Kinodynamic Sampling Based Motion Planning with Massive Parallelism

by

Nicolas Perrault

B.A.,University of Colorado Boulder, 2024

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Science

Department of Aerospace Engineering

2025

Committee Members:

Morteza Lahijanian, Chair

Nisar Ahmed

Eric Frew

Perrault, Nicolas (MS., Aerospace Engineering)

Kinodynamic Sampling Based Motion Planning with Massive Parallelism

Thesis directed by Prof. Morteza Lahijanian

Sampling-based motion planners (SBMPs) are effective for planning with complex kinodynamic constraints in high-dimensional spaces, but they still struggle to achieve **real-time** performance, which is mainly due to their serial computation design. In this thesis we introduce two highly parallel kinodynamic SBMP algorithms designed explicitly for many-core architectures such as GPUs.

Our first algorithm, *Kinodynamic Parallel Accelerated eXpansion* (*Kino-PAX*), rapidly solves initial motion planning problems, whereas our second algorithm, *Asymptotically Near-Optimal Kinodynamic Parallel Accelerated eXpansion* (*Kino-PAX**), targets finding near-optimal trajectories over extended planning durations. Both algorithms concurrently grow a tree of trajectory segments by decomposing the iterative growth process into three massively parallelizable subroutines. We align the design closely with GPU execution hierarchies, through ensuring that threads are largely independent, share equal workloads, and take advantage of low-latency resources while minimizing high-latency data transfers and process synchronizations. This design results in a very efficient GPU implementation. We prove that Kino-PAX is probabilistically complete and analyze its scalability with compute hardware improvements. We also prove that Kino-PAX* achieves probabilistic δ-robust completeness and asymptotic δ-robust near-optimality.

Empirical evaluations demonstrate solutions in the order of 10 ms on a desktop GPU and in the order of 100 ms on an embedded GPU, representing up to 1000$\times$ improvement compared to coarse-grained CPU parallelization of state-of-the-art sequential algorithms over a range of complex environments and systems. In addition, *Kino-PAX** finds high-quality solutions within approximately 10 ms and can also be tuned effectively to converge towards near-optimal solutions over extended planning periods.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

Autonomous robotic systems are increasingly deployed in dynamic environments, requiring fast, reactive motion planning that accounts for the robot's complex kinematics and dynamics. Solving the kinodynamic motion planning problem **quickly** is critical for ensuring both functionality and safety. *Sampling-based motion planners* (SBMPs) have proven effective for various difficult problems, such as complex dynamics [77, 100, 102, 59, 108], complex tasks [9, 83, 71], and stochastic dynamics [79, 34, 35]. Nevertheless, they are typically designed for serial computation, limiting their speed to CPU clock rate. While recent methods can find solutions within seconds for simple systems and tens of seconds for complex ones [77, 100, 102], this is insufficient for **real-time** reactivity. Given the plateau in improvements to serial computation and CPU clock speeds, parallel devices like GPUs offer promising speedups. However, current SBMP algorithms are inherently sequential and inefficient when parallelized. In this work, we aim to enable real-time motion planning for complex and high-dimensional kinodynamical systems by exploiting the parallel architecture of GPU-like devices.

The objective of this work is to present two kinodynamic motion planning methods designed to efficiently leverage highly parallel processors, thereby enabling **real-time** and **low-cost** solutions for high-dimensional and complex problems.

First, we introduce *Kino-PAX*, a highly parallel kinodynamic SBMP, designed to efficiently find solution trajectories in **real-time**. *Kino-PAX* grows a tree of trajectory segments directly in parallel. Our key insight is that the iterative tree growth process can be decomposed into three

massively parallel subroutines. We design *Kino-PAX* to align with the parallel execution hierarchy of these devices, ensuring that threads are largely independent, share equal workloads, and take advantage of low-latency resources while minimizing high-latency data transfers and process synchronizations. We provide an analysis of *Kino-PAX*, showing that it is probabilistically complete. We also demonstrate, through several benchmarks, that *Kino-PAX* is robust to changes in hyperparameters and scalable to large dimensional systems.

Our contributions with *Kino-PAX* are four-fold: (i) *Kino-PAX*, a highly parallel kinodynamic SBMP designed to leverage the parallel architecture of GPU-like devices, (ii) a discussion of *Kino-PAX*'s hyperparameters and its efficient application to highly parallel devices, (iii) a thorough analysis and proof of probabilistic completeness, and (iv) benchmarks showing the efficiency and efficacy of *Kino-PAX* for complex and high-dimensional dynamical systems. Our results show that *Kino-PAX* achieves up to three-orders-of-magnitude improvement in computation time compared to our baselines, which use CPU parallelization. In all evaluated problems, *Kino-PAX* finds solutions in the order of 10 milliseconds, representing significant progress in enabling real-time kinodynamic motion planning.

Building on the work of *Kino-PAX*, we introduce *Kino-PAX**, a **near-optimal**, massively parallel, kinodynamic motion planning algorithm. *Kino-PAX** executes a parallel search through the state space, continuously identifying and refining solution trajectories throughout the planning process. It restructures the classical dynamic programming subroutines of near-optimal SBMPs—node selection, node extension, and node pruning—into three highly parallel subroutines. Through extensive experiments, we demonstrate that *Kino-PAX** is capable of achieving high-quality initial solutions in real-time and can be tuned to refine those solutions over an extended runtime.

Specifically, our contributions with *Kino-PAX** are four-fold: (i) *Kino-PAX**, a highly parallel near-optimal kinodynamic SBMP designed for GPU-like devices; (ii) a proof of near-optimality and δ-robust completeness; (iii) benchmarks demonstrating *Kino-PAX**'s efficiency in quickly achieving initial solutions and improving them to near-optimality over time; (iv) a comprehen-

sive description and explanation of our implementation of *Kino-PAX** on CUDA-enabled GPUs.

1.0.1 Related Work

Geometric Motion Planning: SBMPs have a long-standing history in addressing the geometric motion planning problem, as established by foundational works such as *Probabilistic RoadMaps* (PRM) [50], *Expansive-Space Tree* (EST) [37], *Rapidly-exploring Random Tree* (RRT) [76], etc. In general terms, SBMP techniques involve finding a path from a starting configuration to a goal region by constructing a graph or tree, where nodes represent geometric configurations and straight line edges represent transitions in the configuration space. Traditionally, these algorithms operate serially on CPU devices. However, the increasing demand for rapid replanning for complex systems in unknown and dynamic environments has driven the development of parallelization methods for geometric SBMPs. These parallelized approaches have been applied to both CPU-based planners [97, 101, 110, 39, 40, 114] and GPU-based implementations [42].

The CPU-based methods in [97, 101, 110] use coarse-grained parallelization, where threads independently construct trees and exchange search information. These methods offer straightforward implementation by leveraging existing serial SBMPs but they achieve limited improvements in planning rates, and are not applicable to many-core devices. More similar to our work, the approaches in [39, 40, 114] focus on fine-grained parallelization to accelerate the construction of a single tree. Works [39, 40] introduce parallel RRT and RRT* methods that leverage thread-safe atomic operations and a novel concurrent data structure for efficient nearest neighbor search. In contrast, [114] decomposes core operations of geometric SBMPs into unconventional data layouts that enable parallelism without specialized hardware. While these works present promising fine-grained parallelization techniques, they are not easily adaptable to kinodynamic planning and are unsuitable for many-core technology due to high inter-thread communication costs.

Most similar to our work, [42] leverages GPUs to solve the geometric path planning problem by adapting *FMT** [49] for parallelized graph search. Their implementation achieves orders-of-magnitude performance improvements over its serial counterpart; however, it is strictly limited to

geometric problems.

Kinodynamic Motion Planning: To provide dynamically feasible and collision-free trajectories for systems with complex dynamics, a kinodynamic motion planning algorithm is used, as seen in traditional serial solutions such as [77, 100, 102, 59, 108]. The details of the kinodynamic motion planning problem are formally discussed in Section 2.2; however, in general, these algorithms are tree-based and solve the problem by sequentially randomly extending trajectories until a path from a start state to a goal region satisfying all state constraints can be followed by a sequence of trajectory segments.

To achieve fast planning times, works [102, 108] employ space discretization. Specifically, [102] constructs a graph from discrete regions and uses it as a high-level planner to guide the motion tree. However, this approach can face the **state-explosion** problem as the dimensionality of state space increases.

In contrast, [108] avoids this issue by using the discrete regions to track spatial information about the sparsity of the motion tree without constructing a graph. In the design of *Kino-PAX* and *Kino-PAX**, we take inspirations from those planners, using discrete regions to guide the search. Similar to [108], we use these regions for spatial information to ensure scalability. However, unlike previous work, *Kino-PAX* and *Kino-PAX** performs these operations in parallel subroutines.

In contrast to the parallelized geometric planning solutions discussed above, parallelization for planning under the constraints of general dynamical systems is relatively understudied in the field. An approach to parallelization for the kinodynamic problem is a coarse-grained method, where multiple trees of classical SBMPs are generated in parallel, and the first solution found is returned [10]. This technique improves average-case performance [120], and is employed in Section 2.6 to perform CPU-based parallelization as a baseline. However, the inherent sequential nature of these motion planners make them inefficient for massive parallelization. In this work, we propose two novel algorithms that enable efficient application to highly parallel devices.

Chapter 2

Kinodynamic Parallel Accelerated Expansion

In this chapter, we present *Kinodynamic Parallel Accelerated eXpansion* (*Kino-PAX*), a novel highly parallel kinodynamic SBMP designed for concurrent devices such as GPUs. *Kino-PAX* grows a tree of trajectory segments directly in parallel. Our key insight is how to decompose the iterative tree growth process into three massively parallel subroutines. *Kino-PAX* is designed to align with the parallel device execution hierarchies, through ensuring that threads are largely independent, share equal workloads, and take advantage of low-latency resources while minimizing high-latency data transfers and process synchronization. This design results in a very efficient GPU implementation. We prove that *Kino-PAX* is probabilistically complete and analyze its scalability with compute hardware improvements. Empirical evaluations demonstrate solutions in the order of 10 ms on a desktop GPU and in the order of 100 ms on an embedded GPU, representing up to 1000× improvement compared to coarse-grained CPU parallelization of state-of-the-art sequential algorithms over a range of complex environments and systems.

