2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Roadmaps with Gaps over Controllers: Achieving Efficiency in Planning under Dynamics

Aravind Sivaramakrishnan, Sumanth Tangirala, Edgar Granados, Noah R. Carver, and Kostas E. Bekris

***Abstract*— This paper aims to improve the computational efficiency of motion planning for mobile robots with non-trivial dynamics through the use of learned controllers. Offline, a system-specific controller is first trained in an empty environment. Then, for the target environment, the approach constructs a data structure, a "Roadmap with Gaps," to approximately learn how to solve planning queries using the learned controller. The roadmap nodes correspond to local regions. Edges correspond to applications of the learned controller that approximately connect these regions. Gaps arise as the controller does not perfectly connect pairs of individual states along edges. Online, given a query, a tree sampling-based motion planner uses the roadmap so that the tree's expansion is informed towards the goal region. The tree expansion selects local subgoals given a wavefront on the roadmap that guides towards the goal. When the controller cannot reach a subgoal region, the planner resorts to random exploration to maintain probabilistic completeness and asymptotic optimality. The accompanying experimental evaluation shows that the approach significantly improves the computational efficiency of motion planning on various benchmarks, including physics-based vehicular models on uneven and varying friction terrains as well as a quadrotor under air pressure effects. Website: `https://prx-kinodynamic.github.io/projects/rogue`**

I. Introduction

Kinodynamic motion planning allows mobile robots with non-trivial dynamics to negotiate environments with obstacles, such as a warehouse, or physical features, such as uneven terrain and ice. The problem is challenging when there is no local steering function available that connects two states. Tree sampling-based planners [1] do not need a steering function as they only forward propagate the system's dynamics. Some variants provide Asymptotic Optimality (AO) by propagating randomly sampled controls [2]–[5]. While random controls provide theoretical properties, they result in slow convergence to high-quality solutions in practice.

This work aims to improve the efficiency of such AI kinodynamic planners [6], [7] by integrating controllers trained via Reinforcement Learning (RL) [8]. Supervision has also be used to train controllers that are then integrated with planners [9]–[12]. This is often done, however, under the assumption that a steering function is available for the robot, which is not assumed here. While one may attempt to solve planning problems directly with RL, this often suffers from large training data requirements and requires careful tuning of rewards [13]. Most RL solutions lack long-horizon reasoning, which a planner provides.

The authors are with the Dept. of Computer Science, Rutgers University, NJ, USA. E-mail: {as2578, kb572}@rutgers.edu. This work is partly supported by NSF NRT-FW-HTS award 2021628.

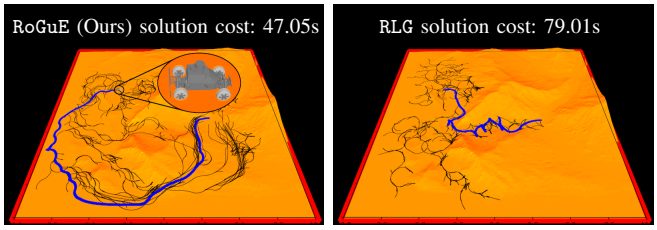


Fig. 1: Solution trajectories (thick lines) and planning trees (thin lines) for a MuSHR vehicle over an uneven terrain in MuJoCo. The cost function is trajectory duration, which is impacted by the uneven terrain. The proposed expansion method for kinodynamic planning (left) leverages a "Roadmap with Gaps" to avoid difficult terrain resulting in shorter duration solutions. The alternative expansion (right), which samples random local goals, navigates the rough hills less effectively. It results in a shorter distance path but a much slower trajectory. Planning time was 60s for both methods.

Several recent works have leveraged RL to build an abstract representation of the planning problem and search for an optimal path between the start and the goal. Search on the Replay Buffer (SoRB) [14] and Sparse Graphical Memory (SGM) [15] build a graph where the nodes correspond to states visited by the RL agent during training. Similar to SoRB and SGM, Deep Skill Graphs [16]–[18] builds a graph representation, where nodes correspond to subgoals, and edges correspond to control policies between them. A common feature of these methods is that the control policy is trained in the planning environment of interest. This requires the learning algorithm to reason jointly about the system's dynamics and the obstacles present in the environment, which is challenging for second-order systems [19].

Prior work by the authors integrates RL in kinodynamic motion planning via a decoupled strategy [20], [21], where a controller is first trained offline in an empty environment and the learning process only deals with the system's dynamics. The absence of obstacles allows learning the controller with significantly fewer data and only a sparse reward. During online planning, these prior efforts engineer local goals that bias expansion toward the global goal via manual engineering. Designing, however, an informed local goal procedure requires manual effort, which is undesirable and may not work well across environments. Furthermore, the local goals these procedures generate are unaware of the trained controller's reachability properties.

