Journal name
():–
©The Author(s) 2010
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI:10.1177/1081286510367554
http://mms.sagepub.com

RRT^{X}: Asymptotically Optimal Single-Query Sampling-Based Motion Planning with Quick Replanning

Michael Otte[*] **and Emilio Frazzoli**
Massachusetts Institute of Technology, Cambridge MA 02139, USA

Abstract
Dynamic environments have obstacles that unpredictably appear, disappear, or move. We present the first sampling-based replanning algorithm that is asymptotically optimal and single-query (designed for situation in which *a priori* offline computation is unavailable). Our algorithm, RRT^{X}, refines and repairs the *same* search-graph over the *entire* duration of navigation (in contrast to previous single-query replanning algorithms that prune and then regrow some or all of the search-tree). Whenever obstacles change and/or the robot moves, a graph rewiring *cascade* quickly remodels the existing search-graph and repairs its shortest-path-to-goal sub-tree to reflect the new information. Both graph and tree are built directly in the robot's state space; thus, the resulting plan(s) respect the kinematics of the robot and continue to improve during navigation. RRT^{X} is probabilistically complete and makes no distinction between local and global planning, yet it reacts quickly enough for real-time high-speed navigation though unpredictably changing environments. Low information transfer time is essential for enabling RRT^{X} to react quickly in dynamic environments; we prove that the information transfer time required to inform a graph of size n about an ϵ-cost decrease is $O\left(n \log n\right)$ for RRT^{X}—faster than other current asymptotically optimal single-query algorithms (we prove RRT* is $\Omega\left(n(\frac{n}{\log n})^{1/D}\right)$ and RRT$^{\#}$ is $\omega\left(n \log^2 n\right)$). In static environments RRT^{X} has the same amortized runtime as RRT and RRT*, $\Theta\left(\log n\right)$, and is faster than RRT$^{\#}$, $\omega\left(\log^2 n\right)$. In order to achieve $O\left(\log n\right)$ iteration time, each node maintains a set of $O\left(\log n\right)$ expected neighbors, and the search-graph maintains ϵ-consistency for a predefined ϵ. Experiments and Simulations confirm our theoretical analysis and demonstrate that RRT^{X} is useful in both static and dynamic environments.

Keywords
real-time, asymptotically optimal, graph consistency, motion planning, replanning, dynamic environments, shortest-path

1. Introduction

Now, more than ever before, autonomous robots are affecting the lives of the average citizen; yet, most current mainstream deployments are also invisible to the mainstream population. Large scale deployments are limited to highly controlled or civilian-free settings such as factories, warehouses, crop fields, space, disaster clean-up, or the battlefield. However,

[*] Corresponding author; e-mail: ottemw@mit.edu

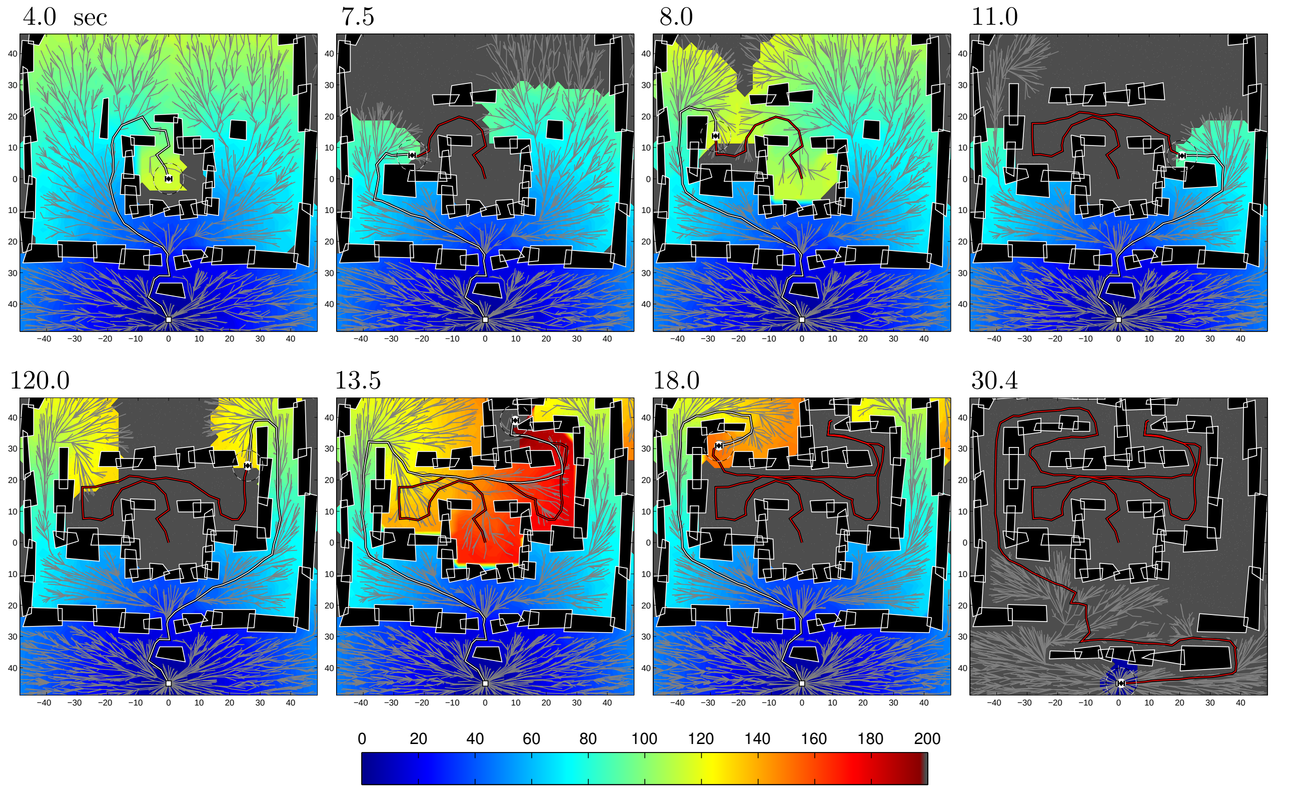
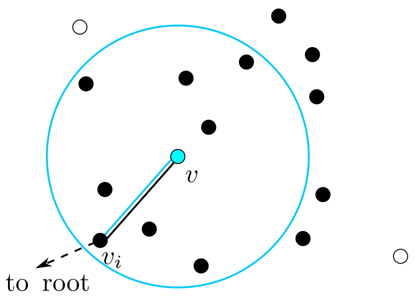


Fig. 1. Holonomic robot using RRT^X to move from start to goal (white square) while repairing its shortest-path tree (light-gray) vs. obstacle changes. Color/grayscale is cost-to-goal. Planned and executed paths are white and red/dark gray. Obstacles are black with white outlines. Time (in seconds) appears above each sub-figure. See `http://tinyurl.com/l53gzgd` for videos of this and other vehicles.

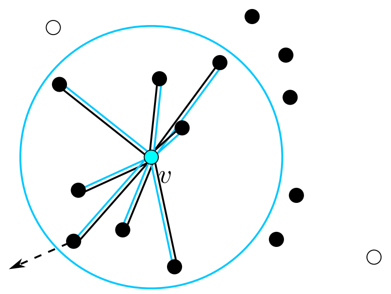
multiple industrial efforts are underway to develop self-driving cars, package delivery aircraft, and countless other every-day applications. This new breed of consumer robotics will interact directly with average humans; and therefore, must be designed to thrive in the relatively *unpredictable* environments in which people live, work, and play. Despite many recent advances, there is still a need for algorithms that enable efficient kinematic real-time autonomous navigation through dynamic environments.

We present RRT^X, the first sampling-based replanning algorithm that is both asymptotically optimal and designed for situation in which *a priori* offline computation is unavailable. RRT^X enables real-time kinodynamic navigation in dynamic environments, i.e., in environments with obstacles that *unpredictably* appear, move, and vanish. In particular, RRT^X calculates an initial plan, then continually refines it toward the optimal solution during navigation, while also repairing it on-the-fly whenever changes to the obstacle set are detected. The plan is stored in a graph and its asymptotically optimal shortest-path sub-tree. An important quality of the algorithm is that it uses the *same* graph/tree even when obstacles change—quickly remodeling and repairing it instead of pruning disconnected branches away (see Figure 1). Both graph and sub-tree exist in the robot's state space, and the tree is rooted at the goal state (allowing it to remain valid as the robot's state changes during navigation). Whenever obstacle changes are detected, e.g., via the robot's sensors, *rewiring* operations *cascade* down the affected branches of the tree repairing the graph and remodeling the shortest-path tree.

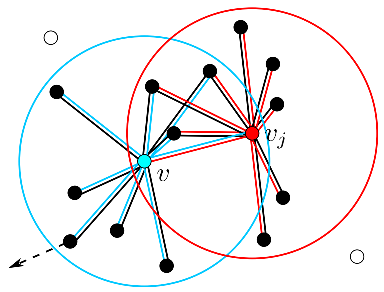
Although RRT^X is designed for dynamic environments, it is also competitive in static environments—where it is asymptotically optimal and has an expected amortized per iteration runtime of $\Theta(\log n)$ for graphs with n nodes. This is similar to RRT and RRT* $\Theta(\log n)$ and faster than RRT$^\#$ $\Theta\left(\log^2 n\right)$.



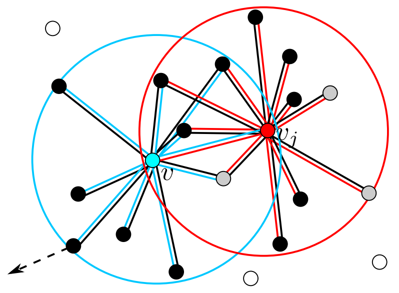
(A) v (light blue) initially achieves the shortest cost-to-goal through neighbor v_i, so v_i is the initial parent of v.



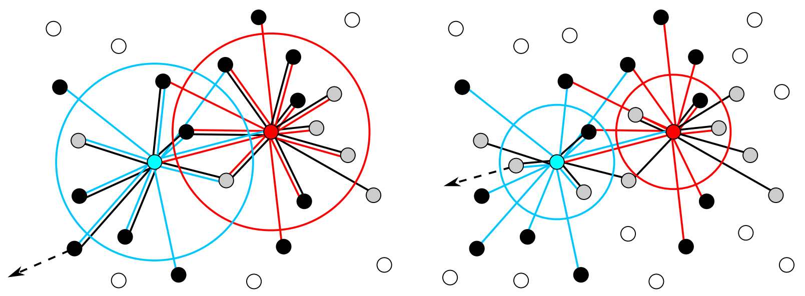
(B) v stores edges to/from its neighbors (light blue lines) and its neighbors store edges to/from v (black lines).



(C) v_j is inserted with parent v. v_j stores edges to/from its neighbors (red lines) and its neighbors store edges to/from v_j (new black/blue lines).



(D) As additional nodes are inserted (gray/white), edges to/from v and v_j are also stored at v, v_j and the neighbors (new light blue, red, black lines, respectively).



(E,F) As the hyperballs shrink, the edges are adjusted. Nodes remember their original neighbors, as well as all neighbors within the current hyperball centered at themselves. Edges to/from parents are kept as long as the parent relationship exists.

Fig. 2. A-F show the evolution of edges between v (light blue) and v_j (red) and neighbors inserted before (black) or after (gray). Non-neighbors are white.

The expected $\Theta(\log n)$ time is achieved, despite rewiring cascades, by using two new graph rewiring strategies: (1) Rewiring cascades are aborted once the graph becomes ϵ-consistent[1], for a predefined $\epsilon > 0$. (2) Graph connectivity information is maintained in local neighbor sets stored at each node, and the usual edge symmetry is allowed to be broken, i.e., the directed edge (u, w) will eventually be forgotten by u but not by w or *vice versa* (See Figure 2). In particular, node v always remembers the original neighbors that were calculated upon its insertion into the search-graph. However, each of those original neighbors will forget its connection to v once it is no longer within an RRT*-like shrinking D-ball centered at v (with the exception that connections within the shortest-path subtree are also remembered). This guarantees: (A) each node maintains expected $O(\log n)$ neighbors, (B) the RRT* solution is always a realizable sub-graph of the RRT^{X} graph—providing an upper-bound on path length, (C) all "edges" are remembered by at least one node. Although (1) and (2) have obvious side-effects[2], they significantly decrease reaction time (i.e, iteration time vs. RRT$^{\#}$ and cost propagation time vs. RRT*) without hindering asymptotic convergence to the optimal solution. Indeed, reaction time is the most important metric for a replanner in dynamic environments.

A YouTube play-list of RRT^{X} movies at `http://tinyurl.com/l53gzgd` shows RRT^{X} solving a variety of motion problems in different spaces (Otte, 2014).

The rest of this paper is organized as follows: Section 2 surveys related work. Notation, assumptions, and the "replanning" problem statement appear in Section 3. The RRT^{X} algorithm is presented in Section 4, and its theoretical analysis in Section 5. Simulations of RRT^{X} in dynamic environments and Experiments comparing RRT^{X} to RRT* and RRT$^{\#}$ in static environments

[1]"ϵ-consistency" means that the cost-to-goal stored at each node is within ϵ of its look-ahead cost-to-goal, where the latter is the minimum sum of distance-to-neighbor plus neighbor's cost-to-goal.
[2](1) allows graph inconsistency. (2) may delay the practical realization of some paths.

appear in Section 6. Sections 7 and 8 contain a discussion of results and conclusions, respectively. Minor subroutines appear in Appendix A, while additional background information regarding RRT* and RRT$^{\#}$ appears in Appendix B.

2. Related Work

RRT$^{\mathrm{X}}$ reflects a convergence of ideas from sampling-based motion planning (i.e., control theory) and discrete graph replanning (i.e., artificial intelligence). RRT$^{\mathrm{X}}$ differs from previous work in that it is the first asymptotically optimal[3] single-query[4] sampling-based motion replanning[5] algorithm.

2.1. Related work on feasible sampling-based replanning

Replanning in the context of *feasible* sampling-based motion planning has undergone a number of incarnations for over a decade. ERRT (Bruce and Veloso, 2002) caches waypoints from early plans, and then regrows the search tree from scratch whenever the environment changes. The cached waypoints are used to bias the direction of tree growth during replanning. DRRT (Ferguson et al., 2006) plans from the goal to the start (like RRT$^{\mathrm{X}}$) in order to reuse the same search tree after an environment change. Branches that have been detached by obstacles are completely pruned and deleted (unlike RRT$^{\mathrm{X}}$), and new sampling is biased toward the affected regions in order to help grow new branches around the obstacles. MP-RRT (Zucker et al., 2007) retains disconnected branches and actively attempts to reconnect them to the rooted tree. LRF (Gayle et al., 2007) additionally builds trees to multiple goals, retains disconnected branches like MP-RRT, and may merge trees or reconnect branches. GRIP (Bekris and Kavraki, 2007) prunes disconnected branches and then bias new growth toward old locations using heuristics in the sampling function. ATS+EB (Hauser, 2012) simultaneously plans vs. two different objective functions (a safety strategy that we discuss in more detail in Section 2.5).

ERRT, MP-RRT, LRF, GRIP, and ATS+EB all use forward search that grows the search tree from the start to the goal. This design choice that is arguably a retrogression from DRRT, given that significant machinery is required to continually update the root of the tree to respect the robot's location as the robot moves. RET (Martin et al., 2007) uses a bi-directional planner (which also requires that the forward tree root be updated vs. a moving robot location) and an offline phase to calculate additional waypoints for sampling bias during replanning.

The main difference between RRT$^{\mathrm{X}}$ and these algorithms is that RRT$^{\mathrm{X}}$ is designed for asymptotically optimal replanning while they are designed for feasible replanning. In particular, although excess computation can sometimes be used to improve path quality (e.g., GRIP), none of the aforementioned algorithms have theoretical guarantees regarding path optimally. Another significant difference is that these algorithms must rely on new samples to regrow or reconnect portions of the search tree that have been disconnected by obstacles. In contrast, RRT$^{\mathrm{X}}$ uses a rewiring cascade to immediately remodel the tree around the obstacles using existing nodes. With the exception of DRRT (and RRT$^{\mathrm{X}}$), all of the aforementioned algorithms plan from start to goal. Replanning, arguably, should be done from goal to start because (1) this allows the same tree *and root* to be used for all replans, and (2), in practice, robot sensors will detect changes near the leaves of the tree instead of the root (thus any rewiring/pruning/regrowing of a node and its descendants will affect a significantly smaller portion of the tree).

[3]Asymptotically optimal algorithms have the property that their solution to a problem approaches the optimal value (e.g., minimum path length) in the limit as the number of search iterations approaches infinity.

[4]Single-query algorithms assume that only a single problem will be solved and make no distinction between precomputation and computation. In contrast, multi-query algorithms assume that a series of problems will be solved in the same state space and with the same obstacle configuration (but with different start and/or goal states), thus justifying the use of significant offline precomputation.

[5]Replanning algorithms find a sequence of solutions"on-the-fly" vs. an evolving obstacle configuration and start state.

2.2. Related work on discrete graph replanning

D* (Stentz, 1995), Lifelong-A* (Koenig et al., 2004), and D*-Lite (Koenig et al., 2002) are discrete graph replanning algorithms designed to repair an A*-like solution after edge weights have changed. Equivalence of "no edge exists" and "an infinite cost edge" allows graph connectivity changes, in general. These early algorithms traditionally plan/replan over a grid embedded in the robot's work-space, and thus are potentially impossible to follow given its kinematics. More recent work addresses this deficiency by using precomputed lattice graphs composed of kinodynamically feasible edges projected into the workspace from the robot's state space. Likhachev and Ferguson (2009) use this technique with AD* (other methods also use this idea in static environments (Pivtoraiko et al., 2009; Frazzoli et al., 2005; Kushleyev and Likhachev, 2009).

These ideas are resolution optimal (or near-optimal in the case of AD*) and complete with respect to the particular lattice used; however, they almost surely find paths that are suboptimal with respect to the robot and environment. The construction of a kinodynamically feasible lattice graph requires significant offline computation. While this is required only once per robot (if the user is satisfied with the level of resolution completeness/optimality), increasing completeness/optimality requires recomputing the lattice. In contrast, RRT^X refines an asymptotically dense search-graph directly in the robot's state-space, and finds paths that are asymptotically optimal and probabilistically complete with respect to the robot and environment. RRT^X also does not require any offline computation.

2.3. Related work on asymptotically optimal single-query planning

RRT* (Karaman and Frazzoli, 2011) is the first asymptotically optimal single-Query sampling-based motion planning algorithm. It introduces the idea that asymptotic optimality can be guaranteed if new nodes, upon insertion, are allowed to "rewire" graph edges within their local neighborhood. They prove that the size of this neighborhood may shrink as a function of graph size without affecting asymptotically optimality or probabilistic completeness. For convenience, a more in-depth description of RRT* appears in Appendix B.

Any-Time SPRT (Otte, 2011) uses node-insertion rewiring and random rewiring to achieve asymptotic optimality, and maintains graph consistency by propagating decreasing cost-to-goal information through the graph using brute-force. RRT$^\#$ (Arslan and Tsiotras, 2013) uses a more efficient rewiring cascade that both propagate decreasing cost-to-goal information through the graph and also rewires within local neighborhoods when even lower cost-to-goal values are possible. A more in-depth description of RRT$^\#$ also appears in Appendix B for convenience. LBT-RRT (Salzman and Halperin, 2014) is designed for static environments and maintains a "lower-bound" graph that returns asymptotically $1 + \hat{\epsilon}$ "near-optimal" solutions.

RRT*, Any-Time SPRT, RRT$^\#$ and LBT-RRT are all designed for static environments and cannot address unforeseen obstacle configuration changes. In contrast, RRT^X repairs the graph to reflect both *increasing* and *decreasing* cost-to-goal values that result when obstacles change. While similar ideas could be used to create a replanning algorithm based on RRT$^\#$, such an algorithm would have asymptotically slower runtime than RRT^X (see proofs in Section 5).

RRT* and LBT-RRT use "lazy-propagation" to spread information through an inconsistent graph (i.e., data is transferred only via new node insertions), while Any-Time SPRT and RRT$^\#$ maintain fully consistent graphs. In contrast RRT^X maintains an ϵ-consistent graph. Note that $\hat{\epsilon}$ in LBT-RRT should not be confused with ϵ in RRT^X. Tuning $\hat{\epsilon}$ changes the performance of LBT-RRT along the optimality spectrum between RRT and RRT*, while tuning ϵ changes the performance of RRT^X along the graph consistency spectrum.

In static environments the asymptotic expected runtime to build a graph with n nodes is $\Theta\left(n \log n\right)$ for RRT^X and RRT*, and $\omega\left(n \log^2 n\right)$ for RRT$^\#$ (see Section 5); it is previously known that LBT-RRT has similar performance to RRT* while Anytime SPRT is $O\left(n^2\right)$. We also prove that the time required to propagate new information through the graph (i.e., such that it becomes incorporated into the robot's current plan) is asymptotically less for RRT^X than for RRT* or RRT$^\#$

(see Section 5). LBT-RRT can theoretically perform no better[6] than RRT*, while Any-Time SPRT must perform worse[7] than RRT$^{\mathrm{X}}$ (the latter are not formally treated in the current paper).