2.1 Introduction

A major challenge for robotic systems operating in real-world, unstructured, and dynamic environments is solving the kinodynamic motion planning problem **quickly**. Sampling-based methods have emerged as effective solutions to these complex challenges, as evidenced by foundational works such as [77, 100, 102, 59, 108, 9, 83, 71, 79, 34, 35]. Historically, these algorithms were designed to operate on the computer hardware available at the time, making them inherently sequential and

dependent on processor core clock speed for runtime improvements. However, with the plateauing of processor speeds and minimal expected advancements, state-of-the-art serial algorithms require seconds for simple systems and tens of seconds for more complex systems, making them impractical for applications that demand **real-time** responsiveness.

We instead propose the development of new algorithms tailored for modern processors, leveraging concurrency to solve the kinodynamic motion planning problem in real-time. However, this design process presents significant challenges, as classical SBMPs are inherently sequential, employing a dynamic programming style for node selection and expansion that is ill-suited to parallel execution.

In this chapter, we propose *Kino-PAX*, a highly-concurrent algorithm optimized for GPU-like processors that efficiently solves the kinodynamic motion planning problem within milliseconds. *Kino-PAX* is a kinodynamic SBMP designed to exploit the parallel architecture of modern processing devices. We discuss its hyperparameters, efficiency on modern many-core processors, and provide a thorough analysis including a proof of probabilistic completeness. Additionally, we present benchmarks that demonstrate the algorithm's effectiveness and efficiency in handling complex, high-dimensional dynamical systems.

The remainder of this chapter is organized as follows. Section 2.2 formally describes the problem of interest. Section 2.3 details the *Kino-PAX* algorithm, including pseudocode for all subprocesses. Section 2.4 discusses the tuning parameters of *Kino-PAX* and their impact on performance. Section 2.5 provides a proof of probabilistic completeness for *Kino-PAX*. Section 2.6 demonstrates the performance of *Kino-PAX* across problems of varying difficulty. Lastly, Section 2.7 summarizes our findings and proposes directions for future work.

2.2 Problem Formulation

Consider a robotic system operating within a bounded workspace $W \subset \mathbb{R}^d$, where $d \in \{2, 3\}$. This workspace contains a finite set of obstacles $\mathcal{O}$, where each obstacle $o \in \mathcal{O}$ is a closed subset of

W, i.e., $o \subset W$. The dynamics of the robot's motion is given by

$$\dot{x}(t) = f(x(t), u(t)), \tag{2.1}$$

where $x(t) \in X \subset \mathbb{R}^n$ and $u(t) \in U \subset \mathbb{R}^N$ are the robot's state and control at time t, respectively, and $f : X \times U \to \mathbb{R}^n$ is the vector field. We assume that f is a Lipschitz continuous function with respect to both arguments, i.e, there exist constants $K_x, K_u > 0$ such that for all $x, x' \in X$ and $u, u' \in U$,

$$\|f(x, u) - f(x', u')\| \leq K_x\|x - x'\| + K_u\|u - u'\|.$$

In addition to motion constraints defined by the dynamics in (2.1) and obstacles in $\mathcal{O}$, we consider state constraints, e.g., bound on the velocity. To this end, we define the set of valid states, i.e., states at which the robot does not violate its state constraints and does not collide with an obstacle, as the valid set and denote it by $X_{\text{valid}} \subseteq X$. Then, given initial state $x_{\text{init}} \in X_{\text{valid}}$, time duration $t_f \geq 0$, and control trajectory $\mathbf{u} : [0, t_f] \to U$, a *state trajectory* $\mathbf{x} : [0, t_f] \to X$ is induced, where

$$\mathbf{x}(t) = x_{\text{init}} + \int_0^t f(\mathbf{x}(\tau), \mathbf{u}(\tau))d\tau \qquad \forall t \in [0, t_f]. \tag{2.2}$$

Trajectory $\mathbf{x}$ is called *valid* if, $\forall t \in [0, t_f]$, $x(t) \in X_{\text{valid}}$.

In motion planning, the interest is to find a valid trajectory $\mathbf{x}$ that visits a given goal set $X_{\text{goal}} \subseteq X_{\text{valid}}$. Therefore, by following this trajectory, the robot is able to respect all of its motion (kinodynamic) constraints, avoid collisions with obstacles, and reach its goal. In this work, we focus on kinodynamic motion planning with an emphasis on computational efficiency through parallelism.

Problem 1 (Kinodynamic Motion Planning)**.** *Consider a robot with dynamics in* (2.1) *in workspace* W *consisting of obstacle set* $\mathcal{O}$*. Given an initial state* $x_{init} \in X_{valid} \subseteq X$ *and goal region* $X_{goal} \subseteq X_{valid}$*,* ***efficiently*** *find a control trajectory* $\mathbf{u} : [0, t_f] \to U$ *such that its induced trajectory* $\mathbf{x}$ *through* (2.2) *is valid and reaches goal, i.e.,* $\mathbf{x}(0) = x_{init}$ *and*

$$\mathbf{x}(t) \in X_{valid} \qquad \forall t \in [0, t_f],$$

$$\mathbf{x}(t) \in X_{goal} \qquad \exists t \in [0, t_f].$$

Note that this is a challenging problem. The simpler problem of geometric motion planning (by ignoring dynamics) is already PSPACE-complete [106], and the addition of kinodynamic constraints makes finding a solution considerably more difficult due to the increase in search space dimension and dynamic complexities [17, 73]. Existing algorithms find solutions in the order of seconds for simple (e.g., linear) systems and tens of seconds for more complex non-linear systems [102, 100, 77] on standard benchmark problems. When combined with the need for fast replanning in, e.g., unknown and changing environments, finding solutions in real-time (milliseconds) becomes crucial for ensuring the functionality and safety of autonomous systems.

With the availability of onboard GPUs, parallel computation provides a promising approach for finding solutions quickly. Hence, in our approach, we focus on achieving efficiency through a highly parallelizable algorithm.

2.3 Kino-PAX

Our approach to Problem 1 is a highly parallel algorithm that is able to exploit the many-core architecture of GPU-like processors. To achieve efficient performance on these high-throughput devices, it is crucial that our algorithm complements the execution hierarchy of such processors to optimize resource utilization. For the development of this algorithm, we follow the guidance of [51, 96] and base our development on three key principles: (i) **thread independence**, the ability for each thread in a program to execute without being dependent on the state or result of other threads; (ii) **even workloads across threads**, each thread is assigned an equal or nearly equal number of operations throughout its execution; (iii) **utilization of low-latency memory**, groups of threads utilize low-latency memory to share information and reduce the number of higher-latency global memory accesses.

With these principles in mind, we introduce **Kinodynamic Parallel Accelerated eXpansion** (*Kino-PAX*), a highly parallel kinodynamic SBMP.

Kino-PAX grows a tree of trajectory segments in parallel. This is achieved by decomposing the iterative tree growth process, i.e., selection of nodes, extension, validity checking, and adding

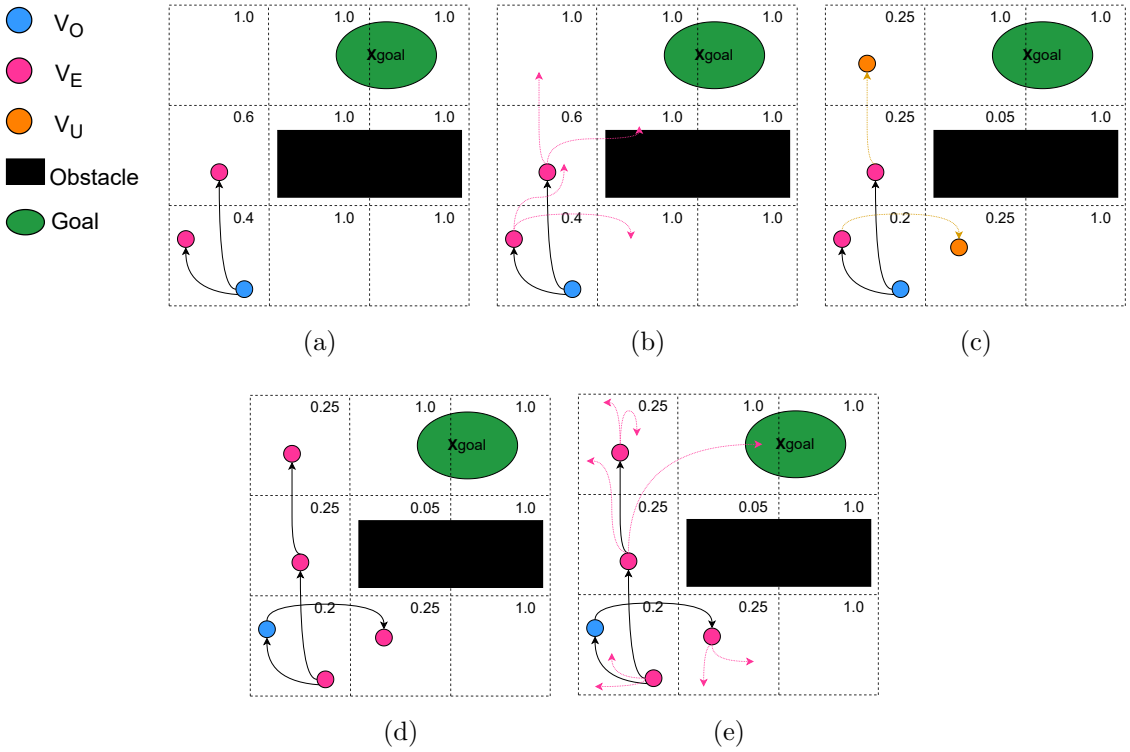
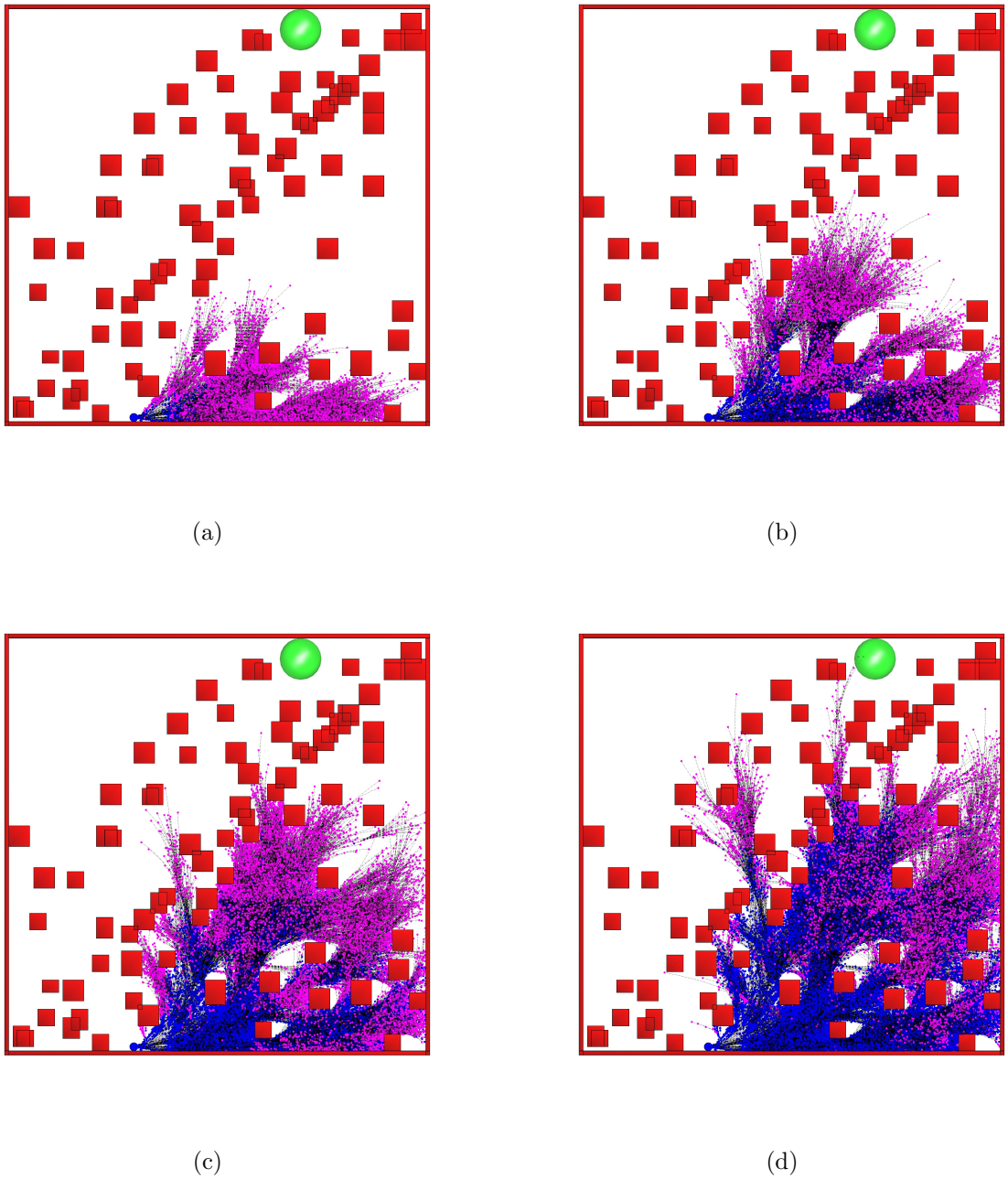


