Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2013-021 August 15, 2013

Optimal Bidirectional Rapidly-Exploring Random Trees

Matthew Jordan and Alejandro Perez

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Optimal Bidirectional Rapidly-Exploring Random Trees

Matthew Jordan and Alejandro Perez

Abstract In this paper we present a simple, computationally-efficient, two-tree variant of the RRT* algorithm along with several heuristics.

1 Introduction

Sampling-based planners such as the Rapidly-exploring Randomized Tree (RRT) [1] and the Probabilistic Road Map (PRM) [2] have been shown to be probabilistically complete and computationally efficient for many motion planning problems. The bidirectional (two-tree) [3] variants of the RRT algorithm have been successfully applied to complex instances of the motion planning where the platform is high-dimensional and must search for paths through narrow corridors, usually referred to as 'bug traps', while leveraging the full capabilities of the robot [4]. In particular, the RRT-Connect algorithm, a bidirectional version of the RRT that attempts to connect both trees with a greedy heuristic, has been empirically observed to show very fast convergence in these scenarios.

More recently, several algorithms with the *asymptotic optimality* property, i.e., almost sure convergence to the optimal solution, have been presented and investigated [5]. One of these algorithms, RRT*, a variant of the RRT algorithm, provides asymptotically-optimal solutions and requires only a constant factor more computation [6]. An asymptotically-optimal version of a bidirectional planner is of great appeal for high-dimensional motion planning problems as it has been empirically observed to yield great performance in these scenarios. However, the Connect heuristic [4], i.e., the procedure that connects trees and ultimately produces solutions with minimal coverage of the space, incurs a large computational burden on asymptotically-optimal planners as they consider $\log n$ neighbors at every iteration.

Matthew Jordan and Alejandro Perez
are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA {jordanm, aperez}@csail.mit.edu

In this paper, we present a simple, two-tree variant of RRT* with several heuristics that greatly improve its computational time. We show that a connecting procedure that meets the requirements shown by Karaman and Frazzoli [5] is needed to guarantee asymptotic-optimality. Second, we show that using a 'one neighbor' RRT-Connect procedure will result in a non-optimal solution with probability one. Third, we show that the computational ratio of the approach converges to a constant factor of that incurred by the RRT-Connect algorithm. Finally, we present various heuristics that lower this constant ratio without affecting the properties of the algorithm.

2 Related Work

The motion planning problem has been investigated for decades [7]. Despite the computational challenge of the problem [8], several efficient approaches have been proposed throughout the years[1], [2], [4], [9]. However, these approaches are focused on finding a single feasible solution. More recently, algorithms that account for optimality have received significant attention.

Applying graph search algorithms such as A* is one classic technique. The configuration space is discretized off-line and then searched for motion plans [9]. This approach has been successfully demonstrated on robotic cars [10], as well as on single-arm [11] and dual-arm [12] instances of the manipulation problem. Although these provide solutions of good quality in a timely manner, they are complete and optimal only with respect to the discretization. Moreover, when planning for complex tasks such as manipulation, the dexterity of the platform is sacrificed for computational time.

Another widely-used approach is to optimize or smooth trajectories after they are obtained. Gradient descent [13], stochastic optimization [14], and shortcutting heuristics [15] have been applied to trajectories performed by high-dimensional robot manipulators. However, these approaches are only locally optimal and decouple the planning and optimization procedures.

Most recently, several heuristics that speed up the convergence of asymptotically-optimal algorithms have been presented. These include, using constantly updated probabilistic models that serve as sampling distributions [16], relaxing optimality to produce sparse graphs [17], using branch-and-bound [18], sampling in task space [19], [20], approximating volumes of free space, and using lazy collision checking [21]. A bidirectional version of the RRT* has been recently presented [22] along with empirical results that indicate faster convergence to an initial solution and monotonic refinement of paths in the tree. However, although samples are rejected with an admissible heuristic, the algorithm still executes all procedures on $\log n$ neighbor nodes and attempts to connect the trees at every iteration regardless of the computational overhead incurred. Moreover, it is not clear how rejecting samples affects the exploration of the space, a property that is important for problems with more than one class of solutions.