In practice, rewiring requires the assumption of a local steering function (used to calculate trajectories between nodes) that solves the two point boundary value problem (either analytically or numerically). This assumption is shared by all of these algorithms and RRT$^{\mathrm{X}}$. Therefore, RRT$^{\mathrm{X}}$ should not be used when this assumption cannot be met.

2.4. *Related work on asymptotically optimal multi-query re/planning*

PRM (Kavraki et al., 1996) is the first asymptotically optimal sampling-based motion planning algorithm. It (and its numerous descendants) use significant offline computation to create a detailed graph of configuration-space connectivity *a priori*. The online planning phase involves connecting the current start and goal states to the graph and then using standard graph search techniques (e.g., A* or Dijkstra) to find a solution. PRM* (Karaman and Frazzoli, 2011) extends the basic idea by using a shrinking neighborhood size (similar to RRT*, described above) to achieve $\Theta(\log n)$ expected per iteration runtime.

Early roadmap algorithms assumed a particular obstacle layout during the precomputation phase and thus were thus ill-suited to replanning in dynamic environments (although replanning vs. new tasks in static environments was very fast). Although not specifically designed for replanning in dynamic environments, Lazy-PRM (Bohlin and Kavraki, 2000) is better suited to the dynamic replanning problem given that it only collision checks edges that are likely to be part of the solution. This idea is formalized in the context of dynamic environments by Leven and Hutchinson (2001) and Leven and Hutchinson (2002) and further developed by Pomarlan and Şucan (2013). The latter three algorithms construct roadmaps in obstacle-free environments, while edges are collision checked vs. obstacles online during planning/replanning and a mapping between workspace and configuration space is used to determine when obstacles invalidate the current path.

The “Elastic Roadmap" (Yang and Brock, 2010) maintains a mapping between workspace and configuration space and fixes a subset of nodes and edges with respect each obstacle’s local coordinate system. When obstacles move, these nodes/edges move with the obstacle and reattach to the global roadmap.

Sparse subgraph *spanners* (Marble and Bekris, 2013; Salzman et al., 2014; Wang et al., 2015) increases speed during the online phase by pruning the *a priori* roadmaps down to useful subset (or spanner) offline. The spanner is designed to maintain near-optimality vs. the original roadmap while using significantly fewer nodes.

The main difference between these methods and RRT$^{\mathrm{X}}$ is that they assume the existence of a significant offline planning phase that is used to create the roadmap *a priori* (in this regard roadmap methods are becoming increasingly similar to lattice based methods). Offline computation makes sense in the context of multi-query algorithms, where the significant cost of offline roadmap construction is amortized over many online plans through the same space. In contrast, RRT$^{\mathrm{X}}$ and other single-query algorithms are designed for systems that assume different spaces may be encountered frequently (e.g., vehicles) and/or assume precomputation is otherwise impossible.

We note that the work on sparse subgraph spanners mentioned above is similar, in spirit, to how RRT$^{\mathrm{X}}$ limits each node’s edge set to $O(\log n)$ by culling outdated edges. Spanner-finding algorithms are designed for the precomputation phase of multi-query algorithms, while RRT$^{\mathrm{X}}$ culls edges online.

2.5. *Other safety considerations in dynamic environments*

“Escape trajectories” (Hsu et al., 2002) are additional backup plans stored at each node in a motion planing that can be used if/when obstacle movement invalidates the current motion plan. Commonly, safety trajectories are a conservative stopping

[6]The time savings achieved in LBT-RRT by not updating some neighbors vs. RRT* is bounded by the runtime difference between RRT* and RRT, which is still not enough to overcome the inherent slow transfer speed of lazy propagation.

[7]Anytime SPRT has similar runtime to inform neighbors about cost changes, but neighboring nodes are not rewired as they are in RRT$^{\mathrm{X}}$.

or holding patterns (e.g., decelerate to a ground vehicle to a stop or fly an aircraft in a circle) that are used while a new plan is calculated. Hauser (2012) takes this a step further by simultaneously calculating both a path to the goal and a path away from obstacles; the robot switches to using the latter whenever it is in danger.

The use of safety trajectories is advisable with any replanning algorithm, including RRT^{X}. Because the mechanism by which they provide safety is orthogonal to the underlying algorithm that is used, we do not focus on their effects in the current paper (all algorithms used in section 6.2 use the same basic "hold pattern" safety trajectories in the event that the robot becomes disconnected from goal-rooted shortest path tree).

Safety can also be increased by considering the the set of nodes that place the robot into an "inevitable collision state" (LaValle and Kuffner, 2001; Reif and Sharir, 1985; Fraichard and Asama, 2003) — a state that, while being safe *itself*, places the robot into a situation in which a future collision cannot be avoided (e.g., flying into the mouth of a narrow box canyon). This is particularly an issue with algorithms that optimize vs. an objective function that does not explicitly require reaching a goal state, but can also be an issue for any algorithm that must explore an unknown or dynamic environment. The latter can be mitigated by requiring a solution to consist only of nodes with safety trajectories (Bekris and Kavraki, 2007), although the added safety may results in a loss of completeness and optimality. Such ideas can also be used with RRT$^{\mathrm{X}}$, although we do not consider them in the current paper.

It is worth remembering that an environment is dynamic if (and only if) an agent's model of obstacle movement may turn out to be wrong. Therefore, it is possible to design situations in which any algorithm will fail in a dynamic environment. While more accurate obstacle models will improve success rates; it is also easy to find problems that require a risk of collision in order to reach the goal (e.g., crossing a busy street). This is a trait shared by all replanning algorithms as well as humans.

2.6. Additional related work

RRT (LaValle and Kuffner, 2001) is the first single-query algorithm with provable asymptotic coverage guarantees. Its main differences vs. RRT$^{\mathrm{X}}$ are: (1) RRT is only concerned with finding *a* (i.e., any) feasible path and does not consider path quality, (2) it is not designed for replanning. That said, due to its ease of implementation and widespread use, RRT is often used inside of a control loop for replanning in practice. A new path is calculated whenever the old plan becomes invalid (e.g., Kuwata et al. (2009) and Bertola and Gonzalez (2013)).

Epsilon A* is an any-time discrete graph algorithm that returns A* with a more and more admissible heuristic while planning time remains. It is similar to RRT$^{\mathrm{X}}$ in the sense that both use a user-defined parameter to control the suboptimality of the shortest-path subtree that is found over a particular graph in order to speed up planning/replanning time. However, path suboptimality in RRT$^{\mathrm{X}}$ vanishes in the limit vs. the number of samples, while the suboptimality in epsilon A* vanishes vs. the shrinking epsilon (over any-time re-runs of the algorithm). Epsilon A* is designed for planning in static environments (and not for replanning in changing environments like RRT$^{\mathrm{X}}$). Being a discrete graph algorithm, it has similar difference vs. RRT$^{\mathrm{X}}$ as D*, D*-Lite, and Lifelong-A* (described above).

Hierarchical planners divide planning into L different scopes. When $L = 2$, as is most common, they are sometimes referred to as the 'local' and 'global' planners, respectively (Otte et al., 2007). The global planner finds an approximate path to the goal (usually either a geometric path (Sugiyama et al., 1994) or a Euclidean-based cost-field constructed in the workspace (Sermanet et al., 2008; Knepper and Mason, 2012)). The local planner is responsible for moving the robot a small distance from its current location while (often) respecting the kinematics of the robot, e.g., to a waypoint located along the global path or by using the global field-cost to help select a local path. Modern hierarchical planners use communication between the local and global planners (Plaku et al., 2010) and/or an assumption of task reversibility Kaelbling and Lozano-Perez (2011) to ensure that the global planner also respects the robot's kinematic constraints. There is significant experimental evidence showing that hierarchical planners work well in many cases, including dynamic environments; however, theoretical guarantees are limited to probabilistic and/or resolution completeness and resolution optimality. RRT$^{\mathrm{X}}$ differs from hierarchical planners in that it is an all-in-one planner (constructing a single tree of valid movement in the state space all the way from the robot's current location to the goal). It is also asymptotically optimal.

Feedback planners generate a continuous control policy over the state space (i.e. instead of embedding a graph in the state space). Most feedback planners do not consider obstacles (Tedrake et al., 2010; Rimon and Koditschek, 1992), while those that do (LaValle and Lindemann, 2009) assume that obstacles are both static and easily representable in the state space. In contrast, sampling-based motion planning algorithms assume that obstacles are not easily representable in the state space, and RRT$^{\mathrm{X}}$ assumes obstacles change unpredictably.

A preliminary version of the current paper appears as a conference paper in Otte and Frazzoli (2014). Major differences and extension that appear in the current paper include: more detailed proofs in Sections 5, a new analysis regarding the information transfer time for RRT$^{\mathrm{X}}$, RRT$^{\#}$, and RRT* in Section 5.5, four additional simulations in Section 6.1 (including instructions on how to use RRT$^{\mathrm{X}}$ in scenarios that contain time $\mathbb{T}$ as a dimension of $\mathcal{X}$), and four new experiments in Section 6.2 verifying the analytical results of RRT$^{\mathrm{X}}$ vs. RRT* and RRT$^{\#}$, and additional discussion in Section 7.

3. Notation, Assumptions, and Problem Statement

3.1. Preliminaries

Let $\mathcal{X}$ denote the robot's D-dimensional state space. $\mathcal{X}$ is a measurable metric space that has finite measure. Formally, $\mathscr{L}(\mathcal{X}) = c$, for some $c < \infty$ and $\mathscr{L}(\cdot)$ is the Lebesgue measure; assuming $\mathrm{d}(x_1, x_2)$ is a distance function on $\mathcal{X}$, then $\mathrm{d}(x_1, x_2) \geq 0$ and $\mathrm{d}(x_1, x_3) \leq \mathrm{d}(x_1, x_2) + \mathrm{d}(x_2, x_3)$ and $\mathrm{d}(x_1, x_2) = \mathrm{d}(x_2, x_1)$ for all $x_1, x_2, x_3 \in \mathcal{X}$. We assume the boundary of $\mathcal{X}$ is both locally Lipschitz-continuous and has finite measure. The obstacle space $\mathcal{X}_{\mathrm{obs}} \subset \mathcal{X}$ is the open subset of $\mathcal{X}$ in which the robot is "in collision" with obstacles or itself. The free space $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{\mathrm{obs}}$ is the closed subset of $\mathcal{X}$ that the robot can reach. We assume $\mathcal{X}_{\mathrm{obs}}$ is defined by a set $\mathcal{O}$ of a finite number of obstacles O, each with a boundary that is both locally Lipschitz-continuous and has finite measure.

The robot's start and goal states are x_{start} and x_{goal}, respectively. At time t the location of the robot is $x_{\mathrm{bot}}(t)$, where $x_{\mathrm{bot}} : [t_0, t_{\mathrm{cur}}] \to \mathcal{X}$ is the traversed path of the robot from the start time t_0 to the current time t_{cur}, and is undefined for $t > t_{\mathrm{cur}}$. The obstacle space (and free space) may change as a function of time and/or robot location, i.e. $\Delta\mathcal{X}_{\mathrm{obs}} = f(t, x_{\mathrm{bot}})$. For example, if there are unpredictably moving obstacles, inaccuracies in *a priori* belief of $\mathcal{X}_{\mathrm{obs}}$, and/or a subset of $\mathcal{X}_{free}$ must be "discovered" via the robot's sensors.

A movement trajectory $\pi(x_1, x_2)$ is the curve defined by a continuous mapping $\pi : [0,1] \to \mathcal{X}$ such that $0 \mapsto x_1$ and $1 \mapsto x_2$. A trajectory is *valid* iff both $\pi(x_1, x_2) \cap \mathcal{X}_{\mathrm{obs}} = \emptyset$ and it is possible for the robot to follow $\pi(x_1, x_2)$ given its kinodynamic and other constraints. $\mathrm{d}_\pi(x_1, x_2)$ is the length of $\pi(x_1, x_2)$.

3.2. Environments: Static vs. Dynamic and Related Assumptions

A *static* environment has an obstacle set that changes deterministically vs. t and x_{bot}, i.e., $\Delta\mathcal{X}_{\mathrm{obs}} = f(t, x_{\mathrm{bot}})$ for f known *a priori*. In the simplest case, $\Delta\mathcal{X}_{\mathrm{obs}} \equiv \emptyset$. In contrast, a *dynamic*[8] environment has an unpredictably changing obstacle set, i.e., f is a "black-box" that cannot be known *a priori*. The assumption of incomplete prior knowledge of $\Delta\mathcal{X}_{\mathrm{obs}}$ guarantees myopia; this assumption is the defining characteristic of replanning algorithms, in general. While nothing prevents us from estimating $\Delta\mathcal{X}_{\mathrm{obs}}$ based on prior data and/or online observations, we cannot guarantee that any such estimate will be correct. Note that $\Delta\mathcal{X}_{\mathrm{obs}} \neq \emptyset$ is not a sufficient condition for $\mathcal{X}$ to be dynamic[9].

[8]The use of the term "dynamic" to indicate that an environment is "unpredictably changing" comes from the artificial intelligence literature. It should not be confused with the "dynamics" of classical mechanics.

[9]For example, if $\mathcal{X} \subset \mathbb{R}^d$ space, $\mathbb{T}$ is time, and obstacle movement is known *a priori*, obstacles are stationary with respect to $\hat{\mathcal{X}} \subset (\mathbb{R}^d \times \mathbb{T})$ space-time.

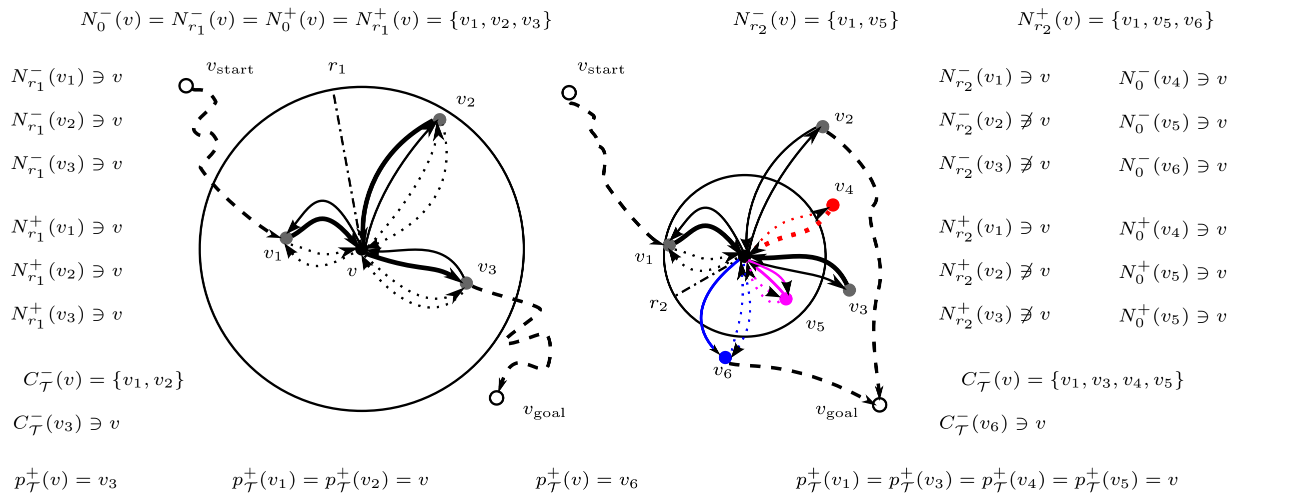

$N_0^-(v) = N_{r_1}^-(v) = N_0^+(v) = N_{r_1}^+(v) = \{v_1, v_2, v_3\}$
$N_{r_2}^-(v) = \{v_1, v_5\}$
$N_{r_2}^+(v) = \{v_1, v_5, v_6\}$
$N_{r_1}^-(v_1) \ni v$
$N_{r_1}^-(v_2) \ni v$
$N_{r_1}^-(v_3) \ni v$
$N_{r_1}^+(v_1) \ni v$
$N_{r_1}^+(v_2) \ni v$
$N_{r_1}^+(v_3) \ni v$
$C_{\mathcal{T}}^-(v) = \{v_1, v_2\}$
$C_{\mathcal{T}}^-(v_3) \ni v$
v_{start}
r_1
v_2
v_1
v
v_3
v_{goal}
v_{start}
v_2
v_4
v_1
r_2
v_5
v_3
v_6
v_{goal}
$N_{r_2}^-(v_1) \ni v$
$N_{r_2}^-(v_2) \not\ni v$
$N_{r_2}^-(v_3) \not\ni v$
$N_{r_2}^+(v_1) \ni v$
$N_{r_2}^+(v_2) \not\ni v$
$N_{r_2}^+(v_3) \not\ni v$
$N_0^-(v_4) \ni v$
$N_0^-(v_5) \ni v$
$N_0^-(v_6) \ni v$
$N_0^+(v_4) \ni v$
$N_0^+(v_5) \ni v$
$N_0^+(v_5) \ni v$
$C_{\mathcal{T}}^-(v) = \{v_1, v_3, v_4, v_5\}$
$C_{\mathcal{T}}^-(v_6) \ni v$
$p_{\mathcal{T}}^+(v) = v_3$
$p_{\mathcal{T}}^+(v_1) = p_{\mathcal{T}}^+(v_2) = v$
$p_{\mathcal{T}}^+(v) = v_6$
$p_{\mathcal{T}}^+(v_1) = p_{\mathcal{T}}^+(v_3) = p_{\mathcal{T}}^+(v_4) = p_{\mathcal{T}}^+(v_5) = v$

Fig. 3. Neighbor sets of/with node v. Left: v is inserted when $r = r_1$. Right: later $r = r_2 < r_1$. Solid: neighbors known to v. Black solid: original in- $N_0^-(v)$ and out-neighbors $N_0^+(v)$ of v. Colored solid: running in- $N_r^-(v)$ and out-neighbors $N_r^+(v)$ of v. Dotted: v is neighbor of $v_i \neq v$. Dotted colored: v is an original neighbor of v_i. Bold: edge in shortest-path subtree. Dashed: other sub-path.

3.3. Problem Statement of "Shortest-Path Replanning"

Given $\mathcal{X}$, $\mathcal{X}_{\mathrm{obs}}$, x_{goal}, $x_{\mathrm{bot}}(0) = x_{\mathrm{start}}$, and unknown $\Delta\mathcal{X}_{\mathrm{obs}} = f(t, x_{\mathrm{bot}})$, find $\pi^*(x_{\mathrm{bot}}, x_{\mathrm{goal}})$ and, until $x_{\mathrm{bot}}(t) = x_{\mathrm{goal}}$, simultaneously update $x_{\mathrm{bot}}(t)$ along $\pi^*(x_{\mathrm{bot}}, x_{\mathrm{goal}})$ while recalculating $\pi^*(x_{\mathrm{bot}}, x_{\mathrm{goal}})$ whenever $\Delta\mathcal{X}_{\mathrm{obs}} \neq \emptyset$, where

$$\pi^*(x_{\mathrm{bot}}, x_{\mathrm{goal}}) = \underset{\pi(x_{\mathrm{bot}}, x_{\mathrm{goal}}) \in \mathcal{X}_{free}}{\arg\min} \; \mathrm{d}_\pi(x_{\mathrm{bot}}, x_{\mathrm{goal}})$$

3.4. Additional Notation Used for the Algorithm and its Analysis

RRT$^{\mathrm{X}}$ constructs a graph $\mathcal{G} := (V, E)$ embedded in $\mathcal{X}$, where V is the node set and E is the edge set. With a slight abuse of notation we will allow $v \in V$ to be used in place of v's corresponding state $x \in \mathcal{X}$, e.g., as a direct input into distance functions. Thus, the robot starts at v_{start} and goes to v_{goal}. The "shortest-path" subtree of $\mathcal{G}$ is $\mathcal{T} := (V_\mathcal{T}, E_\mathcal{T})$, where $\mathcal{T}$ is rooted at v_{goal}, $V_\mathcal{T} \subset V$, and $E_\mathcal{T} \subset E$. The set of 'orphan nodes' is defined $V_\mathcal{T}^{\mathrm{c}} = V \setminus V_\mathcal{T}$ and contains all nodes that have become disconnected from $\mathcal{T}$ due to $\Delta\mathcal{X}_{\mathrm{obs}}$ (c denotes the set complement of $V_\mathcal{T}$ with respect to V and *not* the set of nodes in the complement graph of $\mathcal{T}$).