Figure 2.1: Illustration of the *Kino-PAX* expansion process: (a) The current sets V_E and V_O, where the numbers in each grid cell represent the value of $P_{accept}(\mathcal{R}_i)$. (b) Expansion of V_E with branching factor $\lambda = 2$. (c) Acceptance of promising samples and update of $P_{\text{accept}}(\mathcal{R}_i)$. (d) Updated V_E, ready for the next iteration. (e) Expansion of V_E, producing a valid trajectory to X_{goal}.

new nodes to the tree, into three massively parallel subroutines. Each subroutine follows the key principles of **thread independence**, **balanced workloads**, and **low-latency memory utilization**. Additionally, to ensure fast and efficient planning iterations, we minimize the communication needed in the synchronization steps between subroutines, e.g., CPU-GPU communication.

At each iteration of *Kino-PAX*, a set of nodes in the tree is expanded in parallel. Each sample is extended multiple times through random sampling of controls, also in parallel. We dynamically adjust the number of extensions in each iteration to maintain an effective tree growth rate, ensuring efficient usage of the device's throughput. After extension, a new set of nodes are selected independently to be propagated in the next iteration.

To guide the search process, *Kino-PAX*, similar to [102, 108], employs a high-level space



(a) (b)

(c) (d)

Figure 2.2: Four iterations of the *Kino-PAX* planning process for a Dubins airplane dynamical system, illustrating the growth of the solution tree. Newly added, promising nodes from the unexplored set V_U are shown in pink; existing nodes within the tree in blue; the goal region is highlighted in green; and obstacles are marked in red.

decomposition approach. We designed this method to be well-suited for parallel computation. This

Algorithm 1: *Kino-PAX*

Input : $x_{\text{init}}, X_{\text{goal}}, t_{max}$
Output: Solution trajectory $\mathbf{x}$

$\mathcal{T} \leftarrow$ Initialize tree with root node x_{init}
$V_E \leftarrow \{x_{\text{init}}\}$, $V_U, V_O \leftarrow \emptyset$
Initialize $\mathcal{R}$ with $P_{accept}(\mathcal{R}_i) = 1$ for each $\mathcal{R}_i \in \mathcal{R}$
while $ElapsedTime < t_{max}$ **do**
    `Propagate`$(V_E, V_U, \mathcal{R}, \lambda)$
    `UpdateEstimates`$(\mathcal{R})$
    $\mathbf{x} \leftarrow$ `UpdateNodeSets`$(V_U, V_E, V_O, \mathcal{T}, \mathcal{R}, X_{\mathbf{goal}})$
    if $\mathbf{x} \neq null$ **then return** $\mathbf{x}$;
return *null*

decomposition estimates exploration progress in each region, allowing threads to act independently when adding new nodes to the tree and identifying promising nodes for extension. As more trajectory segment data becomes available, the estimate of promising space regions is improved, allowing *Kino-PAX* to focus on propagating a large number of favorable nodes into less explored areas of the space.

2.3.1 Core Algorithm

Here, we present a detailed description of *Kino-PAX*. Pseudocode of *Kino-PAX* is presented in Alg. 1, with subroutines Algs. 2, 3, and 4, a full planning iteration is illustrated in Fig. 2.1, and several planning iterations are shown in 2.2. *Kino-PAX* organizes samples into three distinct sets: V_U, V_O, V_E. The set V_U consists of newly generated promising samples that have not yet been added to the tree. The set V_O includes tree nodes that are not currently considered for expansion; intuitively, these nodes are located in densely populated or frequently invalid regions of the search space. Finally, V_E comprises the set of nodes that are flagged for parallel expansion. Further, *Kino-PAX* maintains the spatial search progress information using a partition of the state space denoted by $\mathcal{R}$.

2.3.1.1 Initialization

In Alg. 1, *Kino-PAX* takes as input the initial state x_{init}, a goal region X_{goal}, and a maximum execution time t_{max}. In Lines 1-2, a tree $\mathcal{T}$ is initialized with x_{init} at its root. Additionally, the set V_E is initialized with the state x_{init} and the sets V_U and V_O are initialized as empty.

In Line 3, the space decomposition $\mathcal{R}$ is initialized in the state space, where the decomposition consists of non-overlapping regions, such that:

$$X = \cup_{i=1}^{n} \mathcal{R}_i \quad \text{and} \quad \forall i \neq j \in \{1, \ldots, n\}, \quad \text{Int}(\mathcal{R}_i) \cap \text{Int}(\mathcal{R}_j) = \emptyset,$$

where $\text{Int}(\mathcal{R}_i)$ is the interior of $\mathcal{R}_i$. Each $\mathcal{R}_i$ is then further partitioned to a set of finer regions. We denote the k-th sub-region of $\mathcal{R}_i$ by $\mathcal{R}_i^k$, i.e., $\mathcal{R}_i = \cup_{k=1}^{n'} \mathcal{R}_i^k$. For each region $\mathcal{R}_i \in \mathcal{R}$, several metrics are calculated to assess the exploration progress of the tree. These metrics, adapted from [102], are designed to be effective in identifying promising regions for systems with complex dynamics and are well suited for parallelism. Specifically, *Kino-PAX* updates the following metrics for each region, $\mathcal{R}_i$, after each iteration of parallel propagation to continually guide the search process:

- $Cov(\mathcal{R}_i)$: estimates the progress made by the tree planner in covering $\mathcal{R}_i$;
- $FreeVol(\mathcal{R}_i)$: estimates the free volume of $\mathcal{R}_i$.

The exact expressions for $Cov(\mathcal{R}_i)$ and $FreeVol(\mathcal{R}_i)$ are the same as in [102], which we also show in Sec. 2.3.1.3.

These metrics determine a score value, $Score(\mathcal{R}_i)$ and subsequently a probability $P_{accept}(\mathcal{R}_i)$, which aid *Kino-PAX* in adding favorable samples to $\mathcal{T}$ and assessing if an existing node should be extended. During initialization, all $P_{accept}(\mathcal{R}_i)$ values are set to 1.

2.3.1.2 Node Extension

After initialization, the main loop of the algorithm begins (Alg. 1, Lines 4–8). In each iteration, the `Propagate` (Alg. 2) function is called to propagate the set V_E in parallel (Alg. 1,

Algorithm 2: Propagate

Input : $V_E, V_U, \mathcal{R}, \lambda$
Output: Updated V_U

```
1  foreach x ∈ V_E do
2      Map x to region R_i ;
3      for i = 1, ..., λ do
4          Randomly sample u and dt ;
5          x' ← PropagateODE(x, u, dt) ;
6          Map x' to region R_j^k ;
7          if the trajectory from x to x' is valid then
8              Increment n_valid(R_i) ;
9              if R_j^k unvisited or with P_accept(R_j) then
10                 Add x' to V_U ;
11         else
12             Increment n_invalid(R_i) ;
```

Line 5, Fig. 2.1b). Each node $x \in V_E$ is expanded $\lambda \in \mathbb{N}^+$ times using λ threads, where each thread handles one expansion of x (Alg. 2, Lines 1–2). For each thread, a control $u \in U$ and a time duration $dt \in (0, T_{prop}]$, where T_{prop} is a user-defined constant that sets the maximum propagation time, are randomly sampled, and the node's continuous state x is propagated using dynamics in (2.1) to generate a new sample state x' (Alg. 2, Lines 3–4). Next, the corresponding region of x', $\mathcal{R}_i^k$, is calculated (Alg. 2, Line 5). Then, in Line 6, the trajectory segment from x to x' is checked for validity, i.e., if it is in X_{valid}. Throughout the trajectory segment, a user-defined collision check is performed (our implementation uses a coarse-phase bounding volume hierarchies method discussed in [42]).

If the extended segment is valid, we increment the total number of valid samples in $\mathcal{R}_i$ (Alg. 2, Line 7). Next, x' is added to the set V_U if its corresponding region $\mathcal{R}_i^k$ is unvisited; if $\mathcal{R}_i^k$ already contains a node, then x' is added to V_U with probability $P_{accept}(\mathcal{R}_i)$, which favors promising samples (Alg. 2, Lines 8-9, Fig. 2.1c). Alternatively, if the trajectory segment is invalid, we increment the count of invalid samples in $\mathcal{R}_i$, as shown in Line 11. This information guides future propagation iterations away from regions that are frequently invalid, improving search efficiency.