This work proposes a data structure, a **Roadmap with Gaps**, that learns the approximate reachability of local regions in a given environment given a controller. The approach constructs a directed graph in the free configuration space of the robot. An edge exists from a source node/region to a target when the source node's nominal state can reach the

target node's termination set without collisions by applying the controller. Gaps arise since the learned controller does not connect any two exact states exactly. Given a new planning query, a wavefront expressing the cost-to-goal on this roadmap provides every node with the direction of motion towards the goal. This roadmap guidance is integrated with an AO tree sampling-based planner. During the tree planner's expansion, nodes that are closer to the global goal given the roadmap guidance are prioritized.

The proposed approach provides a tradeoff. It incurs an offline computational cost for pre-processing a known environment and the robot's dynamics to provide online computational benefits for multiple planning queries in the same workspace. At the same time it retains AO properties and is an informed, efficient solution by understanding the reachability properties of the learned controller and the environment's connectivity. Prior approaches that guide tree sampling-based planners using graphical abstractions [22], [23] do not provide desirable properties (e.g., AO) or do not reason about the system's dynamics. A related kinodynamic planning approach introduced the *Bundle of Edges (BoE)* notion [24] but relies on random control propagation, which reduces efficiency.

In summary, this paper proposes the **Ro**admap-**Gu**ided **E**xpansion (RoGuE)-Tree method, which:

- Introduces the "Roadmap with Gaps," a data structure that approximately guides how a robot can navigate a target environment given a controller, and
- Adapts an Asymptotically Optimal (AO) framework for kinodynamic planning to utilize the "Roadmap with Gaps".

The accompanying evaluation shows that RoGuE results in lower-cost solutions faster than random propagation given an AO sampling-based kinodynamic planner both on analytically and physically simulated models in various environments. This includes physics-based vehicular models on uneven and varying friction terrains as well as a quadrotor under air pressure effects.

II. Preliminaries

Consider a system with state space $\mathbb{X}$ and control space $\mathbb{U}$. $\mathbb{X}$ is divided into collision-free ($\mathbb{X}_f$) and obstacle ($\mathbb{X}_o$) subsets. The dynamics $\dot{x} = f(x, u)$ (where $x \in \mathbb{X}_f, u \in \mathbb{U}$) govern the robot's motions. The process f can be an analytical ordinary differential equation (ODE) or modeled via a physics engine, e.g., MuJoCo [25]. A function $\mathbb{M} : \mathbb{X} \to \mathbb{Q}$ maps a state $x \in \mathbb{X}$ to its corresponding *configuration space* point $q \in \mathbb{Q}$ ($q = \mathbb{M}(x)$). The inverse mapping $\mathbb{M}^{-1}(q_i)$ returns a state $x_i \in \mathbb{X}$ by setting the dynamics term to some nominal value (e.g., by setting all velocity terms to 0). A distance function $d(\cdot, \cdot)$ is defined over $\mathbb{Q}$, which corresponds to a weighted Euclidean distance metric in SE(2).

A *plan* p_T is a sequence of piecewise-constant controls of duration T that induce a trajectory $\tau \in \mathcal{T}$, where $\tau : [0, T] \mapsto \mathbb{X}_f$. Given a start state $x_0 \in \mathbb{X}_f$ and a goal set $X_G \subset \mathbb{X}_f$, a feasible motion planning problem admits a solution trajectory of the form $\tau(0) = x_0, \tau(T) \in X_G$. The goal set is defined as $X_G = \{x \in \mathbb{X}_f \mid d(\mathbb{M}(x), q_G) < \epsilon\}$, or equivalently, $\mathcal{B}(q_G, \epsilon)$ where ϵ is a tolerance parameter. A heuristic $h : \mathbb{X} \to \mathbb{R}^+$ estimates the *cost-to-go* of an input state x to the goal region X_G. Each solution trajectory has a cost given by $\texttt{cost} : \mathcal{T} \to \mathbb{R}^+$. An optimal motion planning problem aims to minimize the cost of the solution trajectory.

Sampling-Based Planning Framework: The framework uses the forward propagation model f to explore $\mathbb{X}_f$ by incrementally constructing a tree via sampling in $\mathbb{U}$. Algo. 1 outlines the high-level operation of a sampling-based motion planner (Tree-SBMP) that builds a tree of states rooted at the initial state x_0 until it reaches X_G.

Algorithm 1: Tree-SBMP($\mathbb{X}, \mathbb{U}, x_0, X_G$)

```
1 T ← {x_0};
2 while termination condition is not met do
3 |   x_sel ← SELECT-NODE(T);
4 |   u ← EXPAND(x_sel);
5 |   x_new ← PROPAGATE(x_sel, u);
6 |   if (x_sel → x_new) ∈ X_f then
7 |   |   EXTEND-TREE(T, x_sel → x_new);
```

Each iteration of Tree-SBMP selects an existing tree node/state x_{sel} to expand (Line 3). Then, it generates a control sequence u and propagates it from x_{sel} to obtain a new state x_{new} (Lines 4-5). The resulting edge is added to the tree if not in collision (Lines 6-7). Upon termination (e.g., a time threshold), if the tree has states in X_G, the best-found solution according to $cost$ is returned. By varying how the key operations in Algo. 1 are implemented, different variations can be obtained. The specific variant this work adopts is the informed and AO "Dominance-Informed Region Tree" (DIRT) [4]. It uses an admissible state heuristic function h to select nodes in an informed manner. If x_{new} improves upon x_{sel} given h, then it is selected as the next x_{sel}. DIRT also propagates a "blossom" of k controls at every iteration and prioritizes the propagation of the edge that brings the robot closer to the goal given the heuristic.

