Anytime Kinodynamic Motion Planning using Region-Guided Search

Matthew G. Westbrook[1] and Wheeler Ruml[2]

Abstract— **Many kinodynamic motion planners have been developed that guarantee probabilistic completeness and asymptotic optimality for systems for which steering functions are available. Recently, some planners have been developed that achieve these properties of completeness and optimality without requiring a steering function. However, these planners have not taken strong advantage of heuristic guidance to speed their search. This paper introduces Region Informed Optimal Trees (RIOT), a sampling-based, asymptotically optimal motion planner for systems without steering functions. RIOT's search is guided by a low-dimensional abstraction of the state space that is updated during planning for better guidance. Simulation results suggest RIOT is adaptable, scalable, and more effective on difficult problems than previous work.**

Index Terms—**Motion Planning, Dynamics, Non-holonomic systems.**

I. Introduction

This paper addresses *kinodynamic* planning, in which one must find a feasible system trajectory (or equivalently, a sequence of controls) that obeys not just geometric and kinematic constraints but also dynamic ones such as limited acceleration in the face of inertia. There are many types of algorithms for this setting. Algorithms based on heuristic graph search often exploit a heuristic cost-to-go estimate to focus their efforts on states that participate in low-cost trajectories [1]–[3]. The heuristic estimates are often derived from plan costs in low-dimensional abstractions. However, the completeness and optimality of graph-based methods depends heavily on how the state space is discretized; such planners almost never find truly optimal motions and an unlucky discretization can cause incompleteness.

Sampling-based motion planners are not limited by a fixed discretization and provide probabilistic completeness in continuous space. Some methods [4] aim to quickly find a single solution while others, known as *anytime* methods, continue searching to find better solutions until stopped, asymptotically converging on an optimal solution [5]. Some of these planners require a *steering function* that, given two states, returns an optimal feasible trajectory connecting them (without considering obstacles) [5], [6], while others only require the ability to forward simulate the dynamics of the system [4], [7]. Inspired by graph search, some sampling-based methods use abstraction to derive heuristics [8], [9]. These planners are effective at finding single solutions quickly but do not converge to optimal solutions.

Other types of methods include trajectory optimization [10] and potential fields [11]. While these can be very effective for simple spaces, they tend to be susceptible to local minima and have difficulty in complex environments.

In this paper, our aim is to combine the most promising aspects of previous work to develop a relatively simple asymptotically optimal kinodynamic motion planner that exploits abstraction-based heuristics for fast search but does not require a steering function, and is thus widely applicable. Our proposed method, Region Informed Optimal Trees (RIOT), uses an abstraction of the state space to bias where the motion tree is grown. The path costs in the abstraction adapt over time to exploit experience during the search. We compare RIOT experimentally against DIRT [12], the current state-of-the-art in asymptotically optimal kinodynamic motion planning without a steering function. For a variety of problems, RIOT finds lower cost solutions faster than DIRT. For some easy problems, DIRT is faster. We find that, even when provided with the same heuristic information as RIOT, DIRT discovers solutions more slowly. We also show that RIOT works well even when its abstraction is relatively coarse or initially not representative of the state space. RIOT's strong performance supports the continuing trend of integrating ideas from heuristic graph search into sampling-based motion planning.

II. Previous Work

We will touch on only the most closely related work; for thorough background see [13] or [14]. There are many planners that are capable of kinodynamic path planning. Rapidly-exploring Random Trees (RRT) [4] has probabilistic completeness in most cases [15]. The motion tree in standard RRT is extended by choosing a sample uniformly from the entire state space and expanding from the nearest neighbor of the sample in the motion tree. RRT has a natural Voronoi bias to fill the state space and was shown to find solutions to difficult problems and does not require a steering function. RRT is intended to find a single solution and is therefore not intended, and most often does not, converge to optimal [5]. RRT is not guided by a heuristic function so the search tends to go everywhere equally and can waste search effort. A goal bias is often implemented to encourage the search to reach the goal [4] but this typically consists of just occasionally selecting the goal as the target state.

RRT* [5] is a modification of RRT to achieve asymptotic optimality but it requires a steering function (or controllable linear dynamics and a commercial solver [16]). RRT* uses a process of rewiring states in the search graph when there are neighbors nearby which can be used to obtain better trajectories. This rewiring scheme is shown to converge to optimal solutions, but in practice it can be slow. Batch Informed

The authors are with the [1]Department of Mechanical Engineering and the [2]Department of Computer Science at the University of New Hampshire, USA; mgw10@wildcats.unh.edu, ruml@cs.unh.edu.

Trees (BIT*) [6] is a sampling based motion planner that carefully uses heuristics to minimize unnecessary collision checking. Building on the approach of [17], BIT* places a batch of samples at once and connects and rewires them to find the optimal path to the goal with the current samples. This is repeated to converge to an optimal solution. BIT* is much faster than previous sampling-based optimal planners. Like RRT*, BIT* depends on a steering function in order to connect and rewire the samples. Thus, these algorithms are not suitable for robots with dynamic constraints.

Some motion planning algorithms have achieved probabilistic completeness and asymptotic optimality without requiring a steering function. Sparse Stable Trees (SST) [7] uses an RRT-like expansion with a biased selection to get better solutions over time. SST also uses pruning to maintain a sparse tree and achieve near-optimality while lowering the cost of nearest neighbor queries. SST* [7] slowly reduces the sparsity in SST for asymptotic optimality.