The algorithm presented in this paper is a provably asymptotically-optimal bidirectional approach to the RRT* that leverages the rapid convergence of the RRT-Connect algorithm [4] and employs several heuristics to approximate the running time of its suboptimal equivalent and improve its convergence rate.

3 Background

3.1 Problem Definition

Let $X \subseteq \mathbb{R}^d$, referred to as the *configuration space*, be a compact set. The elements of X are called *configurations*. Let $X_{\text{obs}} \subset X$ be an open set called the *obstacle region* and let x_{goal} be the *goal configuration*. The set defined as $X_{\text{free}} := X \setminus X_{\text{obs}}$ is called the *obstacle-free space*. A *path* in X is a continuous function $\sigma : [0,1] \to X$. The path σ is said to be *collision-free*, if $\sigma(\tau) \in X_{\text{free}}$ for all $\tau \in [0,1]$. The set of all collision-free paths is denoted by Σ_{free}.

Given an initial configuration x_{init}, an obstacle region X_{obs}, and a goal configuration x_{goal}, the *motion planning problem* is to find a collision-free path $\sigma : [0,1] \to X_{\text{free}}$ that starts from the initial configuration $\sigma(0) = x_{\text{init}}$ and reaches the goal configuration $\sigma(1) = x_{\text{goal}}$.

Let $c : \Sigma_{\text{free}} \to \mathbb{R}_{\geq 0}$ be a *cost functional* that maps each collision-free trajectory to a non-negative cost. The *optimal motion planning problem* is to find a collision-free path $\sigma^* : [0,1] \to X_{\text{free}}$ that solves the motion planning problem, and minimizes the cost functional $c(\cdot)$, i.e., $c(\sigma^*) = \inf_{\sigma' \in \Sigma_{\text{free}}} c(\sigma')$.

3.2 The RRT* Algorithm

The RRT*, first introduced by Karaman and Frazzoli [5], is an incremental sampling-based motion planning algorithm that provides an asymptotic optimality guarantee, i.e., almost-sure convergence to optimal solutions. The reader is directed to the original publication [5] for details regarding the algorithm.

The primitives of the algorithm are described below.

Sampling: The `Sample` procedure returns independent uniformly distributed samples from the obstacle-free space.

Collision Checking: Given a path $\sigma : [0,1] \to X$, the $\texttt{CollisionFree}(\sigma)$ procedure returns true if σ is collision-free, i.e., $\sigma(\tau) \in X_{\text{free}}$ for all $\tau \in [0,1]$.

Extend: Given two configurations $x, x' \in X$, the $\texttt{Extend}(x, x')$ procedure returns a path $\sigma : [0,1] \to X$ that connects x and x', i.e., $\sigma(0) = x$ and $\sigma(1) = x'$. The `Extend` procedure used in this paper does so with a straight path, i.e., $\sigma(\tau) = (1-\tau)x + \tau x'$ for all $\tau \in [0,1]$.

Nearest Vertex: Given a set $V \subset X$ of configurations and a configuration $x \in X$, the $\texttt{Nearest}(V,x)$ procedure returns the configuration in V that is closest to x with respect to the Euclidean norm, i.e., $\text{argmin}_{x' \in V} \|x' - x\|$.

Near Vertices: Given a finite set $V \subset X$ of configurations and a configuration $x \in X$, roughly speaking, the $\texttt{Near}(V,x)$ procedure returns the set of all configurations in V that are close to x, where we define closeness as follows. Letting $n := |V|$ be the number of configurations in V, we define $\texttt{Near}(V,x) := \left\{x' \in V : \|x' - x\| \leq \gamma((\log n)/n)^{1/d}\right\}$, where γ is a constant independent of n [5]. In other words, $\texttt{Near}(V,x)$ is the set of all configurations in V that lie inside a ball of volume $O((\log n)/n)$ centered at x.

Cost Functional: Given a vertex x of the tree maintained by the RRT* algorithm, we let $\texttt{Cost}(x)$ be the cost of the unique path that starts from the root vertex x_{init} and reaches x along the vertices of the tree. With a slight abuse of notation, we denote the cost $c(\sigma)$ of a path $\sigma : [0,1] \to X$ as $\texttt{Cost}(\sigma)$ for notational simplicity.