$\mathcal{G}$ is built, in part, by drawing nodes at random from a random sample sequence $S = \{v_1, v_2, \ldots\}$. We assume v_i is drawn i.i.d from a uniform distribution over $\mathcal{X}$; however, this can be relaxed to any distribution with a finite probability density for all $v \in \mathcal{X}_{free}$. We use V_n, $\mathcal{G}_n$, $\mathcal{T}_n$ to denote the node set, graph, and tree when the node set contains n nodes, e.g., $\mathcal{G}_n = \mathcal{G}$ s.t. $|V| = n$. Note $m_i = |V_{m_i}|$ at iteration i, but $m_i \neq i$ in general because samples may not always be connectable to $\mathcal{G}$. Indexing on m (and not i) simplifies the analysis.

$\mathbb{E}_n(\cdot)$ denotes the expected value of '$\cdot$' over the set $\mathcal{S}$ of all such sample sequences, conditioned on the event that $n = |V|$. The expectation $\mathbb{E}_{n,v_x}(\cdot)$ is conditioned on both $n = |V|$ and $V_n \setminus V_{n-1} = \{v_x\}$ for v_x at a particular $x \in \mathcal{X}$.

RRT$^{\mathrm{X}}$ uses a number of neighbor sets for each node v, see Figure 3. Edges are directed $(u, v) \neq (v, u)$, and we use a superscript '$-$' and '$+$' to denote association with incoming and outgoing edges, respectively. 'Incoming neighbors' of v is the set $N^-(v)$ s.t. v knows about (u, v). 'Outgoing neighbors' of v is the set $N^+(v)$ s.t. v knows about (v, u). At any instant $N^+(v) = N_0^+(v) \cup N_r^+(v)$ and $N^-(v) = N_0^-(v) \cup N_r^-(v)$, where $N_0^-(v)$ and $N_0^+(v)$ are the original PRM*-like in/out-neighbors (which v always keeps), and $N_r^-(v)$ and $N_r^+(v)$ are the 'running' in/out-neighbors (which v culls as r decreases). The set of all neighbors of v is $N(v) = N^+(v) \cup N^-(v)$. Because $\mathcal{T}$ is rooted at v_{goal}, the parent of v is denoted $p_\mathcal{T}^+(v)$ and the child set of v is denoted $C_\mathcal{T}^-(v)$.

Algorithm 1: `RRT`$^{\text{X}}(\mathcal{X}, S)$

```
1  $V \leftarrow \{v_{\text{goal}}\}$ ;
2  $v_{\text{bot}} \leftarrow v_{\text{start}}$ ;
3  while $v_{\text{bot}} \neq v_{\text{goal}}$ do
4      $r \leftarrow$ shrinkingBallRadius($|V|$) ;
5      if obstacles have changed then
6          updateObstacles() ;
7      if robot is moving then
8          $v_{\text{bot}} \leftarrow$ updateRobot($v_{\text{bot}}$) ;
9      $v \leftarrow$ randomNode($S$) ;
10     $v_{\text{nearest}} \leftarrow$ nearest($v$) ;
11     if $\mathrm{d}(v, v_{\text{nearest}}) > \delta$ then
12         $v \leftarrow$ saturate($v, v_{\text{nearest}}$) ;
13     if $v \notin \mathcal{X}_{\text{obs}}$ then
14         extend($v, r$) ;
15     if $v \in V$ then
16         rewireNeighbors($v$) ;
17         reduceInconsistency() ;
```

$\mathrm{g}(v)$ is the (ϵ-consistent) cost-to-goal of reaching v_{goal} from v through $\mathcal{T}$. The look-ahead estimate of cost-to-goal is $\texttt{lmc}(v)$. Note that the algorithm stores both $\mathrm{g}(v)$ and $\texttt{lmc}(v)$ at each node, and updates $\texttt{lmc}(v) \leftarrow \min_{u \in N^+(v)} \mathrm{d}_\pi(v,u) + \texttt{lmc}(u)$ when appropriate conditions have been met. v is 'ϵ-consistent' iff $\mathrm{g}(v) - \texttt{lmc}(v) < \epsilon$. $\mathrm{g}_m(v)$ is the cost-to-goal of v given $\mathcal{T}_m$. Recall that $\pi^*_{\mathcal{X}}(v, v_{\text{goal}})$ is the optimal path from v to v_{goal} through $\mathcal{X}$; the length of $\pi^*_{\mathcal{X}}(v, v_{\text{goal}})$ is $\mathrm{g}^*(v)$.

$\mathcal{Z}_m$ is the visibility set of a particular graph of size m, where a visibility set of a graph contains all points that can safely connect to any node in that graph.

Q is the priority queue that is used to determine the order in which nodes become ϵ-consistent during the rewiring cascades. The key that is used for Q is the ordered pair $\left(\min(\mathrm{g}(v), \texttt{lmc}(v)), \mathrm{g}(v)\right)$ nodes with smaller keys are popped from Q before nodes with larger keys, where $(a,b) < (c,d)$ iff $a < c \vee (a = c \wedge b < d)$.

4. The RRT$^{\text{X}}$ Algorithm

RRT$^{\text{X}}$ appears in Algorithm 1 and its major subroutines in Algorithms 2-6 (minor subroutines appear in Appendix A). The main control loop, lines 3-17, terminates once the robot reaches the goal state. Each pass begins by updating the RRT*-like neighborhood radius r (line 4), and then accounting for obstacle and/or robot changes (lines 5-8). "Standard" sampling-based motion planning operations improve and refine the graph by drawing new samples and then connecting them to the graph if possible (lines 9-14). RRT*-like graph rewiring (line 16) guarantees asymptotic optimality, while rewiring cascades enforce ϵ-consistency (line 17, and also on line 6 as part of `updateObstacles`()). `saturate`(v, v_{nearest}), line 12, repositions v to be δ away from v_{nearest}.

`extend`(v, r) attempts to insert node v into $\mathcal{G}$ and $\mathcal{T}$ (line 5). If a connection is possible then v is added to its parent's child set (line 6). The edge sets of v and its neighbors are updated (lines 7-13). For each new neighbor u of v, u is added to v's initial neighbors sets $N_0^+(v)$ and $N_0^-(v)$, while v is added to u's running neighbor sets $N_r^+(u)$ and $N_r^-(u)$. This differentiation allows RRT$^{\text{X}}$ to maintain $O(\log n)$ edges at each node, while ensuring $\mathcal{T}$ is no worse than the tree created by RRT* given the same sample sequence.

```
Algorithm 2: extend$(v, r)$
1  $V_{near} \leftarrow$ near$(v, r)$ ;
2  findParent$(v, V_{near})$ ;
3  if $p^+_{\mathcal{T}}(v) = \emptyset$ then
4      return
5  $V \leftarrow V \cup \{v\}$ ;
6  $C^-_{\mathcal{T}}(p^+_{\mathcal{T}}(v)) \leftarrow C^-_{\mathcal{T}}(p^+_{\mathcal{T}}(v)) \cup \{v\}$ ;
7  forall the $u \in V_{near}$ do
8      if $\pi(v, u) \cap \mathcal{X}_{\mathrm{obs}} = \emptyset$ then
9          $N^+_0(v) \leftarrow N^+_0(v) \cup \{u\}$ ;
10         $N^-_r(u) \leftarrow N^-_r(u) \cup \{v\}$ ;
11     if $\pi(u, v) \cap \mathcal{X}_{\mathrm{obs}} = \emptyset$ then
12         $N^+_r(u) \leftarrow N^+_r(u) \cup \{v\}$ ;
13         $N^-_0(v) \leftarrow N^-_0(v) \cup \{u\}$ ;
```

```
Algorithm 3: cullNeighbors$(v, r)$
1  forall the $u \in N^+_r(v)$ do
2      if $r < \mathrm{d}_\pi(v, u)$ and $p^+_{\mathcal{T}}(v) \neq u$ then
3          $N^+_r(v) \leftarrow N^+_r(v) \setminus \{u\}$ ;
4          $N^-_r(u) \leftarrow N^-_r(u) \setminus \{v\}$ ;
```

```
Algorithm 4: rewireNeighbors$(v)$
1  if g$(v)$ − lmc$(v) > \epsilon$ then
2      cullNeighbors$(v, r)$ ;
3      forall the $u \in N^-(v) \setminus \{p^+_{\mathcal{T}}(v)\}$ do
4          if lmc$(u) > \mathrm{d}_\pi(u, v)$ + lmc$(v)$ then
5              lmc$(u) \leftarrow \mathrm{d}_\pi(u, v)$ + lmc$(v)$;
6              makeParentOf$(v, u)$ ;
7              if g$(u)$ − lmc$(u) > \epsilon$ then
8                  verrifyQueue$(u)$ ;
```

cullNeighbors(v, r) updates $N^-_r(v)$, and $N^+_r(v)$ to allow only edges that are shorter than r—with the exceptions that we do not remove edges that are part of $\mathcal{T}$. *RRT*^{X} inherits asymptotic optimality and probabilistic completeness from PRM*/RRT* by never culling $N^-_0(v)$ or $N^+_0(v)$.

rewireNeighbors(v) rewires v's in-neighbors $u \in N^-(v)$ to use v as their parent, if doing so results in a better cost-to-goal at u (lines 3-6). This rewiring is similar to RRT*'s rewiring, except that here we verify that ϵ-inconsistent neighbors are in the priority queue (lines 7-8) in order to set off a rewiring cascade during the next call to reduceInconsistency().

reduceInconsistency() manages the rewiring cascade that propagates cost-to-goal information and maintains ϵ-consistency in $\mathcal{G}$ (at least up to the level-set of lmc$(\cdot)$ containing v_{bot}). It is similar to its namesake from RRT$^\#$, except that in RRT^X the cascade only continues through v's neighbors if v is ϵ-inconsistent (lines 3-5). This is one reason why RRT^X is faster than RRT$^\#$. Note that v is always made locally 0-consistent (line 6).

updateLMC(v) updates lmc(v) based on v's out-neighbors $N^+(v)$ (as in RRT$^\#$).

findParent(v, U) finds the best parent for v from the node set U.

propogateDescendants() performs a cost-to-goal *increase* cascade leaf-ward through $\mathcal{T}$ after an obstacle has been

Algorithm 5: `reduceInconsistency`$()$

```
1 while size($Q$) > 0 and (keyLess(top($Q$), $v_{\mathrm{bot}}$) or lmc($v_{\mathrm{bot}}$) $\neq$ g($v_{\mathrm{bot}}$) or g($v_{\mathrm{bot}}$) $= \infty$ or $Q \ni v_{\mathrm{bot}}$) do
2     $v \leftarrow$ pop($Q$) ;
3     if g($v$) $-$ lmc($v$) $> \epsilon$ then
4         updateLMC($v$) ;
5         rewireNeighbors($v$) ;
6     g($v$) $\leftarrow$ lmc($v$) ;
```

Algorithm 6: `findParent`(v, U)

```
1 forall the $u \in U$ do
2     $\pi(v,u) \leftarrow$ computeTrajectory($\mathcal{X}, v, u$) ;
3     if $\mathrm{d}_\pi(v,u) \leq r$ and lmc($v$) $> \mathrm{d}_\pi(v,u) +$ lmc($u$) and $\pi(v,u) \neq \emptyset$ and $\mathcal{X}_{\mathrm{obs}} \cap \pi(v,u) = \emptyset$ then
4         $p^{+}_{\mathcal{T}}(v) \leftarrow u$ ;
5         lmc($v$) $\leftarrow \mathrm{d}_\pi(v,u) +$ lmc($u$) ;
```

added; the cascade starts at nodes with edge trajectories made invalid by the obstacle and is necessary to ensure that the *decrease* cascade in `reduceInconsistency`$()$ reaches all relevant portions of $\mathcal{T}$ (as in D*). `updateObstacles`$()$ updates $\mathcal{G}$ given $\Delta\mathcal{X}_{\mathrm{obs}}$; affected by nodes are added to Q or $V^{\mathrm{c}}_{\mathcal{T}}$, respectively, and then `reduceInconsistency`$()$ and/or `propogateDescendants`$()$ are called to invoke rewiring cascade(s).

5. Runtime Analysis of RRT, RRT*, RRT$^{\mathrm{X}}$, and RRT$^{\#}$

The replanning problem can be viewed as a static planning problem that is interrupted with repair operations whenever $\Delta\mathcal{X}_{\mathrm{obs}} \neq \emptyset$. In Sections 5.1-5.3 we prove bounds on the time required by RRT, RRT*, RRT$^{\mathrm{X}}$, and RRT$^{\#}$ to build a search-graph containing n nodes in the static case when $\Delta\mathcal{X}_{\mathrm{obs}} = \emptyset$. In Section 5.4 we discuss the extra operations required by RRT$^{\mathrm{X}}$ when $\Delta\mathcal{X}_{\mathrm{obs}} \neq \emptyset$.

Lemma 1. $\sum_{j=1}^{n} \log j = \Theta(n \log n)$.

Proof. $\log(j+1) - \log j \leq 1$ for all $j \geq 1$. Therefore, by construction:
$-n + \int_1^n \log x \; dx \leq \sum_{j=1}^{n} \log j \leq n + \int_1^n \log x \; dx$ for all $n \geq 1$. Calculus gives: $-n + \frac{1-n}{\ln 2} + n \log n \leq \sum_{j=1}^{n} \log j \leq n + \frac{1-n}{\ln 2} + n \log n$ □

Slight modifications to the proof of Lemma 1 yield the following corollaries:

Corollary 1. $\sum_{j=1}^{n} \frac{\log j}{j} = \Theta\left(\log^2 n\right)$.

Corollary 2. $\sum_{j=1}^{n} \log^2 j = \Theta\left(n \log^2 n\right)$.

Let f_i^{RRT} and f_i^{RRT*} denote the runtime of the ith iteration of RRT and RRT*, respectively, assuming samples are drawn uniformly at random from $\mathcal{X}$ according to the sequence $S = \{v_1, v_2, \ldots\}$. Let $f^{RRT}(n)$, $f^{RRT*}(n)$, and $f^{RRT^X}(n)$ denote the *cumulative* time until $n = |V_n|$, i.e., the graph contains n nodes, using RRT, RRT*, and RRT$^{\mathrm{X}}$, respectively.

5.1. Expected Time until $|V| = n$ for RRT and RRT*

LaValle and Kuffner (2001) and Karaman and Frazzoli (2011) give the following propositions, respectively:

Proposition 1. $f_i^{RRT} = \Theta(\log m_i)$ *for all* $i \geq 0$, *where* $m_i = |V_{m_i}|$ *at iteration* i.

Proposition 2. $\mathbb{E}\left(f_i^{RRT*}\right) = \Theta\left(f_i^{RRT}\right)$ *for all* $i \geq 0$.

The dominating term in both RRT and RRT* is due to a nearest neighbor search.

Theorem 1. $\mathbb{E}\left(f^{RRT}(n)\right) = \Theta\left(n \log n\right)$.

Proof. Trivially $\mathbb{P}\left(\{\lim_{i\to\infty} m_i = c < \infty\}\right) = 0$, so we can ignore the measure zero set of all $S \in \mathcal{S}$ that result in the visibility set $\mathcal{Z}_m$ being undefined for all $m > c$, for any $c < \infty$. By construction, $\lim_{i\to\infty} \mathcal{Z}_{m_i} \leq \mathcal{X}_{free}$, and also by construction, $\mathcal{Z}_{m_i} \geq \mathcal{Z}_1$ for all $i > 1$. Thus, $\mathbb{P}\left(\left\{\frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{Z}_1)} \leq \frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{Z}_m)} \leq \frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{X}_{free})}\right\}\right) = 1$ for all $m \geq 1$, and so $\frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{Z}_1)} \leq \mathbb{E}_m\left(\frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{Z}_m)}\right) \leq \frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{X}_{free})}$. Summing m terms yields another inequality: $\sum_{m=1}^{n-1} \frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{Z}_1)} \log m \leq \sum_{m=1}^{n-1} \mathbb{E}_m\left(\frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{Z}_m)}\right) \log m \leq \sum_{m=1}^{n-1} \frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{X}_{free})} \log m$. Note that $\frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{Z}_1)} = c_1$ and $\frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{X}_{free})} = c_2$ for constants c_1 and c_2, and thus $c_1 \sum_{m=1}^{n-1} \log m \leq \sum_{m=1}^{n-1} \mathbb{E}_m\left(\frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{Z}_m)}\right) \log m \leq c_2 \sum_{m=1}^{n-1} \log m$. By definition $\mathbb{E}\left(f^{RRT}(n)\right) = \sum_{m=1}^{n-1} \mathbb{E}_m\left(\frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathcal{Z}_m)}\right) \log m$. The existence of a final term does not affect asymptotics: $\sum_{m=1}^{n} \log m = \log n + \sum_{m=1}^{n-1} \log m = \Theta\left(\sum_{m=1}^{n-1} \log m\right)$. Applying Lemma 1 finishes the proof. □

Corollary 3. $\mathbb{E}\left(f^{RRT*}(n)\right) = \Theta\left(n \log n\right)$.

Theorem 1 and Corollary 3 show that rejecting samples (e.g., samples not connected due to obstacle conflicts) effects performance by at most a constant multiple, and furthermore, that the time required by RRT and RRT* to build a graph of size n is a convenient $\Theta\left(n \log n\right)$.

5.2. *Expected Amortized Time until $|V| = n$ for RRT^X (Static $\mathcal{X}$)*

In this section we prove: $\mathbb{E}_n\left(f^{RRT^X}(n)\right) = \Theta\left(n \log n\right)$. The proof involves a comparison to RRT* and proceeds in the following three steps:

1. RRT* cost-to-goal values approach optimality, in the limit as $n \to \infty$.
2. For RRT*, the summed total difference (i.e., over all nodes) between initial and optimal cost-to-goal values is $O\left(\epsilon n\right)$; thus, when $n = |V_n|$, RRT^X will have performed at most $O\left(n\right)$ cost propagations of size ϵ given the same S.
3. For RRT^X, each propagation of size ϵ requires the same order amortized time as inserting a new node (which is the same order for RRT* and RRT^X).

By construction RRT^X inherits the asymptotically optimal convergence of RRT* (we assume the planning problem, cost function, and ball parameter are defined appropriately). Theorem 38 from Karaman and Frazzoli (2011) has two relevant corollaries:

Corollary 4. $\mathbb{P}\left(\{\limsup_{n\to\infty} \mathrm{g}_n(v) = \mathrm{g}^*(v)\}\right) = 1$ *for all* $v \,:\, v \in V_{n<\infty}$.

Corollary 5. $\lim_{n\to\infty} \mathbb{E}_n\left(\mathrm{g}_n(v) - \mathrm{g}^*(v)\right) = 0$ *for all* $v \,:\, v \in V_{n\leq\infty}$.

Consider the case of adding v_x as the nth node in RRT* (Figure 4), where v_x is located at x. The RRT* parent of v_x is p and $\mathrm{d}(v_x, p)$ is the distance from v_x to p. The length of the trajectory from v_x to p is $\mathrm{d}_\pi(v_x, p)$. The radius of the shrinking neighborhood ball is r. By construction $\mathrm{d}(v_x, p) < r$ and $\mathrm{d}_\pi(v_x, p) < r$. Let $\hat{\mathrm{d}}(v_x, p)$ be a stand in for both $\mathrm{d}(v_x, p)$ and $\mathrm{d}_\pi(v_x, p)$. The following proposition comes from the fact that $\lim_{n\to\infty} r = 0$.

Proposition 3. $\lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(\hat{\mathrm{d}}(v_x, p)\right) = 0$*, where* p *is RRT* parent of* v_x.

We now prove that the expected difference between the *initial* cost-to-goal of a node v_x and its *optimal* cost-to-goal approaches 0, in the limit, as $n \to \infty$ (i.e., as the iteration that v_x is inserted into the graph approaches infinity).

Lemma 2. $\lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(\mathrm{g}_n(v_x) - \mathrm{g}^*(v_x)\right) = 0$.

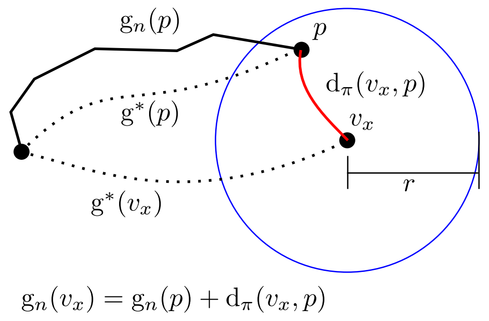


Fig. 4. Node v_x at x is inserted when $n = |V_n|$. The parent of v_x is p. $d_\pi(v_x, p)$ is the distance from v_x to p along the (red) trajectory. $g_n(v_x)$ and $g_n(p)$ are the cost-to-goals of v_x and p when $n = |V_n|$, while $g^*(v_x)$ and $g^*(p)$ are their optimal cost-to-goals, respectively. The neighbor ball (blue) has radius r. Obstacles are not drawn.