Dominance Informed Region Trees (DIRT) [12] uses regions around states in the motion tree to search in promising areas while maintaining completeness and optimality. The regions in DIRT are larger for states that are likely to provide better paths to the goal. When a sample is uniformly taken in the state space, a state to expand is chosen randomly from all states whose dominance regions contain the sample, therefore there is an indirect effect that larger regions will be selected more often. DIRT also uses a "blossom expansion" the first time a state is selected to attempt to create an edge that gets the best path to the goal. The blossom expansion creates many possible trajectories, from a curated set or using a randomly selected control, and selects the best one to be used according to heuristics. Further expansions of a state use a single random control to maintain completeness and asymptotic optimality. A greedy expansion is also used in DIRT to immediately select a new state for expansion whenever it has a better heuristic value than its parent. This encourages fast convergence to the goal in easy areas. DIRT was shown to have significant speed-ups from previous planners such as SST and Rand A* [18]. The disadvantage of DIRT is the optimistic heuristic and low transparency of the algorithm. As the heuristic in DIRT must be admissible and does not take obstacles into account, the search is susceptible to local minima and the algorithm relies on random exploration to escape them rather than adapting. This makes searching for solutions in high dimensional spaces or narrow passages difficult. The dominance informed regions can also be difficult to conceptualize, making it harder to alter the algorithm without losing desired properties.

RRT 2.0 [19] plans in the state-cost space to achieve optimal kinodynamic motion planning. RRT 2.0 is not heuristically guided although hybrid adaptations of RRT 2.0 with other planners are proposed in [19] which provide heuristics and improve performance in some cases. RRT 2.0 was shown to have better performance than SST although with less of a performance increase than DIRT achieved.

There are also many planners which have taken advantage of simplifications or abstractions of the state space in order to guide high dimensional search. SyCLoP [8] uses a decomposition of the state space, thereby taking obstacles into account, to find the most promising regions for the sampling based planner to explore. By searching the decomposition as a graph, these regions are used as a heuristic to guide the search. Because the graph is relatively small, the heuristic values can be quickly computed to bias the search in the continuous space. GUST [9] further develops this idea for a fast search to goal based on an abstraction of the state space. GUST updates the regions based on experience and makes the abstraction more fine grained in difficult areas of the state space. These algorithms are intended to quickly find one solution and therefore are not asymptotically optimal.

III. Problem Statement

We formalize the motion planning problem as an 8-tuple $\{\mathbb{X}, \mathbb{M}, x_{start}, \mathbb{T}_G, \mathbb{U}, Prop, T_{max}, h\}$. The state space of the problem $\mathbb{X}$ can be partitioned into a collision-free subset $\mathbb{X}_f$ and an obstacle subset $\mathbb{X}_{obs}$. There is a start state, $x_{start} \in \mathbb{X}_f$. There is also a task space, $\mathbb{T}$, that is a result of mapping the state space $\mathbb{X}$ using $\mathbb{M} : \mathbb{X} \rightarrow \mathbb{T}$. The goal set is defined in the collision-free subset of the task space, $\mathbb{T}_G \subset \mathbb{T}_f$. Goal states are defined as all states x such that $\mathbb{M}(x) \in \mathbb{T}_G$. The control space is denoted $\mathbb{U}$. A trajectory $\pi(t) : [0, t_\pi] \rightarrow \mathbb{X}_f$ is a function generated by integrating the dynamics function $\dot{x} = Prop(x, u)$ with the applied controls $u \in \mathbb{U}$ and the initial state $x = \pi(0)$ for duration t_π. In our simulations, the controls u are piece-wise constant, although this is not strictly necessary for our formal analysis. Trajectory segments are generated with a varying number $t \in (0, T_{max})$ of fixed Δt time steps for $T_{max} \in \mathbb{Z}^+$ [15], limiting the number of time steps in a segment. Each trajectory from the initial state to a leaf of the motion tree τ has a cost g that we aim to minimize. We also provide a state heuristic function h which is the cost-to-go estimate from a state to the goal set. An *admissible* heuristic function h provides guidance for the search but does not overestimate the optimal cost to the goal: $h(\tau) = 0, \forall \mathbb{M}(\tau) \in \mathbb{T}_G$ and $h(\tau) \leq c^*(\tau), \forall \mathbb{M}(\tau) \in \mathbb{T}_f$ where c^* denotes the optimal cost of a solution trajectory for a given problem. An admissible heuristic is required for pruning trajectories that are more costly than an incumbent solution.

The $f(x)$ value of a state $x \in \mathbb{X}$ is the cost of the trajectory to the state, $g(x)$, plus the heuristic estimate to the goal from the state, $h(x)$. This function is used to determine which states are more likely to provide low cost solutions. It is also used to prune states that are unable to provide a better solution than the incumbent.

We will say that trajectories π_1 and π_2 are δ-*similar* if the end state of π_2 is within a δ radius ball of the end state of π_1 and that trajectory π has ϵ *clearance* if all states in the trajectory are at least ϵ away from obstacles. We will make use of the following relatively weak standard assumptions:
Assumption 1: Chow's condition [20] of Small-time Locally Accessible (STLA) systems is met for the dynamics of the system. This provides that the reachable set of states $A \subset V$ from state x in time $t \leq T_{max}\Delta t$ without leaving

neighborhood $V \subset \mathbb{X}$ has the same dimensionality as $\mathbb{X}$ for any V. This assumption implies the existence of δ similar trajectories for any trajectory π [7].

Assumption 2: The second derivative of the dynamics of the system $\ddot{x}(t)$ are bounded.

Assumption 3: The dynamics of the system are Lipschitz continuous in both $x(t)$ and $u(t)$. The cost function must also be Lipschitz continuous w.r.t $\mathbb{X}$, additive, monotonically increasing, and non-degenerate.

Assumption 4: There exists a ϵ clearance solution trajectory, π_{sol}, generated by a piece-wise constant control function, $\bar{\Upsilon}$. This assumption implies the planning query can be solved given the discrete, piece-wise constant control space $\mathbb{U}$ and dynamics $\dot{x} = Prop(x, u)$.

