Towards Benchmarking Sampling-Based Kinodynamic Motion Planners with ML4KP

Edgar Granados*, Aravind Sivaramakrishnan*, and Kostas E. Bekris.

***Abstract*— This work introduces a benchmarking framework for sampling-based kinodynamic planning algorithms, built on top of ML4KP, a C++ library for efficient kinodynamic planning that allows to incorporate learned primitives for planning purposes. ML4KP is fast, has minimal dependencies, and provides implementations of state-of-the-art kinodynamic planners that can be used with minimal configuration for the included systems. In contrast to other motion planning libraries, ML4KP allows to directly generate solutions for dynamic systems, without the need for additional post-processing of obtained solution paths. ML4KP is intended to be used by the robot learning community that seek to use sampling-based motion planners, as well as members of the motion planning community interested in integrating machine learning tools. Project website: https://sites.google.com/scarletmail.rutgers.edu/ml4kp/**

I. Introduction

Motion planning for robotic systems with significant dynamics is challenging. Often, there is no local planner available, and the only primitive to explore the state space is forward propagation of controls. Tree sampling-based motion planners (SBMPs) have been developed [1], some of which achieve asymptotic optimality (AO) [2]–[5] by propagating random controls during each iteration.

Creating an efficient motion planning pipeline for a new robot or a dynamical system requires effectively evaluating different planning components and framework. The implementation of these planners must therefore be efficient, interpretable, and reliable. Similarly, if the user is prototyping a new planner, or testing a new implementation of a planner's components (like a distance function), it is useful to quickly benchmark its performance against the state-of-the-art. To test the generalization capability of the new approach, the planner must be applicable to different dynamical systems across a variety of clear and well-defined motion planning benchmarks.

To the best of the authors' knowledge, however, many efforts in the related literature rely on either outdated or suboptimal implementations of AO kinodynamic planners, which limits the ability to effectively evaluate performance. Many research papers also define their own benchmarks, thus making it difficult to judge the performance of new methods in a standardized manner. This extended abstract describes an initial effort to address these shortcomings. It proposes a benchmarking framework built on top of ML4KP, a lightweight, fast software library built by the authors with a focus on integrating sampling-based kinodynamic planners with machine learning primitives.

*The first two authors contributed equally to this work. The authors are with the Dept. of Computer Science, Rutgers University, NJ USA 08854. E-mail: {eg585, as2578, kb572}@rutgers.edu

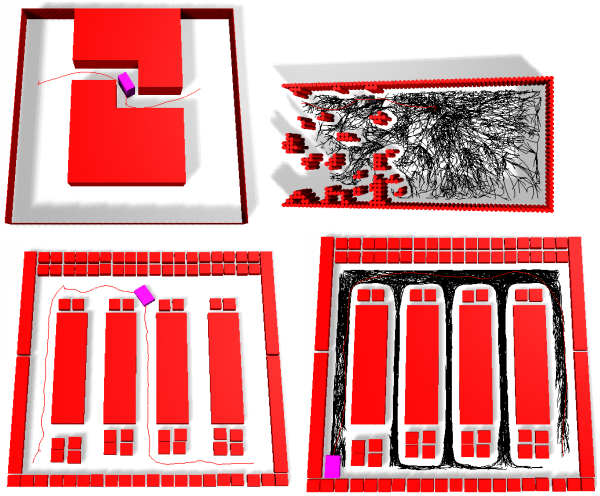

Fig. 1: (Left) Solution paths for different vehicular systems and environments. (Right) Trees for AORRT and DIRT planners.

Existing motion planning benchmarks, such as BARN [6], Bench-MR [7], and MOTIONBENCHMAKER [8] are built on top of existing motion planning libraries, such as the Open Motion Planning Library (OMPL) [9] and MoveIt! [10]. Although they are well suited for classical *kinematic* motion planning problems, they lack features useful for kinodynamic planning. OMPL and MoveIt! do not focus on computing time-optimal kinodynamic trajectories, and do not use execution duration as a cost function to be minimized. The primary design choice in OMPL is to plan for kinematic systems, treating robot dynamics as an extension. ML4KP, on the other hand, is built for systems with significant dynamics.

