a vertex, $\mathsf{Cost}(v)$, is the sum of the costs of the edges connecting the root to v. The paths in $\mathscr{T}$ are asymptotically optimal, as $n \to \infty$ the optimal path from the initial configuration, $x_I \in X_{\text{free}}$, to any other configuration in X_{free} is recovered. The functions involved in the RRT* process are described as follows. With some abuse of notation, we will define a robot configuration as x_v instead of v.

After initializing $\mathscr{T}$ at x_I, the RRT* begins by using the Sample function to output x_{rand}, a uniformly sampled random configuration from X_{free}. The $\mathsf{Nearest}$ function finds the nearest vertex, $x_{\text{nearest}} \in \mathscr{T}$, and extends $\mathscr{T}$ a distance ε from x_{nearest} to get x_{new}.

Next, the set of near vertices from $\mathscr{T}$ with respect to x_{new} are output as the set X_{near} from the function Near. Vertices that are farther than $r = \min(\varepsilon, \gamma(\log(n_v)/n_v)^{(1/d)})$, where n_v is the number of vertices in $\mathscr{T}$, d is the dimension of the configuration space, and γ is an independent parameter, are omitted from X_{near}. The best parent for x_{new}, determined in $\mathsf{FindBestParent}$, is the vertex in X_{near} that has a collision-free path with the lowest $\mathsf{Cost}(x_{\text{new}})$, outlined in Algorithm 2. The paths that connect the vertices to each other (determined using Steer), satisfy the system dynamics. Only collision-free edges are added to $\mathscr{T}$. The collision checker, $\mathsf{CollisionCheck}$, returns true if the edge is collision-free. If x_{new} is added to $\mathscr{T}$, then Rewire, outlined in Algorithm 3, attempts to add the other vertices in X_{near} as children of x_{new} based upon a lower cost and collision-free edge.

Algorithm 1 $\mathscr{T} = (V,E) \leftarrow \mathsf{RRT}^*(x_I, \varepsilon)$

```
$\mathscr{T} \leftarrow \mathsf{InitializeTree}()$;
$\mathscr{T} \leftarrow \mathsf{InsertNode}(\emptyset, x_I, \mathscr{T})$;
for $i = 1$ to $i = N$ do
    $x_{\text{rand}} \leftarrow \mathsf{Sample}(i)$;
    $x_{\text{new}} \leftarrow \mathsf{Nearest}(\mathscr{T}, x_{\text{rand}}, \varepsilon)$;
    $X_{\text{near}} \leftarrow \mathsf{Near}(\mathscr{T}, x_{\text{new}})$;
    $x_{\text{parent}} \leftarrow \mathsf{FindBestParent}(X_{\text{near}}, x_{\text{new}})$;
    if $x_{\text{parent}} \neq \mathsf{NULL}$ then
        $\mathscr{T} \leftarrow \mathsf{InsertNode}((x_{\text{parent}}, x_{\text{new}}), x_{\text{new}}, \mathscr{T})$;
        $\mathscr{T} \leftarrow \mathsf{Rewire}(\mathscr{T}, X_{\text{near}}, x_{\text{new}})$;
    end if
end for
return $\mathscr{T}$
```

Algorithm 2 $x_{\text{parent}} \leftarrow \mathsf{FindBestParent}(X_{\text{near}}, x_{\text{new}})$

```
$x_{\text{parent}} \leftarrow \emptyset$;
$c_{\min} \leftarrow \infty$;
for $x_{\text{near}} \in X_{\text{near}}$ do
    $e_{\text{near,new}} \leftarrow \mathsf{Steer}(x_{\text{near}}, x_{\text{new}})$;
    $c_{\text{near}} \leftarrow \mathsf{Cost}(x_{\text{near}}) + c_{\text{edge}}(e_{\text{near,new}})$;
    if $c_{\text{near}} < c_{\min}$ and $\mathsf{CollisionFree}(e_{\text{near,new}})$ then
        $x_{\text{parent}} \leftarrow x_{\text{near}}$;
        $c_{\min} \leftarrow c_{\text{near}}$;
    end if
end for
return $x_{\text{parent}}$;
```

Algorithm 3 $\mathscr{T} \leftarrow \mathsf{Rewire}(\mathscr{T}, X_{\text{near}}, x_{\text{new}})$

```
for $(x_{\text{near}}) \in X_{\text{near}}$ do
    $e_{\text{near,new}} = \mathsf{Steer}(x_{\text{new}}, x_{\text{near}})$;
    if $\mathsf{Cost}(x_{\text{new}}) + c_{\text{edge}}(e_{\text{near,new}}) < \mathsf{Cost}(x_{\text{near}})$ then
        if $\mathsf{CollisionFree}(e_{\text{near,new}})$ then
            $x_{\text{oldparent}} \leftarrow \mathsf{Parent}(\mathscr{T}, x_{\text{near}})$;
            $\mathscr{T}.\mathsf{remove}((x_{\text{oldparent}}, x_{\text{near}}))$;
            $\mathscr{T}.\mathsf{add}((x_{\text{new}}, x_{\text{near}}))$;
        end if
    end if
end for
return $\mathscr{T}$;
```

3 The Goal Tree Algorithm

In this section, the Goal Tree (GT) algorithm is described in detail. The GT is a method for replanning when unexpected or moving obstacles obstruct the execution of the path determined by the RRT*. The RRT* produces paths that are asymptotically optimal from the initial configuration to any other point in the configuration space. By a slight modification to the RRT* algorithm, one can produce a tree, $\mathscr{T}_G$, rooted at the goal configuration, x_G, such that the asymptotically optimal path from any point in X_{free} to x_G can be recovered. To do this, the cost associated with each edge $e_{1,2} = (v_1, v_2)$ in the RRT* tree becomes the *cost-to-come*; i.e. the cost of traveling from the child v_2 to the parent v_1.

Journal of Dynamic Systems, Measurement and Control. Received May 18, 2017;
Accepted manuscript posted July 24, 2018. doi:10.1115/1.4040970

Once the new obstacle, $\mathscr{O}$, has been discovered, the GT trims $\mathscr{T}_G$ and then it is extended in some subset $R \subseteq X \setminus \mathscr{O}$. To trim $\mathscr{T}_G$, the edges are checked for conflict with $\mathscr{O}$ and removed using PropagateCost. Instead of checking every edge in $\mathscr{T}_G$ we can define a subregion that contains all possible vertices whose trajectories are in conflict with $\mathscr{O}$. We want to define the region as all points within some Euclidean distance from a point in $\mathscr{O}$. Denote the center point of $\mathscr{O}$ as x_c, and the maximum Euclidean distance from x_c to the boundary of $\mathscr{O}$ as $r_{\max}$. A graph search is done over $\mathscr{T}_G$ to determine the maximum edge cost, $r_{\text{cost}} = \max_{e \in \mathscr{T}.E}\{c_{\text{edge}}(e)\}$. Since $c_{\text{edge}}(e_{1,2}) \geq \|x_1 - x_2\|$, the set of vertices whose trajectories are in conflict with $\mathscr{O}$ is contained in

$$V_{\text{conflict}} = \{v \in \mathscr{T}_G.V \mid \|x_c, x_v\| \leq r_{\max} + r_{\text{cost}}\}.$$

All trajectories of the vertices in V_{conflict} are checked for conflict with $\mathscr{O}$. All vertices found in conflict with $\mathscr{O}$, and their descendants, are trimmed from $\mathscr{T}_G$.

4 The Focused-Refinement and Grandparent-Connection Modifications

The following section describes the two extensions to the RRT* in detail. The first is the Focused-Refinement (FR) modification that attempts to recover a near optimal path much quicker than the RRT*. The second extension, the Grandparent-Connection (GP), is aimed at reducing the number of nodes in the path to the goal and reducing the computation time needed to discover a near optimal path.

4.1 The Focused-Refinement algorithm

As shown in [6], the RRT* initially constructs a tree that is the same as the RRT and then, as more nodes are added, the RRT* begins to look at many neighboring vertices to recover an asymptotically optimal path. The RRT* finds and refines all paths in the configuration space. The refinement extension, Focused-Refinement (FR), focuses on refining only those paths that have already reached the goal region in hopes of reducing the amount of time needed to find a sufficiently optimal path.