Assumption 5: The task space mapping function, $\mathbb{M}$, must have topological equivalence w.r.t. the cost, heuristic, and distance functions between the state space, $\mathbb{X}$, and task space, $\mathbb{T}$. This means the task space and state space are capable of being transformed into one another by a continuous one-to-one transformation in both directions. This assumption allows for analysis in both the state and task space and implies accuracy for cost, heuristic, and distance evaluation in the state space.

An algorithm is *probabilistically complete* if the probability that the algorithm will find a solution if one exists approaches 1 as the iterations n approaches infinity, $\lim_{n\to\infty} \mathbb{P}(\exists x_{goal} \in (\mathbb{M}(\tau_n) \cap \mathbb{T}_G)) = 1$ for the motion tree τ which contains all trajectories. An algorithm is *asymptotically optimal* if the probability of finding the optimal solution approaches 1 as the iterations approach infinity, $\mathbb{P}(\limsup_{n\to\infty} Y_n = c^*) = 1$ where Y_n is a random variable representing the minimum cost over all trajectories returned by the end of iteration n of the algorithm.

IV. Algorithm

RIOT combines aspects of GUST, DIRT, and BIT* to produce a heuristically guided, probabilistically complete, asymptotically optimal kinodynamic motion planner that does not require a steering function. The psuedocode is shown in Algorithm 1 where the inputs are the start state x_{start}, goal set x_{goal}, the map of the environment, the blossom number b_n, the control space $\mathbb{U}$, the forward propagation function $Prop$, max propagation time T_{max}, and heuristic function h. The algorithm starts by partitioning an abstraction of the task space, $\mathbb{T}$, into a discretized graph of regions, denoted G at Line 1. The abstraction can be created however desired, for example uniform grid, random sampling and Voronoi decomposition, or triangular decomposition as long as it covers the task space. The purpose of G is to estimate the cost of optimal solutions through every region of the state space. This allows the planner to prioritize searching in the most likely regions to provide high quality solutions. Each region r has the properties $\hat{g}, \hat{h}, \hat{f}, P_s, \hat{g}_{max}$, and $\hat{h}_{max}$ which are also used by any state which maps to that region. The values $\hat{g}, \hat{h}$, and $\hat{f}$ are found by searching G, discussed in more detail below. P_s is the probability of successful propagation to a region from any other region and is used to weight the edge cost between regions during the search of G. The values $\hat{g}_{max}$ and $\hat{h}_{max}$ upper bound $\hat{g}$ and $\hat{h}$ respectively in each region. Edges are created between regions with cost according to the task space cost between their center points, weighted by P_s. We also keep track of all regions touched by the motion tree, denoted $G_{interior}$. We will use the shorthand $G(x)$ to refer to the region that state x maps to.

Algorithm 1 RIOT $(x_{start}, x_{goal}, map, b_n, \mathbb{U}, Prop, T_{max}, h)$

1: $G \leftarrow InitializeAbstraction(map)$
2: $\tau \leftarrow \{x_{start}\}$
3: $\pi_{sol} \leftarrow \emptyset, \pi_{sol_c} = \infty$
4: $x_{new} \leftarrow \emptyset, x_{sel} \leftarrow \emptyset, \beta \leftarrow \emptyset$
5: **while** $TimeRemaining$ **do**
6: **if** $Greedy(G, x_{new})$ **then**
7: $x_{sel} = x_{new}$
8: **else if** $\pi_{sol_c} = \infty$ **then**
9: $x_{sel} = NearestNeighbor(RandomSample(map))$
10: **else**
11: $AbstractionSearch(G, x_{start}, x_{goal})$
12: $r \leftarrow SelectRegion(G_{interior})$
13: $x_{sel} \leftarrow SelectState(G, r)$
14: $x_{new} \leftarrow \emptyset$
15: **if** $f(x_{sel}) \leq \pi_{sol_c}$ **then**
16: **if** $x_{sel} \in \beta$ **then**
17: $E_{cand}(x_{sel}) \leftarrow Blossom(x_{sel}, \mathbb{U}, T_{max}, 1)$
18: **else**
19: $E_{cand}(x_{sel}) \leftarrow Blossom(x_{sel}, \mathbb{U}, T_{max}, b_n)$
20: $\beta \leftarrow \beta \cup x_{sel}$
21: **while** $E_{cand}(x_{sel}) \neq \emptyset$ **do**
22: $C \leftarrow \underset{\hat{f},\hat{h},f}{\operatorname{argmin}} E_{cand}(x_{sel})$
23: $E_{cand}(x_{sel}) \leftarrow E_{cand}(x_{sel}) \backslash C$
24: $x_{new} \leftarrow Propagate(x_{sel}, C)$
25: $UpdateP_s(G, x_{new})$
26: **if** $x_{new} \neq \emptyset$ **and** $f(x_{new}) \leq \pi_{sol_c}$ **then**
27: $\tau \leftarrow \tau \cup x_{new}$
28: $UpdateInterior(G, x_{new})$
29: $Max_g(G, x_{new})$
30: **break**
31: **if** $x_{new} \in x_{goal}$ **then**
32: **if** $g(x_{new}) < \pi_{sol_c}$ **then**
33: $\pi_{sol} = \pi(x_{new}), \pi_{sol_c} = g(x_{new})$
34: $Max_h(G, x_{new})$
35: **return** π_{sol}

Each iteration of the main loop at Line 5 starts by selecting a state in the motion tree to extend from. There is a possibility of greedy selection of a state in Line 6 of each iteration. The greedy test checks if a new state was added in the last iteration. If the new state is in a better heuristic region than its parent ($\hat{h}(x_{new}) < \hat{h}(x_{new.parent})$) or in an equal heuristic region with a better state heuristic, this new state is immediately selected. This greedy selection improves time to find solutions and optimal solution convergence by utilizing the abstraction. If the greedy selection is not used,

and no solution has been found, a random sample is taken in the task space and the nearest state is selected in Line 9. This improves exploration of the search, ensuring it is not too greedy when a solution has not yet been found and the abstraction may be inaccurate. Otherwise the abstraction is searched and a region is selected from $G_{interior}$ at Line 12 with probability proportional to $\frac{1}{\hat{f}(r)}$, inversely proportional to the $\hat{f}$ value of each region. This means that regions likely to provide high quality solutions are selected more often. This is just one of many possible ways to bias the region selection. A state is selected uniformly from all states in the motion tree that map to the selected region. If the selected state is not deemed irrelevant by Line 15 incumbent pruning based on the current best solution cost π_{sol_c}, it is expanded.