3.3 Bidirectional (two-tree) RRT method

Bidirectional variants of sampling-based algorithms are often applied to problems with challenging regions such as narrow corridors, 'bug traps', or high-dimensional configuration spaces with numerous obstacles. In very general terms, constructing opposing trees from x_{init} and x_{goal} can lead to paths resulting from their connection without requiring fine coverage of X_{free}. The first bidirectional variant of RRT in the literature iterated by incrementally extending both trees towards random samples [1]. Soon after, the RRT-Connect planner was proposed as a much greedier version of this algorithm. The algorithm employs the Connect heuristic at the end of every iteration to attempt to create a branch between trees [4]. Although there are no theoretical results describing its rate of convergence, it has been observed to very rapidly provide solutions for high-dimensional instances of the motion planning problem such as manipulation.

4 Optimal Bidirectional Rapidly-Exploring Random Trees

We present a simple bidirectional variant to the RRT* algorithm that is provably asymptotically-optimal and tailored to approximate the computation time of a standard bidirectional RRT algorithm. In this section we present the resulting algorithm along with additional procedures, several heuristics, and modifications tailored to reduce its computational overhead and increase its convergence rate. These are outlined below.

Admissible Heuristic For a vertex x, let c^*_x be the cost of the optimal path that starts at x and reaches the target vertex x_{target}. The cost-to-go function serves as an

equivalent to the admissible heuristic employed by A* planning algorithms. This value is considered to avoid unnecessary procedures that cannot possibly result in monotonic improvement towards the optimal solution. We use the straight-line Euclidean distance between x and x_{target} as our `CostToGo` function.

Sorting We consider lists of cost, configuration, and path triplets, i.e., triplets of the form (c_i, x_i, σ_i), where $c_i \in \mathbb{R}_{\geq 0}$, $x_i \in X$, and $\sigma_i \in \Sigma_{\text{free}}$. Given a list L of such pairs, the L.`sort`() method sorts the elements of L according to their cost in the ascending order. Sorting vertices within a ball allows us to reduce the number of collision checking and connecting procedures required when selecting parent vertices and rewiring the tree. The algorithm iterates through the list until a feasible vertex is found. This allows RRT* to match the computational complexity of the RRT as one operation is performed in the best case, as opposed to the expected $\frac{\zeta_d r_n^d}{\mu(X_{\text{free}})} n$.

Conditional Activation The algorithm only considers a single nearest neighbor until an initial feasible solution is found. This allows the algorithm to match its time to initial solution to its suboptimal equivalent and to use the remaining available time to approximate the optimal solution. More specifically, only vertices returned by the `Nearest` procedure are considered. After a connection is made between the two trees, the algorithm considers all vertices returned by the `Near` procedures.

Conditional Graph Constructing Procedures The `CostToGo` procedure is employed before all graph constructing operations (see Lines 21 and 27 of Algorithm 1, and Line 10 of Algorithm 2). Only operations that can result in possible cost improvement are executed. This greatly reduces the computational overhead over a suboptimal planner as it decreases the likelihood that all $O(\log n)$ operations are carried out during each of these graph constructing steps.

Best Case $O(1)$ Collision-checking The collision checking procedure iterates through a sorted list of $(\log n)$ neighbors. Because this list is sorted by global cost, the procedure can terminate as soon as collision-free edge is found. This results in a best case of $O(1)$ collision-checking procedures per neighborhood and a worst case of $O(\log n)$.

Conditional Connecting The Connect heuristic allows for rapid convergence to solutions with minimal coverage of the space. However, due to its greedy behavior, i.e., extending until a connection or obstacle is found, the procedure incurs a significant computational overhead for optimal planners. The algorithm presented alleviates this problem in two ways. First, it sorts neighbor nodes based on global cost before attempting to connect the trees. This allows for a best case of $O(1)$ connection attempts and a wrost case of $O(\log n)$. Second, the algorithm only attempts to connect the trees if it is determined that the resulting path will provide a lower cost than the current best one in the graph.

Lazy Vertex Contraction The algorithm lazily contracts vertices in the current best paths in the graph. The process of improving a single path is often times referred to as 'smoothing' or 'shortcutting' [15]. However, the algorithm presented contracts various parts of a graph as the algorithm iterates. Note that this procedure is different from improving a single path. We use the term vertex contraction as is done in the graph theory literature [23]. The procedure randomly selects two vertices within a connected path in our graph, and checks to see whether or not a linear obstacle free path exists between the two. Vertices are contracted if such an edge exists.