Algorithm 3: UpdateEstimates

Input : $\mathcal{R}$
Output: Updated estimates for each region $\mathcal{R}_i$
foreach $\mathcal{R}_i \in \mathcal{R}_{avail}$ **do**
  `UpdateFreeVol`($\mathcal{R}_i$);
  `UpdateCoverage`($\mathcal{R}_i$);
  `UpdateScore`($\mathcal{R}_i$);
foreach $\mathcal{R}_i \in \mathcal{R}_{avail}$ **do**
  `UpdateAccept`($\mathcal{R}_i$);

2.3.1.3 Node Selection

After all samples in V_E have been expanded, the `UpdateEstimates` (Alg. 3) subroutine is called (Alg. 1, Line 6). In this subroutine, metrics for each visited region (i.e., a region with a node $x \in \mathcal{T}$), denoted by $\mathcal{R}_{avail}$, are updated in parallel, with a thread handling a unique region $\mathcal{R}_i \in \mathcal{R}_{avail}$, calculating $Cov(\mathcal{R}_i)$ and $FreeVol(\mathcal{R}_i)$ (Alg. 3, Lines 1-3). For each thread, $Cov(\mathcal{R}_i)$ is set to the number of visited sub-regions within $\mathcal{R}_i$, and $FreeVol(\mathcal{R}_i)$ is calculated as

$$FreeVol(\mathcal{R}_i) = \frac{\big(\delta + n_{valid}(\mathcal{R}_i)\big) \cdot vol(\mathcal{R}_i)}{\delta + n_{valid}(\mathcal{R}_i) + n_{invalid}(\mathcal{R}_i)}, \tag{2.3}$$

where $\delta > 0$ is a small constant, and $vol(\mathcal{R}_i)$ represents the mapped workspace volume of the region $\mathcal{R}_i$. Subsequently, on Line 4 of Alg. 3, each thread calculates its corresponding $Score(\mathcal{R}_i)$ value with

$$Score(\mathcal{R}_i) = \frac{\text{FreeVol}^4(\mathcal{R}_i)}{(1 + \text{Cov}(\mathcal{R}_i))(1 + (n_{valid}(\mathcal{R}_i) + n_{invalid}(\mathcal{R}_i))^2)}, \tag{2.4}$$

which prioritizes regions that are less visited and have a high free volume and low coverage.

Once all score values have been updated, each visited region's $P_{accept}(\mathcal{R}_i)$ probability is refined (Alg. 3, Lines 5-6, Fig. 2.1c). This process, again, is done in parallel with a thread being designated to a unique $\mathcal{R}_i \in \mathcal{R}_{avail}$ and $P_{accept}(\mathcal{R}_i)$ being set by

$$P_{accept}(\mathcal{R}_i) = \min\left\{1, \quad \frac{Score(\mathcal{R}_i)}{\sum_{\mathcal{R}_j \in \mathcal{R}_{\text{avail}}} Score(\mathcal{R}_j)} + \epsilon\right\}, \tag{2.5}$$

where $0 < \epsilon \ll 1$ is a constant and $\mathcal{R}_{avail} \in \mathcal{R}$ represents the set of regions that contain a node in $\mathcal{T}$. We note that the expressions for the metrics in (2.3)-(2.5) are taken from [102].

Algorithm 4: UpdateNodeSets

Input : $V_U, V_E, V_O, \mathcal{T}, \mathcal{R}, X_{\text{goal}}$
Output: Trajectory if a goal is found, otherwise *null*

1 **foreach** $x \in V_E$ **do**
2 | Map x to $\mathcal{R}_i$;
3 | Move x to V_O with probability $1 - P_{accept}(\mathcal{R}_i)$;
4 **foreach** $x \in V_U$ **do**
5 | Move x from V_U to V_E and $\mathcal{T}$;
6 | **if** $x \in X_{\textbf{goal}}$ **then return** Trajectory x_{init} to x ;
7 **foreach** $x \in V_O$ **do**
8 | Map x to $\mathcal{R}_i$;
9 | Move x to V_E with probability $P_{accept}(\mathcal{R}_i)$;
10 **return** *null*

Once the $P_{accept}(\mathcal{R}_i)$ probabilities have been updated for all available regions, the `UpdateNodeSets` subroutine is called (Alg. 1, Line 7, Fig. 2.1d). In Lines $1-3$ of Alg. 4, we remove samples from the expansion set V_E randomly with probability $1 - P_{accept}(\mathcal{R}_i)$, ensuring that promising samples remain in V_E. Then, we move the newly generated samples, V_U, to $\mathcal{T}$ and add them to the expansion set V_E. If any newly generated nodes satisfy goal criteria, we return the valid trajectory $\mathbf{x}$ (Alg. 4, Lines 4-6). Finally, we move inactive samples in V_O to the expansion set if deemed promising with the updated search information (Alg. 4, Lines 7-9).

Kino-PAX repeats the main loop of `Propagate`, `UpdateEstimates` and `UpdateNodeSets` until a solution trajectory $\mathbf{x}$ that solves Problem 1 is returned, or a user-defined time limit t_{max} is surpassed.

2.4 Tuning and Performance Discussion

In this section, we discuss how *Kino-PAX* can be tuned to match a problem's difficulty, and present the properties of *Kino-PAX* that enable efficient parallelism.

2.4.1 Tuning Parameter

In practice, due to the limitations of device memory and the high cost of vector resizing operations, we predefine the maximum size of the tree rather than constraining the runtime, similar

to the approach taken in PRM. This introduces a hyperparameter for *Kino-PAX*, denoted as t_e, which we refer to as the expected tree size. Specifically, t_e corresponds to the maximum number of nodes in *Kino-PAX* and should be tuned according to the difficulty of the problem at hand.

Varying, t_e has two main effects on the performance of *Kino-PAX*. Firstly, an increase in t_e increases the branching factor λ which is updated for each iteration of parallel propagation and is set according to

$$\lambda = \min \left\{ \lambda_{\max}, \ \left\lfloor \frac{t_e - |\mathcal{T}|}{|V_E|} \right\rfloor \right\}, \tag{2.6}$$

where λ_{max} is the user-defined maximum branching factor and $|\mathcal{T}|$ and $|V_E|$ are the numbers of nodes in $\mathcal{T}$ and V_E, respectively. Eq. (2.6) ensures that in the early iterations, when $|\mathcal{T}|$ is much smaller than t_e, a larger λ is used. This approach effectively uses available throughput and accelerates the initial stages of tree propagation. As $|\mathcal{T}|$ approaches t_e and $|V_E|$ grows large, a smaller λ is used to stabilize the growth rate of $\mathcal{T}$.

Secondly, t_e affects the number of nodes that can be included in $\mathcal{T}$. As the problem difficulty increases, such as with a system's state dimension, a larger t_e is recommended to sufficiently explore the space. *Kino-PAX*'s search characteristics make finding a suitable t_e relatively straightforward for a given system, as the tree expands in all accessible free space areas. We demonstrate this in Sec. 2.6, where systems with the same state dimension utilize a constant t_e value across all testing environments. In Sec. 2.6, we also show that as t_e increases, the success rate of *Kino-PAX* converges to 100%.

Remark 1. *The maximum tree size can be made adaptive by increasing t_e by a constant multiple if a solution is not found after the tree size nears its threshold. However, this introduces vector resizing operations on high latency device memory, slowing down the search.*

2.4.2 Decomposition Tuning

The search efficiency of *Kino-PAX* is dependent on the choice of space decomposition. An improper decomposition, e.g., one that is too coarse or too fine, can lead to inefficiencies. A

decomposition that is too fine may cause slower updates due to the large number of regions. On the other hand, a decomposition that is too coarse may lead to poor approximation of promising space regions. For instance, if a region frequently generates invalid trajectories but contains critical space for finding a solution, *Kino-PAX* may experience inefficient search. A method to mitigate this is to use a free-space-obeying decomposition, as proposed in [102]. Nonetheless, *Kino-PAX* remains probabilistically complete with any valid decomposition.

2.4.3 Efficient Application to Highly Parallel Devices

Kino-PAX's propagation subroutine implementation is well-suited for parallelism due to three main factors. First, we utilize **low-latency memory** by distributing commonly used data to groups of nearby threads. Specifically, the state $x \in V_E$ is shared via on-chip memory to all λ threads assigned to expand the node. Second, we **balance workloads** across threads by having each thread create a single trajectory segment, minimizing thread divergence—i.e., variations in execution paths that force threads into serial execution. Third, the acceptance of new samples and their addition to V_U is achieved with **thread independence**, using (2.5) and unique thread identifiers.

Kino-PAX maintains its space decomposition in a parallel-friendly manner through the metrics (2.3)-(2.4) that can be calculated independently of other regions and by ensuring that each region's calculations have an equal number of operations. Further, we avoid the need for serial data structures when choosing expansion nodes by adding samples to V_E independently via the acceptance probability (2.5).

Furthermore, the organization of samples into three disjoint sets, V_U, V_O, and V_E, enables a straightforward and memory-efficient representation. In *Kino-PAX*, we do this via a boolean-mask representation that is thoroughly discussed in [89].

Lastly, *Kino-PAX* reduces latency between its subroutines by pre-allocating a large memory chunk on the GPU to accommodate t_e nodes. This allows *Kino-PAX* to construct its tree directly on the GPU, avoiding the transfer of large data structures between devices.

2.5 Analysis

Here, we show that *Kino-PAX* is probabilistically complete for Problem 1 and analyze its scalability.

2.5.1 Probabilistic Completeness

We begin with a definition of probabilistic completeness for algorithms that solve Problem 1.

Definition 1 (Probabilistic Completeness)**.** *A sampling-based algorithm is* ***probabilistically complete*** *if the probability that the algorithm fails to return a solution, given one exists, approaches zero as the number of samples approaches infinity.*

We first show that, in *Kino-PAX*, every tree node has a non-zero probability of being extended.

Lemma 1. *Let $x \in \mathcal{T}$ be a node in the tree. The probability that x is selected for extension is lower bounded by $\epsilon \in (0, 1)$.*

Proof. The probability of extending a node $x \in \mathcal{T}$ in *Kino-PAX* is calculated using (2.5) for its corresponding region $\mathcal{R}_i \in \mathcal{R}$. We demonstrate that every region $\mathcal{R}_i$ falls into one of three cases, and in each case, (2.5) is lower bounded by ϵ.