Expansion starts by creating a set of edges for the selected state. Edges come from $Blossom(x_{sel}, \mathbb{U}, T_{max}, n)$ which returns a set of n edges from controls $u \in \mathbb{U}$ generated for random time steps $[1; T_{max}]$. If x_{sel} is not in the set β that records states that have previously been selected for expansion (Line 16), a set of $|E_{cand}| = b_n$ edges are created by $Blossom$ (Line 19). For all future selections of a state, one edge is randomly generated (Line 17). From this set of edges, edges are collision checked, best first, until a collision free edge is found. For each propagation, the probability of success is updated in $UpdateP_s$ for region $G(x_{new})$ (Line 25) depending on whether it was a collision or successful. The sorting of these edges from the destination is by $\hat{f}$ value of the region first, $\hat{h}$ value of the region second, and f value of the state last. This prioritizes search through low cost regions towards the goal. If the new state is not pruned by the incumbent (Line 26), it is added to the motion tree.

After adding the state to the motion tree, the set of interior regions is updated in $UpdateInterior$ at Line 28. This ensures that if a new region has been touched by the motion tree then it can be selected for expansion. For each state added to the motion tree, Max_g in Line 29 checks if it is the lowest cost trajectory to that region seen so far and if so $\hat{g}_{max}$ of that region becomes the g value of that trajectory. A similar update is done in Max_h in Line 34 to change the $\hat{h}_{max}$ values of regions along the solution trajectory.

$AbstractionSearch$ at Line 11 runs *Dijkstra's* Algorithm on the abstraction from the region that contains the start state $G(x_{start})$ to estimate the cost from the start to each region, $\hat{g}$, and again from the region(s) containing the goal set to estimate the cost from the goal to each region, $\hat{h}$. The sum of the cost-to-come $\hat{g}$ and cost-to-go $\hat{h}$ yields an estimate $\hat{f}$ of the lowest cost path from start to goal through that region. The region costs are initialized to the minimum $\hat{g}$ or $\hat{h}$ value of current trajectories through that region, which upper bound the $AbstractionSearch$ cost with $\hat{g}_{max}$ and $\hat{h}_{max}$ respectively. An example of this $\hat{f}$ estimate can be seen by the color gradient in the House and Obstacle Field maps in Fig. 1 where the dark blue regions are expected to have lower cost trajectories. The abstraction is meant to be as close an approximation to an optimal solution through each region as possible, therefore it is inappropriate to have an abstraction cost greater than a realized cost in the motion tree. However, the abstraction is not forced to be admissible, as this may decrease performance and is not necessary to maintain any desired properties. For certain environments, it may not be possible to determine if regions are completely occluded and initializing the probability of successful propagation may require estimates. The edge costs are weighted by dividing them by $P_s(dest)$, the probability that a state propagation that ends in the region represented by the destination vertex will be collision free. This success probability is updated every time a new motion is computed, so that RIOT can recognize when a region is blocked or difficult to navigate and try growing the motion tree elsewhere, similar to [21]. The weighted edge cost $c = edge(source, dest)/P_s(dest)$ from region $source$ to region $dest$ approaches infinity as the probability of successful propagations approaches 0, $\lim_{P_s(dest)\to 0}(c) \to \infty$. $P_s(r) = 0$ should occur only if a region is completely occluded with obstacles. This requires that P_s is initialized to an optimistic estimate with some number of initialization values. This can be done by random sampling in each region and always including at least 1 success if there is any uncertainty whether a region is completely blocked.

Algorithm 2 NaiveRandomTrees($x_{start}, map, \mathbb{U}, T_{max}$)

1: $\tau \leftarrow \{x_{start}\}$
2: **while** $TimeRemaining$ **do**
3: $\quad x_{selected} \leftarrow UniformSampling(\tau)$
4: $\quad t \leftarrow Sample(0, T_{max}); \Upsilon \leftarrow Sample(\mathbb{U}, t)$
5: $\quad x_{new} \leftarrow \int_0^t f(x(t), \Upsilon(t))dt + x_{prop}$
6: $\quad$ **if** $CollisionFree(\overline{x_{selected} \to x_{new}})$ **then**
7: $\quad\quad \tau \leftarrow \{x_{new}\}$
8: **return** τ

V. Analysis

Following [7], our proof that RIOT is asymptotically optimal will be based on its similarity to a simple method, Naive Random Trees (NRT), shown in Algorithm 2. NRT expands a motion tree by uniformly selecting a state from the motion tree in Line 3. The control and time to propagate the control is then sampled randomly in Line 4 and propagated in Line 5. The trajectory is collision checked in Line 6 and added to the motion tree in Line 7 if collision free. It was shown by [7] that NRT is asymptotically optimal, so our proof merely needs to show how a subset of RIOT's expansions can be seen as a simulation of that method.

Lemma 1: RIOT will use the non-greedy state selection infinitely many times as time approaches infinity.