The FR begins the construction of a tree using the RRT* algorithm until there exists at least one path that reaches the goal region. This set of paths is denoted as Π, with p vertices defining a set V_Π. The FR has two options: exploring the configuration space or exploiting Π to lower its cost. If exploring, the algorithm proceeds as the RRT*, but if exploiting, the set of vertices in Π, V_Π, is determined. The sample x_{new} is determined by perturbing a vertex randomly drawn from the set V_Π. The FR then proceeds as the RRT*.

The pseudo code for the FR is presented in Algorithm 4, and uses three parameters. The first is $C_{\text{exploit}} \in \mathbb{N}$, the number of consecutive iterations the FR will exploit Π. The number of consecutive iterations to explore X_{free} is the second parameter needed, $C_{\text{explore}} \in \mathbb{N}$. The third parameter, $C_{\text{reset}} \in \mathbb{N}$, tells the algorithm when to update V_Π. The sampling region defined by V_Π does not change dramatically every iteration, therefore, to save computation time, the set V_Π is only updated every $C_{\text{reset}} + C_{\text{explore}}$ iterations. If $C_{\text{exploit}} = 0$ and $C_{\text{explore}} = \infty$, the FR becomes the RRT*. The parameters are chosen by the user to best reflect their desired ratio of exploration versus exploitation. It is recommended to keep the numbers small relative to the total number of samples to allow for swapping between the explore and exploit natures of the FR algorithm. In order to take advantage of the exploitation property of the FR, C_{exploit} should be greater than C_{explore}. In environments with multiple routes to the goal, C_{explore} can be increased in hopes of finding a better route than what was initially found.

Exploitation only occurs if GoalReach returns true (there exists at least one path to X_{G}) and exploitation has occurred less than C_{exploit} consecutive times. Once Π has been exploited C_{exploit} iterations, the RRT* is allowed to explore the space for C_{explore} iterations. The following are the details on choosing x_{new} during the exploitation stage of the FR.

The new sample, x_{new}, is determined as illustrated in Fig. 1 and in Algorithm 5. Given a d-dimensional configuration space, $X \subset \mathbb{R}^d$, consider $k \in \{1, 2, \cdots, d\}$. First, the minimum and maximum k-component from $V_\Pi \in \mathbb{R}^{d \times p}$, $w_{\min} = \min V_\Pi^k$ and $w_{\max} = \max V_\Pi^k$, are found. Here, V_Π^k is the set of all k-components of the vertices in V_Π. Next, the k-component of x_{new} (x_{new}^k) is taken as a uniformly random sample between $w_{\min} - \varepsilon$ and $w_{\max} + \varepsilon$, $\varepsilon > 0$. For every $j \neq k$, the j-component of the vertex whose k-component is nearest to x_{new}^k is determined, x_{nearest}^j, $x_{\text{nearest}} = \operatorname{argmin}_{x \in V_\Pi} \|x_{\text{new}}^k - x^k\|$. The j-component of x_{new} is uniformly sampled between $x_{\text{nearest}}^j - \varepsilon$ and $x_{\text{nearest}}^j + \varepsilon$. The FR alternates which k-component is used to determine x_{new}, this provides a uniform distribution of samples around Π. As ε is increased, the entire configuration space is uniformly sampled randomly, thus recovering the original RRT*.

Note that V_Π can consist of multiple distinct paths to the goal. Determining distinct paths is non-trivial and potentially time-consuming. In general, and in the simulation section, V_Π is only the current best path. Efficiently determining distinct paths is a subject of future work. Because the FR restricts exploration of the free configuration space, it is not for use in conjunction with the GT replanning algorithm.

4.2 The Grandparent-Connection modification

The GP was inspired by reducing the number of nodes in, and cost of, a given path. Before adding a node to the tree, the modified algorithm attempts to connect directly to its grandparent node, as outlined in Algorithm 6. A successful connection to the grandparent occurs when a lower cost, collision-free path is found. It is also predicted that the Grandparent Connection will produce smoother paths with fewer nodes. Because the Grandparent-Connection is applied during construction of the tree as every node is added to the tree, the grandparent connection smooths out every path in the tree. The GP gains the advantage over smoothing a single path when paired with the GT or similar replanning algorithms. The grandparent connection can also be used in combination with the FR algorithm.

Journal of Dynamic Systems, Measurement and Control. Received May 18, 2017;
Accepted manuscript posted July 24, 2018. doi:10.1115/1.4040970
Copyright (c) 2018 by ASME

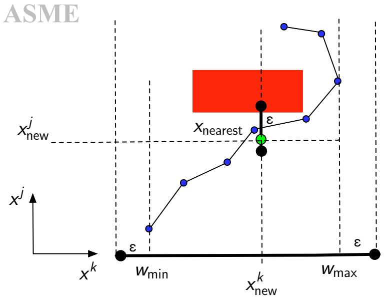


Fig. 1: An illustrative example on choosing x_{new} when refining a single path. The red rectangle is an obstacle in the environment. The blue dots are the the set of vertices, V_Π, used to determine the region from which x_{new} is sampled. The k-component of x_{new} is a uniform random sample between the maximum and minimum (plus and minus ε, respectively) k-component values from V_Π. Next, with respect to x^k_{new}, determine the nearest k-component from V_Π and label its corresponding j-component as x^j_{nearest}. Finally, the x^j_{new} is a random value from between $x^j_{\text{nearest}} - \varepsilon$ and $x^j_{\text{nearest}} + \varepsilon$, $\varepsilon > 0$. Sample x_{new} is represented as the green dot.

Algorithm 4 $\mathscr{T} = (V,E) \leftarrow$ $\mathsf{FR}(x_i, \varepsilon, d, C_{\text{exploit}}, C_{\text{explore}}, C_{\text{reset}})$

```
$\mathscr{T} \leftarrow \mathsf{InitializeTree}()$;
$\mathscr{T} \leftarrow \mathsf{InsertNode}(\emptyset, x_i, \mathscr{T})$;
$c_{\text{reset}} = 1$; $c_{\text{exploit}} = 1$; $c_{\text{explore}} = 1$; $k = 1$;
for $i = 1$ to $i = N$ do
    if GoalReach and $c_{\text{exploit}} < C_{\text{exploit}}$ then
        if $c_{\text{reset}} = 1$; then
            $V_\Pi = \mathsf{PathSet}(\mathscr{T})$;
        end if
        $(c_{\text{reset}}, c_{\text{exploit}}) \leftarrow \mathsf{UpdateParameters}(c_{\text{reset}}, c_{\text{exploit}}, C_{\text{reset}})$;
        $x_{\text{new}} = \mathsf{NewPointPathSet}(V_\Pi, \varepsilon, k, d)$;
        $k \leftarrow \mathsf{UpdateDimension}(d)$;
    else
        $(c_{\text{explore}}, c_{\text{exploit}}) \leftarrow \mathsf{UpdateParameters}(c_{\text{explore}}, c_{\text{exploit}}, C_{\text{explore}})$;
        $x_{\text{rand}} \leftarrow \mathsf{Sample}(i)$;
        $x_{\text{new}} \leftarrow \mathsf{Nearest}(\mathscr{T}, x_{\text{rand}}, \varepsilon)$;
    end if
    $X_{\text{near}} \leftarrow \mathsf{Near}(\mathscr{T}, x_{\text{new}})$;
    $x_{\text{parent}} \leftarrow \mathsf{FindBestParent}(X_{\text{near}}, x_{\text{new}})$;
    if $x_{\text{parent}} \neq$ NULL then
        $\mathscr{T} \leftarrow \mathsf{InsertNode}((x_{\text{parent}}, x_{\text{new}}), x_{\text{new}}, \mathscr{T})$;
        $\mathscr{T} \leftarrow \mathsf{Rewire}(\mathscr{T}, X_{\text{near}}, x_{\text{new}})$;
    end if
end for
return $\mathscr{T}$
```

5 Analysis

In this section, we analyze the optimality of the GT and GP algorithms.