III. Proposed Method

The proposed method has 2 offline stages: (i) training a controller $\pi(x, q)$ in an obstacle-free environment, and (ii) building a "Roadmap with Gaps" $(\mathcal{V}, \mathcal{E})$ in the target environment given the controller. Online, given a motion planning query (x_0, x_G) in the target environment, a sampling-based kinodynamic planner expands a tree of feasible trajectories similar to Algo. 1 above. Line 4 of Algo. 1 is adapted so as to use guidance from a wavefront function computed over the "Roadmap with Gaps". The method computes first a local goal q_{lg} that looks promising given the wavefront. It then propagates a control $u = \pi(x_{\text{sel}}, q_{\text{lg}})$ by using the trained controller π. The controller is applied at the selected node $x_{\text{sel}} \in \mathbb{X}$ and generates a control towards reaching the local goal q_{lg} without considering obstacles.

A. Training a Controller offline

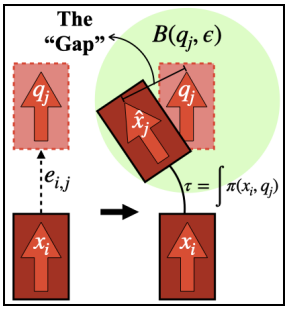
A goal-conditioned controller $u = \pi(x, q)$ is first trained via Reinforcement Learning to reach from an initial state x to a goal set $\mathcal{B}(q, \epsilon)$, i.e., within an ϵ distance of q. For

the training, this is attempted for any $(x, q) \in \mathbb{X} \times \mathbb{Q}$ in an empty environment of given dimensions. The training process collects transitions $(x_t, u_t, \mathcal{C}(x_t, q), x_{t+1}, q)$, which are stored in a replay buffer. The cost function $\mathcal{C} : \mathbb{X} \times \mathbb{Q} \to \{0, -1\}$ has a value of $\mathcal{C}(x_t, q) = 0$ iff $x_t \in \mathcal{B}(q, \epsilon)$, and -1 otherwise. During each iteration, mini-batches of transitions are sampled from the buffer. A Soft Actor-Critic (SAC) [26] algorithm is employed, which minimizes the total cost $\mathbb{E}_{x_0, q_G \sim \mathbb{X} \times \mathbb{Q}}[\sum_{t=0}^{T} \mathcal{C}(x_t, q_G)]$. Concurrently, Hindsight Experience Replay (HER) [27] relabels some transitions with alternative goals to provide additional training signals from past experiences.

B. Building a Roadmap with Gaps offline

The proposed approach then builds a "Roadmap with Gaps" $(\mathcal{V}, \mathcal{E})$ at the target environment, i.e., a graphical representation where nodes $\mathcal{V}$ correspond to configurations q_i of the vehicle. Edges $(q_i, q_j) \in \mathcal{E}$ exist between vertices as long as the application (of maximum duration $T_{\max}$) of the available controller $\pi(\mathbb{M}^{-1}(q_i), q_j)$ at a zero velocity state $\mathbb{M}^{-1}(q_i)$ of the initial configuration q_i brings the system within a hyperball $\mathcal{B}(q_j, \epsilon)$ of the configuration q_j.

The roadmap construction procedure (Fig 2a) samples first configurations $\{q_i\}_{i=1}^{|N|}$, which form the roadmap vertices $\mathcal{V}$. Here, the vertices are sampled over a grid in $\mathbb{Q}$. They are collision-checked and verified to be in $\mathbb{Q}_{\text{f}}$. Then, for the edges $\mathcal{E}$, the procedure selects a random $q_i \in \mathcal{V}$ and obtains two sets: $A(q_i)$ and $D(q_i)$. $A(q_i)$ is the set of vertices $q \in \mathcal{V}$, whose hyperballs $\mathcal{B}(q, \epsilon)$ can be reached from $x_i = \mathbb{M}^{-1}(q_i)$ given π. Similarly, $D(q_i)$ is the set of vertices $q \in \mathcal{V}$, where the state $x = \mathbb{M}^{-1}(q)$ can reach the hyperball $\mathcal{B}(q_i, \epsilon)$ given π. Then, edges from q_i to vertices in $A(q_i)$ and edges from vertices in $D(q_i)$ to q_i are added to the set $\mathcal{E}$.