Proof: The greedy selection only occurs for a state with a lower region heuristic or equal region heuristic and lower state heuristic than its parent. The abstraction heuristic values are not altered until a greedy choice is not selected in Line 11 of Algorithm 1, therefore the region heuristic is constant until the greedy choice has ended. Eventually a greedy choice will not be possible as a constant heuristic can not infinitely decrease. Therefore it is not possible to continually go to a

region with a lower heuristic and thus the non-greedy choice will be selected. This argument holds repeatedly, thus a non-greedy choice will be made infinitely many times as time approaches infinity.

Lemma 2: RIOT's non-greedy choice has positive probability of selecting any state in the motion tree for expansion. (Note this holds regardless of the admissibility of the abstraction heuristic.)

Proof: When the greedy choice is not selected, if a solution has not yet been found, the nearest neighbor of a random sample is selected in Line 9. This means that the algorithm has positive probability of selecting any state. If an incumbent solution exists, RIOT first selects a region from among those that have been touched by the motion tree and contain a state (Lines 12 and 28 of Algorithm 1). The $\hat{f}$ bias for region selection in Line 12 assigns positive probability of selection for any region r with $\hat{f}(r) < \infty$. $\hat{g}(r)$ is upper bounded by the cheapest trajectory to that region, thus clearly finite for all regions in $G_{interior}$. $\hat{h}(r)$ is determined by the cost to the goal in the abstraction, which can only be infinite if $P_s = 0$ for some region on every path to the goal. Because $P_s = 0$ is only for completely blocked regions and a solution exists by assumption 4, the $\hat{f}$ values of reachable regions will remain finite. Once a region is selected, RIOT selects a state from the motion tree in that region uniformly, therefore every state has positive probability of being selected.

Lemma 3: RIOT will generate all trajectories from a state given infinite selections.

Proof: In RIOT there is a Blossom expansion the first time a state is selected for expansion. For all future selections of that state, only a single random control is generated and with infinite attempts all controls will be generated. This implies that with infinite time RIOT will realize the full reachability of each state.

Theorem 1: RIOT is probabilistically complete and asymptotically optimal.

Proof: By Lemma 1 and Lemma 2, RIOT simulates Line 3 of NRT. By Lemma 3, RIOT will generate all trajectories from a given state as in Line 4 of NRT. RIOT will also add the new state to the tree if collision free and not irrelevant due to incumbent pruning as in Lines 6 and 7 of NRT. Thus, RIOT simulates NRT. Theorem 18 of [7] shows that NRT will eventually find a solution if one exists, thus proving probabilistic completeness of RIOT. Theorem 20 in [7] shows that NRT, and therefore RIOT, is also asymptotically optimal.

VI. Simulation Results

Both RIOT and DIRT are asymptotically optimal but their time to find initial solutions and their convergence rates to optimal solutions may differ. DIRT was chosen to compare with RIOT because [12] showed DIRT outperforming other algorithms such as SST and Rand A* [18] for kinodynamic anytime motion planning in almost all cases. To evaluate the performance of DIRT and RIOT experimentally, we implemented them in C++ and in Unreal Engine 4 and tested on a 3.40GHz Intel i5-7500 CPU.[1] For our experiments we used the same blossom number for RIOT and DIRT. This was decided after experimenting with different blossom numbers for each and concluding changing the blossom number similarly effected each algorithm.

A. General Performance

The environments and results for general performance experiments are shown in Fig. 1. Four 2D and 3D environments were represented as occupancy grids. The abstraction in these experiments matches the resolutions of the occupancy grid. The cost function is distance and the state heuristic is Euclidean distance to the goal. Simulations were run for 50 trials of 120 seconds each with the same start/goal pair and random seeds. For all solution costs vs. time plots data is only shown for times after which the algorithm has solved all instances of the problem, allowing us to compare the means and time to find initial solutions.

Three different vehicle dynamics were tested, all represented as points for collision checking. The first vehicle is a dynamic car which operates in 2D space taken from code in OMPL [22]. The state of the dynamic car is 4 dimensional: the xy coordinates, steering angle β, and forward velocity ν. There are two control inputs for the dynamic car: forward acceleration, u_0, and steering rate, u_1. The second vehicle is the hovercraft from [23] with a 6 dimensional state space: the xy coordinates, heading angle ϕ, forward velocity u, sway velocity v, and rotational velocity r. The hovercraft is controlled by a thrust, F_u, and rudder angle δ. The last robot is a generic double integrator point robot operating in 3D space. This robot is the dynamic version of the 3D case of the point robot in [24]. This robot has a 6 dimensional state space for the xyz coordinates and velocity and a 3 dimensional control space for accelerations in each direction.

In all experiments RIOT is able to find initial solutions faster than DIRT. (We will discuss the RIOT+ results below in Section VI-E.) For the large sparse 2D environment with the dynamic car (lower left panel) and the slalom environment with the generic 3D robot (lower right panel), DIRT is able to find lower cost solutions than RIOT. Both environments have a Euclidean distance heuristic similar to an optimal solution. The DIRT blossom in simple problems such as these can more quickly converge to optimal as the discrete nature of the RIOT abstraction makes a larger area share the same priority. For the other four problems RIOT quickly found lower cost solutions than DIRT. The RIOT abstraction was very effective for guiding the search in harder problems with a less direct solution. The house and 3D maze environments (upper panels) had large local minima and very indirect paths to the goal, making the performance of RIOT better than DIRT. The higher dimensional state space of the hovercraft (middle panels) vs. the dynamic car (left panels) made it much harder for both algorithms to find solutions in the obstacle field. This difficulty lead to RIOT outperforming

[1]We thank Aravind Sivaramakrishnan, Zakary Littlefield, and Kostas Bekris for sharing their code for DIRT, which informed our implementation. Our code is available at `https://github.com/mattgw10/RIOT`.