Algorithm 5 $x_{\text{new}} \leftarrow \mathsf{NewPointPathSet}(V, \varepsilon, k, d)$

```
$w_{\min} = \min(V^k)$;
$w_{\max} = \max(V^k)$;
$x^k_{\text{new}} = \mathsf{Rand}(w_{\min} - \varepsilon, w_{\max} + \varepsilon)$;
$x_{\text{nearest}} = \mathsf{NearestComponent}(V, x^k_{\text{new}})$;
for $j = 1$ to $j = d$; do
    if $j \neq k$ then
        $v_{\min} = x^j_{\text{nearest}} - \varepsilon$;
        $v_{\max} = x^j_{\text{nearest}} + \varepsilon$;
        $x^j_{\text{new}} = \mathsf{Rand}(v_{\min}, v_{\max})$;
    end if
end for
```

Algorithm 6 $x_{\text{parent}} \leftarrow \mathsf{FindBestParent}(X_{\text{near}}, x_{\text{new}})$

```
$x_{\text{parent}} \leftarrow \emptyset$;
$c_{\min} \leftarrow \infty$;
for $x_{\text{near}} \in X_{\text{near}}$ do
    $e_{\text{near,new}} \leftarrow \mathsf{Steer}(x_{\text{near}}, x_{\text{new}})$;
    $c_{\text{near}} \leftarrow \mathsf{Cost}(x_{\text{near}}) + c_{\text{edge}}(e_{\text{near,new}})$;
    if $c_{\text{near}} < c_{\min}$ and $\mathsf{CollisionFree}(e_{\text{near,new}})$ then
        $x_{\text{parent}} \leftarrow x_{\text{near}}$;
        $c_{\min} \leftarrow c_{\text{near}}$;
    end if
    if $x_{\text{parent}} \neq \emptyset$ then
        $x_{\text{grandparent}} \leftarrow \mathscr{T}.\mathsf{parent}(x_{\text{parent}})$;
        $e_{\text{grandparent,new}} \leftarrow \mathsf{Steer}(x_{\text{grandparent}}, x_{\text{new}})$;
        $c_{\text{grandparent}} \leftarrow \mathsf{Cost}(x_{\text{grandparent}}) + c_{\text{edge}}(e_{\text{grandparent,new}})$;
        if $c_{\text{grandparent}} < c_{\min}$ and $\mathsf{CollisionFree}(e_{\text{grandparent,new}})$ then
            $x_{\text{parent}} \leftarrow x_{\text{grandparent}}$;
            $c_{\min} \leftarrow c_{\text{grandparent}}$;
        end if
    end if
end for
return $x_{\text{parent}}$;
```

5.1 Optimality of Goal Tree

Reducing the sampling region for rebuilding $\mathscr{T}_G$ can lead to faster cost rate-of-change but can also prevent global asymptotic optimality. We prove that there exist a generic restricted region of the space which when used to extend $\mathscr{T}_G$ guarantees convergence to a globally optimal path. Then we aim to characterize these regions for common cases.

Theorem 1. *Let $X = [0,1]^d$ be a d-dimensional C-space, $d \in \mathbb{N}$ and $d \geq 2$. Let X_{obs} be the C-obstacles space. Assume $\mathscr{O}$ is newly found obstacle information, i.e. $\mathscr{O} \not\subset X_{\text{obs}}$, and there exists a ball, $B(x_G, r) \subset (X_{\text{obs}} \cup \mathscr{O})^c$, $r > 0$. Suppose the feasible dynamic paths of vehicles in a free environment are at least $\mathscr{C}^3$. Then, there exists a generic $R \subsetneq X$ such that if $\mathscr{T}_G$ is originally built in X using the RRT* with information X_{obs}, then trimmed using $\mathscr{O}$, and finally extended in R using the RRT* with information $X_{\text{obs}} \cup \mathscr{O}$, then an optimal path, $\pi : x_I \to x_G$, can be asymptotically recovered by the GT* *algorithm as $n \to \infty$.*

Proof. The ball $B(x_G, r)$ is an obstacle free environment where the restricted optimal solution is a sufficiently smooth curve. A generic property of smooth curves is that they have a finite number of inflection points and vertices; see [26] focusing on planar curves, but from which results are valid for curves in any dimensions. Thus, there exists a final piece of the optimal path, say σ, to x_G which is convex or concave and does not contain any vertex or inflection point in it. Using σ, there exists a smaller radius $r' < r$ such that $B(x_G, r') \cap \sigma$ reduces to a single intersection point. The optimal path π from $x_{I'}$ to x_G must go through the $\partial B(x_G, r')$ at this point. Then, taking $R = X \setminus B(x_G, r')$ will yield π asymptotically. □

In d dimensional environments, a rebuilding region guaranteed to recovery an asymptotically optimal trajectory can be found as follows. Consider the goal and new initial configurations, $x_G$and $x_{I'}$, and a new obstacle $\mathscr{O}$ such that $x_{I'}, x_G \notin \mathscr{O}$. For simplicity, assume that $\mathscr{O} \cap X_{\text{obs}} = \varnothing$.

First, a region in the environment is defined and then, using this region for sampling, the GT is proven to recover a geodesic from $x_{I'}$ to x_G. Due to the obstacles in the environment, any configuration in X_{free} could have more than one geodesic to x_G. Note that in the following, the distinction is made between position and configuration. Position is the $(p_1, p_2, \ldots)$ position in the environment, while configuration can also include orientations or velocities.

Definition 1. Shadow Set: *The* shadow *of x_G on $\mathscr{O}$, $\mathscr{S}_{\mathscr{O}}$, is the envelope or hull, as defined by position rather than configuration, formed by the geodesics from all configurations in X_{free} going to x_G that are in conflict with $\mathscr{O}$.*

Note that $x_{I'} \in \mathscr{S}_{\mathscr{O}}$ must be true, otherwise, there is no need for replanning. Also note that $\mathscr{S}_{\mathscr{O}}$ is a set of positions and not configurations. In this way each position could have an infinite number of possible configurations associated with it.

Definition 2. Outgoing Configuration*: Let $S \subset X$ be a set such that $x_{I'} \in S$ and whose boundary is denoted as ∂S. Then, an* outgoing configuration *on ∂S is defined as a configuration whose position is in ∂S and whose orientation or velocity will force the vehicle to leave S.*

Lemma 1. *All outgoing configurations on $\partial \mathscr{S}_{\mathscr{O}}$ have geodesics to x_G that are not in conflict with $\mathscr{O}$.*

Proof. Let x be an outgoing configuration on $\partial \mathscr{S}_{\mathscr{O}}$. Consider a geodesic from x to x_G which is in conflict with $\mathscr{O}$, then by the definition of an outgoing configuration on $\partial \mathscr{S}_{\mathscr{O}}$, any motion from x forces the vehicle position strictly outside $\partial \mathscr{S}_{\mathscr{O}}$. However, this is in contradiction with the definition of $\mathscr{S}_{\mathscr{O}}$, which contains all positions obtained from geodesics to x_G that are in conflict with $\mathscr{O}$. Therefore, there must only exist geodesics from x to x_G that are not in conflict with $\mathscr{O}$□

The main result, Theorem 2, states that using the shadow of x_G on $\mathscr{O}$ as the new sampling region will allow the Goal Tree to asymptotically recover an optimal path from $x_{I'}$ to x_G. Due to the tree structure used by the GT, only one of the geodesics from $x_{I'}$ to x_G will be recovered.