Proof. By the triangle inequality $g^*(v_x) + d(p, v_x) \geq g^*(p)$. Rearranging and then adding $g_n(v_x)$ to either side:

$$g_n(v_x) - g^*(v_x) \leq g_n(v_x) - g^*(p) + d(p, v_x). \tag{1}$$

(1) holds over all $S \in \{\hat{S} : V_n \setminus V_{n-1} = \{v_x\}\} \subset \mathcal{S}$, thus

$$\mathbb{E}_{n,v_x}\left(g_n(v_x) - g^*(v_x)\right) \leq \mathbb{E}_{n,v_x}\left(g_n(v_x) - g^*(p) + d(p, v_x)\right). \tag{2}$$

By construction $g_n(v_x) = g_n(p) + d_\pi(v_x, p)$. Substituting into (2), using the linearity of expectation, and taking the limit of either side:

$$\lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(g_n(v_x) - g^*(v_x)\right) \leq$$
$$\lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(g_n(p) - g^*(p)\right) + \lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(d_\pi(v_x, p)\right) + \lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(d(p, v_x)\right).$$

The law of large numbers guarantees that x (i.e., location of v_x) becomes uncorrelated with the cost-to-goal of p, in the limit as $n \to \infty$. Thus, consequently: $\lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(g_n(p) - g^*(p)\right) = \lim_{n\to\infty} \mathbb{E}_n\left(g_n(p) - g^*(p)\right)$. Using Corollary 5 and Proposition 3 (twice) finishes the proof. □

Applying the law of total expectation yields the following corollary regarding the nth node added to V, and completes step 1 of the overall proof.

Corollary 6. $\lim_{n\to\infty} \mathbb{E}_n\left(g_n(v) - g^*(v)\right) = 0$, *where* $V_n \setminus V_{n-1} = \{v\}$.

We now show that, for RRT*, the expected cumulative difference between initial and final cost-to-go over all nodes in $\mathcal{G}_n$ is dominated by a linear function of n (i.e., is $O(n)$). Note that this is the sum of all cost that was, is, or ever will be propagated through any node in $\mathcal{G}_n$.

Lemma 3. $\sum_{m=1}^{n} \mathbb{E}_m\left(g_m(v_m) - g^*(v_m)\right) = O(\epsilon n)$ *for all* n *such that* $1 \leq n < \infty$ *and where* $V_m \setminus V_{m-1} = \{v_m\}$ *for all* m *s.t.* $1 \leq m \leq \infty$.

Proof. Using the definition of a limit with Corollary 6 shows that for any $c_1 > 0$ there must exist some $n_c < \infty$ such that $\mathbb{E}_n\left(g_n(v) - g^*(v)\right) < c_1$ for all $n > n_c$. We choose $c_1 = \epsilon$ and define $c_2 = \sum_{m=1}^{n_c} \mathbb{E}_m\left(g_m(v_m) - g^*(v_m)\right)$ so that by construction $\sum_{m=1}^{n} \mathbb{E}_m\left(g_m(v_n) - g^*(v_n)\right) \leq c_2 + \epsilon n$ for all n s.t. $1 \leq n < \infty$. □

Let $f^{pr}(n)$ denote the total number of cost propagations that occur (i.e., through any and all nodes) in RRT$^{\text{X}}$ as a function of $n = |V_n|$.

Lemma 4. $\mathbb{E}_n\left(f^{pr}(n)\right) = O(n)$

Proof. When $m = |V_m|$ a propagation is possible only if there exists some node v such that $\mathrm{g}_m(v) - \mathrm{g}^*(v) > \epsilon$. Assuming RRT* and RRT$^{\mathrm{X}}$ use the same S, then by construction $\mathrm{g}_m(v)$ for RRT* is an upper bound on $\mathrm{g}_m(v)$ for RRT$^{\mathrm{X}}$ for all $v \in V_m$ and m such that $1 \leq m < \infty$. Thus, $f^{pr}(n) \leq (1/\epsilon) \sum_{m=1}^{n} \mathrm{g}_m(v_m) - \mathrm{g}^*(v_m)$, where $\mathrm{g}_m(v_m)$ is the RRT*-value of this quantity. Using the linearity of expectation to apply Lemma 3 we find that $\mathbb{E}_n\left(f^{pr}(n)\right) \leq (1/\epsilon) O\left(\epsilon n\right)$. □

Corollary 7. $\lim\limits_{n\to\infty} \frac{\mathbb{E}_n(f^{pr}(n))}{cn} \leq 1$ *for some constant* $c < \infty$.

Corollary 7 concludes step two of the overall proof. The following Lemma 5 uses the notion of runtime amortization[10]. Let $\hat{f}^{single}(n)$ denote the *amortized* time to propagate an ϵ-cost reduction from node v to $N(v)$ when $n = |V_n|$.

Lemma 5. $\mathbb{P}\left(\{\lim\limits_{n\to\infty} \frac{\hat{f}^{single}(n)}{c\log n} \leq 1\}\right) = 1$ *for some constant* $c < \infty$.

Proof. By construction, a single propagation through v requires interaction with $|N(v)|$ neighbors. Each interaction normally requires $\Theta\left(1\right)$ time–except when the interaction results in $u \in N(v)$ receiving an ϵ-cost decrease. In the latter case u is added/updated in the priority queue in $O\left(\log n\right)$ time; however, we add this $O\left(\log n\right)$ time to u's *next* propagation time, so that the current propagation from v only incurs $\Theta\left(1\right)$ *amortized* time per each $u \in N(v)$. To be fair, v must account for any similar $O\left(\log n\right)$ time that it has absorbed from each of the c_1 nodes that have given it an ϵ-cost reduction since the last propagation from v. But, for $c_1 \geq 1$ the current propagation from v is at least $c_1\epsilon$ and so we can count it as c_1 different ϵ-cost decreases from v to $N(v)$ (and v only touches each $u \in N(v)$ once). Hence, $c_1\hat{f}^{single}(n) = |N(v)| + c_1 O\left(\log n\right)$. By the law of large numbers, $\mathbb{P}\left(\{\lim_{n\to\infty} |N(v)| = c_2 \log n)\}\right) = 1$, for some constant c_2 s.t. $0 < c_2 < \infty$. Hence, $\mathbb{P}\left(\{\lim_{n\to\infty} \hat{f}^{single}(n) \leq (c_2/c_1)\log n + \log n\}\right) = 1$, setting $c = 1 + c_2/c_1$ finishes the proof. □

Corollary 8. $\lim\limits_{n\to\infty} \frac{\mathbb{E}_n\left(\hat{f}^{single}(n)\right)}{c\log n} \leq 1$ *for some constant* $c < \infty$.

Let $f^{all}(n)$ denote the *total* runtime associated with cost propagations by the iteration that $n = |V_n|$, where $f^{all}(n) = \sum_{j=1}^{f^{pr}(n)} \hat{f}^{single}(m_j)$ for a particular run of RRT$^{\mathrm{X}}$ resulting in $n = |V_n|$ and $f^{pr}(n)$ individual ϵ-cost decreases.

Lemma 6. $\lim\limits_{n\to\infty} \frac{\mathbb{E}_n\left(f^{all}(n)\right)}{cn\log n} < 1$ *for* $c \leq \infty$.

Proof. $\lim_{n\to\infty} \mathbb{E}_n\left(f^{pr}(n)\right) \neq 0$ and $\lim_{n\to\infty} \mathbb{E}_n\left(\hat{f}^{single}(n)\right) \neq 0$, so obviously $\lim\limits_{n\to\infty} \frac{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)}{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)} = 1$. Although $\hat{f}^{single}(n)$ and $f^{pr}(n)$ are mutually dependent, in general, they become *independent*[11] in the limit as $n \to \infty$. Thus, $\lim\limits_{n\to\infty} \frac{\mathbb{E}_n\left(f^{pr}(n)\hat{f}^{single}(n)\right)}{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)} = \lim\limits_{n\to\infty} \frac{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)}{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)} = 1$. Note that the previous step would not have been allowed *outside* the limit. Using algebra: $\lim\limits_{n\to\infty} \frac{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr}(n)} \hat{f}^{single}(n)\right)}{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)} = 1$. Using Corollaries 7 and 8:
$\lim\limits_{n\to\infty} \frac{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr}(n)} \hat{f}^{single}(n)\right)}{c_1c_2 n\log n} \leq \lim\limits_{n\to\infty} \frac{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr}(n)} \hat{f}^{single}(n)\right)}{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)} = 1$ for some $c_1, c_2 < \infty$. Using algebra and defining $c = c_1c_2$ finishes the proof. □

Corollary 9. $\mathbb{E}_n\left(f^{all}(n)\right) = O\left(n\log n\right)$.

Theorem 2. $\mathbb{E}_n\left(f^{RRT^X}(n)\right) = \Theta\left(n\log n\right)$.

Proof. When $\Delta\mathcal{X}_{\mathrm{obs}} = \emptyset$ RRT$^{\mathrm{X}}$ differs from RRT* in 3 ways: (1) ϵ-cost propagation, (2) neighbor list storage, and (3) neighbor list culling[12]. The runtime of RRT$^{\mathrm{X}}$ can be found by adding the extra time of (1), (2), and (3) to that of RRT*.

[10] In particular, if a node u receives an ϵ-cost decrease $> \epsilon$ via another node v, then u agrees to take responsibility for the runtime associated with that exchange (i.e., including it as part u's next propagation time).

[11] i.e., because the number of neighbors of a node converges to the function $\log n$ with probability 1 (as explained in Lemma 5)

[12] Note that neighbors that are not removed during a cull are touched again during the RRT*-like rewiring operation that necessarily follows a cull operation.

Corollary 9 gives the asymptotic time of (1). (2) and (3) are $O\left(|N(v)|\right)$, the same as finding a new node's neighbors in RRT*. Therefore, $\mathbb{E}_n\left(f^{RRT^X}(n)\right) = \mathbb{E}_n\left(f^{RRT*}(n)\right) + \mathbb{E}_n\left(f^{all}(n)\right)$. Trivially:
$\mathbb{E}_n\left(f^{RRT*}(n)\right) = \Theta\left(\mathbb{E}_n\left(f^{RRT*}(n)\right)\right)$, and by Theorem 3 and Corollary 9:
$\mathbb{E}_n\left(f^{RRT^X}(n)\right) = \Theta\left(n\log n\right) + O\left(n\log n\right) = \Theta\left(n\log n\right)$ □

5.3. Expected Time until $|V| = n$ for RRT#

RRT# does not cull neighbors (in contrast to RRT^X) and so all nodes continue to accumulate neighbors forever.

Lemma 7. $\mathbb{E}_n\left(|N(v)|\right) = \Theta\left(\log^2 n\right)$ *for all* v *s.t.* $v \in V_m$ *for some* $m < n < \infty$.

Proof. Assuming v is inserted when $m = |V_m|$, the expected value of $\mathbb{E}_n\left(|N(v)|\right)$, where $n = |V_n|$ for some $n > m$ is:
$\mathbb{E}_n\left(|N(v)|\right) = c\log m + \sum_{j=m+1}^{n}\frac{c\log j}{j} = c\left(\log m + \left(\sum_{j=1}^{n}\frac{\log j}{j}\right) - \left(\sum_{j=1}^{m}\frac{\log j}{j}\right)\right)$ where c is constant. Corollary 1 finishes the proof. □

RRT# propagates all cost changes (in contrast, RRT^X only propagates those larger than ϵ). Thus, *any* cost decrease at v is propagated to *all* descendants of v, *plus* any additional nodes that become new descendants of v due to the propagation. Let $f^{pr\#}(n)$ be the number of propagations (i.e., through a single node) that have occurred in RRT# by the iteration that $n = |V_n|$. We now prove that $f^{pr\#}(n)$ dominates n in the limit as n approaches infinity.

Lemma 8. $\mathbb{P}(\lim_{n\to\infty}\frac{n}{f^{pr\#}(n)} = 0) = 1$ *with respect to* $\mathcal{S}$.

Proof. By contradiction. Assume that $\mathbb{P}(\lim_{n\to\infty}\frac{n}{f^{pr\#}(n)} = 0) = c_1 < 1$. Then there exists some c_2 and c_3 such that $\mathbb{P}(\lim_{n\to\infty}\frac{n}{f^{pr\#}(n)} \geq c_2 > 0) = c_3 > 0$ and therefore $\mathbb{E}(\lim_{n\to\infty}\frac{n}{f^{pr\#}(n)}) \geq c_2 c_3 > 0$, and the expected number of cost propagations to each v s.t. $v \in V_n$ is $c_4 = \frac{1}{c_2 c_3} < \infty$, in the limit as $n \to \infty$. This is a contradiction because v experiences an infinite number of cost *decreases* with probability 1 as a result of RRT#'s asymptotic optimal convergence, and each decrease (at a non-leaf node) causes at least one propagation. □

The runtime of RRT# can be expressed in terms of the runtime of RRT* plus the extra work required to keep the graph consistent (cost propagations):

$$f_{m_i}^{RRT\#} = \Theta\left(f_{m_i}^{RRT*}\right) + \sum_{j=1}^{f^{pr\#}(m_i)} f_j^{pr\#}. \tag{3}$$

Here, $f^{pr\#}(m_i)$ is the total number of cost propagations (i.e., through a single node) by iteration i when $m_i = |V|$, and $f_j^{pr\#}$ is the time required for the jth propagation (i.e., through a single node). Obviously $f_j^{pr\#} > c$ for all j, where $c > 0$. Also, for all $j \geq 1$ and all m the following holds, due to non-decreasing expected neighbor set size vs. j:

$$\mathbb{E}_m\left(f_j^{pr\#}\right) \leq \mathbb{E}_m\left(f_{j+1}^{pr\#}\right) \tag{4}$$

Lemma 9. $\lim\limits_{n\to\infty}\frac{n\log^2 n}{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} = 0.$

Proof. $\lim\limits_{n\to\infty}\frac{\mathbb{E}_n\left(\sum_{j=1}^{n} f_j^{pr\#}\right)}{\mathbb{E}_n\left(\sum_{j=n+1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} = 0$ because the ratio between the number of terms in the numerator vs. denominator approaches 0, in the limit, by Lemma 8, and the smallest term in the denominator is no smaller than the largest term in the numerator, by (4). Obviously, $\lim\limits_{n\to\infty}\frac{\mathbb{E}_n\left(\sum_{j=1}^{n} f_j^{pr\#}\right)}{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} \leq \lim\limits_{n\to\infty}\frac{\mathbb{E}_n\left(\sum_{j=1}^{n} f_j^{pr\#}\right)}{\mathbb{E}_n\left(\sum_{j=n+1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} = 0$. Rearranging: $\lim\limits_{n\to\infty}\frac{\sum_{j=1}^{n}\mathbb{E}_{m_j}\left(f_j^{pr\#}\right)}{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} = 0$. By Lemma 7: $\lim\limits_{n\to\infty}\frac{\sum_{j=1}^{n}\mathbb{E}_{m_j}\left(c\log^2 m_j\right)}{\sum_{j=1}^{n}\mathbb{E}_{m_j}\left(f_j^{pr\#}\right)} = 1$ for some constant c such that $0 < c < \infty$. Hence,

$$\lim_{n\to\infty}\frac{\sum_{j=1}^{n}\mathbb{E}_{m_j}\left(f_j^{pr\#}\right)}{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr\#}(n)} f_j^{pr\#}\right)}\;\frac{\sum_{j=1}^{n}\mathbb{E}_{m_j}\left(c\log^2 m_j\right)}{\sum_{j=1}^{n}\mathbb{E}_{m_j}\left(f_j^{pr\#}\right)} = \frac{c\sum_{j=1}^{n}\mathbb{E}_{m_j}\left(\log^2 m_j\right)}{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} = 0$$

Corollary 2 with the linearity of expectation finishes the proof. □

Corollary 10. $\mathbb{E}_n\left(\sum_{j=1}^{f^{pr\#}(n)} f_j^{pr\#}\right) = \omega\left(n \log^2 n\right)$.

In the above, $\omega\left(n \log^2 n\right)$ is a stronger statement than $\Omega\left(n \log^2 n\right)$. We are now ready to prove the asymptotic runtime of RRT$^\#$.

Theorem 3. $\mathbb{E}_n\left(f_n^{RRT\#}\right) = \omega\left(n \log^2 n\right)$

Proof. Taking the limit (as $m_i = n \to \infty$) of the expectation of either side of (3) and then using Corollaries 3 and 10, we see that the expected runtime of RRT$^\#$ is dominated by propagations: $\mathbb{E}_n\left(f_n^{RRT\#}\right) = \Theta\left(n \log n\right) + \omega\left(n \log^2 n\right)$. □

5.4. *RRT^X Obstacle Addition/Removal in Dynamic Environments*

The addition of an obstacle requires finding V_{obs}, the set of all nodes with trajectories through the obstacle, and takes expected $O\left(|\mathcal{D}(V_{\text{obs}})| \log n\right)$ time. The resulting call to `reduceInconsistency`() interacts with each $u \in \mathcal{D}(V_{\text{obs}})$, where $\mathcal{D}(V_{\text{obs}})$ is the set of all descendants of all $u \in V_{\text{obs}}$, and each interaction takes expected time $O\left(\log n\right)$ due to neighbor sets and heap operations. Thus, adding an obstacle requires expected time $O\left(|\mathcal{D}(V_{\text{obs}})| \log n\right)$. Removing an obstacle requires similar operations and thus the same order of expected time. In the special case that the obstacle has existed since t_0, then $\mathcal{D}(V_{\text{obs}}) = \emptyset$ and time is $O\left(1\right)$.

5.5. *Information transfer time of RRT^X vs. RRT* and RRT$^\#$*

Another important performance metric is the time required by an algorithm to transfer information. In particular, how long does it take for knowledge of a cost decrease at node v to be transferred to the relevant portions of the graph? e.g., to v_{robot}? Information transfer time is essentially the reaction time of the robot, and quicker information transfer enables more nimble replanning.

RRT* and RRT$^\#$ were originally designed to solve problems in static environments and cannot handle cost increases (for instance, due to the appearance of an unexpected obstacle). Consequently, information transfer time of RRT* and RRT$^\#$ are technically only defined for cost decreases (such as when a shortcut is discovered due to random sampling — a common event when planning in static environments). That said, it is conceivable that RRT$^\#$ could be modified to handle both increases and decreases by using some of the techniques proposed in the current paper. The discussion that follows regarding information transfer time for RRT$^\#$ is simultaneously valid for both the original RRT$^\#$ and an augmented version of the algorithm that handles cost increases with an increase propagation wave (similar to RRT^X). On the other hand, RRT* cannot be modified to handle cost increases without significantly altering the runtime properties of the algorithm; the following discussion regarding information transfer time for RRT* is therefore valid only for static environments as well as the special case of dynamic environments where obstacles may disappear but *not* appear.

Let $t_{v,u}^{RRT\#}(n)$ denote the time required in RRT$^\#$ to transfer information from node v to node u when $n = |V|$, and where we assume that $u \in \mathcal{D}(v)$ after the transfer. Let $t_{v,u}^{RRT^X}(n)$ denote a similar quantity for RRT^X.

Let $\Delta\text{g}(v)$ denote the magnitude of the cost decrease at v. Let $\hat{\mathcal{D}}(v)$ be the set of all nodes to which information is propagated during a rewiring cascade. By construction, $\hat{\mathcal{D}}(v) = \{u : u \in \mathcal{D}(v) \wedge \text{g}(u) \leq \text{g}(v_{\text{bot}})\}$, where $\mathcal{D}(v)$ is the descendant set of v calculated *after* the cascade. By inspection, it is easy to see that if $\Delta\text{g}(v) > \epsilon$, then RRT^X should have a quicker information transfer time than RRT$^\#$. This is formalized in the following theorem.