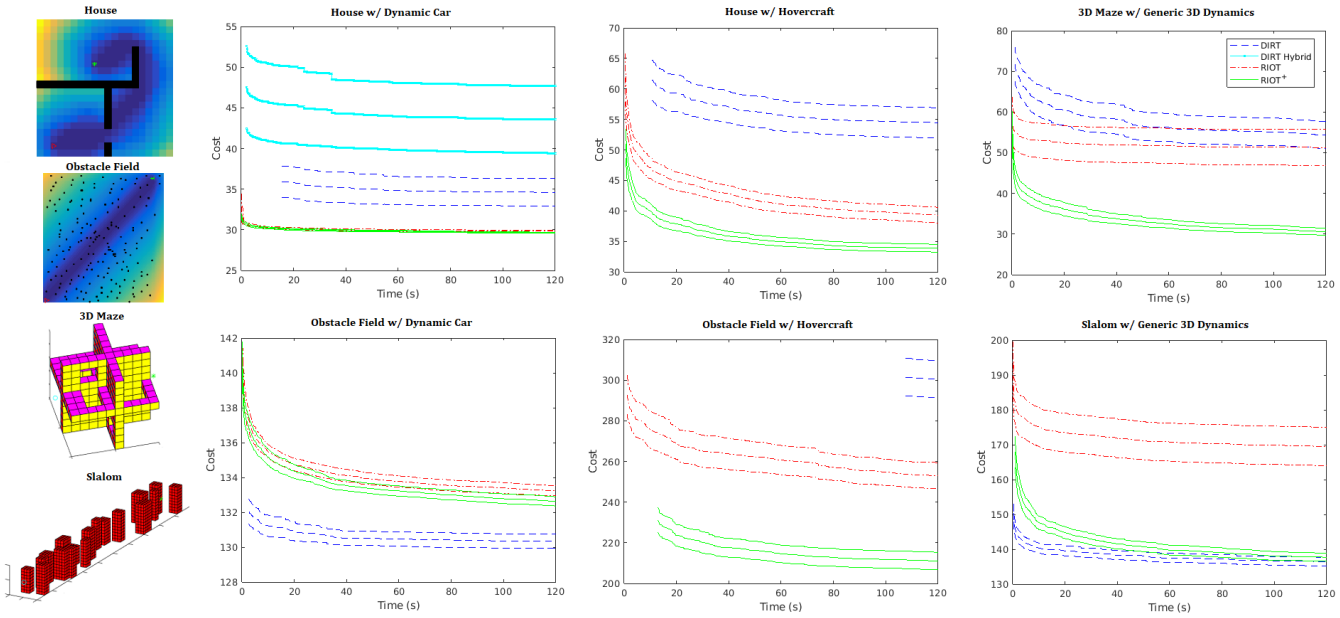

Fig. 1. Environments and solution cost vs. time comparison of DIRT and RIOT for general motion planning with abstraction resolution equal to occupancy grid resolution. Lines show average and 95 percent confidence intervals for each experiment.

DIRT as the blossom in DIRT only sorts using f values while RIOT gives priority to states closer to the goal by using region $\hat{f}$, region $\hat{h}$, then state f.

To ascertain whether RIOT's performance is merely a result of its abstraction-based heuristic, the plot for the house with dynamic car also includes a hybrid version of DIRT that uses the initial abstraction from RIOT as its heuristic. Using this abstraction, DIRT hybrid was able to solve all trials faster than regular DIRT but the solutions were of lower quality. This shows that even with access to a heuristic that guides around obstacles, the search strategy of DIRT does not utilize this information as effectively as RIOT.

B. Varying Abstraction Resolution

Given that RIOT relies heavily on the abstraction to bias its search, we conducted experiments to assess the effect of varying the abstraction resolution. Three environments were used (Fig. 2). The first was chosen as a difficult case for RIOT: a 100x100 2-D ladder-like maze with thin walls. The initial abstraction will view each wall with some probability of going through it, thus skewing the cost of each region. The other two maps are taken from [25]: orz100d is 395x412 and comes from the video game Dragon Age: Origins2D and Boston_0_256 is 256x256. For these maps, we used 200 start/goal pairs specified by [25] (bins 50-69). The vehicle is the dynamic car and the cost function is distance with a Euclidean distance heuristic. Each trial is 120 seconds long. Neither algorithm was able to solve every instance, so we plot solution cost for each algorithm as the average over those instances solved by that time.

Results are shown in Fig. 3. RIOT was able to solve more instances than DIRT in every problem. DIRT was unable to find solutions in any of the 2D maze instances. Abstraction A1 is at the full occupancy grid resolution in each case and abstraction A2 is lower resolution (maze 50x50, orz100d 103x79, Boston 128x128). For all environments, the higher resolution abstractions resulted in better performance for RIOT. The effect of a low-resolution abstraction was most significant in the 2D maze. RIOT initially believes there is some probability of going through each wall to the goal due to the low resolution. The probability of successful propagation to each region is updated, eventually informing RIOT to make these edges more costly. However, RIOT still greatly outperforms DIRT for this challenging problem. For the other maps, DIRT initially appears to find lower cost solutions than the lower resolution RIOT, but it is solving significantly fewer problems and its average cost increases over time, leading us to conclude that RIOT performs better overall, even when the abstraction is relatively coarse.