The application of the controller for the edge construction, especially a learned one for a non-linear dynamical system, implies that for all edges (q_i, q_j) in the roadmap, there is no guarantee that the resulting configuration q_j can be achieved exactly from q_i. Instead, the corresponding edge only guarantees that the system will be within a hyperball $\mathcal{B}(q_G, \epsilon)$ if it is initialized at $\mathbb{M}^{-1}(q_i)$. Consequently, the edges of the roadmap introduce **"gaps"** as highlighted by the figure. This issue is exacerbated if the controller is used to follow a sequence of edges on the roadmap, e.g., (q_i, q_j) and (q_j, q_k). Since the controller's application over the first edge can only bring the system in the vicinity of q_j, the application of the controller corresponding to the second edge at the resulting configuration may not even bring the system within an ϵ-hyperball of q_k. Consequently, paths on the "Roadmap with Gaps" do not respect the dynamic constraints of the vehicle and cannot be used to directly solve kinodynamic planning problems. They can still provide, however, useful guidance for a kinodynamic planner.

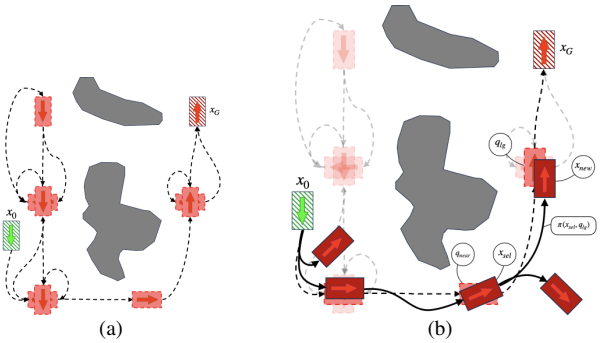


Fig. 2: (a) A "Roadmap with Gaps" consists of vertices (configurations shown as dotted boxes). The roadmap's directional edges (dotted lines) correspond to where the controller executed from the source successfully reached the vicinity of the target. For a new query (x_0, x_G), the start q_0 and goal q_G are added to the roadmap (transparent). (b) During the expansion of the planning tree (opaque), RoGuE selects informed local goals q_{lg} for the controller π. The closest roadmap node q_{near} to the selected tree node x_{sel} is identified. Its roadmap successor is passed as the local goal q_{lg} to the controller. This expansion adds a new tree node x_{new}.

C. RoGuE: Guiding expansion via a roadmap with gaps

Algo. 2 describes the **Ro**admap-**Gu**ided **E**xpansion (RoGuE)-Tree method, which uses the "Roadmap with Gaps" to expand online an AO Tree Sampling-based Motion Planner, given a new query (x_0, x_G). It first adds vertices $q_0 = \mathbb{M}(x_0)$ and $q_G = \mathbb{M}(x_G)$ to the roadmap and connects them appropriately, i.e., it adds edges from q_0 to the set $A(q_0)$ and edges from the set $D(q_G)$ to q_G.

Then, it computes a *wavefront* $\mathcal{W} : \mathcal{V} \mapsto \mathbb{R}^+$ over the roadmap. Initially, all vertices have an infinite value but the goal, where $\mathcal{W}(q_G) = 0$. Then, a backward search computes the value of every node according to $\mathcal{W}(v) = min_{\mathcal{V}}\{c(v, v') + \mathcal{W}(v') \mid \in \mathcal{E}\}$, where $c(v, v')$ is the cost of edge (v, v'), i.e., the duration required to reach the vicinity of v' from v given π. This also allows to compute the Successor(v) of a node, which is the out-neighbor that defines the wavefront value of v. For a node with an infinite value at the end of the process, its successor is undefined.

Algorithm 2: RoGuE-Tree $(\mathbb{X}, \mathbb{U}, x_0, x_G, \pi, (\mathcal{V}, \mathcal{E}))$

```
Add q_0, q_G to roadmap (V, E) as start and goal;
W ← GET-WAVEFRONT((V, E));
T ← {x_0};
while termination condition is not met do
    x_sel ← SELECT-NODE(T);
    u ← RoGuE(x_sel, V, W, π);
    x_new ← PROPAGATE(x_sel, u);
    if (x_sel → x_new) ∈ X_f then
        EXTEND-TREE(T, x_sel → x_new);
```

Then, a tree T data structure is expanded by selecting a tree node x_{sel}, propagating a control u from x_{sel} and generating a new tree node x_{new}. For the selection, and following the DIRT planner [4], when the cost-to-go value of the roadmap node closer to x_{new} is lower than that of its parent's, then x_{new} is selected for expansion at the consecutive step so the tree can make progress along a promising path.

The RoGuE function selects the control u and is detailed in Algo. 3 and Fig 2b). It uses the wavefront $\mathcal{W}$ to provide an informed local goal. The first time a tree node x_{sel} is selected, RoGuE identifies the closest roadmap node $q_{\text{near}} \in \mathcal{V}$ according to a distance function $d(\cdot,\cdot)$ (Line 2). It then queries the Successor(q_{near}). If defined, it is used as the local goal q_{lg} for the controller π (Lines 3-5). If the successor is undefined, a random local goal in $\mathbb{Q}$ is chosen (Line 6-7).