Theorem 4. *If* $\Delta\text{g}(v) > \epsilon$ *then* $\mathbb{E}_n\left(t_{v,v_{\text{bot}}}^{RRT^X}(n)\right) = O\left(\mathbb{E}_n\left(t_{v,v_{\text{bot}}}^{RRT\#}(n)\right)\right)$

Proof. When $\Delta\text{g}(v) > \epsilon$ then both RRT^X and RRT$^\#$ use rewiring cascades to propagate knowledge to all $\hat{\mathcal{D}}(v)$ before information transfer is complete. Each step of the cascade (i.e. through a particular node $u \in \hat{\mathcal{D}}(v)$) of either algorithm requires touching $|N(u)|$ neighbors of u. Let $|N(u)|_\#$ be the $|N(u)|$ of RRT$^\#$ and let $|N(u)|_{\text{x}}$ be the $|N(u)|$ of RRT^X,

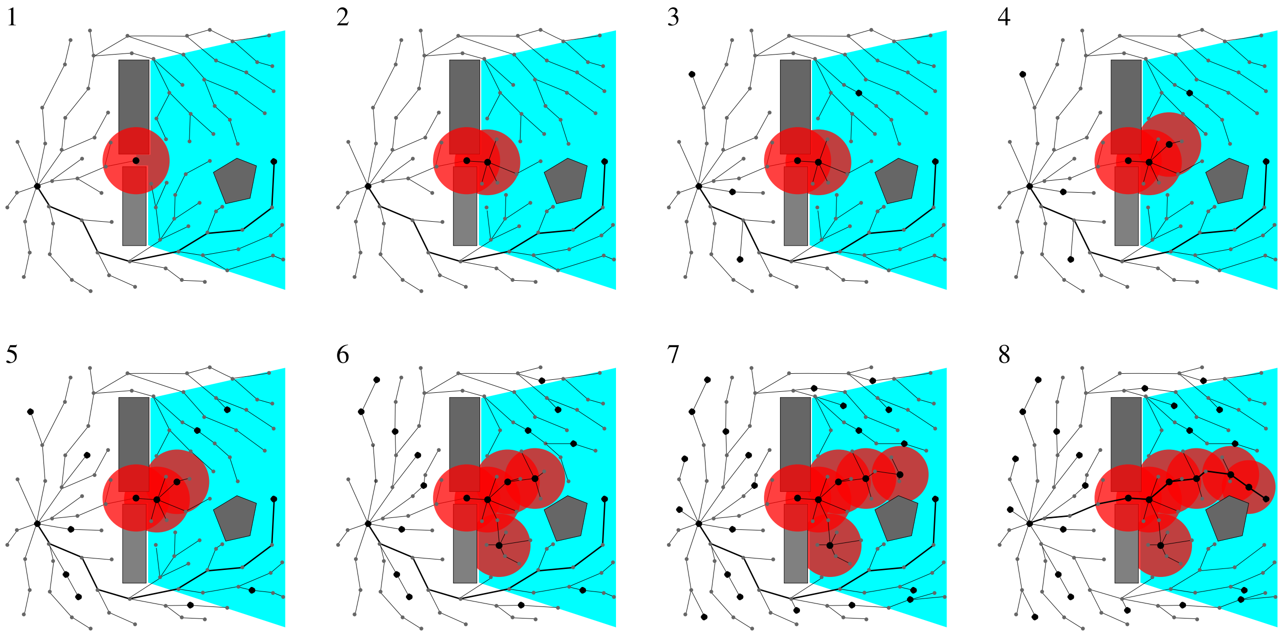


Fig. 5. Lazy information propagation in RRT* after a shortcut is found in between two obstacles. Light blue is the region where information can propagate, red is the region where information has propagated. Obstacles are gray, the search tree has black edges and gray/black nodes. Start and robot nodes are black, as are all nodes added after the shortcut is found. Smaller gray nodes are from before the shortcut was found at v. 1: The original shortcut is found. 2: a new node u can connect to v and information is propagated further. 3: four nodes are inserted that do not propagate information. 4-7: nodes continue to be added (some contribute to propagation, others do not). 8: The information about the shortcut is finally propagated to v_{bot}.

assuming RRT$^{\#}$ and RRT$^{\mathrm{X}}$ use the same random sample sequence S. By construction, for all u the following holds: $|N(u)|_{\mathrm{x}} \leq |N(u)|_{\#}$. Indeed, $\lim_{n\to\infty} |N(u)|_{\mathrm{x}}/|N(u)|_{\#} = 0$ with probability 1, i.e., because the neighborhood of a particular node u is maintained to be constant in RRT$^{\mathrm{X}}$ and increases without bound in RRT$^{\#}$. Note that while $|\mathcal{D}(v)|$ may not be the same for RRT$^{\mathrm{X}}$ and RRT$^{\#}$, the ratio between the two approaches 1 in the limit, as $n \to \infty$. □

Alternatively, for changes less than ϵ, we should expect the information transfer time of RRT$^{\mathrm{X}}$ to be similar to that of RRT*.

Theorem 5. *If* $\Delta\mathrm{g}(v) < \epsilon$ *then* $\mathbb{E}_n\left(t^{RRT^X}_{v,v_{\mathrm{bot}}}(n)\right) = O\left(\mathbb{E}_n\left(t^{RRT*}_{v,v_{\mathrm{bot}}}(n)\right)\right)$

Proof. When $< \epsilon$ both RRT* and RRT$^{\mathrm{X}}$ use the exact same information transfer process (i.e., lazy rewiring). □

However, how RRT$^{\mathrm{X}}$ compares to RRT* when $\Delta\mathrm{g}(v) > \epsilon$ is not obvious. The rest of this section is dedicated to characterizing the performance of RRT*, in general, as well as the performance of RRT^{X} when $\Delta\mathrm{g}(v) > \epsilon$ in order to make this comparison.

In RRT* information transfer happens over many iterations, and each iteration runs in expected time $O(\log n)$. In contrast, RRT$^{\mathrm{X}}$ propagates all changes in a single iteration but that particular iteration typically requires much more time than the average iteration of RRT$^{\mathrm{X}}$. In order to formalize comparison between RRT* and RRT$^{\mathrm{X}}$, we assuming both algorithms use the same S and then define that RRT* has "completed" an "ϵ-close" information transfer when $\mathrm{g}(v_{\mathrm{bot}})$ of RRT* is $\leq \epsilon + \mathrm{g}(v_{\mathrm{bot}})$ of RRT$^{\mathrm{X}}$. Let $t^{RRT*}_{\epsilon,v,v_{\mathrm{bot}}}(n)$ denote the time required for an ϵ-close information transfer in RRT* from v to v_{bot}.

RRT* uses "Lazy-propagation" as depicted in Figure 5. The particular process by which information is transferred through the graph depends on $\mathcal{X}$ and $\mathcal{X}_{\mathrm{obs}}$ and is difficult to analyze, in general. On the other hand, it is straightforward to

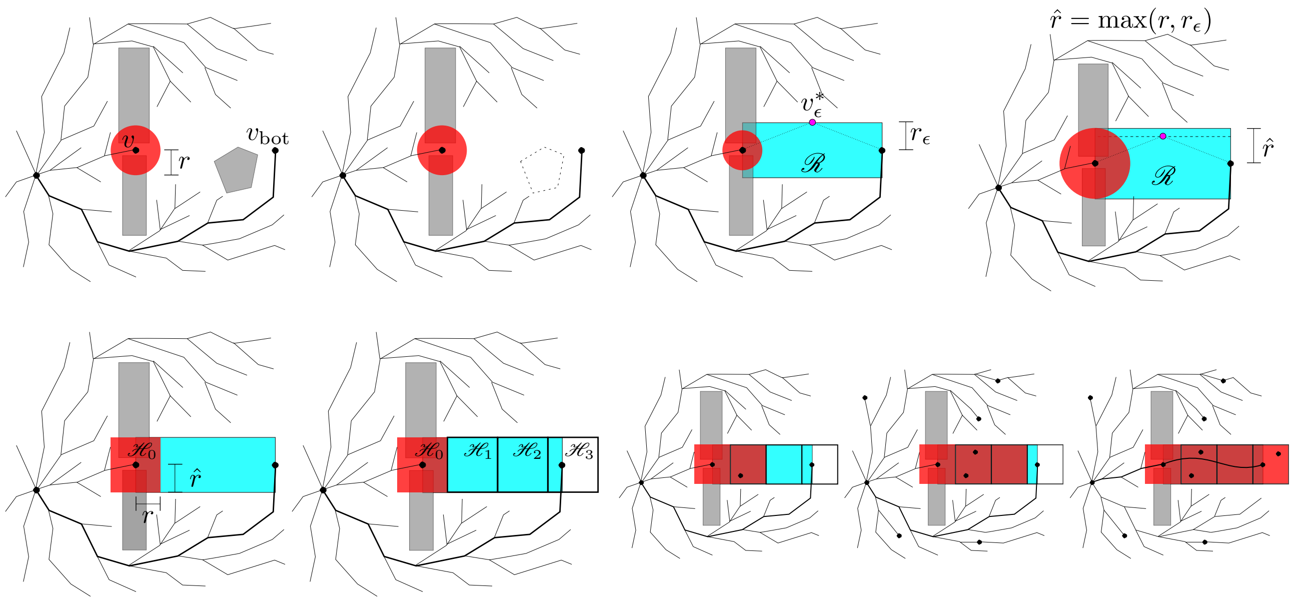


Fig. 6. The simplified process we evaluate to find a lower bound on the information transfer time of RRT*. r is the radius of the neighbor ball. r_ϵ is defined by the point $v_\epsilon^* = \arg\min u \,:\, \mathrm{g}(v) + \mathrm{d}(v,u) + \mathrm{d}(u, v_{\text{bot}}) \geq \mathrm{g}(v_{\text{bot}}) + \epsilon$, where the final $\mathrm{g}(v_{\text{bot}})$ is the post-cascade value from RRT$^{\text{X}}$. $\hat{r} = \max(r, r_\epsilon)$. The hyperrectangle $\mathscr{R}$ (light blue) has volume $\hat{r}^{D-1}\mathrm{d}(v, v_{\text{bot}})$ Assuming information has propagated to $\mathscr{H}_{k-1}$ (small red hyperrectangles) then sampling from within $\mathscr{H}_{k-1} \cup \mathscr{H}_k$ causes information to propagate to $\mathscr{H}_k$. Volume $\mathscr{L}\left(\mathscr{H}_k\right) = 2^D r \hat{r}^{(D-1)}$.

calculate a *loose lower bound* on $\mathbb{E}_n\left(t^{RRT*}_{\epsilon,v,v_{\text{bot}}}(n)\right)$ by considering the simplified process (constructed in Figure 6) that is guaranteed to transfer information more quickly than RRT*.

We now walk through the differences/considerations between/of general information propagation in RRT* and the simplified process. Note that each simplification results in a slightly looser bound on $\mathbb{E}_n\left(t^{RRT*}_{\epsilon,v,v_{\text{bot}}}(n)\right)$.

1. Starting with the same $\mathcal{X}$ and $\mathcal{X}_{\text{obs}}$ as RRT*.
2. Obstacles between v and v_{robot} are removed in the simplified process (thus, transfer time may be artificially shorter due to the triangle inequality since it does not have to go around obstacles).
3. Consider the point $v_\epsilon^* = \arg\min u \,:\, \mathrm{g}(v) + \mathrm{d}(v,u) + \mathrm{d}(u, v_{\text{bot}}) \geq \mathrm{g}(v_{\text{bot}}) + \epsilon$, where the final $\mathrm{g}(v_{\text{bot}})$ is the post-cascade value from RRT$^{\text{X}}$. Let r_ϵ be the shortest distance from v_ϵ^* to the line segment (v, v_{bot}).
4. Let $\hat{r} = \max(r, r_\epsilon)$ and consider the D-dimensional hyperrectangle $\mathscr{R}$ that has v and v_{bot} in the center of its 'left' and 'right' faces, and the other faces are located $\hat{r}$ from (v, v_{bot}). The volume of $\mathscr{R}$ is $\mathscr{L}\left(\mathscr{R}\right) = \hat{r}^{(D-1)}\mathrm{d}(v, v_{\text{bot}})$. By construction, any ϵ-close information transfer in RRT* must follow a path through the interior of $\mathscr{R}$.
5. We assume that inserting v causes information transfer to all points $x \in \mathscr{H}_0$ at no *additional* runtime cost beyond the usual rewiring of neighbors in $N(v)$, where $\mathscr{H}_0$ is the hyperrectangle with volume $\mathscr{L}\left(\mathscr{H}_0\right) = 2^D r \hat{r}^{(D-1)}$ that is centered at v and aligned with $\mathscr{R}$. This loosens the bound vs. RRT* because an insertion causes more nodes to receive the new information than in RRT*.
6. We cover the rest of $\mathscr{R}$ with non-overlapping hyperrectangles $\mathscr{H}_k$ for $k = 1, \ldots, K$, where $K = \lceil(\mathrm{d}(v, v_{\text{bot}}) - r)/(2r)\rceil$, allowing $\mathscr{H}_K$ to go past the end of $\mathscr{R}$. (Note that $\mathscr{L}\left(\mathscr{H}_k\right) = \mathscr{L}\left(\mathscr{H}_0\right)$, i.e. information transfer volume does not shrink from the insertion of v until ϵ-close information transfer has been completed, this loosens the bound vs. RRT*.
7. We assume that if information has propagated to any point in $\mathscr{H}_{k-1}$, then inserting a point in $\mathscr{H}_{k-1} \cup \mathscr{H}_k$ causes "perfect" information transfer to all $x \in \mathscr{H}_k$. This artificially increases the probability of sampling near a part of the graph that has already received the information, and thus decreases the expected time of information transfer. Furthermore, we assume there is no *additional* runtime cost beyond rewiring of neighbors in $N(v)$ where we define $\mathbb{E}\left(|N(v)|\right)$ to be

constant until information propagates to v_{bot}—further loosening the bound vs. RRT* since RRT* has increasing $|N(v)|$ vs. n.

8. Finally, we assume that once information has propagated to $\mathscr{H}_K$ then ϵ-close information propagation is complete. This further loosens the bound vs. RRT* because there are many paths through $\mathscr{R}$ that are not ϵ-close.

By Construction an ϵ-close information transfer in the simplified process happens if and only if information travels through $\mathscr{H}_1, \ldots, \mathscr{H}_K$.

Theorem 6. $\mathbb{E}_n\left(t^{RRT*}_{\epsilon,v,v_{\mathrm{bot}}}(n)\right) = \Omega\left(n\left(\frac{n}{\log n}\right)^{1/D}\right)$

Proof. There are $K = \lceil(\mathrm{d}(v, v_{\mathrm{bot}}) - r)/(2r)\rceil$ information transfers (one each for $\mathscr{H}_1, \ldots, \mathscr{H}_K$). Assuming information has propagated to $\mathscr{H}_{k-1}$, the chances that a new point causes a transfer to $\mathscr{H}_{k-1}$ is given by $\frac{\mathscr{L}(\mathscr{H}_{k-1}\cup\mathscr{H}_k)}{\mathscr{L}(\mathcal{X})}$ and so the expected time to propagate from $\mathscr{H}_{k-1}$ to $\mathscr{H}_k$ is $\frac{\mathscr{L}(\mathcal{X})}{\mathscr{L}(\mathscr{H}_{k-1}\cup\mathscr{H}_k)}$. By construction $\mathscr{L}\left(\mathscr{H}_{k-1}\cup\mathscr{H}_k\right) = 2^{(D+1)}r\hat{r}^{(D-1)}$, and $\mathscr{L}\left(\mathcal{X}\right)$ can be considered a constant for the purposes of this analysis. In our simplified process the expected time required for neighbor interactions is held fixed at $\Theta\left(\log n\right)$ (a lower bound on the actual case). Thus, recursion can be used to calculate the expected time for an ϵ-close information transfer, i.e.,

$$\mathbb{E}_n\left(t^{RRT*}_{\epsilon,v,v_{\mathrm{bot}}}(n)\right) \geq \sum_{k=1}^{K} c_1 \frac{\mathscr{L}\left(\mathcal{X}_{free}\right)}{\mathscr{L}\left(\mathscr{H}\right)} \log n = c_2 \frac{\lceil(\mathrm{d}(v, v_{\mathrm{bot}}) - r)/(2r)\rceil}{r\hat{r}^{(D-1)}} \log n \tag{5}$$

for constants c_1 and c_2. By construction $r \leq \hat{r}$ and from Karaman and Frazzoli (2011) we know that $r = \gamma\left(\frac{\log n}{n}\right)^{1/D}$. Substituting into (5) and simplifying yields the following looser bound:

$$\mathbb{E}_n\left(t^{RRT*}_{\epsilon,v,v_{\mathrm{bot}}}(n)\right) \geq c_3\left(\frac{\mathrm{d}(v, v_{\mathrm{bot}})}{(\gamma^D+1)}\frac{n}{\log n}\left(\frac{n}{\log n}\right)^{1/D} - \frac{n}{\gamma^D \log n}\right)\log n$$

and noting that γ is constant given D and $\mathrm{d}(v, v_{\mathrm{bot}})$ is constant given v and v_{bot},

$$\mathbb{E}_n\left(t^{RRT*}_{\epsilon,v,v_{\mathrm{bot}}}(n)\right) = \Omega\left(c_D\ n\left(\frac{n}{\log n}\right)^{1/D}\right)$$

where $c_D = \Theta\left(\frac{\mathrm{d}(v,v_{robot})\mathscr{L}(\mathcal{X})}{(2\gamma)^{d+1}}\right)$ is a constant depending on d. □

Theorem 7. *If* $\Delta\mathrm{g}(v) > \epsilon$ *then* $\mathbb{E}_n\left(t^{RRT^X}_{v,v_{\mathrm{bot}}}(n)\right) = O\left(n\log n\right)$

Proof. In a "worst-case" scenario v is the root node, and so a change must propagate through all n nodes in the graph. Each node u has $\mathbb{E}_n\left(|N(u)|\right) = \Theta\left(\log n\right)$. As discussed in the derivation of Lemma 5, the total amortized time for a propagation through u is $\Theta\left(1\right)$ per each $w \in N(u)$, and hence using linearity of expectation, $\mathbb{E}_n\left(t^{RRT^X}_{v,v_{\mathrm{bot}}}(n)\right) \leq c\sum_{j=1}^{n}\log n$, for some constant c, where the bound is loose, since we are considering a "worst case" scenario. Indeed, the full amortization is even repaid in this worst case, since leaf nodes do not propagate cost further. □

The fact that $\left(\frac{n}{\log n}\right)^{1/D} = \omega\left(\log n\right)$ for all $D < \infty$ gives the final corollary:

Corollary 11. *If* $\Delta\mathrm{g}(v) > \epsilon$ *then* $\mathbb{E}_n\left(t^{RRT*}_{\epsilon,v,v_{\mathrm{bot}}}(n)\right) = \omega\left(\mathbb{E}_n\left(t^{RRT^X}_{v,v_{\mathrm{bot}}}(n)\right)\right)$

For cost changes $> \epsilon$, RRT^{X} has a faster information transfer time than RRT*.

6. Simulations and Experiments

In Section 6.1 we show how RRT^X can be applied in a variety of dynamic environments. While we include still-frame image sequences of examples in the current paper (limited to two examples for brevity), videos are a much better medium in which to observer the utility of RRT^X in dynamic environments. We have uploaded a number of videos showing the real-time performance of RRT^X at (Otte, 2014), and we encourage readers to watch them.

In Section 6.2 we run a number of experiments in both static and dynamic environments comparing the performance of RRT^X vs. other algorithms as well as evaluating the effects of different ϵ. In dynamic environments we compare RRT^X to both DRRT Ferguson et al. (2006) and the naive approach of re-starting RRT whenever the current path becomes invalid. DRRT was chosen because it is the only other single-query sampling-based motion planning algorithm we know of that plans from goal to start (this is advantageous for a number of reasons discussed in Section 2. The comparison to restarting RRT is included as a baseline.

Experiments in static environment involve a bottleneck environment designed to highlight the benefits of graph consistency and ϵ-consistency with respect to information transfer time (i.e. the time it takes new information to become actionable with respect to the robot's path).

Experiments 1 and 2 are run on a Dell OptiPlex 3020 with Intel Core i5 and 8 GB of RAM, all other experiments use a Dell OptiPlex 790 with Intel Core i7 and 16 GM of RAM (note that only a single processor core is used in all experiments).

6.1. Simulations in Dynamic Environments

Holonomic robot in a $\mathbb{R}^D$. The state space is a D-dimensional Euclidean space $\mathcal{X} \subset \mathbb{R}^D$, and the workspace is equivalent to the configuration space. The robot is Holonomic with arbitrary volume and boundary. In the case that the robot is a point-robot than this reduces to case of geometric path-planning. We assuming that collision vs. obstacles is straightforward to determine[13]. This is arguably the simplest type of planning and $\pi(v, u)$ is a straight line between v and u. The distance between two points $x, y \in \mathcal{X}$ is defined as $\mathrm{d}(x, y) = \sqrt{\sum_{i=1}^{D}(x_i - y_i)^2}$, where x_i is the coordinate of x with respect to the i-th dimension. In this case the trajectory length function is equivalent to the distance function $\mathrm{d}_\pi(x, y) = \mathrm{d}(x, y)$.

Examples of this type of environment include scenarios where:

1. Obstacles randomly appear and disappear vs. time.
2. An inaccurate map of the world is provided and obstacles "disappear" or "appear" (i.e., from the robot's point of view) as their absence or existence is discovered via a ranged sensor.

Videos showing simulations in both of these situations are available at: Otte (2014).

Holonomic robot in $\mathbb{R}^D \times \mathbb{T}$. This is similar to the scenario described in the previous section except for the addition of a time dimension $\mathbb{T}$ to the state space, i.e., $\mathcal{X} \subset \mathbb{R}^D \times \mathbb{T}$. We follow the recommendation of LaValle (2006) about how to handle distance in a state space that is a Cartesian product of subspaces. In particular, distance from v to u, given $v, u \in \mathcal{X}$ is defined $\mathrm{d}(x, y) = \sqrt{c_t(v_t - u_t)^2 + \sum_{i=1}^{D}(v_i - u_i)^2}$, where x_t is the time coordinate of x and x_i is the coordinate of x with respect to the i-th dimension for i such that $1 \leq i \leq D$. The user defined constant c_t determines the importance of time vs. space distance (note the trade-off is non-linear). The trajectory length function is similar except that it accounts for the fact that time moves forward:

$$\mathrm{d}_\pi(x, y) = \begin{cases} \mathrm{d}(x, y) & \text{if } v_t < u_t \\ \infty & \text{otherwise} \end{cases}.$$

[13]In practice we can locate all graph edges in conflict with a particular obstacle in two steps: (1) First finding the set V_{close} of all nodes that are within a distance c of the obstacle center, where $c = c_{robot} + c_{obstacle} + c_{edge}$ and where c_{robot} and $c_{obstacle}$ are radii of the smallest D-Ball that contain the robot and obstacle, respectively, and c_{edge} is the length of the largest edge in E. (2) examine all edges (v, u) such that either $v \in V_{close}$ or $u \in V_{close}$. This is easy to do by examining the neighbor sets of all nodes in V_{close}.