C. Robotic Arm

While these results have demonstrated the effectiveness of abstraction for mobile robots, it is important to assess if RIOT can be effective for other systems, such as manipulators. Simulations were run on a dynamic robot arm to test how RIOT performs with non-point robots which may have a less obvious state space representation in the abstraction. Time is used as the cost function. These experiments were implemented in Unreal Engine 4 with a 5-link robotic arm. The robotic arm is a dynamic version of the arm in [24]. It has a 10 dimensional state space for each angle and velocity and is controlled by accelerations at each joint for a 5 dimensional control space. Collision checking in these environments is done on meshes of the robot and environment, with no defined occupancy grid. The abstraction represents the x, y position of the end effector using a grid with boundaries at the max extension of the arm. The heuristic is the Euclidean distance of the end

Fig. 2. Environments used for varying abstraction resolution, dynamic arm, and scaling. From left to right 2D maze environment, orz100d environment from [25], Boston_0_256 map from [25], arm around wall environment, and 2D example of Kink problem from [24].

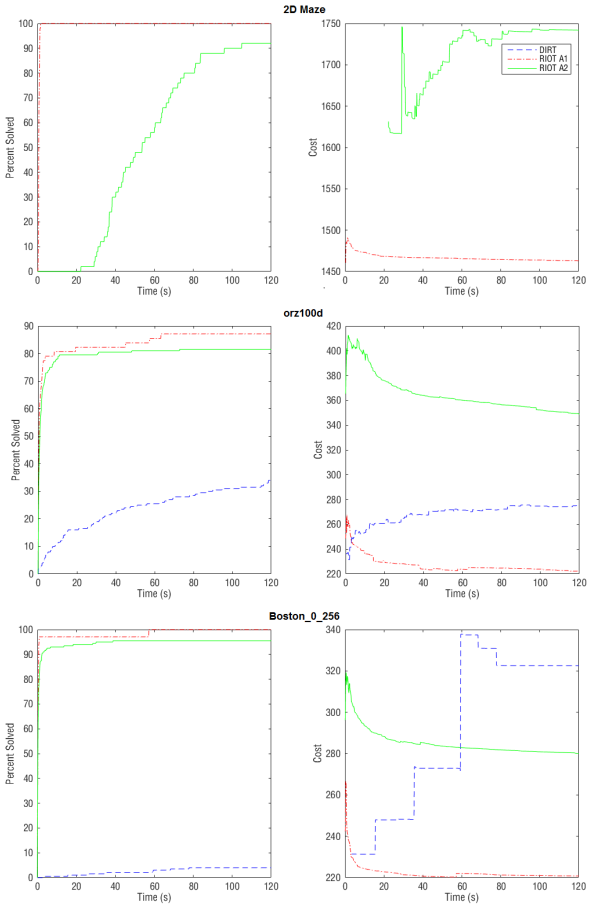

Fig. 3. Percent of problems solved and solution cost vs. time comparison of RIOT at different resolutions for the 2-D maze, orz100d map, and Boston_0_256 map with the dynamic car.

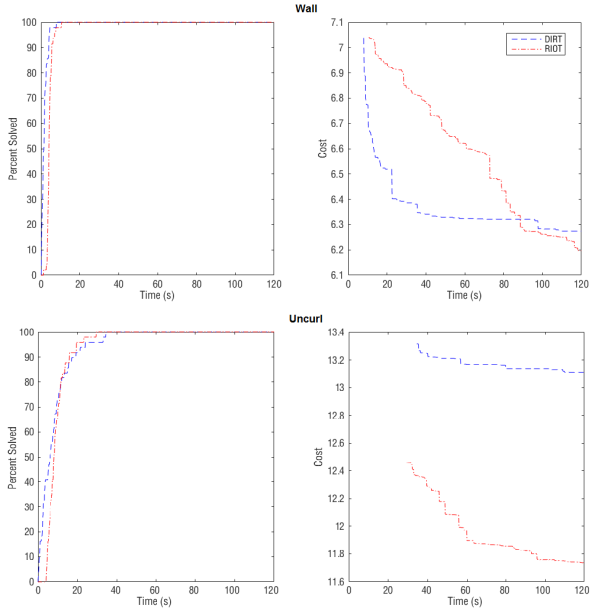

Fig. 4. Percent of problems solved and solution cost vs. time comparison of DIRT and RIOT for the dynamic arm trying to reach around obstacle to goal and trying to extend from the curled position.

effector position to the goal. At least 1 initialization success is used in each region as no region is immediately known as unreachable. For both environments the abstraction size used was 100x100 and the regions are initialized by sampling 10,000 random linkage angles and checking if the robot is in collision. The first robot arm environment is obstacle free but the arm starts in a curled position and can be in collision with itself, as in [24]. The goal is for the arm to reach the forward extended position to reach the goal region. The second robot arm environment shown in Fig. 2 has the arm initially wrapped around a wall with the end effector very close to the goal. Due to the presence of the wall, the end effector is unable to directly move towards the goal and the arm must retract away from the goal to reach around the other side.

Results are shown in Fig. 4. For both environments DIRT and RIOT solved instances in similar times. In these environments the heuristic for both algorithms is initially very inaccurate. The random sampling allows both algorithms to have a Voronoi bias to explore areas of the state space which the heuristic implies are not valuable to explore. After finding an initial solution the RIOT abstraction updates so the search can be guided accordingly with the remaining time. In the wall scenario DIRT initially finds lower cost solutions but eventually RIOT surpasses DIRT. In the uncurling arm scenario RIOT finds lower cost solutions than DIRT.