In addition, there has been significant interest recently in applying machine learning techniques to improve the practical efficiency of motion planning algorithms. ML4KP was designed with the intention of integrating machine learning with motion planning. The modular and functional structure of ML4KP allows a user to integrate learned components into the planner alongside traditional implementations of existing methods.

II. Preliminary Material

A. Notation

Let $\mathbb{X} \subset \mathbb{R}^n$ be the *state space*, $\mathbb{X}_{free} \subset \mathbb{X}$ the obstacle-free space and the obstacle space $\mathbb{X}_{obs}$ the complement of $\mathbb{X}_{free}$, i.e. $\mathbb{X}_{obs} = \mathbb{X} \setminus \mathbb{X}_{free}$. The *control space* is $\mathbb{U} \subset \mathbb{R}^m$. Given the *state* at time t $x(t) \in \mathbb{X}$ and a

control $u(t) \in \mathbb{U}$, a time-invariant *dynamical system* has the form $\dot{x}(t) = f(x(t), u(t))$ with some initial state $x(0)$. A *plan* Υ is the sequence of controls $u(0), u(1), \ldots, u(T-1)$ executed in order, inducing a valid *trajectory* $\pi : [0, T] \mapsto \mathbb{X}_{free}$. The *cost* of a trajectory is given by $\texttt{cost}(\pi) = \int_0^T g(\pi(t), \Upsilon(t))dt$ where $g : \mathbb{X} \times \mathbb{U} \mapsto \mathbb{R}^+$ is a cost derivative. `cost` has to be a monotonically increasing function.

B. Kinodynamic Motion Planning

Given a dynamical system at some initial state $x(0) \in \mathbb{X}_{free}$, a *goal region* $\mathbb{X}_G \subset \mathbb{X}_{free}$, the *feasible* kindoynamic motion planning problem is to find a plan of some duration T such that $\pi(T) \in \mathbb{X}_G$. That is, a trajectory is found that drives the system from the initial state to the goal region. An algorithm is *probabilistically complete* if the probability of finding a solution (if one exists) converges to 1 as time budget allowed goes to infinity.

The *optimal* kinodynamic motion planning problem is a special case of the feasible problem, where the solution plan Υ^* has the minimal $\texttt{cost}(\pi)$ over the set of all possible plans. An algorithm is said to be *asymptotically optimal* (AO) if the cost of the solution plan converges to $\texttt{cost}(\Upsilon^*)$ as time goes to infinity.

III. ML4KP Architecture

`ML4KP` consists of two core modules: **Simulation** and **Planning**. These modules interact together to implement different dynamical systems and kinodynamic planners.

A. Ground Truth Simulation

The fundamental abstraction in the simulator is the *prx::system_t* (system) class. The system class uses the *prx::space_t* (space) abstraction, which provides a way for users to define abstract spaces (i.e., state space or control space). This also allows to deal with non-Euclidean spaces. Each space dimension can be Euclidean, rotational, or discrete. A *prx::space_point_t* is created by a space, and can represent states or controls in the system class. Every system has a state and an input control space associated with it.

The system class also implements a *compute control* method that receives a control and potentially manipulates it. The *propagate* functionality implements the forward dynamics propagation for a system according to its dynamics. This is achieved either through numerical methods, or through a black-box physics engine.

The *prx::plant_t* (plant) class extends the abstract system and associates it with a set of rigid bodies, each with their own geometric configuration. Additionally, for systems simulated using analytical expressions of their dynamics, a definition for the *compute_derivative* function is needed to compute the forward propagation and update the geometries for the resulting state.

Finally, the *prx::world_model_t* (world model) class is used to represent the physical world as it is observed or known to the motion planner. The world model can be used to define multiple *contexts* for the same physical world. Each context consists of a *prx::system_group_t* (system group) and a *prx::collision_group_t* (collision group). Collision detection is implemented using PQP [11], [12].

B. Functional Motion Planning

A planner contains the main logic of the algorithm while the *planner specification* includes parameters as well as functions that have a default implementation. The functions are called during the execution of the planner, modifying the operation of the algorithm. Finally, the *planner query* specifies the parameters and functions that define the planning problem and determine its solution.