It is often realistic to bound the robot velocity at $\delta x_{\max}$, in which case $\mathrm{d}_\pi(x,y) = \infty$ unless y is in the reachibility cone of x, i.e.,

$$\mathrm{d}_\pi(x,y) = \begin{cases} \mathrm{d}(x,y) & \text{if} \qquad (u_t - v_t)\delta x_{\max} \geq \sqrt{\sum_{i=1}^{D}(v_i - u_i)^2} \\ \infty & \text{otherwise} \end{cases}.$$

If a minimum speed is also defined then there is a second cone within the first representing more points that cannot be reached.

To maintain the graph connectivity guarantees introduced by RRT*, the size of the shrinking the D-Ball needs to be adjusted to account for the fact that only a subset of its original space can be reached.

If an estimate of obstacles motion exists (i.e., based on current trajectory or past behavior) than space-time obstacles are defined by sweeping space obstacles along their estimated path in the time dimension. A simple model may assume, for instance, an obstacle will move along its current heading forever; thus creating volume in space-time. In the latter case, space-time obstacles are adjusted only if/when they deviate from their predicted behavior. Uncertainty about obstacle movement can be modeled by expanding the size of the obstacle vs. time. When a model of obstacle behavior does not exist, then we assume that it remains stationary, i.e., the time-obstacle is a cylinder with its longitudinal access parallel to the time dimension.

For practicality we must impose time bounds on $\mathcal{X}$. We have also found it useful to insert a number of points at the spatial goal location at incremental values of time, and then define that each of these points is distance 0 to an abstract notion of a goal. This allows the robot to arrive at the goal location early without incurring a time-based cost penalty.

Videos showing simulations in both of these situations are available at: Otte (2014).

Dubins model in $\mathbb{R}^2 \times \mathbb{S}^1$. The state space is defined $\mathcal{X} \subset \mathbb{R}^2 \times \mathbb{S}^1$. The robot moves at a constant speed and has a predefined minimum turning radius $r_{\min}$ LaValle (2006). Distance between two points $x, y \in \mathcal{X}$ is defined $\mathrm{d}(x,y) = \sqrt{c_\theta(x_\theta - y_\theta)^2 + \sum_{i=1}^{2}(x_i - y_i)^2}$, where x_θ is heading and x_i is the coordinate of x with respect to the ith dimension of $\mathbb{R}^2$, and assuming the identity $\theta = \theta + 2\pi$ is obeyed. The constant c_θ determines the cost trade-off between a difference in location vs. heading. $\mathrm{d}(x,y)$ is the length of the geodesic between x and y through $\mathbb{R}^2 \times \mathbb{S}^1$. $\mathrm{d}_\pi(x,y)$ is the length of the Dubins trajectory that moves the robot from x to y. In general, $\mathrm{d}_\pi(x,y) \neq \mathrm{d}(x,y)$; however, by defining c_θ appropriately (e.g., $c_\theta = 1$) we can guarantee $\mathrm{d}_\pi(x,y) \geq \mathrm{d}(x,y)$ so that $\mathrm{d}(x,y)$ is an admissible heuristic on $\mathrm{d}_\pi(x,y)$. The nearest-neighbor data structure must also account for the identity $\theta = \theta + 2\pi$. A videos of this simulation appears at Otte (2014).

Extended Dubins model in $\mathbb{R}^2 \times \mathbb{S}^1 \times \mathbb{T}$. The standard Dubins model does not work well when time is added as a dimension in the state space. This is due to the fact that the set of potential neighbors for any given node v has measure 0 if the robot is constrained to move at a constant speed. Therefore, we need to use an extended model in which the robot speed is allowed to vary. In particular, we define speed bounds $\delta x_{\max}$ and $\delta x_{\min}$ and allow instantaneous velocity change. This modifies the connection set to exist as a positive measure subset of the reachibility cone that was discussed earlier (the shrinking ball radius must be adjusted accordingly).

The distance between two points $x, y \in \mathcal{X}$ is defined as $\mathrm{d}(x,y) = \sqrt{c_t(x_t - y_t)^2 + c_\theta(x_\theta - y_\theta)^2 + \sum_{i=1}^{D}(x_i - y_i)^2}$ and represents the length of the straight line between x and y through $\mathbb{R}^2 \times \mathbb{S}^1 \times \mathbb{T}$. As in the normal Dubins case described in the previous section, this straight line through the state space is not followed by the robot, in general; thus, $\mathrm{d}_\pi(x,y) \neq \mathrm{d}(x,y)$.

Let $\mathrm{d}^{\mathrm{dub}}_{\mathbb{R}^2 \times \mathbb{S}^1 \times \mathbb{T}}(x,y)$ be the space-time distance along $\pi(x,y)$, and let $\mathrm{d}^{\mathrm{dub}}_{\mathbb{R}^2 \times \mathbb{S}^1}(x,y)$ be the distance along the projection of $\pi(x,y)$ onto $\mathbb{R}^2 \times \mathbb{S}^1$. We define $\mathrm{d}_\pi(x,y)$ as the distance along the space-time Dubins curve (unless velocity constraints

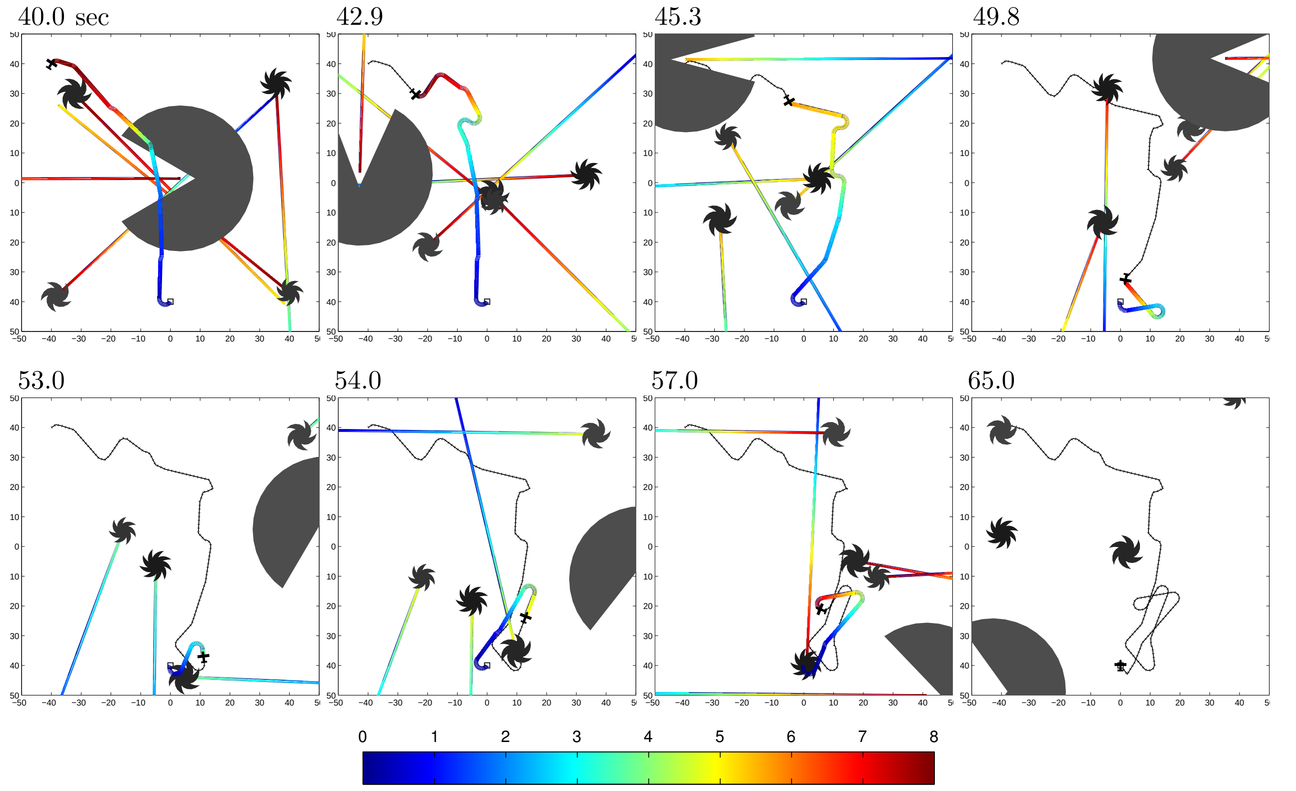


Fig. 7. (Extended) Dubins robot using RRT^X to move from start to goal (white square) while repairing its shortest-path tree vs. moving obstacle changes. Color paths show the planned position of the robot and the estimated positions of the obstacles with respect to the robot's time-to-goal. The path executed by the robot is black with hatch marks. Moving obstacles are dark gray. Time (in seconds) appears above each sub-figure (Note the robot begins moving at 40 seconds). See `http://tinyurl.com/l53gzgd` for video.

prevent the robot from following $\mathrm{d}_\pi(x,y)$).

$$\mathrm{d}_\pi(x,y) = \begin{cases} \mathrm{d}^{\mathrm{dub}}_{\mathbb{R}^2\times\mathbb{S}^1\times\mathbb{T}}(x,y) & \text{if } (u_t - v_t)\delta x_{\min} \le \mathrm{d}^{\mathrm{dub}}_{\mathbb{R}^2\times\mathbb{S}^1}(x,y) \le (u_t - v_t)\delta x_{\max} \\ \infty & \text{otherwise} \end{cases}$$

As in the normal Dubins case, it is convenient to define the distance scaling constants such that $\mathrm{d}(x,y)$ returns a reasonable set of nodes with respect to $\mathrm{d}_\pi(x,y)$.

Figure 7 shows a simulation of this vehicle in a dynamic environment with moving obstacles. A videos of this simulation also appears at (Otte, 2014).

3rd-Order Thruster in $\mathbb{R}^3_x \times \mathbb{R}^3_{dx/dt} \times \mathbb{T}$. An axis aligned thruster vehicle controls velocity and position using six thrusters that point in the positive and negative directions along each axis and produce constant acceleration a_c when fired (affects on vehicle mass are assumed negligible). The state space used for planning consists of three dimensions of position $\mathbb{R}^3_x$, three dimensions of velocity $\mathbb{R}^3_{dx/dt}$, and one dimension of time $\mathbb{T}$ for a total of seven dimensions. Velocities $dx/dt \in \mathbb{R}^3_{dx/dt}$ are the derivatives of positions $x \in \mathbb{R}^3_x$ with respect to time $\mathbb{T}$.

Let $\mathbf{x} = [x, \frac{dx}{dt}, t] \in \mathbb{R}^3_x \times \mathbb{R}^3_{dx/dt} \times \mathbb{T}$ be a point in the vehicle's state-space. The steering function for this system is calculated by solving the system of differential equations for the time parameterized control inputs $u(\mathbf{x}_1, \mathbf{x}_2)$ between two points $\mathbf{x}_1$ and $\mathbf{x}_2$) as a function of starting and ending velocities, positions, and times $u(\mathbf{x}_1, \mathbf{x}_2) = \frac{d^2\mathbf{x}}{dt^2} = f(\mathbf{x}_1, \mathbf{x}_2)$.

The trajectory associated with edge is calculated once, when that edges is inserted into the graph (edges without a solution are not added to the graph). As in previous sections, $\mathrm{d}_\pi(\mathbf{x}_1, \mathbf{x}_2)$ is defined by the state-space length of the trajectory, calculated using numerical integration, and scaling constants are defined such that $\mathrm{d}(\mathbf{x}_1, \mathbf{x}_2)$ returns a reasonable set of nodes with respect $\mathrm{d}_\pi(\mathbf{x}_1, \mathbf{x}_2)$.

Videos of this simulation, as well as a similar system in $\mathbb{R}^2_x \times \mathbb{R}^2_{dx/dt} \times \mathbb{T}$ appears at (Otte, 2014).

Cartesian Products of $\mathbb{R}^a$, $\mathbb{S}^b$, $\mathrm{SO}(c)$, *and* $\mathbb{T}$ Using RRT$^{\mathrm{X}}$ in many other higher dimensional spaces will likely follow the pattern of considerations that has been established above. In general, one needs to:

- Define $\mathrm{d}(x, y)$ such that the it is easy to do proximity queries on $\mathcal{X}$.
- Define $\mathrm{d}_\pi(x, y)$ as the distance that the robot actually follows along $\pi(x, y)$.
- It is helpful if $\mathrm{d}(x, y)$ is defined such that it provides a useful potential neighbor set of nodes with respect to $\mathrm{d}_\pi(x, y)$ (the shrinking ball radius must be modified accordingly).
- Additional constraints on $\mathrm{d}(x, y)$ are imposed by the fact that $\mathbb{T}$ must move forward as well as any minimum or maximum constraints on velocity, etc.
- RRT$^{\mathrm{X}}$ (and all sampling-based motion planning algorithms) the family of possible trajectories must be such that from almost all $x \in \mathcal{X}$ the set of reachable points has non-zero measure $\mathscr{L}\left(\{y : \exists \mathrm{d}_\pi(x, y) < \infty\}\right) > 0$. This especially an issue when $\mathbb{T}$ is concerned.

Previous work (LaValle, 2006; Bialkowski, 2014) indicates that all sampling-based motion planning algorithms, including RRT$^{\mathrm{X}}$, can be used for Cartesian products of $\mathbb{R}^a$, $\mathbb{S}^b$, and $\mathrm{SO}(c)$. Following the above considerations allows RRT$^{\mathrm{X}}$ to be used with Cartesian products of $\mathbb{R}^a$, $\mathbb{S}^b$, $\mathrm{SO}(c)$, and $\mathbb{T}$.

Handling Dynamics (e.g., $d\mathbb{R}^a$, $d\mathbb{S}^b$, $d\mathrm{SO}(c)$) involves steering functions that require more complicated two-boundary value problems to be solved. In the most general case numerical solutions can be used (e.g., shooting methods), which may necessitate the use of more powerful computing resources. That said, some problems may be impossible to solve with RRT$^{\mathrm{X}}$ given the current state of computing technology (these situations are currently addressed using different classes of algorithms that either allow offline computation and/or trade asymptotic optimality and probabilistic completeness for resolution optimality/completeness, see Section 2).

6.2. Experiments

Experiment 1: Performance in randomly generated environments with moving obstacles Using the extended Dubins vehicle, we compare RRT$^{\mathrm{X}}$ to DRRT and RRT (i.e., placing RRT inside a control loop) in randomly generated environments. 10 environments are randomly generated for each possible combination of obstacle density $\{.01, .05, .10, .15, .20, .25\}$ and obstacle speed $\{0, 5, 10, 15, 20, 25, 30, 35, 40\}$. Each environment is used in two trials, for a total of 20 runs per each combination of density and speed.

The robot has a radius of 2, Dubins turning radius is 2, and is allowed forward velocity on $[10, 25]$. Environments measure 100 by 100. Obstacles are circular with radius 5. Obstacle movement iterates along a random heading (chosen uniformly on $[0, 2\pi]$) for a random distance (chosen uniformly on $[0, c]$, where c is the diameter of the workspace). Obstacles that touch the workspace boundary are reflected and continue moving along the reflection heading. Obstacle density is defined by the total obstacle volume divided by total volume of the workspace (rounded up to an integer value). If a robot collides with an obstacle then the trial ends in failure and path length is defined to be 800 (which is larger than any successful run). Obstacles do not interact with each other. Results appear in Figure 8.

Experiment 2: 3rd-Order Thruster Vehicle in Dynamic Environment Using the extended 3rd-Order vehicle, we compare the performance of RRT$^{\mathrm{X}}$ using $\epsilon = \{10^{-2}, 10^{-1}, 10^0, 10^1\}$ in a dynamic forest environment where trees randomly appear

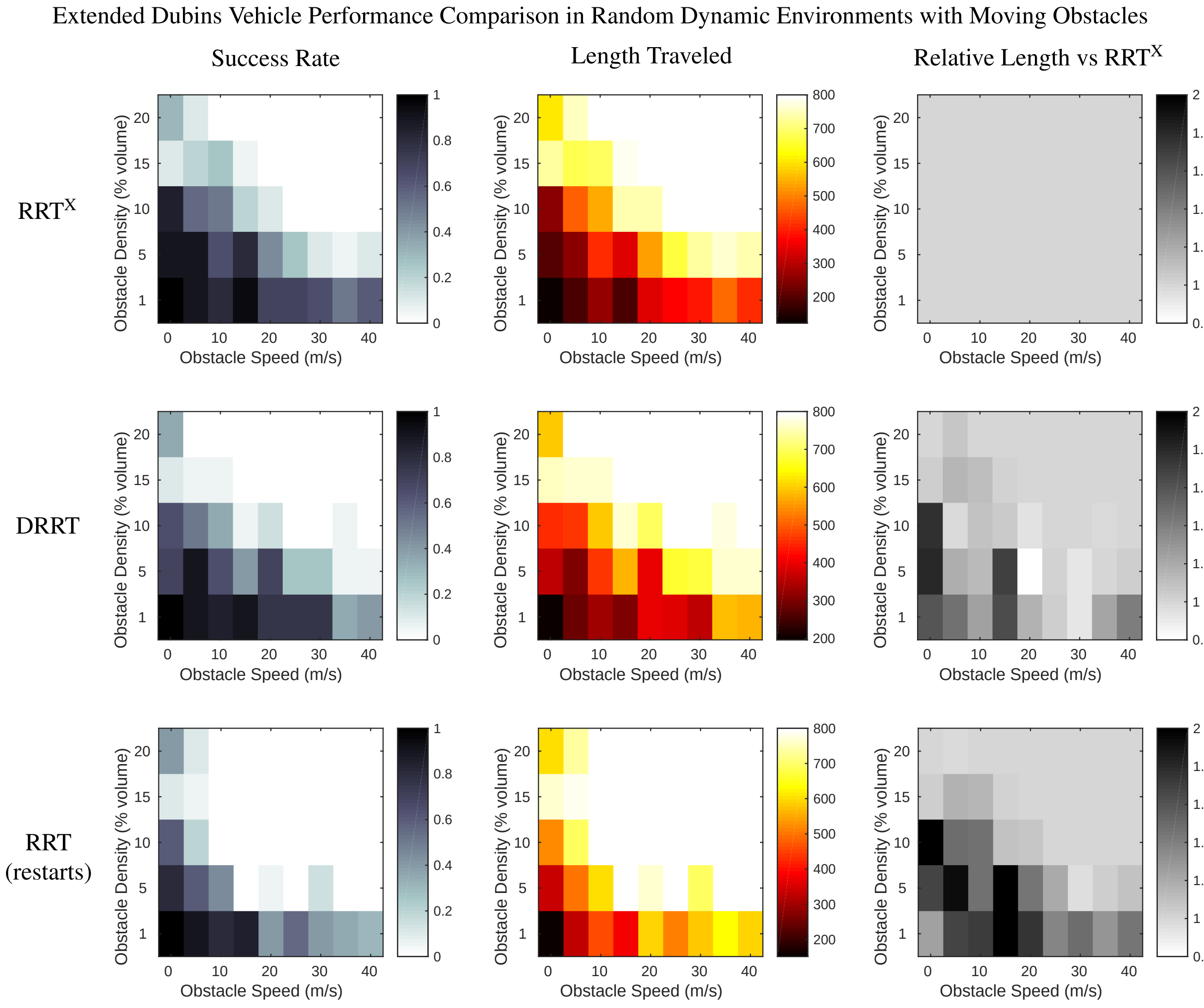


Fig. 8. Success rates and average path lengths vs. obstacle speed and density. 20 trials per datapoint for extended dubins vehicle performance comparison in random dynamic environments with moving obstacles

and disappear. The environment is randomly generated and contains 20 trees of radii 5, each tree is given 1/3 probability *a priori* of appearing, disappearing or doing nothing when the robot comes within sensor range (20). The robot has a radius of 2, and an acceleration of 2 along each axis, and a maximum velocity magnitude of 20 along each axis. The environment measures $100 \times 100 \times 20$ (length, width, height), and the robot must travel from $[20, 20, 0]$ to $[80, 80, 0]$, starting and ending at rest. Results appear in Figure 9.