D. Scaling

[24] recommends evaluating motion planners using simple benchmarks, such as the kink shown in Figure 2, that can be scaled in difficulty, such as by varying the passage width and

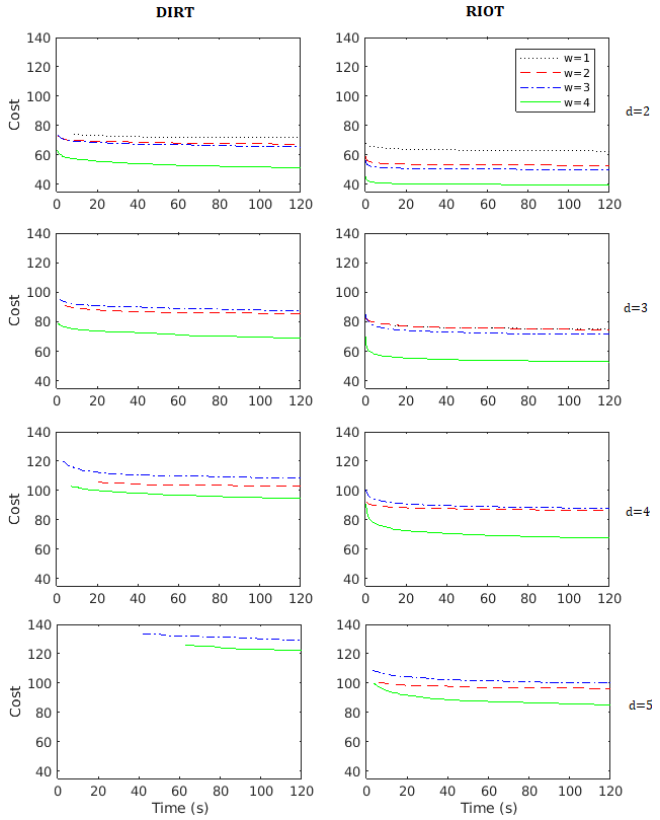

Fig. 5. Solution cost vs. time comparison of DIRT and RIOT for the generic point robot finding a solution varying width and dimensionality.

the state space dimensionality. We used the kink benchmark with a double integrator point robot with passage widths 1-4 and dimensionality 2-5. Cost results are shown in Fig. 5 for times after which all instances are solved. In every case, RIOT finds solutions faster and converges to lower cost solutions than DIRT. The performance of RIOT and DIRT is better for the trials with a larger path width and lower dimensionality as expected. DIRT is more significantly impacted by both dimensionality and path width.

E. A Less Exploratory Variant

We also performed experiments for a variant of RIOT called RIOT+, in which lines 8 and 9 of Algorithm 1 are omitted, so that Voronoi-biased selection is never used and either greedy selection or abstraction-biased region selection are always used. For domains in which the initial abstraction accurately approximates the distance-to-go, RIOT+ found lower cost solutions but sometimes took longer to find solutions. Example results are shown in Figure 1. The only domains where RIOT+ performed significantly worse than plain RIOT was the 5-link arm.

F. Sensitivity Analysis

We also performed a sensitivity analysis of RIOT, varying the number of controls b_n in a blossom between 1 and 100. While space limitations preclude presenting detailed results, the best value for b_n depended on the vehicle and environment, with larger values giving better performance in higher dimensional state spaces. However, RIOT was robust to many different values without large performance decreases. Disabling the greedy selection showed a significant decrease in performance, validating the decision to use greedy selection.

REFERENCES

[1] N. Nilsson, "Problem-solving methods in artificial intelligence," *McGraw-Hill*, 1971.

[2] M. Likhachev, G. Gordon, and S. Thrun, "Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, 2004.

[3] A. Stentz, "The focussed D* algorithm for real-time replanning," in *IJCAI*, 1995, pp. 1652—-1659.

[4] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *IJRR*, vol. 20, no. 5, pp. 378–400, May 2001.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *IJRR*, vol. 30, no. 7, pp. 846–894, 2011.

[6] J. Gammell, S. Srinivasa, and T. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *ICRA*, May 2015, pp. 3067–3074.

[7] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *IJRR*, vol. 35, no. 5, pp. 528–564, 2016.

[8] E. Plaku, L. Kavraki, and M. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 469–482, 2010.

[9] E. Plaku, "Region-guided and sampling-based tree search for motion planning with dynamics," *IEEE Transactions on Robotics*, vol. 31, pp. 723–735, 2015.

[10] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *ICRA*, 2009, pp. 489–494.

[11] C. Tingbin and Z. Qisong, "Robot motion planning based on improved artificial potential field," in *Proceedings 3rd International Conference on Computer Science and Network Technology*, 2013, pp. 1208–1211.

[12] Z. Littlefield and K. E. Bekris, "Efficient and asymptotically optimal kinodynamic motion planning via dominance-informed regions," in *IROS*, October 2018.

[13] S. LaValle, "Planning algorithms," *Cambridge University Press*, 2006.

[14] H. Choset and et al., "Principles of robot motion: Theory, algorithms, and implementations," *Boston, MA: The MIT Press*, 2005.

[15] T. Kunz and M. Stilman, "Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete," in *WAFR*, 2014, pp. 233–244.

[16] D. Webb and J. van Den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear differential contstraints," in *ICRA*, 2013.

[17] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *IJRR*, vol. 34, no. 7, pp. 883–921, 2015.

[18] S. M. Persson and I. Sharf, "Sampling-based A* algorithm for robot path-planning," *IJRR*, vol. 33, no. 13, pp. 1683–1708, 2014.

[19] M. Kleinbort, K. Solovey, R. Bonalli, E. Granados, K. E. Bekris, and D. Halperin, "An empirical study of optimal motion planning," in *ICRA*, 2020.

[20] W. Chow, "Über systeme von linearen partiellen differentialgleichungen erster ordnung," *Mathematische Annalen*, vol. 117, pp. 98–105, 1940/1941.

[21] S. Kiesel, T. Gu, and W. Ruml, "An effort bias for sampling-based motion planning," in *IROS*, September 2017.

[22] A. I. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE R&A Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[23] "Flight control of a hovercraft," 2009. [Online]. Available: \url{shorturl.at/abduU}

[24] J. Luo and K. Hauser, "An empirical study of optimal motion planning," in *IROS*, vol. 117, 2014, pp. 1761–1768.

[25] N. Sturtevant, "Benchmarks for grid-based pathfinding," *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144–148, 2012.