Theorem 2. *Let $\mathscr{S}_{\mathscr{O}}$ be as in Definition 1. If the Goal Tree algorithm uses $\mathscr{S}_{\mathscr{O}}$ as the new sampling region to rebuild $\mathscr{T}_G$, then it will converge to a globally optimal path as $n \to \infty$.*

Proof. Let π be an optimal path from $x_{I'}$ to x_G. If π lies entirely in $\mathscr{S}_{\mathscr{O}}$, then, it will be recovered by sampling in $\mathscr{S}_{\mathscr{O}}$. Otherwise, π must cross $\partial \mathscr{S}_{\mathscr{O}}$ at an outgoing configuration. Let x_1 be the outgoing configuration in π that first crosses $\partial \mathscr{S}_{\mathscr{O}}$. Then the subpath of π, from $x_{I'}$ to x_1, lies entirely in $\mathscr{S}_{\mathscr{O}}$ and can be recovered by sampling in $\mathscr{S}_{\mathscr{O}}$. By Lemma 1, a geodesic from x_1 to x_G is in $\mathscr{T}_G$. Thus, the GT algorithm can recover a geodesic from $x_{I'}$ to x_G by sampling in $\mathscr{S}_{\mathscr{O}}$. □

By exploiting what is known about geodesics in the Euclidean metric, we can provide an alternative characterization of a feasible sampling region for use in the GT by a robot with no differential constraints.

Theorem 3. *Let X be a d-dimensional C-space such that $d \in \mathbb{N}$ and $d \geq 2$. Let the initial obstacle space be X_{obs} and let $\mathscr{O} \not\subset X_{\text{obs}}$ be new obstacle information. For simplicity, assume that $\mathscr{O} \cap X_{\text{obs}} = \varnothing$. If*

1. *X is the Euclidean metric space,*
2. *$\mathscr{O} \subset R \subset X$,*
3. *R is convex, and*
4. *$x_{I'} \in R$*

then the GT algorithm will converge to a globally optimal path, π, as $n \to \infty$ by employing $\mathscr{T}_G$ with the previous X_{obs} and trimming $\mathscr{T}_G$ using the $\mathscr{O}$ information and then extending $\mathscr{T}_G$ in R.

Proof. In Euclidean space, an optimal path, π, is composed of straight lines and segments that follow the boundary of the obstacles. In particular, an optimal path from $x_{I'}$ to any point on ∂R, with respect to the new obstacle information, is a concatenation of path segments included among the following:

1. collision-free straight line paths from $x_{I'}$ to a point on the boundary of $\mathscr{O}$; i.e., a *visible* point on $\partial \mathscr{O}$ from $x_{I'}$.
2. any path along the boundary of $\mathscr{O}$ and the boundary of the convex hull of $\mathscr{O}$, and
3. collision-free straight line paths from $\partial \mathscr{O}$ to the visible boundary of R.

The convexity of R implies that all straight lines that begin and end in R are entirely contained in R. Any path that follows $\partial \mathscr{O}$ is entirely in R because $\mathscr{O} \subset R$. A globally optimal path from $x_{I'}$ to x_G will have to cross ∂R if $x_G \notin R$. Let the boundary point at this crossing be x_B. By the above discussion, the subpath from $x_{I'}$ to x_B can be recovered asymptotically by means of sampling in R with the new obstacle information. Consider the optimal subpath from x_B to x_G with respect to the old obstacle information X_{obs}. By the same considerations as above, this optimal subpath is made of a concatenation of segments from the list above but with respect to X_{obs}. Thus, it can be asymptotically recovered by means of $\mathscr{T}_G$ with information in X_{obs}. □

Note that, if $\mathscr{O} \cap X_{\text{obs}} \neq \varnothing$, then R would have to be a convex region containing the connected component of $\mathscr{O} \cup X_{\text{obs}}$ that contains $\mathscr{O}$. This connected set would then be used in the above proof in place of $\mathscr{O}$.

The region characterization from Definition 1 can be used to approximately determine where to sample from the geodesics obtained from the initial tree for planning problems. However, and as for the Euclidean case, alternative regions can be used if the particular dynamics are amenable to direct analysis. The following leads to a characterization of a new sampling region, R, for use in rebuilding $\mathscr{T}_G$ during replanning with the Dubins' vehicle. The Dubins' vehicle has three states, x- and y-position and orientation θ. The dynamics for the Dubins' vehicle are

$$\begin{aligned}\dot{x}(t) &= v\cos(\theta)\\ \dot{y}(t) &= v\sin(\theta)\\ \dot{\theta}(t) &= u, \quad |u| \leq \frac{v}{\rho},\end{aligned}$$

where v is the speed of the vehicle and ρ is the minimum turning radius. It is assume that both v and ρ are constant. The optimal trajectory between two configurations for these dynamics are discussed in [27]. The locally optimal trajectory defined by the above dynamics is one of six paths, RSL, LSR, RSR, LSL, RLR, and LRL, where L means left, R means right, and S means straight. Geodesics with respect to Euclidean length are characterized as concatenations of circular arcs and straight lines. The minimum turning radius for the Dubins' vehicle is denoted as ρ.

The following lemmas are useful in obtaining the main Dubins' vehicle result of this subsection.

Lemma 2. *Given a circular arc that begins at x_1 and ends at x_2, that has an angle strictly less than π radians, let x_c be the point where the tangent lines of the arc at x_1 and x_2 cross. Then, the* outer approximation *is defined as the union of the line from x_1 to x_c with the line from x_c to x_2. Then, the length of the outer approximation of a given circular arc is greater than or equal to the arc length.*

Lemma 3. *Given a circular arc that begins at x_1 and ends at x_2, define the* inner approximation *as the straight line connecting x_1 to x_2. Then, the length of the inner approximation of a given circular arc is less than or equal to the arc length.*

The proof of Lemma 2 and 3 follows directly from basic geometric considerations employing the triangular inequality and the convexity of circular arcs. It can be seen that the result can be extended to any convex curve and any inner approximation defined using points on the curve and joining them through lines in a similar way.

Now, using $\mathscr{O}$, a region that contains at least one valid path around $\mathscr{O}$ is defined.

Definition 3. $R_{\mathscr{O}}$ Region*: Define the region $R_{\mathscr{O}}$ as the smallest convex set that contains the union of $\mathscr{O}$ with circles of radius 2ρ centered at each corner of $\mathscr{O}$.*

Now, $R_{\mathscr{O}}$ is extended to contain feasible paths from $x_{I'}$ to the previous region $R_{\mathscr{O}}$.

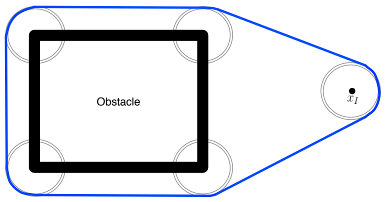


Fig. 2: The sampling region for a Dubins' vehicle as described in Definition 4.

Definition 4. R Region*: Define the region R, as the smallest convex region that contains $R_{\mathscr{O}}$ and $B(x_{I'}, 2\rho)$.*

Fig. 2 pictorially explains the Dubins' vehicle sampling region from Definition 4. The first Dubins' vehicle result of this subsection states the existence of valid trajectories in R.

Lemma 4. *The region R, as in Definition 4, contains at least one feasible Dubins' vehicle trajectory from $x_{I'}$ to any outgoing configuration on ∂R.*

Proof. First, it is shown that the Dubins' vehicle dynamics can be satisfied while traveling from $x_{I'}$ to any point on $\partial B(x_{I'}, 2\rho)$. To do this, note that the Dubins' vehicle can leave $x_{I'}$ and travel on a curve of radius ρ for π radians that places the vehicle on the boundary of R. This will put the Dubins' vehicle 2ρ way from $x_{I'}$ and the vehicle can now travel along $\partial B(x_{I'}, 2\rho)$. From $\partial B(x_{I'}, 2\rho)$, the Dubins' vehicle can travel to $R_{\mathscr{O}}$ along the tangent line forming the boundary of the convex hull between $R_{\mathscr{O}}$ and $B(x_{I'}, 2\rho)$. The Dubins' vehicle is now on $\partial R_{\mathscr{O}}$, the circles centered at the corners with radius 2ρ allow it to stay on $\partial R_{\mathscr{O}}$ traveling completely around the obstacle. Given that ∂R is 2ρ from $\partial\mathscr{O}$, implies that the vehicle can do a circular maneuver from some point on ∂R to get to a specific outgoing configuration on ∂R. Thus, R will contain at least one trajectory, not necessarily optimal, from $x_{I'}$ to any outgoing configuration on ∂R. □

The second result is that, the optimal path, from a configuration inside R to an arbitrary outgoing configuration on ∂R, will be entirely inside R.