Experiment 3: Information propagation starting at time t. We compare the performance of RRT^X, RRT$^\#$, and RRT* when a shortcut through the environment is found at a particular time t. This illustrates the relative effects of graph ϵ-consistency, consistency, vs. lazy cost propagation on cost propagation speed[14].

We assume the robot is a Holonomic first-order system and the environment contains two long obstacles with a small gap between them (see Figure 10). In order to guarantee that all three algorithms discover the shortcut at the same time, we define that the shortcut contains an obstacle for the first $t = 10$ seconds of planning (but not after), and the first sample

[14]i.e., how quickly (and to what extent) knowledge of an attraction basin shift spreads through the search tree and thus informs the robot's motion plan

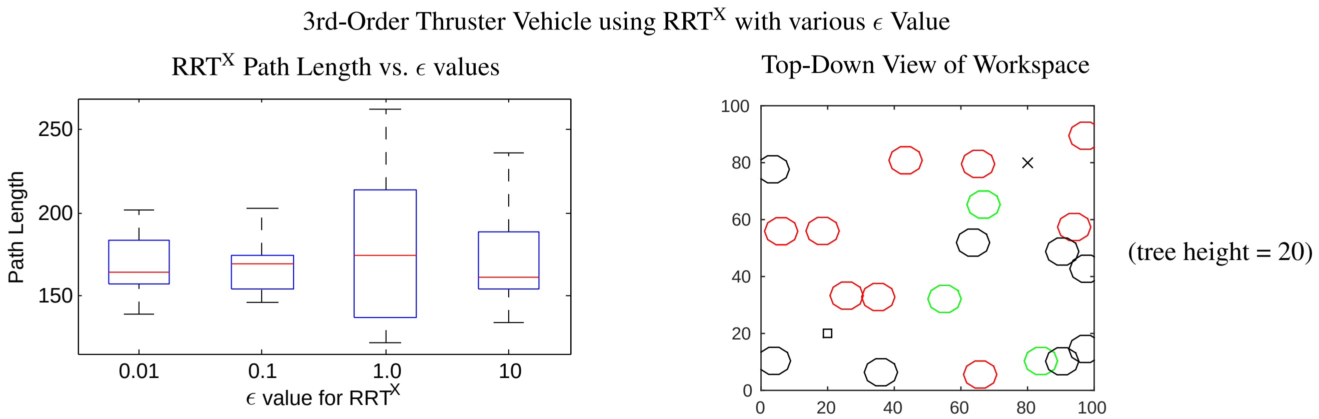


Fig. 9. Left: Results of 3rd-Order thruster vehicle using RRT^X with various ϵ values in a randomly generated dynamic forest environment with appearing and disappearing trees. Box-plots are for 10 runs of data per ϵ value. Right: Workspace of the forest environment, robot must start at the square and reach the X. Note that the state space used for planning is 7-dimensional; additionally containing height, velocity, and time; the robot can only directly control thrust.

after t is explicitly drawn from the center of the gap (all other samples are drawn uniformly at random). The experiment is halted after 60 seconds. Forcing the shortcut to be found at t eliminates one source of variability without compromising the comparison between the random sampling-based algorithms.

Figure 11 shows mean solution lengths and graph sizes over 50 trials. For RRT^X we set ϵ to be 0.5. Results are depicted in. A video of this experiment is accessible online at (Otte, 2014).

Experiment 4: Information propagation starting at iteration i. In the previous experiment (i.e., Experiment 1) all three algorithms discovered the shortcut at the same *time*—a scenario that is representative of planning in a dynamic environment when an obstacle disappears. This experiment (i.e., Experiment 2) is similar, except that we remove the obstacle and sample the shortcut at iteration $i = 6000$. This is more representative of planning in a static environment (where the expected time to sample from within the bottleneck is a function of the number of samples that are drawn).

Although RRT* and RRT^X both have iteration time $O(\log n)$, the constant factor associated with RRT* is less than that of RRT^X. The experiment is halted after 60 seconds of planning. Results appear in Figure 12, and represent the mean over 50 trials.

Experiment 5: Information transfer time. We evaluate the time it takes to propagate new data about the environment through the graph with RRT^X, RRT$^\#$, and RRT*. We use the same environment as in Experiments 3 and 4 (i.e., Figure 10), and record the duration between the time when the shortcut is removed/sampled and the time the best-path adjusts to utilize the shortcut. Results appear in Figure 13.

7. Discussion

7.1. RRT^X in Dynamic Environments

RRT^X is the first asymptotically optimal sampling-based motion planning designed for kinodynamic replanning in environments with unpredictably changing obstacles. Analysis shows that the amortized time required to add a new node is $\Theta(\log n)$, assuming a graph with n nodes, and that handling obstacle changes requires a rewiring cascade that takes $O(n \log n)$ time.

Simulations in Section 6.1 show that RRT^X produces effective real-time navigation in many different dynamic environments, and for state spaces $\mathcal{X}$ for which time $\mathbb{T}$ is a dimension, and/or while respecting robot kinematics in state spaces that

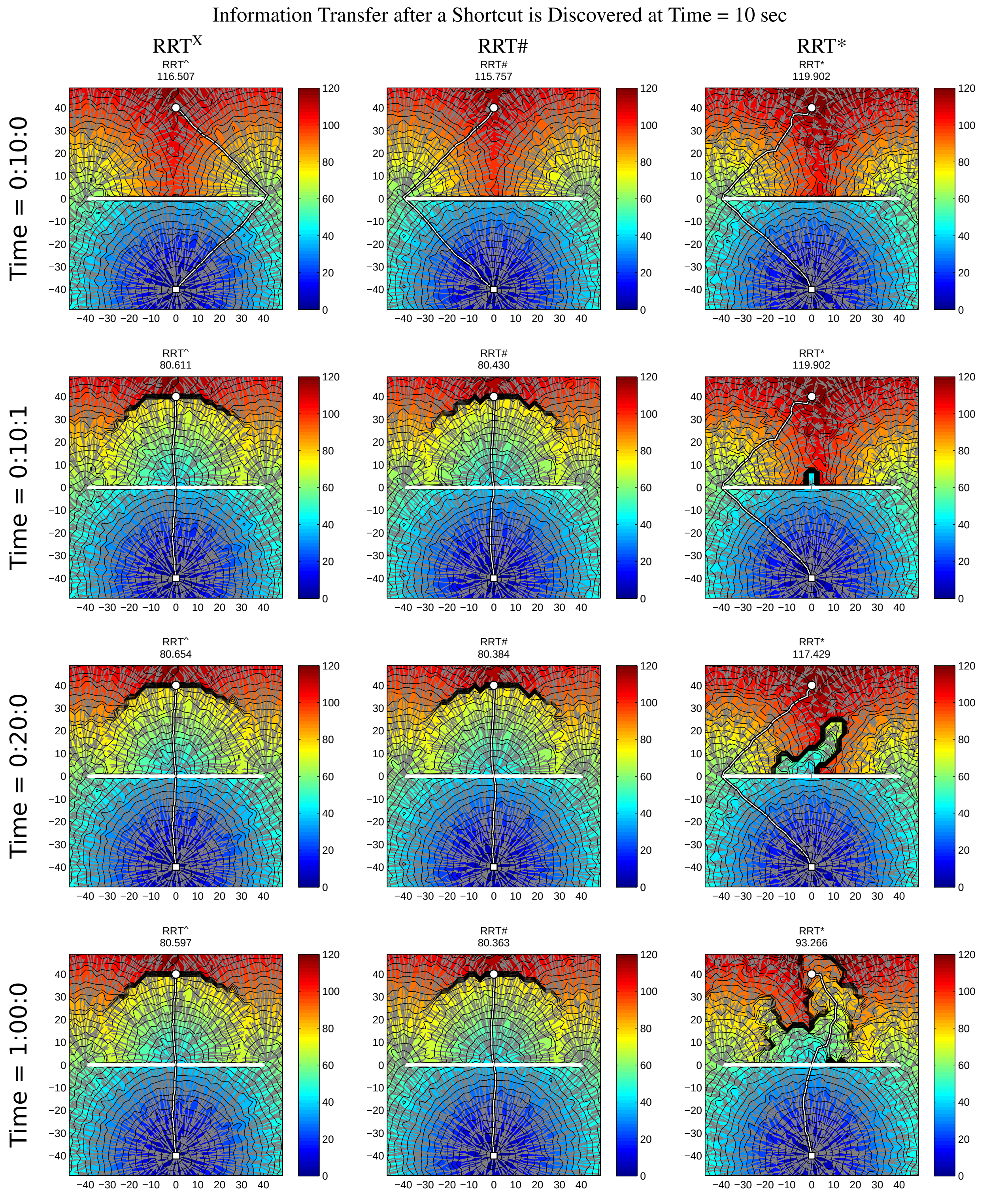


Fig. 10. RRT^X, RRT#, and RRT* (left to right, respectively) at different times (top to bottom), in a scenario where a shortcut is discovered at time 10 seconds. Color and black topo-lines show cost-to-goal. White circle and square are the start and goal states, respectively. The wall obstacle is also white. Paths are white with black outline and the search tree is gray. Path length appears at the top of each sub-figure. RRT^X and RRT# maintain graph consistency, allowing them to react to the shortcut more quickly than RRT*. A video of this experiment is available online at `http://tinyurl.com/l53gzgd`

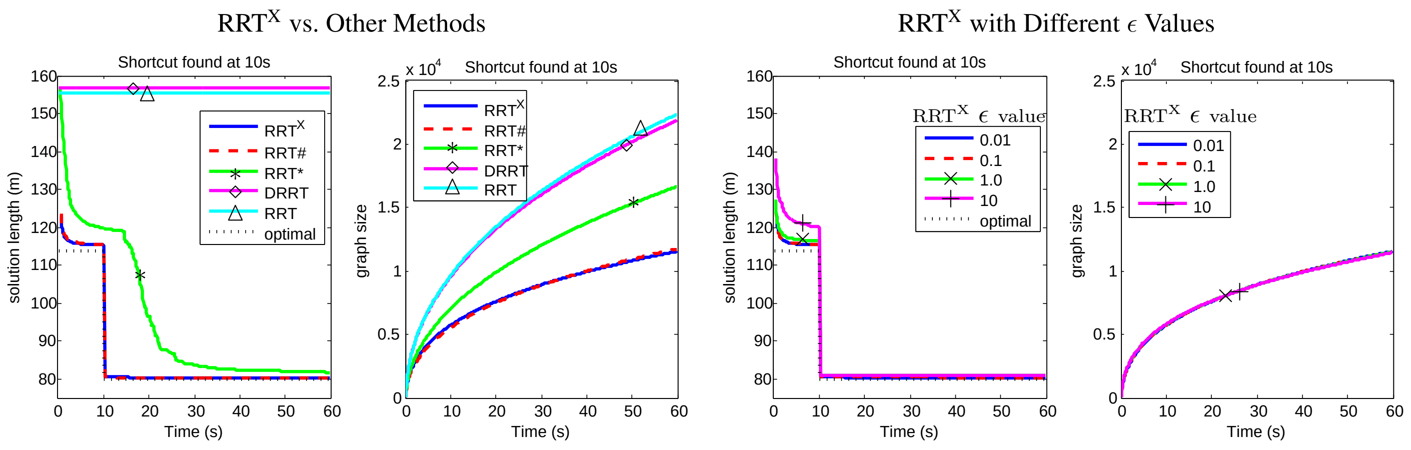


Fig. 11. Comparison of path length (Left) and graph size (right) vs. time, when a shortcut through a wall-like obstacle is discovered at 10 seconds. Graph consistency and graph ϵ-consistency allows RRT$^\#$ and RRT^X to react more quickly than RRT*, which does not use consistency, even despite the latter having more nodes. The length of the optimal path is also depicted (dashed-line, Left).

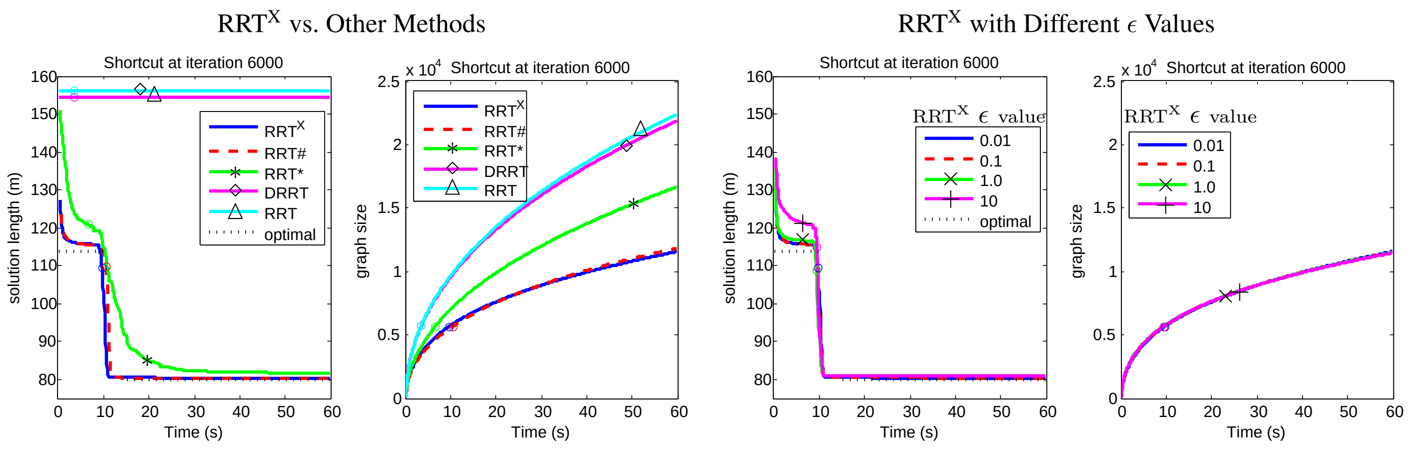


Fig. 12. Comparison of path length (Left) and graph size (right) vs. time, when a shortcut through a wall-like obstacle is discovered at iteration 6000. While RRT*'s faster iteration time enable it to find the shortcut more quickly that RRT^X and RRT$^\#$, graph consistency and graph ϵ-consistency allows RRT$^\#$ and RRT^X to react more quickly than RRT*. The length of the optimal path is also depicted, dashed-line (Left); as are the mean times corresponding to iteration 6000 (circles). Note, at iteration 6000 there are less than 6000 nodes in the graph.

are non-euclidean. The benefits of using an asymptotically optimal algorithm instead of a feasible algorithm are clearly demonstrated in Experiments 1, 3, and 4, where RRT^X achieves much shorter paths than DRRT and RRT without sacrificing success rate.

When time is included in the state space, then RRT^X can improve path quality by estimating the trajectories of moving obstacles. The replanning problem assumes myopia; thus, it is impossible for RRT^X to avoid collisions with obstacles that appear suddenly and/or move significantly faster than the robot. This limitation is faced by all replanning algorithms, and by any agent that must make decisions given incomplete information about the world. However, the use of ϵ, as well as neighbor-pruning, enables RRT^X to quickly react to changing obstacles. For example, quick iteration times allows RRT^X to start repairing its plan soon after a change is detected, and quick information transfer time allows that repair to happen quickly. These are the primary advantages of RRT^X over other possible replanning formulations of asymptotically optimal sampling-based methods. Both theoretical analysis and experimental results support this claim.

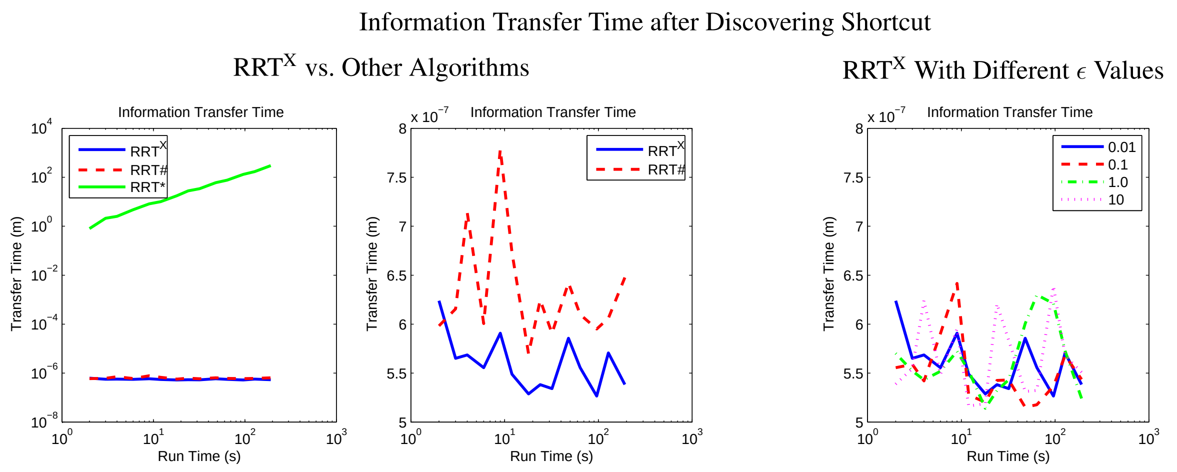


Fig. 13. The time that it takes to propagate new information about the environment to the robot's position with RRT^X, RRT$^\#$, and RRT* (colors) when a bottleneck is detected at a variety of planning times (x axis). Each datapoint represents the mean results over 30 trials. Left depicts all three algorithms on a logarithmic scale, Right shows a close up of RRT^X and RRT$^\#$. The environment that is used appears in Figure 10.

7.2. RRT^X in Static Environments

Although RRT^X is designed primarily for use in dynamic environments, both analysis and experiments shows that it is also competitive with all state-of-the-art asymptotically optimal single-query motion planning algorithms. RRT^X has the same order expected runtime as RRT and RRT*, and is theoretically quicker than RRT$^\#$. RRT^X inherits probabilistic completeness and asymptotic optimality from RRT*. By maintaining a ϵ-consistent graph, RRT^X has similar behavior to RRT$^\#$ for cost changes larger than ϵ; which translates into faster convergence than RRT* in practice.

The two major differences between RRT^X and RRT$^\#$ in static environments are: (1) RRT^X's use of ϵ to maintain ϵ-consistency vs. the full consistency maintained by RRT$^\#$. (2) active maintenance of the neighborhood size that causes each node to have a roughly constant sized neighborhood in RRT^X vs. the increasing neighborhood size of RRT$^\#$ (i.e., because old nodes retain their original neighbors, plus any new nodes for which they are within the shrinking hyperball).

RRT^X only propagates ϵ-cost decreases. Once all bottle-necks have been discovered and once $r < \epsilon$, where r is the radius of the shrinking D-Ball, then RRT^X behaves similarly to RRT* (i.e., convergence rate is determined by lazy propagation). Because reaction time is not a consideration in static environment, it may be advisable to use RRT^X with $\epsilon = 0$ in static environments. While this will increase RRT^X's expected runtime, it will still remain less than RRT$^\#$'s due to the former's maintenance of constant neighborhood size. That said, if planning time does not have a hard limit, then another alternative (in static environments only) is to use RRT*-like lazy propagation for nearly the entire planning time (ensuring a large number of nodes in the graph), and then running a single corrective cascade starting at v_{goal} with $\epsilon = 0$ to make the graph fully consistent.

7.3. Information Transfer Time

Both analysis and Experiments 3-5 show that for ϵ-cost decreases RRT^X has $O\left(n \log n\right)$ information transfer time. This is better than that of RRT$^\#$ $\omega\left(n \log^2 n\right)$ and also better than RRT* $\Omega\left(n(\frac{n}{\log n})^{1/D}\right)$. For cost increase information RRT^X also has $O\left(n \log n\right)$ transfer time, while RRT* and RRT$^\#$ are not designed to handle cost increases.

The quick information transfer time of RRT^X is one thing that enables it to perform well in dynamic environments. The other factor is iteration time, which determines how long the algorithm expects to wait after a change happens and before it is detected. When obstacles change, the expected reaction time of the algorithm is essentially the information transfer time plus one-half of the iteration time. However, for RRT$^{\mathrm{X}}$, information transfer time is likely to be much larger than iteration time (note that the two are identical for RRT$^{\#}$), and therefore represents most of the reaction time of RRT$^{\mathrm{X}}$.

7.4. Graph Consistency and ϵ-Consistency

We verify experimentally and analytically the result by RRT$^{\#}$ that maintaining graph consistency increases the rate of convergence toward an optimal solution. That said, we also find that for real-time motion replanning problems, maintaining an ϵ-consistent graph allows quicker reaction time to "important" changes.

7.5. Choosing ϵ

The consistency parameter ϵ allows RRT$^{\mathrm{X}}$ to focus effort on propagating information changes that are critical for safe real-time navigation, while ignoring those that do not significantly affect the motion plan. ϵ must be greater than 0 to guarantee $\Theta(\log n)$ expected iteration time. In general, using smaller ϵ causes graphs to be more consistent, but also decreases the number of nodes that can be added to the graph within a particular amount of time. Experiments 2-5 demonstrate that RRT$^{\mathrm{X}}$ is relatively insensitive to ϵ. Different ϵ values have little effect on graph size or the transfer of information $> \epsilon$. The most noticeable effect is that smaller ϵ values improve short-term convergence rates in static environments (i.e., when maintenance rewiring causes cost changes $< \epsilon$).