Branch-and-bound: The branch-and-bound algorithm is used for many domains in optimization and artificial intelligence [24]. In our approach, the algorithm works by keeping track of all vertices with additive costs larger than the cost of the best solution in the graph and periodically removing them from the trees. More specifically, let V' denote the set of vertices x in T_a to be removed, then $V' = \{x \in V \,|\, \texttt{Cost}(x) + \texttt{CostToGo}(x_{\text{goal}}) > c_{\text{best}}\}$.

4.1 Optimal Bidirectional Rapidly-Exploring Random Trees

The resulting algorithm is presented in Algorithm 1. Two trees are maintained. These are denoted as T_a and T_b. In the first phase, T_a iterates by sampling a new configuration x_{rand} from X_{free} (Line 5–9), extending towards it from the nearest vertex in the tree (Lines 10–11), and computing the set X_{near} of all vertices that are close to the resulting x_{new} (Line 12). In the second phase, the algorithm proceeds to calculate a parent vertex for x_{new} (Lines 13–25) and to attempt to rewire the branches in X_{near} (Lines 26–31). These operations are carried out as described by Karaman and Frazzoli [5] with slight modifications that invoke the heuristics described above (Lines 18, 21, 27).

After the current tree is finished with an RRT* iteration, it attempts to connect the closest vertex in the opposing tree, x_{connect}, to the resulting vertex, x_{new}. The `ConnectGraph` procedure is a variant of the Connect heuristic [4] tailored to provably result in a connected graph [25]. The procedure is described in Algorithm 2. Vertices x_{i} and x_{f} are taken as input. These are evaluated as a typical RRT* iteration where x_{f} plays the role of x_{rand}. A set of vertices is calculated from the opposing tree (Lines 4–8). At this point, the algorithm goes through the sorted list and attempts to connect to the vertex on the other tree only if the resulting solution is of cost lower than the current best, c_{best}. Finally, the algorithm updates its current best solution if necessary, prunes the tree with the `BranchAndBound` procedure described above, and swaps the trees.

Algorithm 1: Optimal Bidirectional RRT $(x_{\text{init}}, x_{\text{goal}})$

$V \leftarrow \{x_{\text{init}}, x_{\text{goal}}\}$; $E \leftarrow \emptyset$;
$T_a \leftarrow (x_{\text{init}}, E)$; $T_b \leftarrow (x_{\text{goal}}, E)$;
$c_{best} \leftarrow \infty$; $\sigma_{\text{best}} \leftarrow \emptyset$;
for $i = 1$ *to* N **do**
  $p \leftarrow \texttt{UniformSample}([0,1])$;
  if $p < P_{TS}$ **then**
    $x_{\text{rand}} \leftarrow \texttt{SampleTaskSpace}$;
  else
    $x_{\text{rand}} \leftarrow \texttt{Sample}$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(T_a, x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  $X_{\text{near}} \leftarrow \texttt{Near}(T_a, x_{\text{new}})$;
  $L_{\text{near}} \leftarrow \emptyset$;
  for $x_{\text{near}} \in X_{\text{near}}$ **do**
    $\sigma_{\text{near}} \leftarrow \texttt{Steer}(x_{\text{near}}, x_{\text{new}})$;
    $c_{\text{near}} \leftarrow \texttt{Cost}(x_{\text{near}}) + \texttt{Cost}(\sigma_{near})$;
    $L_{\text{near}} \leftarrow L_{\text{near}} \cup (\, (c_{\text{near}}, x_{\text{near}}, \sigma_{\text{near}}) \,)$;
  $L_{\text{near}}.\texttt{sort}()$;
  for $(c_{\text{near}}, x_{\text{near}}, \sigma_{\text{near}}) \in L$ **do**
    if $\texttt{CollisionFree}(\sigma_{\text{near}})$ **then**
      if $c_{\text{near}} + \texttt{CostToGo}(x_{\text{near}}) < c_{\text{best}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$;
        $V \leftarrow V \cup (x_{\text{new}})$;
        $E \leftarrow E \cup (\, (x_{\text{min}}, x_{\text{new}}) \,)$;
        break;
  for $(c_{\text{near}}, x_{\text{near}}, \sigma_{\text{near}}) \in L$ **do**
    if $\texttt{Cost}(x_{\text{new}}) + c_{\text{near}} < \texttt{Cost}(x_{\text{near}})$ **then**
      if $\texttt{CollisionFree}(\sigma_{\text{near}})$ **then**
        $x_{\text{oldparent}} \leftarrow \texttt{Parent}(E, x_{\text{near}})$;
        $E \leftarrow E \backslash (\, (x_{\text{oldparent}}, x_{\text{near}}) \,)$;
        $E \leftarrow E \cup (\, (x_{\text{new}}, x_{\text{near}}) \,)$;
  $x_{\text{connect}} \leftarrow \texttt{Nearest}(T_b, x_{\text{new}})$;
  $(c_{\text{sol}}, \sigma_{\text{sol}}) \leftarrow \texttt{ConnectGraphs}(T_b, x_{\text{connect}}, x_{\text{new}})$;
  if $c_{\text{sol}} < c_{\text{best}}$ **then**
    $c_{\text{best}} \leftarrow c_{\text{sol}}$;
    $\sigma_{\text{best}} \leftarrow \sigma_{\text{sol}}$;
  $p \leftarrow \texttt{UniformSample}([0,1])$;
  if $p < P_{VC}$ **then**
    $\texttt{RandomVertexContraction}(\sigma_{\text{best}})$;
  $\texttt{BranchAndBound}(T_a, T_b)$;
  $\texttt{SwapTrees}(T_a, T_b)$;
return $T_a, T_b = (V, E)$.