Lemma 5. *Let σ be a feasible path for a Dubins' vehicle that starts at $x_{I'}$ and ends at an outgoing configuration $x_{\text{end}} \in \partial R$. If σ leaves and returns to R, then there exists another path, π, from $x_{I'}$ to x_{end}, that is entirely in R and that has a lower path length than σ.*

Proof. Define B_ρ to be a ball

1. of radius ρ
2. that is tangent to both ∂R and σ
3. that is contained in R, $B_\rho \subseteq R$.

Every time σ crosses ∂R, two such balls can be created. Take $B_{\rho 1}$ to be the B_ρ that is tangent to σ before leaving R and that is closest to where σ crosses back into R. Let x_1 be the configuration in σ that is tangent to $B_{\rho 1}$. Now, take $B_{\rho 2}$ to be the B_ρ that is tangent to σ after returning to R and that

Journal of Dynamic Systems, Measurement and Control. Received May 18, 2017;
Accepted manuscript posted July 24, 2018. doi:10.1115/1.4040970

is closest to where σ crossed outside of R. Let x_2 be the configuration in σ that is tangent to $B_{\rho 2}$.

Break σ into three subpaths: $\sigma_{I'1}$ from $x_{I'}$ to x_1, σ_{12} from x_1 to x_2, and $\sigma_{2\text{end}}$ from x_2 to x_{end}. Now, construct π as follows. Let $\pi_{I'1} = \sigma_{I'1}$ and $\pi_{2\text{end}} = \sigma_{2\text{end}}$. The subpath π_{12} is taken to be the path from x_1 that travels along $\partial B_{\rho 1}$ until it reaches ∂R, concatenated with the path that travels along ∂R until it reaches $\partial B_{\rho 2}$, and concatenated with the path that travels along $\partial B_{\rho 2}$ until it reaches x_2.

Now, π has been constructed to be a path, from $x_{I'}$ to x_{end}, that is entirely inside R. To prove that π has a shorter path length than σ, outer approximate every arc in π_{12}, π_{12}^o, and inner approximate every arc in σ_{12}, σ_{12}^i. This leaves us with two paths that start at x_1 and end at x_2 and that are only composed of straight lines. By construction, π_{12}^o is convex (because R and the outer approximation are convex). From the triangle inequality and the convexity of π_{12}^o, the length of σ_{12}^i cannot be longer than the length of π_{12}^o. Which means the length of π_{12} must be less than the length of σ_{12}. Combining this with the fact that the length of $\pi_{I'1}$ is equal to the length of $\sigma_{I'1}$ and the length of $\pi_{2\text{end}}$ is equal to the length of $\sigma_{2\text{end}}$, the length of π is less than the length of σ. Therefore, any feasible Dubins' vehicle path that leaves and returns to R can be shortened to a path that is entirely inside R. ☐

The main Dubins' vehicle result of this subsection says that R is a sampling region that allows the GT to recover the optimal path. This is stated more precisely in Theorem 4.

Theorem 4. *Consider a Dubins' vehicle at $x_{I'}$ for which minimum-length paths are to be found to x_G. Assume that the new obstacle, $\mathscr{O}$, is a convex polygon and does not intersect any other obstacles. Let R be as in Definition 4 and assume $x_G \notin R$. If the Goal Tree algorithm uses R as the new sampling region to rebuild $\mathscr{T}_G$, then it will converge to a globally optimal path as $n \to \infty$.*

Proof. By Lemma 4, there exists a path in R from $x_{I'}$ to any x_{end}, an outgoing configuration on ∂R. Then, by Lemma 5, if the optimal path π^* to any x_{end} leaves and returns to R, a shorter path can be found, which contradicts the optimality of π^*. Therefore π^* must be entirely in R and can be found by sampling in R. Let x_B be the outgoing configuration on ∂R where the optimal path from $x_{I'}$ to x_G crosses outside of R. From the above, the path from $x_{I'}$ to x_B can be recovered asymptotically by sampling in R with the information from $\mathscr{O}$. Now consider the path from x_B to x_G, this path is outside of R and can be asymptotically constructed by sampling outside of R with respect to X_{obs}. Thus, the optimal path from x_B to x_G can be recovered asymptotically from $\mathscr{T}_G$. ☐

5.2 Analysis of Grandparent-Connection

The same probabilistic completeness and asymptotic optimality results for the RRT* algorithm in [6], also hold true for the GP algorithm. Theorem 5 is a restatement of Theorems 23 and 38 from [6] but for the GP algorithm.

Proposition 1. *If the RRT* and GP algorithms solve the same path planning problem, using the same parameters, then the vertex sets of both graphs are equal, $V_n^{\text{RRT*}} = V_n^{\text{GP}}$, $\forall\, n \in \mathbb{N}$.*

Proof. Proof by induction. When $n = 1$, no Grandparent-Connection is possible, therefore, both the vertex and edge sets are identical. When $n = 2$, even though in the last step of the iteration the added vertex may connect to its grandparent the vertex is still added to the tree, making both vertex sets identical but the edges set different. When $n = 3$, because the vertex sets are the same, the vertex to be added, x_{new}, is the same for both algorithms, so is the set X_{near}. Then, if an edge exists between x_{new} and an $x \in X_{\text{near}}$ in one algorithm it also exists in the other. Therefore, even if the parents are different x_{new} will be added to both trees, maintaining the identical vertex sets. ☐

Theorem 5. *The GP algorithm is probabilistically complete. Furthermore, for any robustly feasible path planning problem $(X_{\text{free}}, x_i, X_{\text{goal}})$, there exist constants $a > 0$ and $n_0 \in \mathbb{N}$, both dependent only on X_{free} and X_{goal}, such that,*

$$\mathbb{P}(\{V_n^{\text{GP}} \cap X_{\text{goal}} \neq \emptyset\}) > 1 - e^{-an} \quad \forall\, n > n_0.$$

Also, if $\gamma > (2(1+\frac{1}{d})^{\frac{1}{d}}(\frac{\mu(X_{\text{free}})}{\zeta_d})^{\frac{1}{d}}$, then the GP algorithm is asymptotically optimal.

Proof. First, from Prop. 1, we have $V_n^{\text{RRT*}} = V_n^{\text{GP}}$. Then, by construction, the GP builds a connected graph. Therefore, the probabilistic completeness of the GP follows from the probabilistic completeness of the RRT*. Finally, the asymptotic optimality result comes from $\text{cost}_n^{\text{RRT*}}(x_i, x) \geq \text{cost}_n^{\text{GP}}(x_i, x)$ for all $x \in (V_n^{\text{RRT*}} = V_n^{\text{GP}})$. ☐

Lemma 6. *When X_{free} is convex, the GP recovers the optimal path from x_i to X_{goal}. Furthermore, define the visibility set of x_i as the subset of X_{free} such that $\forall\, x \in Vis(x_i)$ there exists a collision-free geodesic from x_i to x. Then, the GP algorithm will recover the optimal paths in $Vis(x_i)$.*

Proof. By construction, with a convex X_{free}, every vertex in the graph has x_i as its parent. Then, extending to other metric spaces, any node in the visibility set has x_i as its parent. ☐

6 Simulations

There are three different robot types simulated in this section: Euclidean metric, Dubins' vehicle, and a seven degree-of-freedom manipulator. The Euclidean metric and Dubins' vehicle simulations were run on a MacBook Pro with a 2.6GHz Intel Core i7 processor and 8 GB of memory. The manipulator simulation was run on a HP Z640 with 2.20 GHz Intel Xeon processor and 32 GB of memory. The metrics used for the comparison are the average time it takes for an algorithm to first return a path (Initial Time), and the cost of that initial path (Initial Cost). The percent difference compares the algorithms to the RRT* as a benchmark. All the algorithms in a specific simulation are run for the same amount of time, the cost of the best path returned by the algorithm at that time is labeled the Final Cost. There are two different types of environments: static and dynamic. In the static environments, the obstacles are completely known to the robot. The FR, GP, and FR-GP algorithms were developed for use in static environments. In a dynamic environment the obstacles are changing in a manner that is not completely known to the robot. In these simulations an obstacle will appear in the robot's path that was not previously known to the robot. The GT algorithm was developed for dynamic environments. While the nature of the FR algorithm does not make it suitable for dynamic environments, the GP algorithm integrates easily with the GT algorithm. The Goal Tree simulations find an approximation of the shadow for use as their sampling set when replanning. The shadow approximation, $\tilde{\mathscr{S}}$, is a collection of all the vertices removed from the original tree as a consequence of $\mathscr{O}$. The $\tilde{\mathscr{S}}$ can be made denser by adding samples via the primitive NewPointPathSet.