Case 1: a region $\mathcal{R}_i$ is entirely in invalid space, i.e., $\mathcal{R}_i \cap X_{free} = \emptyset$, meaning $\mathcal{R}_i$ cannot contain a tree node and $\mathcal{R}_i \notin \mathcal{R}_{avail}$. Thus, (2.5) is set to $1 > \epsilon$, as per Line 3 of Alg. 1.

Case 2: a region $\mathcal{R}_i$ is entirely within free space, i.e., $\mathcal{R}_i \subseteq X_{\text{valid}}$. If $\mathcal{T}$ does not contain a node in $\mathcal{R}_i$, then $\mathcal{R}_i \notin \mathcal{R}_{avail}$, and (2.5) is set to $1 > \epsilon$, as by Line 3 of Alg. 1. However, if $\mathcal{T}$ does contain a node in $\mathcal{R}_i$, we examine the worst-case scenario that produces the lowest probability from (2.5). In this scenario, the number of nodes in $\mathcal{R}_i$ approaches infinity, driving the score of $\mathcal{R}_i$ to 0 by (2.4). Additionally, in a worst-case scenario, the score of all other available regions approaches infinity. In this case, (2.5) trivially approaches ϵ.

Case 3: $\mathcal{R}_i$ is partially in X_{valid}, i.e., $\mathcal{R}_i \cap X_{\text{valid}} \neq \emptyset$ and $\mathcal{R}_i \cap X_{\text{valid}} \neq \mathcal{R}_i$. In this case, similar to above, if $\mathcal{R}_i \notin \mathcal{R}_{avail}$, (2.5) is set to 1 by Line 3 of Alg. 1. If $\mathcal{R}_i \in \mathcal{R}_{avail}$, as in *Case 2*, (2.5) approaches ϵ in the worst-case, when the number of nodes in $\mathcal{R}_i$ approaches infinity. □

Finally, we can state our main analysis result.

Theorem 1. *Kino-PAX is probabilistically complete.*

Proof Sketch. Our proof is an adaptation of the result from [52], originally established for kino-dynamic RRT, for *Kino-PAX*. Specifically, the conditions laid out in [52, Lemma 2–3] hold under the assumptions of Problem 1 in this paper. Furthermore, Lemma 1 provides a lower bound $\epsilon > 0$ on the probability of extending from an arbitrary tree node. By combining [52, Lemma 2-3] with Lemma 1, we can follow the same logical structure as the proof of [52, Theorem 2]. Thus, as the number of samples approaches infinity, *Kino-PAX* asymptotically almost-surely finds a valid trajectory from x_{init} to X_{goal} that solves Problem 1. □

2.5.2 Scalability

Here, we discuss the scalability of *Kino-PAX*. Firstly, *Kino-PAX*'s scalability increases as the number of cores in the parallel device increases. Since *Kino-PAX* supports adaptive tuning of the branching factor λ through varying t_e, an increased number of cores can be leveraged by increasing t_e and setting a higher λ_{max}. This results in computation time improvements as the number of cores increase, allowing *Kino-PAX* to scale as parallel hardware computation power improves.

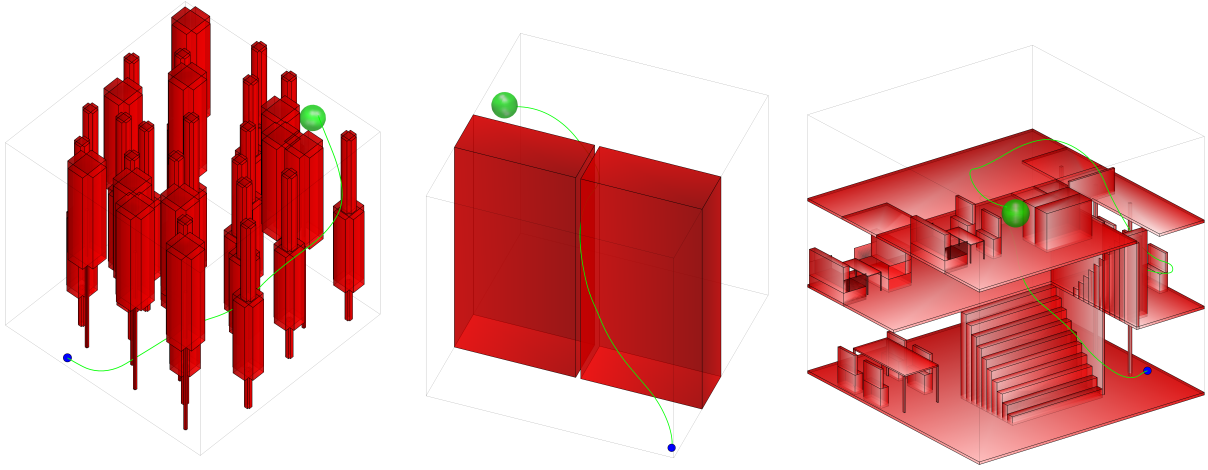
Secondly, as the problem becomes more complex, i.e., requiring a larger tree (more nodes) to find a solution, traditional tree-based SBMPs suffer. That is, they slow down significantly as the number of nodes increases due to the sequential nature of those algorithms. However, *Kino-PAX* does not struggle with increasing number of nodes, unless the tree size nears its threshold and a resizing operation is needed.

This property makes *Kino-PAX* particularly more advantageous for planning for systems with large dimensional state spaces since they often require large trees to sufficiently search the

space.

2.6 Experiments

We demonstrate the performance of *Kino-PAX* in planning for various dynamical systems across three 3D environments shown in Fig. 2.4. The considered systems are: (i) 6D double integrator, (ii) 6D Dubins airplane [12], and (iii) 12D highly nonlinear quadcopter [21].



(a) Forest (b) Narrow Passage (c) Building

Figure 2.3: Environments used throughout the experiments with the solution trajectory produced by *Kino-PAX*. Initial position and goal region are shown by blue and green spheres, respectively. Environments (a) and (c) are taken from [42].

For each pair of dynamical system and environment, we benchmark the performance and scalability of *Kino-PAX* against four traditional SBMPs: RRT [77], EST [37], PDST [59] and SyCLoP [102]. To ensure fairness, for these comparison planners, we used a coarse-grained CPU parallelization method where multiple trees grow in parallel as suggested by [10]. Each tree is managed by a separate thread, and the planner returns the first solution found.

We implemented *Kino-PAX* in CUDA C and performed benchmarks on two GPUs with different capabilities. We used an NVIDIA RTX 4090 as a baseline, which has 16,384 CUDA

cores and 24 GB of RAM. Further, to test the efficiency of *Kino-PAX* on an embedded GPU, we ran benchmarks on an NVIDIA Jetson Orin Nano, which has 1,024 cores and 8 GB of RAM. The comparison algorithms, are implemented in C++ using *OMPL* [109] and executed on an Intel Core i9-14900K CPU with 24 cores, a base clock speed of 4.4 GHz, and 128 GB of RAM. Our implementation of *Kino-PAX* is publicly available: `https://github.com/aria-systems-group/Kino-PAX`.

We followed the standard setup configurations for all comparison algorithms as recommended by the *OMPL* documentation. All algorithms, including *Kino-PAX*, were configured to use the same methods for state propagation, state validity checking and space decomposition. For all experiments, a grid-based decomposition was used with the dimensionality of the grid equal to the system's state-space dimension. For *Kino-PAX*'s hyperparameters, λ_{max} was set to 32 and t_e was set to 2×10^5 for all 6D systems and 4×10^5 for the 12D system. For each combination of algorithm, dynamic model, and workspace, we performed 50 queries, each with a maximum runtime of 60 seconds.

2.6.1 Benchmark Results

Table 2.1 shows the mean runtime, the speed ratio relative to the desktop GPU implementation of *Kino-PAX*, and the success rate within the allotted planning time for each combination of algorithm, environment, and dynamics.

For both 6D systems, *Kino-PAX* finds a solution trajectory in less than 8 ms across all testing environments. For the 6D Double Integrator, *Kino-PAX* is on average 85×, 287×, 433× faster in Environments 1, 2, 3, respectively, compared to the baseline algorithms.

For the Dubins Airplane system, the performance gap of *Kino-PAX* widens further. For instance, in Environment 3, *Kino-PAX* experiences a slowdown of less than 1.3× (∼2 ms) compared to RRT's 22× (∼20,000 ms), EST's 1.8× (∼4,000 ms), PDST's 8.9× (∼16,000 ms), and SyCLoP's 22× (∼24,000 ms). Additionally, the embedded GPU implementation of *Kino-PAX* outperforms all serial baseline methods, finding valid trajectories for all 6D problems in under 115 ms. When

dealing with the more challenging 12D nonlinear quadcopter problem, *Kino-PAX* finds solutions in less than 25 ms across all environments. On average, this marks an improvement of **three orders of magnitude** over all reference serial solutions. In the most challenging environment (Environment 3), the best-performing baseline algorithm (EST) is 1720× slower than the desktop implementation of *Kino-PAX* and 44× slower than the embedded GPU implementation.

As evident from the results, *Kino-PAX* outperforms baseline algorithms more significantly as the problem becomes more challenging; in other words, the performance gap significantly widens in favor of *Kino-PAX*. This is due to two main factors. First, as the dimensionality of the search space increases, exponentially more trajectory segments are required to find a valid solution. This suits *Kino-PAX* particularly well, as it is designed to propagate a massive number of nodes efficiently in parallel. Second, as the problem difficulty increases, the efficiency of *Kino-PAX* becomes more prominent. This is because unlike traditional tree-based SBMPs that slow down as the number of samples increases, *Kino-PAX* does not suffer as much with the size of the tree.

Table 2.1: Benchmark results over 50 trials with a 60-second maximum planning time. CPU-based algorithms used coarse-grained parallelization, growing multiple trees in parallel and utilizing all available cores, denoted by "Par" before the algorithm name.