Algorithm 1: `Tree-SBMP`

```
1  Initialize planner tree rooted at start state
2  while stopping criterion is not met do
3      Select a node on the planner tree to expand
4      Compute a control to be applied from that state
5      Propagate the dynamics to obtain a trajectory
6      if trajectory is valid then
7          Add trajectory as an edge to the tree
8          if trajectory terminates in goal region then
9              Mark a solution has been found
10         end
11     end
12 end
```

The generic tree sample-based motion planning algorithmic framework is shown in Algorithm 1. Line 4-5 together form a planner's `expand` function, with the control computed via its `sample_plan` function. Line 9 uses a planner's `goal_check` implementation to check whether the end state of a trajectory fulfills the goal criteria. Highly configurable algorithms are achieved by defining key parts of the algorithms as functions - either as auxiliary functions, or specified through the planner's query and specification.

Planner	Memory	Parameters	Informed
`AORRT [5]`	++	−	✗
`SST [2]`	+	++	✗
`DIRT [4]`	+	−	✓

TABLE I: AO planners implemented in `ML4KP` and their properties. *Memory* indicates the memory usage of each planner for high-dimensional planning problems. *Parameters* indicates the amount of parameter tuning for finding good solutions. *Informed* planners make use of guidance, e.g. in the form of a heuristic.

`ML4KP` has optimized implementations of the AO motion planning algorithms shown in table I. These algorithms can be executed *out of the box* with minimum parameter specifications. By changing appropriate functions, additional functionality can be provided. For instance,

- A strategy for propagating multiple controls out of a selected states, similar to `RRT-Blossom` [13], can be implemented by changing the implementations of the `expand` function.

- A cost map over the planning environment can be used to specify costs [14] inside the `cost` function of the planning problem.
- A custom `goal_check` can be used to guarantee the convergence of the system [15].

C. Benchmark Definitions

`ML4KP` provides a simple interface to benchmark AO kinodynamic planners on a new planning problem for a dynamical system. It is assumed that for a given dynamical system, the equations that govern its motion and configurations of its underlying rigid bodies are fixed. The following parameters need be to be specified as part of each benchmark.

- **State and Control space bounds**: minimum and maximum values of each variable that describes the system's dynamics.
- **Start and Goal states**: initial condition of the system and a desired goal, that are guaranteed to be within the state space bounds.
- **Minimum and Maximum propagation step size**: After computing a control (line 4 of Alg 1), the propagation duration will typically be uniformly at random sampled from these bounds.
- **Distance function**: While selecting the next node to expand (line 3 of Alg 1), all AO planners use a nearest neighbor query that calls a distance function for the system. For informed planners (like `DIRT`), a **heuristic** function may be defined as well.
- **Planner-specific parameters** like the blossom number for `DIRT`, or selection radius for `SST`.
- **Goal check function**: A function that returns true iff. the state passed as the argument satisfies the desired goal criteria.
- **Environment file**: A file that specifies the dimensions and configurations of the static obstacles in the planning environment. `ML4KP` uses the *YAML* library to define different planning environments.

IV. Experiments with ML4KP

The tools provided by `ML4KP` allow the users to compare various kinodynamic planners on different scenarios with alternative evaluation criteria. This section describes experiments performed with `ML4KP` and their results. The software is available in the `ML4KP` GitHub repository.

A. Planners and Evaluation Metrics

Following related work [16], the following statistics are reported for every planner:

i) Success Rate (p): The ratio of trials where a solution was found within the planning budget,
ii) Average time taken to find the solution across trials where a solution was found (t^{st}),
iii) Average cost of the first found solution across all trials where a solution was found (J^{st}),
iv) Cost of the final solution found within the planning budget (J^{f}), averaged across all trials where a solution was found.

On some benchmark environments, multiple starts and goals may be sampled to test the robustness of the planner. To account for the difficulty of different planning problems, path costs can be normalized by dividing by the best path cost found for a problem across any planner. Similarly, for reporting average time / planning iterations taken to found a solution, the normalization factor can be the maximum time/number of iterations required by any planner to find its first solution. Other metrics that can be measured include the number of nodes in the tree as a function of time/number of iterations, number of invalid states encountered, etc.