6.1 Euclidean Metric

This set of simulation results is for a point robot with no dynamics and Euclidean Metric edge cost. A collision checker that checks for the intersection between a line and a polygon is used. There are three environments, 25, 50, or 75 obstacles, which were chosen to compare how the different algorithms do with various obstacle densities. The results are an average of 25 simulations.

First, the Grandparent-Connection and Focused-Refinement are compared to the RRT*, RRT*-Smart, and RRT with path smoothing, Table 1. The algorithms were run for 100 seconds in the 25 obstacle environment, 120 seconds in the 50 obstacle environment, and 450 seconds in the 75 obstacle environment. The mean initial cost of each algorithm is compared to the RRT*'s mean initial cost to obtain the percent difference. A negative percent difference indicates that the cost is less than the RRT*'s cost. As expected the GP algorithms find the lowest cost initial path, without incurring much of a time increase. While the GP algorithms' initial path cost is about the same as the path found by an RRT with a post-processing path smoothing technique, the cost difference in the final path is much larger. The GP and FR algorithms have lower final path costs than the RRT*. It is also of interest to take note of how the increase in obstacles affects the performance on the various algorithms. The 75 obstacle environment while not increasing the best cost found much, does increase the cost to find that path. Specifically, note the initial cost for the GP algorithm with 25 obstacles is 14.78 while the initial cost with 75 obstacles is 17.90, only an increase of 3.12. But, the difference in time it take to reach those costs increases by 102.7 seconds. Also, note the increase in standard deviation. These trends can be seen across all the algorithms.

Table 2 compares replanning in the 50 obstacle environment using the GT with the RRT* algorithms. The replanning region R is a box that is just large enough to contain the new obstacle and current robot position. The Goal Tree's initial path cost and its path cost at the mean time the RRT* first finds a path are compared to the initial path cost of the RRT*. The GT algorithm out performs the RRT* in initial path cost by 13%. The cost of the final path found, after 80 seconds, by each algorithm is much closer, only a 2.7% difference.

6.2 Dubins' Vehicle

The simulations results in this section are for a Dubins' vehicle in the 25 and 50 obstacle environment. The cost minimized is the distance traveled by the vehicle. The primitive CollisionCheck discretizes the path and then checks that none of the configurations in the path are inside an obstacle. Each algorithm is averaged over 10 runs in each environments. Typical trees found by the RRT* and GT algorithm's are in Figs. 3a- 3f. The tree produced by the RRT* is the only one with many excessive loops in its final path.

Table 3 compares the Grandparent-Connection and Focused-Refinement algorithms to the RRT*, RRT*-smart, and RRT with path smoothing. The loops and curves of the path found by the RRT* increase the path cost significantly. This translates to the GP algorithms finding initial paths that have costs 50% less than the initial cost of the RRT*. The RRT with path smoothing and GP algorithms have initial path costs that are about the same. The excess loops and curves are not present in the GP algorithms final paths, and are reduced in the FR and RRT*-smart algorithms' final path. The GP algorithm has the highest average Initial Times, 63.5%. Running the RRT*-Smart to the GP algorithm's Initial Time (47.64 seconds for 25 obstacles and 130.5 for 50 obstacles) produces an average path cost of 131.68 for 25 obstacles, very close to the GP algorithm's cost, and 151.87 for the 50 obstacle case, which is 7% higher than the GP algorithm's cost at that same time. Our results indicate that it is best to use the first path returned by the GP algorithm. A benefit of executing the first path is removing the need to determine when to terminate the algorithm based on algorithm run-time or number of samples.

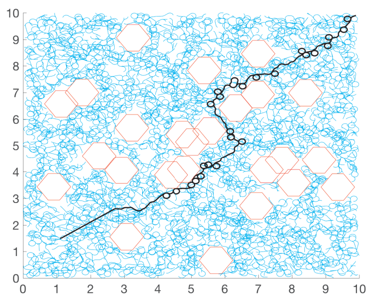
Figs. 4a and 4b show two typical trees produced by the RRT* and GT algorithms when replanning. The original environment has 25 obstacles, one unknown obstacle is found, therefore the replanning is done in a 26 obstacle environment. The empty space in Fig. 4b is from trimming. The Dubins' vehicle Goal Tree simulation comparison results are in Table 4. The GT has a mean initial cost that is higher than the RRT*, but that path is found much quicker. By the time the GT algorithm reaches the mean time it take the RRT* to find an initial path (6 seconds) the GT has reduced the path cost significantly. The GT, after 6 seconds, has a path cost 8% lower than the RRT* initial path cost. The comparison of the final path cost (at 80 seconds) drops to 3%.

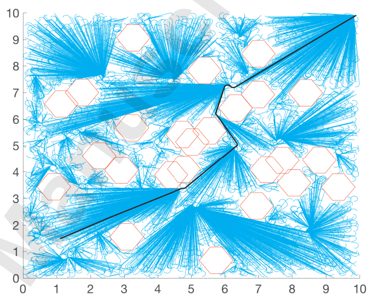
The Grandparent-Connection can be incorporated into the Goal Tree Algorithm. Table 5 compares the Goal Tree with Grandparent-Connection (GT-GP) to the Grandparent-Connection algorithm. The GT with GP was able to find an initial path quicker in comparison to the GP. At the mean time the GP finds an initial path to the goal, the GT with GP has a path to the goal whose cost is 5.4% better. The final cost for both algorithms, after running for 2500 seconds, is very close, with only a 0.48% difference.

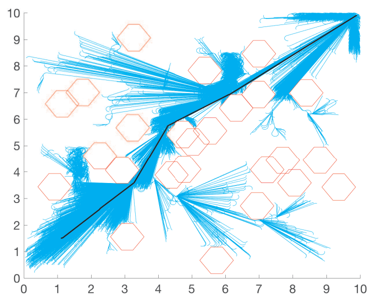
Journal of Dynamic Systems, Measurement and Control. Received May 18, 2017;
Accepted manuscript posted July 24, 2018. doi:10.1115/1.4040970
Copyright (c) 2018 by ASME

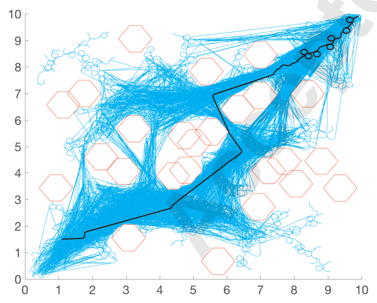
Table 1: Mean with standard deviation Euclidean metric results summarizing the comparison of the Grandparent-Connection and Focused-Refinement algorithms to the RRT*, RRT*-Smart, and RRT with path smoothing.