Algorithm	Device	Environment 1			Environment 2			Environment 3		
		Time (ms)	$t_{alg}/t_{Kino\text{-}PAX}$	Succ %	Time (ms)	$t_{alg}/t_{Kino\text{-}PAX}$	Succ %	Time (ms)	$t_{alg}/t_{Kino\text{-}PAX}$	Succ %
6D Double Integrator										
Par RRT	CPU	157.0	42.0	100.0	598.1	184.1	100.0	993.7	183.4	100.0
Par EST	CPU	416.1	111.4	100.0	1822.5	561	100.0	5122.8	945.5	100.0
Par PDST	CPU	486.0	130.1	100.0	1144.7	352.4	100.0	2094.5	386.6	100.0
Par SyCLoP	CPU	216.6	58.0	100.0	168.1	51.8	100.0	1183.3	218.4	100.0
Kino-PAX	Embd. GPU	62.3	16.7	100.0	29.1	9.0	100.0	78.6	14.5	100.0
Kino-PAX	GPU	**3.7**	1.0	100.0	**3.3**	1.0	100.0	**5.4**	1.0	100.0
Dubins Airplane										
Par RRT	CPU	632.5	165.0	100.0	4973.9	1389.9	100.0	22698.7	3207.4	98.0
Par EST	CPU	275.7	71.9	100.0	3443.6	962.3	100.0	9108.5	1287.1	100.0
Par PDST	CPU	515.6	134.5	100.0	12410.3	3467.9	98.0	18570.2	2624.0	100.0
Par SyCLoP	CPU	234.4	61.2	100.0	936.2	261.6	100.0	25544.7	3610.0	100.0
Kino-PAX	Embd. GPU	67.5	17.6	100.0	43.1	12.0	100.0	110.5	15.6	100.0
Kino-PAX	GPU	**3.8**	1.0	100.0	**3.6**	1.0	100.0	**7.1**	1.0	100.0
12D Non Linear Quadcopter										
Par RRT	CPU	40694.7	2262.5	72.0	91023.1	5306.9	12.0	93144.7	3873.2	8.0
Par EST	CPU	10034.4	557.9	100.0	35435.9	2066	90.0	41381.0	1720.8	84.0
Par PDST	CPU	23726.9	1319.1	92.0	51203.4	2985.3	70.0	53169.8	2211	68.0
Par SyCLoP	CPU	3384.2	188.2	100.0	10181.6	593.6	100.0	86558.3	3599.4	16.0
Kino-PAX	Embd. GPU	797.0	44.3	100.0	681.8	39.8	100.0	935.7	38.9	100.0
Kino-PAX	GPU	**18.0**	1.0	100.0	**17.2**	1.0	100.0	**24.1**	1.0	100.0

2.6.2 Effects Of Tuning Parameter t_e

To support the point made in Sec. 2.4 that *Kino-PAX*'s hyperparameter t_e is easy to tune and that *Kino-PAX* remains efficient across a wide range of values, we present a numerical experiment showing that as *Kino-PAX* is provided with a sufficiently large t_e, its failure rate converges to zero. We also examine the impact on runtime as t_e increases. Fig. 2.4 presents the results for planning with the 12D nonlinear quadcopter system in Environment 1.

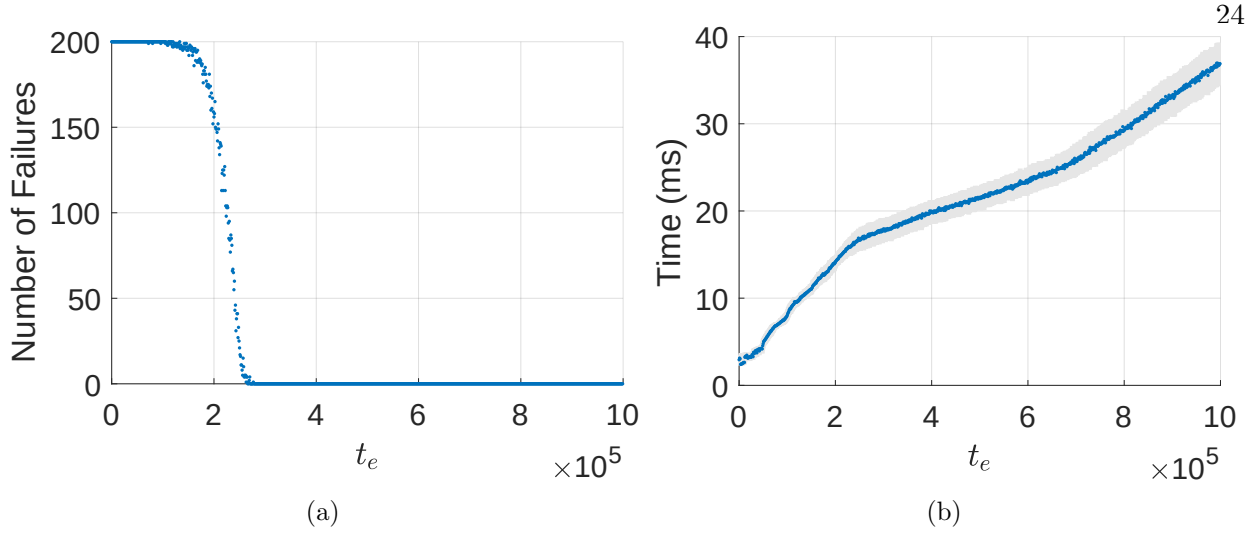


Figure 2.4: Results of varying the expected tree size t_e for the 12D nonlinear quadcopter system in environment 2.3a. (a) Number of Failures vs. t_e. (b) Mean runtime and variance of *Kino-PAX* vs. t_e.

As shown in Fig. 2.4a, for small values of t_e, *Kino-PAX* is unable to find solutions, indicating that the number of samples required exceeds t_e. As t_e increases beyond 2.8×10^5, *Kino-PAX* achieves a 100% success rate, demonstrating that t_e is not a sensitive tuning parameter with respect to finding solutions.

Fig. 2.4b shows an increase in the hyperparameter t_e also increases the runtime. This can be attributed to two main factors. First, as t_e increases, *Kino-PAX* typically uses a larger branching factor λ, resulting in a tree with more nodes. In this experiment, the number of nodes in the tree increases by approximately 1×10^5 for every 2×10^5 increase in t_e. Specifically, we observed 3.1×10^5 nodes when $t_e = 4 \times 10^5$ and 6.1×10^5 nodes when $t_e = 10 \times 10^5$. Second, the larger t_e demands more memory, leading to more frequent cache misses in the implementation of *Kino-PAX*, which causes data to be fetched from slower global memory more often.

2.7 Conclusion

We have introduced a novel motion planning algorithm for kinodynamic systems that enables a significant parallelization of a process previously considered inherently sequential. Our algorithm

is well suited to exploit the recent advancements of modern computing devices and is equipped to scale as hardware continues to improve. Benchmark results show planning times of less than 8 ms for 6-dimensional systems and less than 25 ms for a 12-dimensional nonlinear system, representing an improvement of up to three orders of magnitude compared to traditional motion planning algorithms. For future work, we plan to make the hyperparameter t_e dynamic (adaptive) and extend *Kino-PAX* to a near-optimal planner.

Chapter 3

Asymptotically Near Optimal Kinodynamic Parallel Accelerated Expansion

3.1 Introduction

In addition to solving the kinodynamic motion planning problem **quickly**, it is essential for real-world robotics to also achieve **near-optimal** solutions. This challenge has been extensively explored in foundational works such as [78], which examines various methods enabling SBMP algorithms to asymptotically approach near-optimal solutions. However, similar to classical SBMP algorithms discussed in [77, 100, 102, 59, 108, 9, 83, 71, 79, 34, 35], existing near-optimal kinodynamic algorithms typically rely on serial dynamic programming approaches. Their inherently sequential nature significantly limits their real-time applicability and scalability in complex robotic scenarios.

Given the recent stagnation in improvements to CPU core clock speeds, substantial advancements in near-optimal kinodynamic planning necessitate a fundamental redesign of existing algorithms. We therefore propose developing new SBMPs specifically optimized for manycore processors that can leverage the massive parallelism offered by modern hardware.

However, designing algorithms suitable for highly parallel architectures poses significant challenges. Traditional near-optimal kinodynamic SBMPs employ sequential dynamic programming steps—such as selecting a minimum-cost node, extending this node, evaluating its inclusion into the tree, and pruning existing nodes—that do not directly translate to efficient concurrent execution.

In this chapter, we present *Kino-PAX**, a massively parallel near-optimal kinodynamic SBMP

designed for efficient execution on modern manycore processors. *Kino-PAX** delivers **real-time**, **low-cost** initial solutions and guarantees asymptotic **near-optimality** over extended planning durations. We provide detailed pseudocode for *Kino-PAX**, discuss hyperparameter tuning options to achieve desired behaviors, and present a proof of asymptotic near-optimality and δ-robust completeness. Furthermore, we benchmark *Kino-PAX** to demonstrate its effectiveness across problems of varying difficulty. Finally, we thoroughly describe our implementation of *Kino-PAX** on CUDA-enabled GPUs.

3.2 Problem Formulation

Consider a robotic system operating under the assumptions outlined in Section 2.2. In addition to solving Problem 1, we are interested in computing trajectories that are near-optimal with respect to a specified cost metric. Recall from Section 2.2 and Eq. 2.1 that a system's trajectory is denoted by $\mathbf{x}$. We denote the set of all possible trajectories by $\mathbf{X}$ and define a cost function on this set as follows.

Definition 2 (Cost Function)**.** *The cost function* $cost : \mathbf{X} \to \mathbb{R}_{\geq 0}$ *maps each system trajectory* $\mathbf{x} \in \mathbf{X}$ *to a non-negative real number* $cost(\mathbf{x}) \geq 0$*.*

To formally define the near-optimal motion planning problem, we make assumptions about the cost function, similar to [78]:

Assumption 1 (Lipschitz Continuous Cost Function)**.** *The cost function* $cost(\mathbf{x})$ *of a trajectory is assumed to be Lipschitz continuous. Specifically, there exists* $K_c > 0$ *such that:*

$$|cost(\mathbf{x}_1) - cost(\mathbf{x}_2)| \leq K_c \cdot \sup_{\forall t} \|\mathbf{x}_1(t) - \mathbf{x}_2(t)\|,$$

for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathbf{X}$ *with the same start state. Consider two trajectories* $\mathbf{x}_1, \mathbf{x}_2$ *such that their concatenation is* $\mathbf{x}_1|\mathbf{x}_2$ *(i.e., following trajectory* $\mathbf{x}_2$ *after trajectory* $\mathbf{x}_1$*), the cost function satisfies:*

- $cost(\mathbf{x}_1|\mathbf{x}_2) = cost(\mathbf{x}_1) + cost(\mathbf{x}_2)$ *(additivity)*
- $cost(\mathbf{x}_1) \leq cost(\mathbf{x}_1|\mathbf{x}_2)$ *(monotonicity)*

- $\forall t_2 \geq t_1 \geq 0, \exists M_c > 0, t_2 - t_1 \leq M_c \cdot |cost(\mathbf{x}(t_1)) - cost(\mathbf{x}(t_2))|$ *(non-degeneracy)*

In addition, we assume that the system dynamics satisfy Chow's condition [13], which implies that the system is small-time locally accessible.