Algorithm 3: RoGuE $(x_{\text{sel}}, \mathcal{V}, \mathcal{W}, \pi)$

```
1  if first time x_sel is expanded then
2      q_near ← ClosestRoadmapNode(x_sel, V);
3      q_lg ← Successor(q_near, W);
4      if q_lg is defined then
5          u ← π(x_sel, q_lg);
6      else
7          u ← π(x_sel, Q.sample());
8  else
9      u ← U.random-sample();
10 return u;
```

Maintaining Asymptotic Optimality: When tree nodes have a non-zero probability of being selected, and the probability of expansion along the lowest-cost trajectory is non-zero, the planner is AO. RoGuE adopts the node selection process of DIRT [4], which, while informed, maintains a positive probability for all tree nodes. Similarly, while RoGuE initially applies informed expansions from each node, subsequent expansions apply random controls from the set $\mathbb{U}$ (Algo 3, Line 8-9) to retain AO properties [4].

IV. Experimental Evaluation

The **robot systems** considered in the evaluation are: (i) an analytically simulated second-order differential-drive, (ii) an analytically simulated car-like vehicle (where $\dim(\mathbb{X}) = 5, \dim(\mathbb{U}) = 2$), (iii) a MuSHR car [28] physically simulated on MuJoCo [25] ($\dim(\mathbb{X}) = 27, \dim(\mathbb{U}) = 2$), and (iv) a Skydio X2 Autonomous Drone ($\dim(\mathbb{X}) = 13, \dim(\mathbb{U}) = 4$). For the drone, the distance function $d(\cdot,\cdot)$ considers its center of mass 3D position. For all systems, the parameter ϵ is set to the same value 0.5. All planning experiments are implemented using the ML4KP software library [29] and executed on a cluster with Intel(R) Xeon(R) Gold 5220 CPU @ 2.20GHz and 512 GB of RAM.

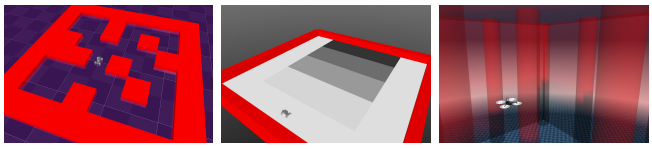


Fig. 3: Physically simulated benchmarks using MuJoCo. (L-R) Maze, Friction, Quadrotor.

Across experiments, the following **metrics** are measured: (1) Average normalized cost of solutions found over computation time; and (2) Ratio of problems solved over time. For all problems, the path cost is the solution plan's duration. To better reflect the performance of different methods and account for the difficulty of different problems, path costs reported are normalized by dividing by the best path cost ever found for a problem across all planners.

Two **comparison expansion functions** are considered in the evaluation: (a) Random uses a blossom expansion of random controls in $\mathbb{U}$, and (b) RLG samples Random Local Goals as input to π and outputs the controls returned. In terms of **comparison motion planners**, both an uninformed Rapidly-exploring Randomized Tree (RRT) [1] and the informed but AO DIRT are considered. For the Random and RLG expansion strategies of DIRT, a blossom of 5 controls is implemented.

Additional AO planners were considered for experimentation but it was difficult to achieve useful results with them. The *Bundle of Edges (BoE)* [24] approach failed to find solutions while using a similarly-sized roadmap. The *discontinuity-bounded A** (dbA*) [30] relies on motion primitives, which are not available, however, for the second-order robots considered here.

All (start, goal) pairs in the experiments are included as milestones during the roadmap construction to reduce connection time during online planning. These connections also lend themselves to multi-threading or alternatively the closest roadmap nodes can be used as surrogates for the actual start and goal states.

Analytically simulated benchmarks: The performance of the expansion functions is measured on 3 sets of planning benchmarks for the analytically simulated vehicular systems: (a) 8 problems in an environment with Narrow passages, (b) 6 problems in the Indoor environment from Arena-bench [31], and (c) 8 problems in the Warehouse environment from Bench-MR [32]. Figure 4 provides the numerical results. Across all benchmarks, for the differential drive and car-like dynamics, DIRT-RoGuE finds the lowest cost solutions. It is also the only expansion strategy that returns solutions in all trials. Among the RRT expansion strategies, RRT-RoGuE achieves the highest success rate and the lowest cost solutions.

Comparison to Path Following: A naïve alternative to the proposed solution is to use the configurations along a path on the roadmap as consecutive local goals for a path following controller. For the car-like system, a pose-reaching controller [33] was employed to drive the robot to a given pose $\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} k_\rho \rho \\ k_\alpha \alpha + k_\beta \beta \end{pmatrix}$, where each k parameter is a gain term, ρ and α are the distance and bearing to the local goal respectively, and β is the angle difference between α and the current angle. The controller is tested on paths retrieved from the Roadmap with Gaps for the benchmarks of Fig 4. Only 2 such executions, however, returned collision-free trajectories. This is due to (a) the environments containing multiple narrow passages and (b) the paths returned by the roadmap still contain "gaps" that the controller cannot easily negotiate.