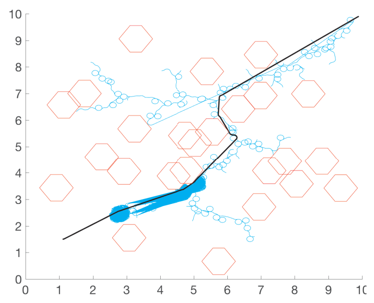
		RRT*	GP	FR	FR-GP	RRT*-Smart	Smoothing
Initial Cost	25	17.65 (0.83)	14.78 (0.55)	17.65 (0.83)	14.78 (0.55)	17.65 (0.83)	15.13 (0.65)
	50	18.08 (1.01)	15.99 (1.02)	18.08 (1.01)	15.99 (1.02)	18.08 (1.01)	16.39 (0.98)
	75	18.84 (1.23)	17.90 (1.20)	18.84 (1.23)	17.90 (1.20)	18.84 (1.23)	18.74 (1.23)
% Difference	25	0	-16.30	0	-16.30	0	-14.29
	50	0	-11.53	0	-11.53	0	-9.32
	75	0	-4.98	0	-4.98	0	-0.53
Initial Time	25	13.77 (7.39)	12.98 (6.62)	13.77 (7.39)	16.20 (9.31)	9.69 (6.07)	11.92 (5.23)
	50	37.72 (16.82)	32.92 (15.38)	35.61 (16.53)	32.87 (15.30)	22.10 (11.32)	34.00 (13.64)
	75	139.3 (47.4)	115.7 (55.4)	140.2 (47.4)	119.4 (57.1)	137.2 (46.7)	81.19 (30.72)
Final Cost	25	15.33 (0.32)	14.60 (0.34)	14.59 (0.37)	14.54 (0.27)	15.01 (0.59)	15.13 (0.65)
	50	15.93 (0.50)	15.45 (0.78)	15.15 (0.64)	14.98 (0.64)	15.80 (0.72)	16.39 (0.98)
	75	16.85 (0.90)	16.60 (1.05)	16.67 (0.98)	16.46 (0.94)	16.46 (0.94)	18.74 (1.23)
% Difference	25	0	-4.79	-4.89	-5.21	-2.15	-1.33
	50	0	-2.98	-4.91	-5.95	-0.78	2.92
	75	0	-1.52	-1.12	-2.31	-2.31	11.2


(a) RRT*


(b) Grandparent-Connection


(c) Grandparent and Focused-Refinement


(d) Focuses Refinement


(e) RRT*-Smart

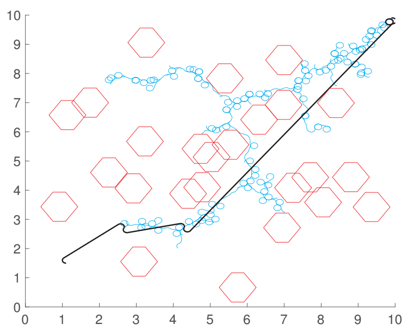
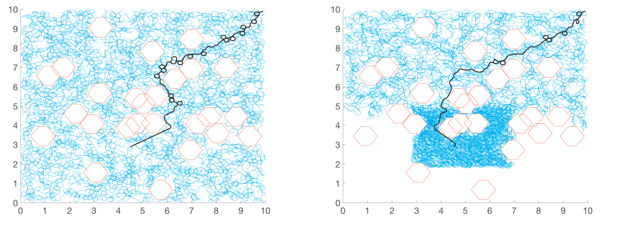

(f) RRT with Path Smoothing

Fig. 3: Typical Dubins' vehicle trees in the 25 obstacle environment found by the RRT*, Grandparent-Connection, Grandparent Connectio with Focused-Refinement, Focused-Refinement, RRT*-Smart, and RRT with path smoothing algorithms.

Table 2: Mean with standard deviation Euclidean metric results the comparing the Goal Tree and RRT* Algorithms.

		Time (s)	Cost	%
Initial	Goal Tree	0.69 (0.76)	7.23 (0.90)	-13.5
		2.02 (0.01)	7.19 (0.90)	-13.9
	RRT*	2.02 (2.16)	8.35 (1.33)	0
Final	Goal Tree	80.01 (0.01)	6.37 (0.14)	-2.8
	RRT*	80.01 (0.01)	6.55 (0.22)	0



(a) RRT* (b) Goal Tree

Fig. 4: Typical Dubins' vehicle trees after replanning in the 26 obstacle environment using the RRT* and GT algorithms.

6.3 Seven Degree-of-Freedom Manipulator

A Motoman seven degree-of-freedom manipulator is simulated using MoveIt! in Robot Operating System (ROS). The Goal Tree and RRT* algorithms are run in the Open Motion Planning Library (OMPL), using the default collision checker. The RRT* finds the best path for the manipulator in an obstacle free environment. Next, a box that is in conflict with the manipulator's path is introduced to the environment, Fig. 5. The Goal Tree and RRT* algorithms are used to replan and find a collsion-free path. The RRT* algorithm in the obstacle free environment is run 10 times. For each of these trees, the Goal Tree algorithm is run 10 times. Each 10 runs of the Goal Tree algorithm is referred to as a set, therefore there are 10 sets of 10 runs. Fig. 6 is a typical collision-free path found by the Goal Tree algorithm. For comparison, the RRT* algorithm is run in the same environment 25 times. The results of this setup are summarized in Table 6.

Table 6 compares the mean costs and initial iterations. The table shows that the Goal Tree algorithm out performs the RRT* in the quality of the initial path. While the mean cost of the final path found by the Goal Tree algorithm is better than the mean cost of the RRT* algorithm it is not significantly less. The percent difference is calculated as the difference between the mean costs divided by the RRT* mean cost. These percentages reiterate that the Goal Tree finds a much better mean initial path compared to the RRT*. The mean number of iterations to find an initial path is significantly lower for the Goal Tree compared to the RRT*. This is because the Goal Tree algorithm reuses part of the original tree, therefore it already starts with nodes in the tree. At the bottom of the table is the failure rate that was encountered

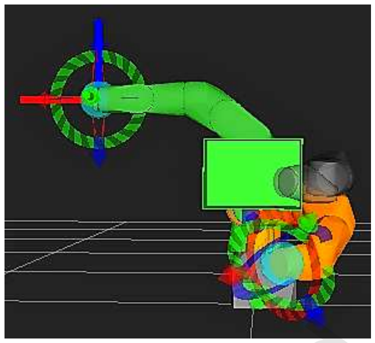


Fig. 5: The box is added to the environment so that it is in conflict with the manipulator's path.

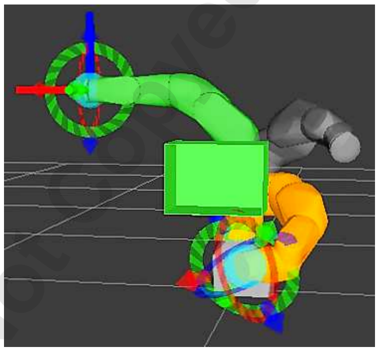


Fig. 6: The Goal Tree algorithm successfully replans to find a collision-free path.

during the simulations. The percentage is calculated as the number of failures divided by the total number of attempted runs for each algorithm. The Goal Tree algorithm is probably less likely to fail because it reuses part of the original tree, therefore has a shorter path to determine.

7 Conclusion

The Goal Tree, Focused-Refinement, and Grandparent-Connection are the three algorithms presented in this paper to improve the performance of the asymptotically optimal Rapidly-exploring Random Tree (RRT*). The GT and GP algorithms are proven to maintain the asymptotic optimally of the RRT*. The simulations for the Euclidean metric show that our algorithms improve performance in the initial cost. The Dubins' vehicle simulations show the GP and FR algorithms can lower the initial and final path cost significantly. The GP algorithm simulations revealed that it is best to use the initial path found due to the already low cost; continued sampling only decreased the cost marginally. The GT algorithm simulation, when run on the seven degree-of-freedom manipulator, showed the most improvement in the initial path cost and that the initial path was found more quickly.

Future work includes implementing the three algorithm modifications on robots in more complex environments. Another research direction is using the Goal Tree algorithm with

Table 3: Mean with standard deviation results summarizing the comparison of the Grandparent-Connection and Focused-Refinement algorithms to the RRT*, RRT*-Smart, and RRT with path smoothing for Dubins' Vehicle.