Algorithm 2: `ConnectGraphs`$(T_b, x_\text{i}, x_\text{f})$

$x_\text{new} \leftarrow$ `Steer`(x_i, x_f);
$X_\text{near} \leftarrow$ `Near`(T_b, x_new);
$L_\text{near} \leftarrow \emptyset$;
for $x_\text{near} \in X_\text{near}$ **do**
    $\sigma_\text{near} \leftarrow$ `Steer`$(x_\text{near}, x_\text{f})$;
    $c_\text{near} \leftarrow$ `Cost`(x_near) + `Cost`(σ_{near}) + `Cost`(x_f);
    $L_\text{near} \leftarrow L_\text{near} \cup (\,(c_\text{near}, x_\text{near}, \sigma_\text{near})\,)$;
L_near.`sort`();
for $(c_\text{near}, x_\text{near}, \sigma_\text{near}) \in L$ **do**
    if c_near + `CostToGo`$(x_\text{near}) < c_\text{best}$ **then**
        if `CollisionFree`(σ_near) **then**
            $E \leftarrow E \cup ((x_\text{near}, x_\text{connect}))$;
            $\sigma_\text{connect} \leftarrow$ `GeneratePath`$(x_\text{near}, x_\text{connect})$;
            return $(c_\text{near}, \sigma_\text{connect})$;
return NULL

5 Analysis

In this section the properties of the algorithm presented are evaluated. First, the probabilistic completeness and exponential decay of the probability of failure are considered. Second, the optimality properties are shown. Finally, the computational complexity of the bidirectional RRT and the bidirectional RRT* are compared.

5.1 Probabilistic Completeness

As shown by LaValle and Kuffner, RRT is probabilistically complete and exponentially converges to a uniform distribution over X_free [1]. It has also been shown that these same properties are present in the bidirectional version of RRT [4]. Additionally, Karaman and Frazzoli have shown that the optimal variants of this algorithm, i.e., RRG, RRT*, inherit these properties as well. The proposed connecting procedure meets the requirements needed for connected graph construction [25] and asymptotically-optimal trees [5]. Therefore, the result follows directly from the probabilistic completeness of RRT, bidirectional RRT, and RRT*.

5.2 Asymptotic Optimality

In this section, we analyze the optimality of two algorithms. It is shown that a bidirectional version of an asymptotically-optimal algorithm, in this case, a two-tree version of RRT* constructed with the Connect heuristic [4], will converge to a solu-

tion of non-optimal cost almost surely. Additionally, it is shown that the algorithm presented in this paper will converge to the optimal solution with probability one.