Physically simulated benchmarks: The performance of the expansion functions is measured on the following benchmarks for MuSHR: (a) a Maze from D4RL [34], (b) uneven Terrain environment [35], (c) and a varying friction environment (Friction). The controller is trained in an empty environment with flat terrain and uniform friction.

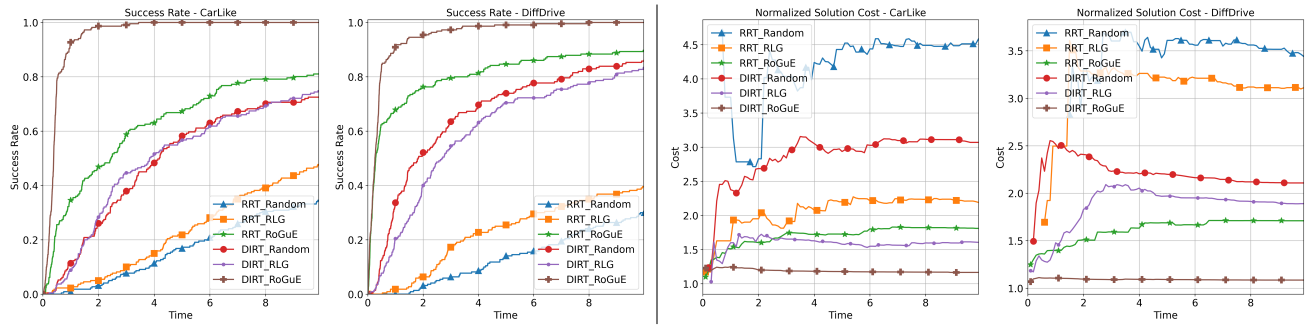
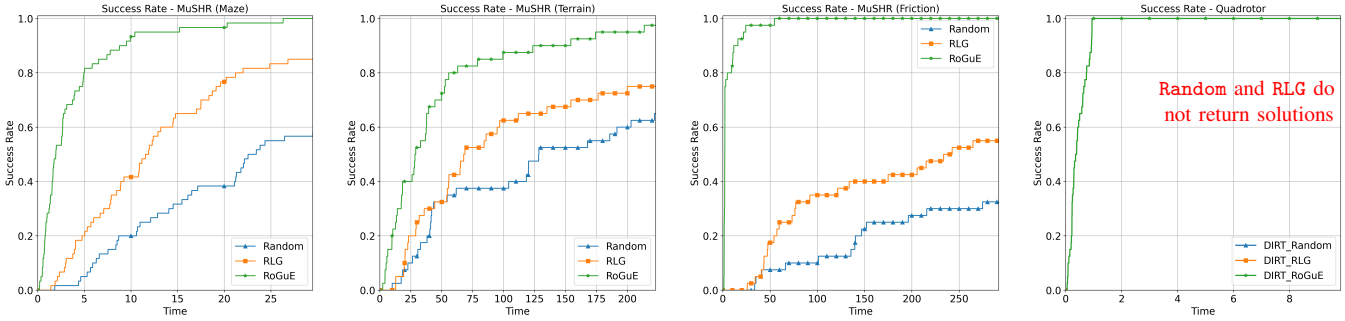
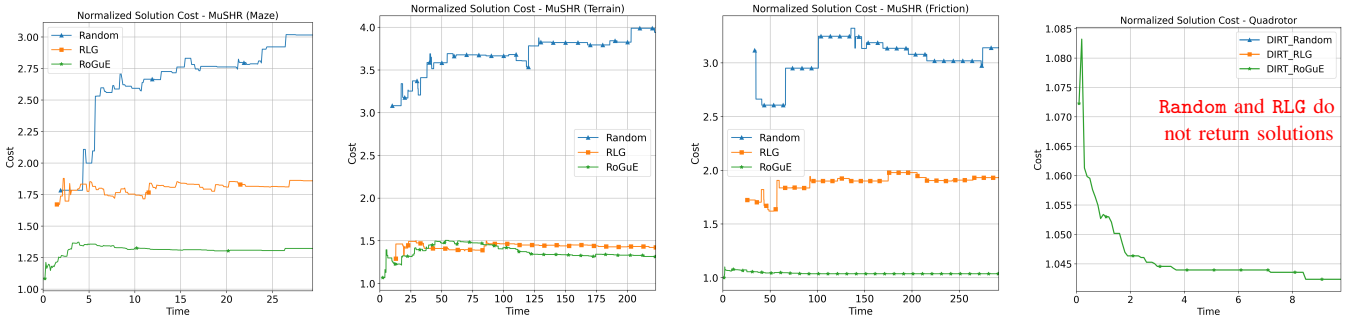


Fig. 4: Analytical benchmarks: Left & middle-left: higher success rate earlier is better. Middle-right & Right: lower path cost is better. As more solutions are discovered, the average cost may increase. Each problem instance is run 10 times to account for different seeds.



(a) Success rates of the planners on each physically simulated benchmark. Higher success rate earlier is better.



(b) Solution quality. Lower values are better. As more solutions are discovered, the average cost may increase over time.