Next, we adopt the notion of a δ-robust trajectory from [78]:

Definition 3 (δ-Robust Trajectory)**.** *A trajectory* $\mathbf{x}$*, defined by Eq.* (2.2)*, is* δ*-robust if both its obstacle clearance* ϵ *(the minimum distance from obstacles along the trajectory) and dynamic clearance* δ_0 *(the maximum permissible perturbation in states preserving feasibility) exceed a positive threshold* δ*.*

Given the assumption of at least one δ-robust trajectory's existence, we formally define the δ-Robust Feasible Motion Planning problem:

Definition 4 (δ-Robust Feasible Motion Planning)**.** *Given a dynamical system following Eq. 2.1, the collision-free subset* $X_{valid} \subset X$*, an initial state* $x_0 \in X_{valid}$*, a target region* $X_{goal} \subset X_{valid}$*, and that a* δ*-robust trajectory that connects* x_0 *with a state in* X_{goal} *exists, find a solution trajectory* $\mathbf{x}$ *for which* $\mathbf{x}(0) = x_0$ *and* $\mathbf{x}(t_f) \in X_{goal}$*.*

Under these assumptions, we present definitions for a kinodynamic SBMP that probabilistically guarantees eventual feasibility and asymptotically approaches near-optimality. These definitions are adapted from [78]:

Definition 5 (Near-Optimal Trajectory)**.** *Let* c^* *denote the minimum achievable cost among all* δ*-robust solution trajectories to a* δ*-Robust Feasible Motion Planning problem. A trajectory* $\mathbf{x}$ *with cost* c *is considered near-optimal if it satisfies:*

$$c \leq h(c^*, \delta),$$

where the function h *is defined as:*

$$h(c^*, \delta) = (1 + \alpha \cdot \delta) \cdot c^*,$$

for some constant $\alpha \geq 0$*.*

Definition 6 (Asymptotic δ-robust Near-Optimality)**.** *Consider a δ-robust feasible motion planning problem with minimum achievable cost c^* as defined in Definition 5. Let Y_n^{ALG} denote a random variable representing the minimum cost among all trajectories returned by algorithm ALG after iteration n. The algorithm ALG is asymptotically δ-robust near-optimal if it satisfies:*

$$\mathbb{P}\left(\limsup_{n\to\infty} Y_n^{ALG} \leq h(c^*, \delta)\right) = 1,$$

where $h(c^*, \delta)$ is as defined in Definition 5.

Definition 7 (Probabilistic δ-Robust Completeness)**.** *Let $\mathbf{X}_n^{ALG}$ denote the set of trajectories discovered by an algorithm ALG at iteration n. Algorithm ALG is probabilistically δ-robustly complete, if for any δ-robustly feasible motion planning problem $(X_{valid}, x_0, X_{goal}, \delta)$ the following holds:*

$$\liminf_{n\to\infty} \mathbb{P}\left(\exists \mathbf{x} \in \mathbf{X}_n^{ALG} : \mathbf{x} \text{ solution to } (X_{valid}, x_0, X_{goal}, \delta)\right) = 1.$$

In this work, we address the problem of asymptotic δ-robust near-optimal kinodynamic motion planning, with an emphasis of computational efficiency achieved through parallelization. Formally, the problem is defined as follows:

Problem 2 (Near-Optimal Kinodynamic Motion Planning)**.** *Consider a robot with dynamics described by Eq.* (2.1) *operating in workspace W containing an obstacle set $\mathcal{O}$. Given an initial state $x_{init} \in X_{valid} \subseteq X$, a goal region $X_{goal} \subseteq X_{valid}$, and a cost function satisfying Assumption 1, the objective is to* ***efficiently*** *compute a control trajectory $\mathbf{u} : [0, t_f] \to U$ whose induced trajectory $\mathbf{x}$ solves Problem 1 and meets the criteria for* ***near-optimality*** *as defined in Definition 5.*

3.3 Kino-PAX*

Our approach to Problem 2 involves designing a highly parallel algorithm specifically designed for the many-core architecture of GPU-like processors. Building upon principles introduced in Section 2.3, this approach adheres to the guidelines from [51, 96], emphasizing thread independence, balanced workloads across threads, and efficient use of low-latency memory. Unlike *Kino-PAX*, which focuses on rapidly solving Problem 1 and terminating, our proposed method aims not only to

quickly identify low-cost initial solutions but also to iteratively refine these solutions over extended planning durations.

Motivated by these concepts, we introduce **Near Optimal Kinodynamic Parallel Accelerated eXpansion** (*Kino-PAX**), a highly parallel kinodynamic SBMP designed to efficiently solve Problem 2. *Kino-PAX** builds a sparse trajectory tree by decomposing the traditionally iterative operations—node selection, node extension, and node pruning—into three massively parallelized subroutines. Each subroutine is optimized for efficient execution on highly parallel processors, significantly reducing communication overhead such as CPU-GPU interactions.

During each iteration of *Kino-PAX**, multiple nodes from the existing tree are expanded concurrently. Each node undergoes multiple parallel extensions through random control sampling, with the number of extensions per node dynamically adjusted based on the current expansion set size and allotted planning time. After extensions are completed, non-promising nodes are pruned in parallel, and a new set of nodes is selected concurrently for expansion in the subsequent iteration.

*Kino-PAX** leverages a spatial decomposition $\mathcal{R}$ to guide the search process and systematically refine discovered solutions. This decomposition promotes thread independence during node addition and selection phases. By continuously tracking the lowest-cost node per region, *Kino-PAX** progressively refines cost estimates, selecting only the most promising nodes for further propagation. Consequently, the algorithm efficiently focuses computational effort on extending a select set of promising nodes into underexplored regions of the state space.

3.3.1 Core Algorithm

Here, we present a detailed description of *Kino-PAX**, accompanied by pseudocode in Algorithms 5, 6, 7, and 8, as well as an illustration of one cycle of *Kino-PAX** in Figure 3.1. *Kino-PAX** organizes tree nodes into four distinct sets: V_A, V_I, V_T, and V_U. Specifically, V_A contains nodes selected for parallel expansion (Algorithm 6); V_I includes temporarily inactive nodes that may be reactivated for expansion in future iterations (Algorithm 7); V_T comprises terminal nodes permanently pruned from the tree, and thus not considered in subsequent iterations (Algorithm 7); and

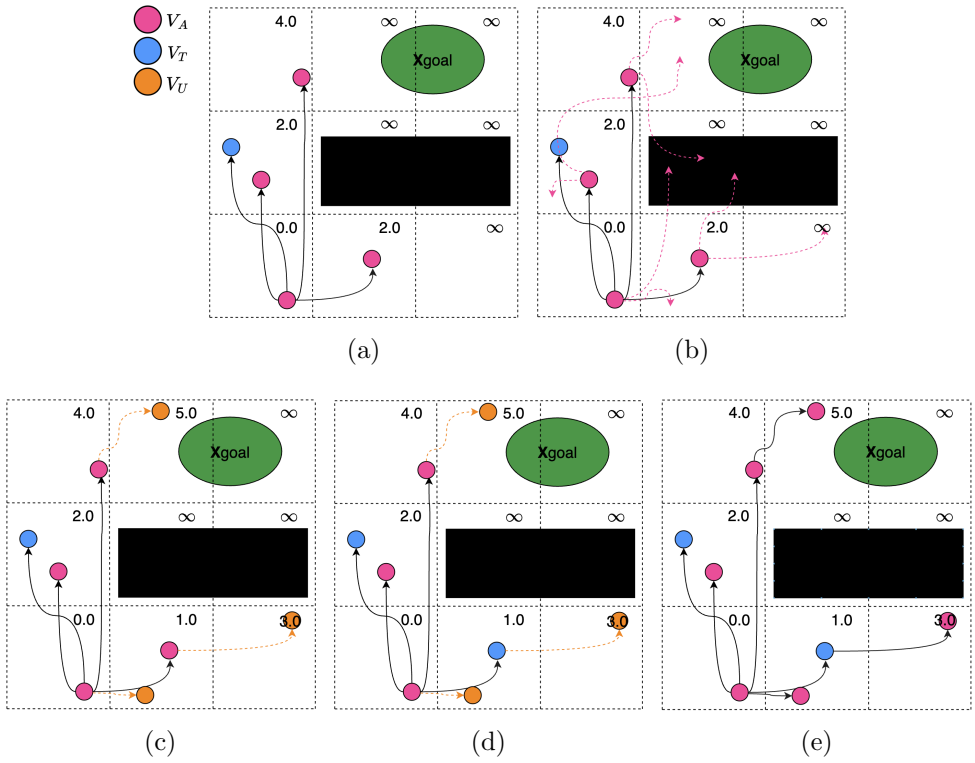


Figure 3.1: Illustration of the *Kino-PAX** expansion process: (a) Current node sets V_A and V_T. Numbers within grid cells indicate the lowest-cost trajectories reaching each region $\mathcal{R}_i$.
(b) Parallel expansion of nodes in V_A with branching factor $\lambda = 2$.
(c) Addition of selected nodes to the unexplored set V_U and corresponding updates to region costs.
(d) Pruning of nodes from V_A based on newly acquired trajectory cost information.
(e) Updated active set V_A, ready for the next iteration of node expansion.

finally, V_U consists of newly generated promising nodes produced during node propagation (Algorithm 6). Additionally, *Kino-PAX** partitions the state space into distinct hyper-cube regions, denoted by $\mathcal{R}$, and maintains the lowest-cost node within each region to guide efficient exploration.

3.3.1.1 Initialization

In Algorithm 5, *Kino-PAX** takes as input an initial state $x_{\text{init}} \in X_{\text{valid}}$, a goal region $X_{\text{goal}} \subseteq X_{\text{valid}}$, a maximum execution time $t_{\max}$, a branching factor λ, and a maximum inactive node count $I_{\max}$. In lines 1–2, the initial state x_{init} is set as the root node of the tree $\mathcal{T}$ and placed into the active set V_A. The inactive set V_I, terminal set V_T, and unexplored set V_U are initialized as empty. Lines 3–4 initialize each region in the spatial decomposition $\mathcal{R}$ with infinite cost, while the solution

Algorithm 5: *Kino-PAX**

Input : $x_{\text{init}}, X_{\text{goal}}, t_{max}, \lambda, I_{max}$
Output: Solution trajectory $\mathbf{x}$
1 $\mathcal{T} \leftarrow$ Initialize tree with root node x_{init}
2 $V_A \leftarrow \{x_{\text{init}}\}$, $V_I, V_U, V_T \leftarrow \emptyset$
3 Initialize $\mathcal{R}$ with $cost(\mathcal{R}_i) = \infty$ for each $\mathcal{R}_i \in \mathcal{R}$
4 Initialize $\mathbf{x} = null$ with $cost(\mathbf{x}) = \infty$
5 **while** $ElapsedTime < t_{max}$ **do**
6 `Propagate`$(V_A, V_U, \mathcal{R}, \lambda)$
7 `PruneNodes`$(V_A, V_I, V_T, \mathcal{T}, \mathcal{R})$
8 `UpdateTree`$(V_A, V_U, \mathcal{T}, \mathcal{R}, \mathbf{x})$
9 **return x**

trajectory $\mathbf{x}$ is initialized as `null` with infinite cost.