Theorem 1 (Non-optimality of a nearest neighbor Bidirectional RRT*) *A two-tree method of RRT* that employs the Connect heuristic [4] on the nearest neighbor is not asymptotically optimal.*

The proof of this theorem is similar to that of Theorem 33 by Karaman and Frazzoli [5]. Clearly, each tree follows the RRT* procedure when adding branches to the tree. However, the Connect heuristic [4] attempts to create an edge from a vertex in T_b to the locally nearest vertex in T_a, i.e., additive costs incurred by vertices within radius $r_n = \gamma ALG(\frac{\log n}{n})^{1/d}$ are not considered when choosing a parent. Therefore, the resulting vertices and their corresponding branches are identical to those obtained by an RRT iteration where the newest vertex in T_a, x_{new}, corresponds to the sample. In fact, solutions are only obtained by following this procedure. Therefore, every path will contain at least one branch whose construction does not meet the requirements specified by Karaman and Frazzoli for asymptotically-optimal planners, i.e., PRM*, RRG, RRT*.

Proof (Sketch) As shown by Muthukrishnan and Pandurangan [25], a random geometric graph with n vertices constructed by connecting all vertices within a distance $d_n = \gamma'(\log n/n)^{1/d}$ will result in a connected graph with probability one as $n \to \infty$ if $\gamma' > \gamma_1$ where γ_1 is a lower bound. They also show that if $\gamma' < \gamma_1$, the resulting graph will be disconnected almost surely [25]. Edges between both graphs are constructed with the Connect procedure [4], which considers a single vertex. This is equivalent to constructing an edge in a graph with $\gamma' = 0$. Therefore, $\gamma' < \gamma_1$. As shown by Karaman and Frazzoli, asymptotic optimality is only obtained if the tree is constructed from an RRG that converges to a connected graph [5]. When $\gamma' = 0$, the probability that a connected graph is returned as $n \to \infty$ is zero [25]. Therefore, $\mathbb{P}(\{\lim_{n\to\infty} \sigma'_{T_a,T_b} = \sigma^*_{T_a,T_b}\}) = 0$. ■

Theorem 2 (Asymptotic optimality of Bidirectional RRT*) *If $\gamma ALG \leq (2(1 + 1/d))^{1/d}((\mu(X_{free}))/\varepsilon_d)^{1/d}$, ALG is asymptotically optimal.*

Proof (Sketch) The proof of this theorem follows directly from Theorem 38 by Karaman and Frazzoli [5]. At the end of every iteration, each tree attempts to create an edge from a vertex within radius $r_n = \gamma ALG(\frac{\log n}{n})^{1/d}$ to the most recent vertex in the opposing tree. Indeed, this procedure is equivalent to an RRT* iteration where the vertex in the opposing tree assumes the role of x_{rand}. Therefore, the $\mathbb{P}(\{\lim_{n\to\infty} \sigma'_{T_a,T_b} = \sigma^*_{T_a,T_b}\}) = 1$ result follows directly from Lemmas 56, 71, and 72 by Karaman and Frazzoli [5].

■

5.3 Computational Complexity

In this section, the computational complexity of the bidirectional RRT is compared to that of bidirectional RRT*. It is shown that these algorithms converge to a constant number of calls to the collision checking, nearest neighbor search, and extension procedures per iteration.

Theorem 3 (Computational ratio of bidirectional RRT and bidirectional RRT*) *There exists a constant* ϕ *such that* $\limsup_{n\to\infty} \mathbb{E}[\frac{M_n^{BiRRT^*}}{M_n^{BiRRT}}] \leq \phi$.

This result follows directly from Theorem 18 by Karaman and Frazzoli [6]. The ratio of steps performed by the RRT and RRT* algorithms converges to a constant. More specifically, There exists a constant ϕ such that $\limsup_{n\to\infty} \mathbb{E}[\frac{M_n^{RRT^*}}{M_n^{RRT}}] \leq \phi$. As shown by Karaman and Frazzoli with Lemma 42 [5], the expected number of vertices in a ball of radius r_n centered at vertex x_n is no more than $\frac{\zeta_d r_n^d}{\mu(X_{\text{free}})} n$. The two-tree versions of RRT and RRT* employ an extra procedure at every iteration to attempt connecting the trees. This procedure incurs the same computational cost of a single-tree iteration where the extension step size parameter is large enough, i.e., $\eta \geq \text{diam}(X_{\text{free}})$. Therefore, the computational cost is trivially inherited from Theorem 18. ■