B. Dynamical systems and Environments

Each planner's performance is evaluated on the following proposed benchmarks:

i) First-order ($dim(\mathbb{X}) = 3$) and second-order ($dim(\mathbb{X}) = 5$) unicycles ($dim(\mathbb{U}) = 2$, Eqs (13.18) and (13.46) from [17]) are evaluated on the kinodynamic motion planning benchmark originally presented in earlier work [16].
ii) First-order ($dim(\mathbb{X}) = 3$) and second-order ($dim(\mathbb{X}) = 5$) differential drive vehicles ($dim(\mathbb{U}) = 2$) are evaluated on the `Warehouse` polygon environment from BenchMR [7].
iii) A first order omnidirectional robot with Mecanum wheels (developed in [18]) is evaluated on the most difficult `BARN` environment [6].
iv) A two-link acrobot ($dim(\mathbb{X}) = 4, dim(\mathbb{U}) = 1$, from [19]) is tasked with swinging up to the upright pose in the presence of obstacles (`Boxes`).

Solutions found by the different planners on each of the benchmark classes are visualized in Fig 2. The planning results are presented in Table II. For the vehicular systems, the `DIRT` algorithm uses the default implementation of a heuristic, which is the 2D Euclidean distance divided by the maximum velocity. Although this heuristic is admissible, it may lead to suboptimal performance.

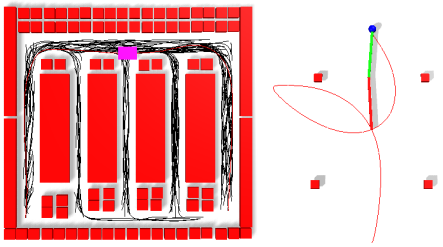

Fig. 2: The ML4KP library includes vehicular and non-vehicular systems. (Left) An example of an SST tree for a vehicular system. The acrobot (right) is a challenging system for low torque values within the presence of obstacles

C. Integration with machine learning primitives

Beyond evaluating traditional properties important for SBMPs (fast, memory-efficient, anytime, probabilistically complete and asymptotic optimal implementations), `ML4KP` is also designed to leverage the practical benefits obtained by integrating machine learning tools. The functional design of `ML4KP` allows users to wrap machine learning methods

#	System	Instance	SST				AO-RRT 2				DIRT			
			p	$t^{st}[s]$	$J^{st}[s]$	$J^{f}[s]$	p	$t^{st}[s]$	$J^{st}[s]$	$J^{f}[s]$	p	$t^{st}[s]$	$J^{st}[s]$	$J^{f}[s]$
1	Unicycle (FO)	`Kink` (60)	0.1	0.77	38.9	38.9	**1.0**	**1.38**	**40.71**	40.07	0.9	8.87	45.28	**37.08**
2	Unicycle (SO)	`Kink` (300)	0.8	76.15	68.91	67.9	1.0	0.441	48.32	46.67	1.0	6.33	55.71	39.22
3	DiffDrive (FO)	`Bench-MR` (60)	1.0	5.74	**97.22**	**46.7**	**1.0**	**0.04**	106.98	105.15	1.0	0.91	101.82	69.32
4	DiffDrive (SO)	`Bench-MR` (60)	1.0	6.23	**48.49**	**39.55**	**1.0**	**0.18**	49.61	49.43	1.0	4.69	66.49	60.21
5	Omnirobot (FO)	`BARN` (10)	1.0	1.98	15.39	11.14	**1.0**	**0.31**	**11.79**	10.84	1.0	1.53	14.66	**8.71**
6	Acrobot	`Boxes` (60)	0.4	10.69	3.14	3.13	**1.0**	**0.04**	**1.58**	**1.33**	0.6	21.32	3.43	3.43

TABLE II: Benchmarking results comparing each planner (SST, AO-RRT 2 and DIRT) on the metrics described in Section IV. Best values are highlighted in bold. For each instance, the maximum planning time (in seconds) is indicated within parentheses.

and use them without changing the underlying planner's source code (for e.g., using a learned distance function or a heuristic).