In dynamic environments, ϵ should be small enough such that a rewiring cascade is triggered whenever obstacle changes require a course correction by the robot. For example, in a Euclidean space (and assuming the robot's position is defined as its center point) we suggest using an ϵ no larger than 1/2 the robot width. Alternatively, if a safety distance d_{safe} is defined (i.e., the robot is not allowed to come within d_{safe} of an obstacle), then $\epsilon < d_{\mathrm{safe}}/2$ is advisable.

7.6. Neighborhood Size

Culling neighbors to keep neighborhood size $O(\log n)$ in expectation is another necessity to maintain an $\Theta(\log n)$ expected per iteration runtime. While we have investigated retaining neighbors within r in the current paper, other culling strategies may be able to increase the utility of sampling-based algorithms in special cases. This topic is beyond the scope of the current paper.

8. Summary and Conclusions

We present RRT$^{\mathrm{X}}$, the first asymptotically optimal single-query sampling-based replanning algorithm. In other words, RRT$^{\mathrm{X}}$ is designed to improve its solution during movement, react quickly when the environment changes, and to be used when *a priori* offline computation is unavailable.

RRT$^{\mathrm{X}}$ uses the same search tree for the duration of navigation, continually refining it and repairing it as changes to the state space are detected. The resulting motion plans are both valid, with respect to the dynamics of the robot, and asymptotically optimal with respect to the current obstacle configuration, i.e., in static environments. Analysis, simulations, and experiments show that RRT$^{\mathrm{X}}$ works well in both static and dynamic environments.

RRT$^{\mathrm{X}}$ achieves a quick reaction time in dynamic environments by maintaining an ϵ-consistent graph, constant neighborhood size at each node, and then using rewiring cascades to transfer information through the graph whenever obstacles change. For changes larger than ϵ, RRT$^{\mathrm{X}}$ has $O(n \log n)$ information transfer time, which is quicker than other sampling-based motion planning algorithms, i.e., RRT* $\Omega\left(n(\frac{n}{\log n})^{1/D}\right)$ and RRT$^{\#}$ $\omega\left(n \log^2 n\right)$.

Algorithm 7: `shrinkingBallRadius`$(|V|)$

```
1 return $\min\left\{\left(\frac{\gamma}{\zeta_D}\frac{\log|V|}{|V|}\right)^{1/D}, \delta\right\}$
```

Algorithm 8: `updateObstacles`$()$

```
1 if $\exists O : O \in \mathcal{O} \wedge O$ has vanished then
2     forall the $O : O \in \mathcal{O} \wedge O$ has vanished do
3         removeObstacle$(O)$ ;
4     reduceInconsistency$()$ ;
5 if $\exists O : O \notin \mathcal{O} \wedge O$ has appeared then
6     forall the $O : O \notin \mathcal{O} \wedge O$ has appeared do
7         addNewObstacle$(O)$ ;
8     propogateDescendants$()$ ;
9     verrifyQueue$(v_{\mathrm{bot}})$ ;
10    reduceInconsistency$()$ ;
```

Algorithm 9: `propogateDescendants`$()$

```
1 forall the $v \in V_{\mathcal{T}}^{\mathrm{c}}$ do
2     $V_{\mathcal{T}}^{\mathrm{c}} \leftarrow V_{\mathcal{T}}^{\mathrm{c}} \cup C_{\mathcal{T}}^{-}(v)$ ;
3 forall the $v \in V_{\mathcal{T}}^{\mathrm{c}}$ do
4     forall the $u \in \left(N^{+}(v) \cup \{p_{\mathcal{T}}^{+}(v)\}\right) \setminus V_{\mathcal{T}}^{\mathrm{c}}$ do
5         g$(u) \leftarrow \infty$ ;
6         verrifyQueue$(u)$ ;
7 forall the $v \in V_{\mathcal{T}}^{\mathrm{c}}$ do
8     $V_{\mathcal{T}}^{\mathrm{c}} \leftarrow V_{\mathcal{T}}^{\mathrm{c}} \setminus \{v\}$ ;
9     g$(v) \leftarrow \infty$ ;
10    lmc$(v) \leftarrow \infty$ ;
11    if $p_{\mathcal{T}}^{+}(v) \neq \emptyset$ then
12        $C_{\mathcal{T}}^{-}(p_{\mathcal{T}}^{+}(v)) \leftarrow C_{\mathcal{T}}^{-}(p_{\mathcal{T}}^{+}(v)) \setminus \{v\}$ ;
13        $p_{\mathcal{T}}^{+}(v) \leftarrow \emptyset$
```

9. Acknowledgments

This work was supported by U.S. Department of Defense-Air Force Office of Scientific Research #FA-8650-07-2-3744, U.S. Department of Defense-Office of Naval Research MURI grant #N00014-09-1-1051, and by the Control Science Center of Excellence at the Air Force Research Laboratory (AFRL).

A. Minor Subroutines

Minor subroutines appear in Algorithms 7-14. The functionality of `shrinkingBallRadius`$(|V|)$ was introduced in the RRT* algorithm, and returns the radius of $\mathcal{B}$, the shrinking D-Ball, such that $\mathcal{B}$ is expected to contain a number of nodes proportional to γ (at least, once $\mathcal{B}$ shrinks such that its radius is less than the RRT-like steering saturation distance δ), both γ and δ are user defined parameters, Karaman and Frazzoli (2011) recommend using $\gamma > 2^D(1 + 1/D)\mathscr{L}(\mathcal{X}_{\mathrm{free}})$ to ensure a connected graph, where the volume of the free space, $\mathscr{L}(\mathcal{X}_{\mathrm{free}})$, is usually estimated in practice. ζ_D is the volume of a unit D-Ball.

```
Algorithm 10: verrifyOrphan($v$)
1 if $v \in Q$ then remove($Q, v$) ;
2 $V^{c}_{\mathcal{T}} \leftarrow V^{c}_{\mathcal{T}} \cup \{v\}$ ;
```

```
Algorithm 11: removeObstacle($O$)
1 $E_O \leftarrow \{(v,u) \in E : \pi(v,u) \cap O \neq \emptyset\}$ ;
2 $\mathcal{O} \leftarrow \mathcal{O} \setminus \{O\}$ ;
3 $E_O \leftarrow E_O \setminus \{(v,u) \in E : \pi(v,u) \cap O' \neq \emptyset$ for some $O' \in \mathcal{O}\}$ ;
4 $V_O \leftarrow \{v : (v,u) \in E_O\}$ ;
5 forall the $v \in V_O$ do
6     forall the $u : (v,u) \in E_O$ do
7         $\mathrm{d}_\pi(v,u) \leftarrow$ recalculate $\mathrm{d}_\pi(v,u)$ ;
8     updateLMC($v$) ;
9     if lmc($v$) $\neq$ g($v$) then verrifyQueue($v$);
```

```
Algorithm 12: addNewObstacle($O$)
1 $\mathcal{O} \leftarrow \mathcal{O} \cup \{O\}$ ;
2 $E_O \leftarrow \{(v,u) \in E : \pi(v,u) \cap O \neq \emptyset\}$ ;
3 forall the $(v,u) \in E_O$ do
4     $\mathrm{d}_\pi(v,u) \leftarrow \infty$ ;
5     if $p^{+}_{\mathcal{T}}(v) = u$ then verrifyOrphan($v$) ;
6     if $v_{\mathrm{bot}} \in \pi(v,u)$ then $\pi_{\mathrm{bot}} = \emptyset$ ;
```

```
Algorithm 13: verrifyQueue($v$)
1 if $v \in Q$ then update($Q, v$) ;
2 else add($Q, v$) ;
```

```
Algorithm 14: updateLMC($v$)
1 cullNeighbors($v, r$) ;
2 forall the $u \in N^{+}(v) \setminus V^{c}_{\mathcal{T}} : p^{+}_{\mathcal{T}}(u) \neq v$ do
3     if lmc($v$) $> \mathrm{d}_\pi(v,u) +$ lmc($u$) then
4         $p' \leftarrow u$ ;
5 makeParentOf($p', v$)
```

B. Additional Background on RRT* and RRT#

This section contains additional background on RRT* (Karaman and Frazzoli, 2011) and RRT# (Arslan and Tsiotras, 2013). The overarching difference between these algorithm and RRT$^{\mathrm{X}}$ is that RRT$^{\mathrm{X}}$ is a shortest-path replanning algorithm while RRT* and RRT# are shortest-path planning algorithms.

Pseudocode for RRT* and RRT# appear side-by-side in Algorithms 15 and 16, respectively, while subroutines differing significantly from those used by RRT$^{\mathrm{X}}$ appear in Algorithms 16-19. We continue to use notation consistent with a directed graph implementation to facilitate comparison with RRT$^{\mathrm{X}}$ as presented in the current paper.

The stopping criteria for RRT* and RRT# are user defined; common choices include: (1) a time-out, (2) a predefined number of nodes sampled, or (3) a predefined number of nodes added to the search-tree/graph. Both RRT* and RRT#

Algorithm 15: $\texttt{RRT*}(\mathcal{X}, S)$

```
1  $V \leftarrow \{v_{\text{goal}}\}$ ;
2  while stopping criteria not met do
3      $r \leftarrow \texttt{shrinkingBallRadius}(|V|)$ ;
4      $v \leftarrow \texttt{randomNode}(S)$ ;
5      $v_{\text{nearest}} \leftarrow \texttt{nearest}(v)$ ;
6      if $\text{d}(v, v_{\text{nearest}}) > \delta$ then
7          $v \leftarrow \texttt{saturate}(v, v_{\text{nearest}})$ ;
8      if $v \notin \mathcal{X}_{\text{obs}}$ then
9          $V_{near} \leftarrow \texttt{extendStar}(v, r)$ ;
10     if $v \in V$ then
11         $\texttt{rewireNeighborsStar}(v, V_{near})$ ;
```

Algorithm 16: $\texttt{RRT\#}(\mathcal{X}, S)$

```
1  $V \leftarrow \{v_{\text{goal}}\}$ ;
2  while stopping criteria not met do
3      $r \leftarrow \texttt{shrinkingBallRadius}(|V|)$ ;
4      $v \leftarrow \texttt{randomNode}(S)$ ;
5      $v_{\text{nearest}} \leftarrow \texttt{nearest}(v)$ ;
6      if $\text{d}(v, v_{\text{nearest}}) > \delta$ then
7          $v \leftarrow \texttt{saturate}(v, v_{\text{nearest}})$ ;
8      if $v \notin \mathcal{X}_{\text{obs}}$ then
9          $\texttt{extend}(v, r)$ ;
10     if $v \in V$ then
11         $\texttt{rewireNeighborsSharp}(v)$ ;
12         $\texttt{reduceInconsistency}()$ ;
```

Algorithm 17: $\texttt{rewireNeighborsStar}(v, V_{near})$

```
1  forall the $u \in V_{near} \setminus \{p^{+}_{T}(v)\}$ do
2      if $\texttt{g}(u) > \text{d}_{\pi}(u, v) + \texttt{g}(v)$ then
3          $\texttt{g}(u) \leftarrow \text{d}_{\pi}(u, v) + \texttt{g}(v)$;
4          $\texttt{makeParentOf}(v, u)$ ;
```

Algorithm 18: $\texttt{rewireNeighborsSharp}(v)$

```
1  if $\texttt{g}(v) > \texttt{lmc}(v)$ then
2      forall the $u \in N^{-}(v) \setminus \{p^{+}_{T}(v)\}$ do
3          if $\texttt{lmc}(u) > \text{d}_{\pi}(u, v) + \texttt{lmc}(v)$ then
4              $\texttt{lmc}(u) \leftarrow \text{d}_{\pi}(u, v) + \texttt{lmc}(v)$;
5              $\texttt{makeParentOf}(v, u)$ ;
6              if $\texttt{g}(u) > \texttt{lmc}(u)$ then
7                  $\texttt{verrifyQueue}(u)$ ;
```

iteratively perform the following steps: update the shrinking ball radius (line 3), sample a new node v (line 4), find its nearest neighbor (line 5), perform an RRT-like steering/saturation to resample v if the original v is further than δ from its nearest neighbor (lines 6-7), collision check v (line 8), extend the search-graph to include v (line 9, which includes trajectory collision checking), and then rewire neighbors of v to use v as their parent if doing so is advantageous (line 11). RRT# additionally propagates any potential improvement throughout the entire search-graph (line 12) in case the advantages of traveling through v can benefit other nodes further leafward.

A significant difference between RRT* and RRT# is that the former maintains only a shortest-path-tree, while the latter maintains both a search-graph and a shortest-path tree (note that RRT$^{\text{X}}$ also maintains both graph and tree). This has ramifications for the implementation details of graph extension and neighbor rewiring. Rewiring subroutines appear in Algorithms 17 and 18, respectively. The graph extension subroutine used by RRT# appears in Algorithm 19. RRT# uses the same $\texttt{extend}(v, r)$ and $\texttt{reduceInconsistency}()$ subroutines as RRT$^{\text{X}}$ (although the rewiring subroutine within $\texttt{extend}(v, r)$ is replaced by the RRT# version that appears in Algorithm 18). The other subroutines involved in RRT* and RRT# are identical to those of the same name used for RRT$^{\text{X}}$, except that RRT* does not use $\texttt{lmc}(v)$, i.e., cost-to-goal via neighbor values, and instead works directly with $\texttt{g}(v)$, i.e., cost-to-goal values.

References

Arslan, O. and Tsiotras, P. (2013). Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2421–2428. IEEE.

Bekris, K. E. and Kavraki, L. E. (2007). Greedy but safe replanning under kinodynamic constraints. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 704–710.

Algorithm 19: `extendStar`(v, r)

```
1 $V_{near} \leftarrow$ near$(v, r)$ ;
2 findParent$(v, V_{near})$ ;
3 if $p_{\mathcal{T}}^{+}(v) = \emptyset$ then
4     return $\emptyset$ ;
5 $V \leftarrow V \cup \{v\}$ ;
6 return $V_{near}$ ;
```

Bertola, A. and Gonzalez, L. F. (2013). Adaptive dynamic path re-planning rrt algorithms with game theory for UAVs. In *15th Australian International Aerospace Congress (AIAC15)*.

Bialkowski, J. (2014). *Optimizations for Sampling-Based Motion Planning Algorithms*. PhD thesis, Massachusetts Institute of Technology.

Bohlin, R. and Kavraki, L. E. (2000). Path planning using Lazy PRM. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 521–528 vol.1.

Bruce, J. and Veloso, M. (2002). Real-time randomized path planning for robot navigation. In *IEEE Conference of Automation Science and Engineering*. IEEE.

Ferguson, D., Kalra, N., and Stentz, A. (2006). Replanning with RRTs. In *Robotics and Automation (ICRA), IEEE International Conference on*. IEEE.

Fraichard, T. and Asama, H. (2003). Inevitable collision states. a step towards safer robots? In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 388–393 vol.1.

Frazzoli, E., Dahleh, M. A., and Feron, E. (2005). Maneuver-based motion planning for nonlinear systems with symmetries. *Robotics, IEEE Transactions on*, 21(6):1077–1091.

Gayle, R., Klinger, K. R., and Xavier, P. G. (2007). Lazy reconfigurattion forest (LRF) — an approach for motion planning with multiple tasks in dynamic environments. In *Robotics and Automation (ICRA), IEEE International Conference on*. IEEE.

Hauser, K. (2012). On responsiveness, safety, and completeness in real-time motion planning. *Autonomous Robots*, 32(1):35–48.

Hsu, D., Kindel, R., Latombe, J.-C., and Rock, S. (2002). Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255.

Kaelbling, L. P. and Lozano-Perez, T. (2011). Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1470–1477.

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *Int. Journal of Robotics Research*, 30(7):846–894.

Kavraki, L., Svestka, P., Latombe, J., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*.

Knepper, R. A. and Mason, M. T. (2012). Real-time informed path sampling for motion planning search. *The International Journal of Robotics Research*, 31(11):1231–1250.

Koenig, S., Likhachev, M., and Furcy, D. (2002). D* lite. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 476–483.

Koenig, S., Likhachev, M., and Furcy, D. (2004). Lifelong planning A*. *Artificial Intelligence Journal*, 155(1-2):93–146.

Kushleyev, A. and Likhachev, M. (2009). Time-bounded lattice for efficient planning in dynamic environments. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 1662–1668.

Kuwata, Y., Karaman, S., Teo, J., Frazzoli, E., How, J., and Fiore, G. (2009). Real-time motion planning with applications to autonomous urban driving. *Control Systems Technology, IEEE Transactions on*, 17(5):1105–1118.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

LaValle, S. M. and Kuffner, J. J. (2001). Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400.

LaValle, S. M. and Lindemann, S. (2009). Simple and efficient algorithms for computing smooth, collision-free feedback laws over given cell decompositions. *The International Journal of Robotics Research*, 28(5):600–621.

Leven, P. and Hutchinson, S. (2001). Algorithmic and computational robotics: New directions. In Donald, B., Lynch, K., and Rus, D., editors, *Algorithmic and Computational Robotics: New Directions 2000 WAFR*. AK Peters/CRC Press.

Leven, P. and Hutchinson, S. (2002). A framework for real-time path planning in changing environments. *The International Journal of Robotics Research*, 21(12):999–1030.

Likhachev, M. and Ferguson, D. (2009). Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945.

Marble, J. D. and Bekris, K. E. (2013). Asymptotically near-optimal planning with probabilistic roadmap spanners. *IEEE Transactions on Robotics*, 29(2):432–444.

Martin, S. R., Wright, S. E., and Sheppard, J. W. (2007). Offline and online evolutionary bidirectional RRT algorithms for efficient re-planning in dynamic environments. In *IEEE Conference of Automation Science and Engineering*. IEEE.

Otte, M. (2011). *Any-Com Multi-Robot Path Planning*. PhD thesis, University of Colorado at Boulder, Dept. Computer Science.

Otte, M. (2014). Videos of RRT^X simulations in various state spaces. `http://tinyurl.com/l53gzgd`.

Otte, M. and Frazzoli, E. (2014). RRT^X: Real-time motion planning/replanning for environments with unpredictable obstacles. In *Proc. International Workshop on the Algorithmic Foundations of Robotics*.

Otte, M., Richardson, S. G., Mulligan, J., and Grudic, G. (2007). Local path planning in image space for autonomous robot navigation in unstructured environments. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2819–2826. IEEE.

Pivtoraiko, M., Knepper, R. A., and Kelly, A. (2009). Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333.

Plaku, E., Kavraki, L. E., and Vardi, M. Y. (2010). Motion planning with dynamics by a synergistic combination of layers of planning. *Robotics, IEEE Transactions on*, 26(3):469–482.

Pomarlan, M. and Şucan, I. A. (2013). Motion planning for manipulators in dynamically changing environments using real-time mapping of free workspace. In *Computational Intelligence and Informatics (CINTI), 2013 IEEE 14th International Symposium on*, pages 483–487.

Reif, J. and Sharir, M. (1985). Motion planning in the presence of moving obstacles. In *Foundations of Computer Science, 1985., 26th Annual Symposium on*, pages 144–154.

Rimon, E. and Koditschek, D. E. (1992). Exact robot navigation using artificial potential functions. *Robotics and Automation, IEEE Transactions on*, 8(5):501–518.

Salzman, O. and Halperin, D. (2014). Asymptotically near-optimal RRT for fast, high-quality, motion planning. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 4680–4685.

Salzman, O., Shaharabani, D., Agarwal, P. K., and Halperin, D. (2014). Sparsification of motion-planning roadmaps by edge contraction. *The International Journal of Robotics Research*, 33(14):1711–1725.

Sermanet, P., Scoffier, M., Crudele, C., Muller, U., and LeCun, Y. (2008). Learning maneuver dictionaries for ground robot planning. In *Proc. 39th International Symposium on Robotics (ISR'08)*.

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

Sugiyama, M., Kawano, Y., Niizuma, M., Takagaki, M., Tomizawa, M., and Degawa, S. (1994). Navigation system for an autonomous vehicle with hierarchical map and planner. In *Intelligent Vehicles' 94 Symposium, Proceedings of the*, pages 50–55. IEEE.

Tedrake, R., Manchester, I. R., Tobenkin, M., and Roberts, J. W. (2010). LQR-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052.

Wang, W., Balkcom, D., and Chakrabarti, A. (2015). A fast online spanner for roadmap construction. *The International Journal of Robotics Research*.

Yang, Y. and Brock, O. (2010). Elastic roadmaps-motion generation for autonomous mobile manipulation. *Autonomous Robots*, 28(1):113–130.

Zucker, M., Kuffner, J., and Branicky, M. (2007). Multipartite RRTs for rapid replanning in dynamic environments. In *Robotics and Automation (ICRA), IEEE International Conference on*. IEEE.