Fig. 5: Physically simulated systems on MuJoC: Each instance is executed 10 times with different random seeds. For the `Quadrotor`, the `Random` and `RLG` strategies consistently fail to return solutions and hence their results are not displayed.

The roadmap captures the traversability of different parts of each environment using the controller. For the `Quadrotor` benchmark, the X2 drone must navigate an indoor environment with pillars and air pressure effects.

Figs. 1 and 3 visualize the different environments in MuJoCo, and Fig. 5 provides the experimental results. Only the DIRT motion planner is reported in these experiments as the RRT-based solutions cannot find a solution within the allotted times. Since calls to the MuJoCo engine are expensive, all expansion strategies use a blossom $k = 1$.

`RoGuE` finds the lowest cost solution across all benchmarks. `Random` and `RLG` consistently fail to find solutions but `RLG` returns better solutions relative to `Random`. In the `Maze` and `Friction` benchmarks, `RoGuE` finds solutions across trials quickly. `RoGuE` also returned the most solutions in the `Terrain` benchmark given the same planning budget. In the `Quadrotor` benchmark, `RoGuE` is the only expansion method that discovers any solutions given the tight placement of obstacles and the speed of the X2 drone.

Comparison to RL solutions: Table I evaluates two RL-based strategies trained on the benchmarks of Fig 5 in terms of success rate `Succ` and offline cost (`Offl` (# of calls made to the MuJoCo engine). The offline cost is reported when the best performance is achieved and success rate no longer improves. The online cost `Onl` for sampling-based planners is also reported. The online cost for the RL solutions is minimal. `SAC+HER` trains a goal-conditioned controller $u = \pi(x, q)$ directly in the planning environment. The approach achieves a low success rate, especially for the `Quadrotor` given the joint challenge of complex dynamics and obstacle avoidance. `H-SAC+HER` follows a hierarchical approach similar to `RoGuE` by training a policy to predict local goals for the controller to reach at every step, i.e., $q_{\text{lg}} = \phi(x)$. This slightly improves success rate relative to `SAC+HER` on `Quadrotor` but lowers it on `Friction`. This shows the difficulty of RL strategies in identifying informed local goals, which `RoGuE` is able to achieve.

	RoGuE			SAC+HER		H-SAC+HER		Random		RLG		
Benchmark	Offl	Onl	Succ	Offl	Succ	Offl	Succ	Onl	Succ	Offl	Onl	Succ
Friction	2.5M	58.15	**100%**	0.81M	35%	1.48M	13%	295.225	37.5%	1M	609.225	60%
Maze	2.05M	241.62	**100%**	0.54M	34%	1.74M	31%	623.57	58%	1M	693.73	87%
Quadrotor	1M	736.3	**100%**	1.3M	5%	1.01M	13%	14k	0%	100k	13.7k	0%

TABLE I: Physically-simulated benchmarks: RoGuE vs RL in terms of computation cost (# of calls to physics engine) and success rate.

V. Conclusion

This paper proposes a strategy that can benefit from learned controllers to improve the efficiency of kinodynamic planning for robots with significant dynamics. It utilizes a controller trained offline in an empty environment. The target environment is represented via a "Roadmap with Gaps" over local regions and applications of the controller between them. Given a wavefront over the roadmap for a specific goal, a tree sampling-based motion planner generates informed subgoals and uses the controller to reach them. When the controller cannot reach a subgoal, the planner resorts to random exploration. Evaluation shows the significant improvement in planning efficiency.

For higher-dimensional systems, the memory requirements of the roadmap can be improved by considering sparse representations [36]. Furthermore, learned reachability estimators can assist in efficient roadmap construction and online queries. This work assumes an accurate model of the environment and the robot, which complicates deployment on real systems. This motivates integrating the proposed motion planner with system identification, state estimation and feedback control to track the planned trajectory.

References

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

[2] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *IJRR*, 2016.

[3] K. Hauser and Y. Zhou, "Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space," *T-RO*, 2016.

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

[5] M. Kleinbort, E. Granados, K. Solovey, R. Bonalli, K. E. Bekris, and D. Halperin, "Refined analysis of asymptotically-optimal kinodynamic planning in the state-cost space," in *ICRA*, 2020.

[6] A. Faust, O. Ramírez, M. Fiser, K. Oslund, A. Francis, J. O. Davidson, and L. Tapia, "PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," *ICRA*, 2018.

[7] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "RL-RRT: Kinodynamic motion planning via learning reachability estimators from RL policies," *RA-L*, vol. 4, no. 4, pp. 4298–4305, 2019.

[8] T. McMahon, A. Sivaramakrishnan, E. Granados, and K. E. Bekris, "A survey on the integration of machine learning with sampling-based motion planning," *Foundations and Trends in Robotics*, 2022.

[9] Y. Li, R. Cui, Z. Li, and D. Xu, "Neural network approximation based near-optimal motion planning with kinodynamic constraints using RRT," *IEEE Transactions on Industrial Electronics*, vol. 65, pp. 8718–8729, 2018.