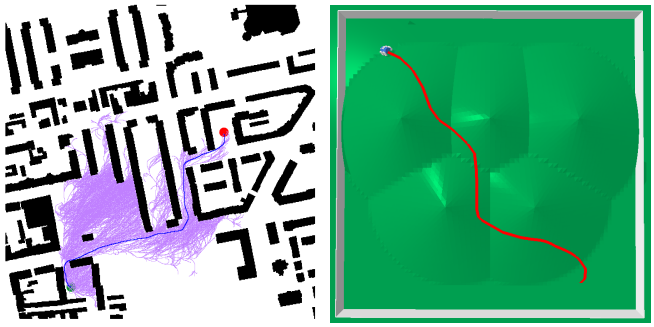

Fig. 3: Solutions paths found by the DIRT algorithm for: (left) DiffDrive (SO) system on a map of Berlin from [20], and (right) a physically simulated Segway navigating a complex terrain. In both approaches, an informed machine-learning based node expansion is integrated in `ML4KP` to find high-quality (low cost) solutions fast.

Fig 3 illustrates an example of an implementation of the `DIRT` algorithm where a controller trained in a flat, obstacle-free environment computes the control after a node is selected for expansion. In environments with obstacles or terrain features, informed local goals are generated as input to the controller [21], [22].

V. Discussion and Future Work

`ML4KP` provides a fast, lightweight framework to benchmark state-of-the-art sampling-based kinodynamic planners on new problems, characterized by a dynamical system operating in an environment. Ongoing work includes the benchmarking of SBMPs on high-dimensional dynamical systems, including physics simulated robots. Further development is also required for considering planning environments with dynamic obstacles, as well as dynamical systems with epistemic and/or aleatoric uncertainty.

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*, vol. 35, no. 5, pp. 528–564, 2016.

[3] K. Hauser and Y. Zhou, "Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space," *T-RO*, vol. 32, no. 6, pp. 1431–1443, 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] D. Perille, A. Truong, X. Xiao, and P. Stone, "Benchmarking metric ground navigation," in *2020 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, 2020.

[7] 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.

[8] C. Chamzas, C. Quintero-Pena, Z. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki, "Motionbenchmaker: A tool to generate and benchmark motion planning datasets," *RA-L*, vol. 7, no. 2, pp. 882–889, 2021.

[9] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.

[10] D. Coleman, I. A. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," *CoRR*, vol. abs/1404.3785, 2014. [Online]. Available: http://arxiv.org/abs/1404.3785

[11] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 171–180.

[12] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," Technical Report TR99-018, Department of Computer Science, University of ..., Tech. Rep., 1999.

[13] M. Kalisiak and M. van de Panne, "Rrt-blossom: Rrt with a local flood-fill behavior," in *ICRA*, 2006.

[14] L. Jaillet, J. Cortes, and T. Simeon, "Transition-based rrt for path planning in continuous cost spaces," in *IROS*, 2008.

[15] E. R. Vieira, E. Granados, A. Sivaramakrishnan, M. Gameiro, K. Mischaikow, and K. E. Bekris, "Morse graphs: Topological tools for analyzing the global dynamics of robot controllers," *WAFR*, 2022.

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

[17] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[18] E. Granados, A. Boularias, K. Bekris, and M. Aanjaneya, "Model identification and control of a low-cost mobile robot with omnidirectional wheels using differentiable physics," in *ICRA*, 2022.

[19] M. W. Spong, "The swing up control problem for the acrobot," *IEEE control systems magazine*, vol. 15, no. 1, pp. 49–55, 1995.

[20] N. Sturtevant, "Benchmarks for grid-based pathfinding," *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144 – 148, 2012. [Online]. Available: http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf

[21] 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.

[22] T. McMahon, A. Sivaramakrishnan, K. Kedia, E. Granados, and K. E. Bekris, "Improving kinodynamic planners for vehicular navigation with learned goal-reaching controllers," in *IROS*, 2022.