		RRT*	GP	FR	FR-GP	RRT*-Smart	Smoothing
Initial Cost	25	342.1 (39.1)	132.6 (7.8)	342.1 (39.1)	132.6 (7.8)	342.1 (39.1)	126.9 (3.0)
	50	365.6 (45.4)	140.9 (51.5)	365.6 (45.4)	158.3 (19.2)	365.6 (45.4)	131.6 (7.4)
% Difference	25	0	-61.23	0	-61.27	0	-62.90
	50	0	-61.45	0	-56.71	0	-64.00
Initial Time	25	9.46 (9.02)	47.64 (44.17)	9.05 (8.71)	34.27 (17.51)	14.63 (15.93)	18.73 (11.73)
	50	34.70 (19.79)	130.5 (102.4)	44.63 (26.37)	116.2 (44.1)	38.82 (21.21)	187.6 (226.8)
Final Cost	25	300.1 (61.1)	131.2 (7.2)	260.0 (52.0)	128.7 (4.0)	134.4 (6.0)	126.9 (3.0)
	50	349.4 (32.2)	152.3 (12.0)	280.9 (28.3)	149.9 (16.2)	151.9 (11.8)	131.6 (7.4)
% Difference	25	0	-56.29	-13.36	-57.12	-55.72	-57.70
	50	0	-56.40	-19.59	-57.09	-56.53	-62.33

Table 4: Mean with standard deviation results summarizing the comparison the Goal Tree and RRT* Algorithms for Dubins' Vehicles in the 25 obstacle environment.

		Time (s)	Cost	%
Initial	Goal Tree	0.16 (0.03)	260.8 (68.1)	4.5
		6.13 (0.03)	228.4 (44.5)	-8.4
	RRT*	6.10 (5.15)	249.6 (47.8)	0
Final	Goal Tree	80.03 (0.03)	209.8 (16.1)	-3.2
	RRT*	80.01 (0.01)	216.8 (21.8)	0

Table 5: Mean with standard deviation results summarizing the comparison of the Goal Tree and Grandparent-Connection Algorithms for Dubins' Vehicles in the 25 obstacle environment.

		Time (s)	Cost	%
Initial	GT-GP	4.97 (1.45)	106.9 (22.1)	3.2
		241.5 (0.3)	98.1 (15.1)	-5.4
	GP	240.6 (201.8)	103.6 (17.1)	0
Final	GT-GP	2500.6 (0.5)	92.92 (5.36)	-0.48
	GP	2500.7 (0.6)	93.36 (5.74)	0

multiple robots in the same environment. The GT could also handle the removal of an obstacle, such as running the existing path through a path smoothing algorithm. Another research direction is to extend the GT algorithm for partially known moving obstacles. These dynamic obstacles would introduce uncertainty into the problem that the motion planning tree would have to consider. One option is to handle the dynamic obstacles using a reactive collision avoidance algorithm and then fuse that information with the motion planning tree to determine how best to move.

Table 6: Mean results summarizing the comparison between the Goal Tree algorithm and RRT* algorithm for the seven degree-of-freedom manipulator.

Initial Cost	Goal Tree	8.83 (0.84)
	RRT*	9.66 (1.95)
Final Cost	Goal Tree	7.694 (0.65)
	RRT*	7.78 (0.93)
% Cost Difference	Initial Cost	-8.7%
	Final Cost	-1.1%
Initial Iterations	Goal Tree	19.54 (17.66)
	RRT*	143.0 (151.2)
% Failure	Goal Tree	-23.7%
	RRT*	-39.0%

Acknowledgements

This work was supported by Los Alamos National Laboratory and is approved for release under LA-UR-17-24020.

References

[1] LaValle, S. M., 2006. *Planning algorithms*. Cambridge University Press.

[2] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H., 1996. "Probabilistic roadmaps for path

planning in high-dimensional configuration spaces". *IEEE Transactions on Robotics and Automation,* **12**(4), pp. 566–580.

[3] Waringo, M., and Henrich, D., 2006. "Efficient smoothing of piecewise linear paths with minimal deviation". In IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, pp. 3867–3872.

[4] Carpin, S., and Pillonetto, G., 2005. "Motion planning using adaptive random walks". *IEEE Transactions on Robotics,* **21**(1), pp. 129–136.

[5] Sánchez, G., and Latombe, J.-C., 2003. "A single-query bi-directional probabilistic roadmap planner with lazy collision checking". In *International Symposium on Robotic Research*. Springer, pp. 403–417.

[6] Karaman, S., and Frazzoli, E., 2011. "Sampling-based algorithms for optimal motion planning". *International Journal of Robotics Research,* **30**(7), pp. 846–894.

[7] Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., and Teller, S., 2011. "Anytime motion planning using the RRT*". In IEEE Int. Conf. on Robotics and Automation, pp. 1478–1483.

[8] Perez, A., Karaman, S., Shkolnik, A., Frazzoli, E., Teller, S., and Walter, M. R., 2011. "Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms". In IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, pp. 4307–4313.

[9] Shkolnik, A., and Tedrake, R., 2011. "Sample-based planning with volumes in configuration space". *Computing Research Repository,* **arXiv:1109.3145**.

[10] Arslan, O., and Tsiotras, P., 2013. "Use of relaxation methods in sampling-based algorithms for optimal motion planning". In IEEE Int. Conf. on Robotics and Automation, pp. 2421–2428.

[11] Janson, L., and Pavone, M., 2015. "Fast marching tree: a fast marching sampling-based method for optimal motion planning in many dimensions". In International Symposium on Robotic Research, Vol. 34, pp. 883–921.

[12] Akgun, B., and Stilman, M., 2011. "Sampling heuristics for optimal motion planning in high dimensions". In IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, pp. 2640–2645.

[13] Islam, F., Nasir, J., Malik, U., Ayaz, Y., and Hasan, O., 2012. "RRT-smart: Rapid convergence implementation of RRT towards optimal solution". In IEEE Int. Conf. on Mechatronics and Automation, pp. 1651–1656.

[14] Koenig, S., and Likhachev, M., 2002. "$D^\star$ lite". In American Association for Artificial Intelligence, Vol. 15.

[15] Stentz, A., 1995. "The focussed D* algorithm for real-time replanning". In International Joint Conference on Artificial Intelligence, Vol. 95, pp. 1652–1659.

[16] Ferguson, D., Kalra, N., and Stentz, A., 2006. "Replanning with RRTs". In IEEE Int. Conf. on Robotics and Automation, pp. 1243–1248.

[17] Zucker, M., Kuffner, J., and Branicky, M., 2007. "Multipartite RRTs for rapid replanning in dynamic environments". In IEEE Int. Conf. on Robotics and Automation, pp. 1603–1609.

[18] Bruce, J., and Veloso, M., 2002. "Real-time randomized path planning for robot navigation". In IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, Vol. 3, pp. 2383–2388.

[19] Li, T.-Y., and Shie, Y.-C., 2002. "An incremental learning approach to motion planning with roadmap management". In IEEE Int. Conf. on Robotics and Automation, Vol. 4, pp. 3411–3416.

[20] Gayle, R., Klingler, K. R., and Xavier, P. G., 2007. "Lazy Reconfiguration Forest (LRF) - An approach for motion planning with multiple tasks in dynamic environments.". In IEEE Int. Conf. on Robotics and Automation, pp. 1316–1323.

[21] J, J, K. J., and LaValle, S. M., 2000. "RRT-connect: An efficient approach to single-query path planning". In IEEE Int. Conf. on Robotics and Automation, Vol. 2, pp. 995–1001.

[22] Otte, M., and Frazzoli, S., 2015. "*RRT^x*: Asymptotically optimal single-query sampling-based motion planning with quick replanning". *International Journal of Robotics Research*, pp. 797–822.

[23] Boardman, B., Harden, T., and Martínez, S., 2014. "Optimal kinodynamic motion planning in environments with unexpected obstacles". In Allerton Conf. on Communications, Control and Computing, pp. 1026–1032.

[24] Boardman, B., Harden, T., and Martínez, S., 2015. "Focused refinenment in the RRT*: Trading optimality for improved performance". In ASME Int. Design Eng. Technical Conferences & Computers and Information in Engineering Conference, p. V05CT08A006.

[25] Karaman, S., and Frazzoli, E., 2010. "Optimal kinodynamic motion planning using incremental sampling-based methods". In IEEE Int. Conf. on Decision and Control, pp. 7681–7687.

[26] Bruce, J. W., and Giblin, P. J., 1984. "An elementary approach to generic properties of plane curves". *Proceedings of the American Mathematical Society*, pp. 455–458.

[27] Dubins, L. E., 1957. "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents". *American Journal of Mathematics*, pp. 497–516.