[10] G. P. Kontoudis and K. G. Vamvoudakis, "Kinodynamic motion planning with continuous-time q-learning: An online, model-free, and safe navigation framework," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 12, pp. 3803–3817, 2019.

[11] J. J. Johnson, L. Li, F. Liu, A. H. Qureshi, and M. C. Yip, "Dynamically constrained motion planning networks for non-holonomic robots," in *IROS*, 2020, pp. 6937–6943.

[12] L. Li, Y. Miao, A. H. Qureshi, and M. C. Yip, "Mpc-mpnet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints," *RA-L*, vol. 6, no. 3, 2021.

[13] Z. Xu, B. Liu, X. Xiao, A. Nair, and P. Stone, "Benchmarking reinforcement learning techniques for autonomous navigation," in *ICRA*, 2023.

[14] B. Eysenbach, R. R. Salakhutdinov, and S. Levine, "Search on the replay buffer: Bridging planning and reinforcement learning," *NeurIPS*, 2019.

[15] S. Emmons, A. Jain, M. Laskin, T. Kurutach, P. Abbeel, and D. Pathak, "Sparse graphical memory for robust planning," *NeurIPS*, 2020.

[16] A. Bagaria and G. Konidaris, "Option discovery using deep skill chaining," in *ICLR*, 2019.

[17] A. Bagaria, J. K. Senthil, M. Slivinski, and G. Konidaris, "Robustly learning composable options in deep reinforcement learning." in *IJCAI*, 2021.

[18] A. Bagaria, J. K. Senthil, and G. Konidaris, "Skill discovery for exploration and planning using deep skill graphs," in *International Conference on Machine Learning*. PMLR, 2021, pp. 521–531.

[19] R. Strudel, R. Garcia, J. Carpentier, J.-P. Laumond, I. Laptev, and C. Schmid, "Learning obstacle representations for neural motion planning," in *CoRL*, 2020.

[20] A. Sivaramakrishnan, E. Granados, S. Karten, T. McMahon, and K. E. Bekris, "Improving kinodynamic planners for vehicular navigation with learned goal-reaching controllers," in *IROS*, 2021.

[21] T. McMahon, A. Sivaramakrishnan, K. Kedia, E. Granados, and K. E. Bekris, "Terrain-aware learned controllers for sampling-based kinodynamic planning over physically simulated terrains," in *IROS*, 2022.

[22] D. Le and E. Plaku, "Guiding sampling-based tree search for motion planning with dynamics via probabilistic roadmap abstractions," in *IROS*, 2014.

[23] M. G. Westbrook and W. Ruml, "Anytime kinodynamic motion planning using region-guided search," in *IROS*, 2020.

[24] R. Shome and L. E. Kavraki, "Asymptotically optimal kinodynamic planning using bundles of edges," in *ICRA*, 2021.

[25] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*, 2012.

[26] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018.

[27] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. H. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *NeurIPS*, 2017.

[28] S. S. Srinivasa, P. Lancaster, J. Michalove, M. Schmittle, C. Summers, M. Rockett, J. R. Smith, S. Chouhury, C. Mavrogiannis, and F. Sadeghi, "MuSHR: A low-cost, open-source robotic racecar for education and research," *CoRR*, vol. abs/1908.08031, 2019.

[29] E. Granados, A. Sivaramakrishnan, T. McMahon, Z. Littlefield, and K. E. Bekris, "Machine learning for kinodynamic planning (ml4kp)," https://github.com/PRX-Kinodynamic/ML4KP, 2021–2022.

[30] W. Hönig, J. Ortiz-Haro, and M. Toussaint, "db-a*: Discontinuity-bounded search for kinodynamic mobile robot motion planning," in *IROS*, 2022.

[31] L. Kästner, T. Bhuiyan, T. A. Le, E. Treis, J. Cox, B. Meinardus, J. Kmiecik, R. Carstens, D. Pichel, B. Fatloun, *et al.*, "Arena-bench: A benchmarking suite for obstacle avoidance approaches in highly dynamic environments," *RA-L*, vol. 7, no. 4, 2022.

[32] E. Heiden, L. Palmieri, L. Bruns, K. O. Arras, G. S. Sukhatme, and S. Koenig, "Bench-mr: A motion planning benchmark for wheeled mobile robots," *RA-L*, vol. 6, no. 3, 2021.

[33] P. I. Corke, W. Jachimczyk, and R. Pillat, *Robotics, vision and control: fundamental algorithms in MATLAB*. Springer, 2011, vol. 73.

[34] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," 2020. [Online]. Available: https://arxiv.org/abs/2004.07219

[35] K. Stachowicz, A. Bhorkar, D. Shah, I. Kostrikov, and S. Levine, "FastRLAP: A System for Learning High-Speed Driving via Deep RL and Autonomous Practicing," 2023.

[36] A. Dobson and K. E. Bekris, "Sparse roadmap spanners for asymptotically near-optimal motion planning," *IJRR*, 2014.