6 Conclusion

Incremental sampling-based planners with the asymptotic optimality property have been successfully applied for various robotic applications. Most recently, there is increased interest in the development of heuristics that very rapidly converge to initial solutions allowing monotonic improvements within specified time budgets. Two-tree methods have been empirically observed to yield great performance in high-dimensional scenarios. Therefore, an asymptotically-optimal version of a bidirectional planner is of great appeal to motion planning for mobile manipulation.

This paper presented a simple, two-tree variant of the RRT* algorithm along with several heuristics and modifications that greatly improve its computational time. We showed that our connecting procedure guarantees asymptotic-optimality and showed that using a 'one neighbor' RRT-Connect procedure will result in a non-optimal solution with probability one. Finally, we showed that the computational ratio of the approach converges to a constant factor of that incurred by the RRT-Connect algorithm.

Additional information such as videos, images, data, extra material, and stand-alone code will be made available at

http://people.csail.mit.edu/aperez/obirrt.

References

1. S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, pp. 378–400, May 2001.
2. L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
3. S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.
4. J. J. K. Jr. and S. M. Lavalle, "Rrt-connect: An efficient approach to single-query path planning," in *Proc. IEEE Intl Conf. on Robotics and Automation*, pp. 995–1001, 2000.
5. S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, June 2011.
6. S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Robotics: Science and Systems (RSS)*, (Zaragoza, Spain), June 2010.
7. T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.
8. J. Canny and J. H. Reif, "New lower bound techniques for robot motion planning problems," in *IEEE Symp. on Foundations of Computer Science (FoCS)*, (Los Angeles, CA), pp. 49–60, October 1987.
9. M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *J. Artificial Intelligence*, vol. 172, pp. 1613–1643, Sept. 2008.
10. M. Likhachev and D. Ferguson, "Planning long dynamically-feasible maneuvers for autonomous vehicles," *Int'l J. of Robotics Research*, vol. 28, pp. 933–945, August 2009.
11. B. Cohen, G. Subramanian, S. Chitta, and M. Likhachev, "Planning for manipulation with adaptive manipulation primitives," in *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA)*, May 2011.
12. B. Cohen, S. Chitta, and M. Likhachev, "Search-based planning for dual-arm manipulation with upright orientation constraints," in *IEEE International Conference on Robotics and Automation*, May 2012.
13. N. Ratliff, M. Zucker, J. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA)*, pp. 489–494, May 2009.
14. M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA)*, May 2011.
15. K. Hauser and V. Ng-Thow-Hing, "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts," in *IEEE Conference on Robotics and Automation (ICRA)*, May 2010.
16. M. Kobilarov, "Cross-entropy motion planning," *International Journal of Robotics Research*, 2012.
17. A. Dobson, A. Krontiris, and K. E. Bekris, "Sparse roadmap spanners," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, June 2012 2012.
18. S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Real-time motion planning using the RRT*," in *IEEE Conference on Robotics and Automation (ICRA)*, April 2011.
19. A. Shkolnik and R. Tedrake, "Path planning in 1000+ dimensions using a task-space voronoi bias," in *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ICRA'09, (Piscataway, NJ, USA), pp. 2892–2898, IEEE Press, 2009.
20. J. Jeon, S. Karaman, and E. Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*," in *IEEE Conference on Decision and Control (CDC)*, 2011.
21. A. Perez, S. Karaman, M. Walter, A. Shkolnik, E. Frazzoli, and S. Teller, "Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

22. B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimension," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'11)*, September 2011.
23. S. S. Skiena, *Implementing discrete mathematics : combinatorics and graph theory with Mathematica*. Redwood City (Calif.), Menlo Park (Calif.), Reading (Mass.): Addison-Wesley publ, 1990.
24. J. Clausen, "Branch and bound algorithms - principles and examples," 2003.
25. S. Muthukrishnan and G. Pandurangan, "The bin-covering technique for thresholding random geometric graph properties," in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '05, (Philadelphia, PA, USA), pp. 989–998, Society for Industrial and Applied Mathematics, 2005.