3.3.1.2 Node Extension

After initialization, the main loop of *Kino-PAX** begins (Algorithm 5, Lines 5–8). In each iteration, the `Propagate` subroutine (Algorithm 6, Figures 3.1a and 3.1b) propagates nodes in the active set V_A using massive parallelism. The inputs to `Propagate` include V_A, V_U, $\mathcal{R}$, and λ. Specifically, each node $x \in V_A$ undergoes λ parallel expansions, resulting in a total of $\lambda \cdot |V_A|$ parallel threads, where $|V_A|$ denotes the cardinality of V_A. For each thread, a random control $u \in U$ and a duration $dt \in (0, T_{\text{prop}}]$ are sampled, with T_{prop} being a user-defined maximum propagation time. The node's state x is then propagated using the system dynamics defined in Eq. (2.1), generating a new candidate state x' (Algorithm 6, Lines 3–4, Figure 3.1b).

Subsequently, the validity of the trajectory segment between states x and x' is evaluated against X_{valid}. If the trajectory segment is valid, the spatial region $\mathcal{R}_i$ corresponding to the new state x' is identified (Algorithm 6, Line 6). The incremental cost from x to x' is computed according to the user-defined cost metric satisfying Assumption 1, and the cumulative cost from the root x_{init} to x' is updated accordingly (Algorithm 6, Line 7).

If the new state x' improves upon the previously recorded cost for region $\mathcal{R}_i$, the cost associated with that region is updated (Algorithm 6, Line 8). Furthermore, if x' remains the lowest recorded cost for region $\mathcal{R}_i$, it is added to the unexplored node set V_U (Algorithm 6, Lines 9–

Algorithm 6: Propagate

Input : $V_A, V_U, \mathcal{R}, \lambda$
Output: Updated V_U

foreach $x \in V_A$ **do**
  for $i = 1, \ldots, \lambda$ **do**
    Randomly sample u and dt ;
    $x' \leftarrow$ `PropagateODE`(x, u, dt) ;
    if the trajectory from x **to** x' **is valid then**
      Map x' to region $\mathcal{R}_i$;
      $cost(x') \leftarrow cost(x) + getCost(x, x')$;
      $cost(R_i) \leftarrow min(cost(R_i), cost(x')$;
      if $cost(x') == cost(R_i)$ **then**
        Add x' to V_U ;

10, Figure 3.1c). This selective admission criterion ensures that nodes added to V_U consistently improve upon the best trajectories found, guiding the search toward near-optimal solutions while maintaining memory efficiency.

Remark 2. *In our implementation, to avoid data races when updating the region cost for $\mathcal{R}_i$, we perform the cost-update operation atomically. Atomic operations ensure an ordering constraint on memory accesses, avoiding degenerate behavior. Section 4.3 provides further explanation of atomic operations and highlights their significance in concurrent programming.*

Remark 3. *Since the* `Propagate` *routine executes as a parallel process, a thread may insert a node into the unexplored set V_U even if another thread later identifies a lower-cost trajectory to the same region. The higher-cost nodes introduced initially are removed during the* `TreeUpdate` *(Algorithm 8) subroutine, ensuring only the lowest-cost nodes remain in the tree $\mathcal{T}$.*

Remark 4. *If collision checking requires significant computational resources, particularly in obstacle-dense environments, performance can be improved by first comparing the cost to reach x' against the lowest recorded cost for region $\mathcal{R}_i$. Collision checking is then performed only if x' is the best recorded node in $\mathcal{R}_i$.*

3.3.1.3 Node Selection

After propagating nodes in V_A, the algorithm executes the `PruneNodes` subroutine (Algorithm 5, Line 7; Algorithm 7), which classifies each node $x \in \mathcal{T}$ as active, inactive, or terminal based on newly acquired search information. This subroutine operates as a massively parallel procedure, assigning one thread per node.

Each thread begins by mapping its node x to its corresponding region $\mathcal{R}_i$ (Algorithm 7, Line 2). If a node $x \in V_I$ (currently inactive) no longer represents the lowest cost in its region, it is moved permanently to the terminal set V_T (Algorithm 7, Lines 10–12). Conversely, if $x \in V_I$ remains the lowest-cost node in its region, its inactivity counter $I_{\text{count}}(x)$ is incremented (Algorithm 7, Lines 3–4). If this counter exceeds a user-defined threshold $I_{\max}$, indicating prolonged inactivity without improvement in its region, node x is reactivated and returned to the set V_A for further expansion (Algorithm 7, Lines 5–7). This mechanism ensures essential nodes re-enter active exploration, maintaining completeness.

Nodes currently in V_A are also evaluated for pruning. A node $x \in V_A$ is pruned and moved to V_T if its trajectory cost exceeds the lowest recorded cost for its region (Algorithm 7, Lines 10–12). Alternatively, if node x has the lowest cost within its region but any of its parent nodes exceeds the minimum recorded cost for their respective regions, node x is moved to the set V_I (Algorithm 7, Lines 13–15, Figure 3.1d). This selective pruning strategy enables *Kino-PAX** to concentrate computational resources on expanding a focused subset of promising nodes with a higher branching factor.

After pruning, the `UpdateTree` subroutine is invoked (Algorithm 5, Line 8; Algorithm 8). This subroutine concurrently removes nodes in V_U that are not the minimum-cost nodes in their respective regions, adds the remaining nodes in V_U to $\mathcal{T}$, and checks whether a better solution has been found. `UpdateTree` receives as input $\mathcal{T}$, V_A, V_U, and $\mathcal{R}$, with each thread responsible for processing a single node from V_U. Each thread retrieves the corresponding region $\mathcal{R}_i$ for its node x (Algorithm 8, Line 2). If x represents the lowest-cost node to reach region $\mathcal{R}_i$, it is added to

Algorithm 7: PruneNodes

Input : $\mathcal{T}, V_A, V_I, V_T, \mathcal{R}$
Output: Updated V_A, V_I, V_T

```
1  foreach x ∈ T do
2      Map x to region R_i ;
3      if cost(x) == cost(R_i) and x ∈ V_I then
4          Increment I_count(x);
5          if I_count(x) > I_max then
6              Add x to V_A;
7              return;
8      if I_count(x) > I_max and cost(x) == cost(R_i) then
9          return;
10     if cost(x) > cost(R_i) then
11         Add x to V_T;
12         return;
13     if any parent of x, x_p, has cost(x_p) > cost(R_p) then
14         Add x to V_I;
15         return;
```

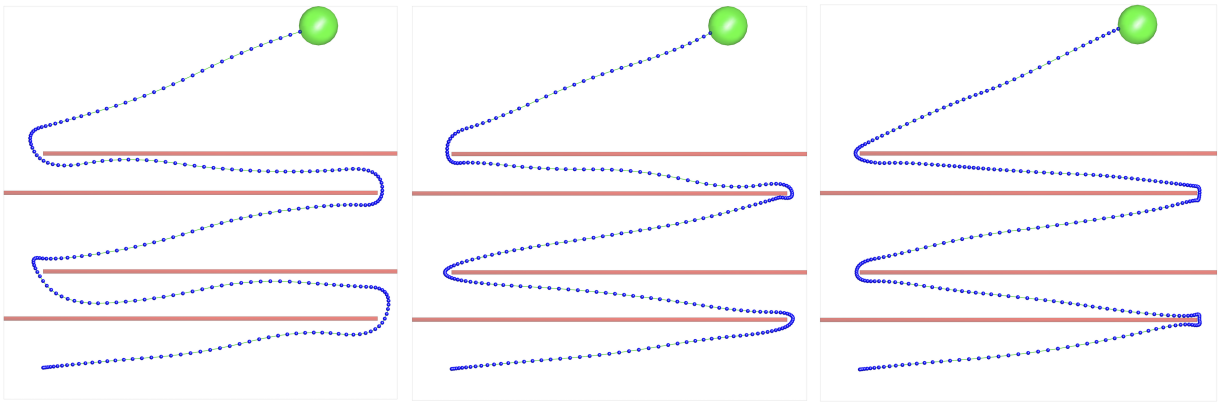
both V_A and $\mathcal{T}$ (Algorithm 8, Lines 3–4, Algorithm 3.1e). Subsequently, if node x satisfies the goal criteria and its cost improves upon the current best-known solution, the solution trajectory $\mathbf{x}$ and its associated cost are updated accordingly (Algorithm 8, Lines 5–7).

*Kino-PAX** repeats the main loop of `Propagate`, `PruneNodes`, and `UpdateTree` until the user-defined time limit $t_{\max}$ is exceeded, at which point the best-found solution trajectory $\mathbf{x}$ is returned. Figure 3.2 illustrates how *Kino-PAX** progressively improves the quality of its solution over time.

Algorithm 8: UpdateTree

Input : $\mathcal{T}, V_A, V_U, \mathcal{R}$
Output: Updated $V_A, \mathcal{T}, \mathbf{x}$

```
foreach x ∈ V_U do
    Map x to region R_i ;
    if cost(x) == cost(R_i) then
        Add x to V_A and T;
        if x ∈ X_goal and cost(x) < cost(x) then
            x ← trajectory from x_init to x;
            cost(x) ← cost(x);
```



(a) First Solution (b) Intermediate Solution (c) Final Solution

Figure 3.2: Three solutions found by *Kino-PAX** using 6D Double Integrator dynamics and minimizing path length cost. Subfigure (a) is the first solution found, subfigure (b) is an intermediate solution, subfigure (c) is the final solution found.

3.4 Tuning Parameters

3.4.1 Expected tree size tuning

In practice, the expected tree size t_e is a fixed hyperparameter that specifies the maximum number of nodes allowed in $\mathcal{T}$ and sets bounds for the sizes of V_A, V_I, V_T, and V_U. Fixing t_e improves the computational efficiency of *Kino-PAX** by eliminating high-latency operations associated with dynamic memory resizing on many-core devices. Additionally, a fixed tree size enables users to precisely tune algorithm performance, balancing the trade-off between runtime

and solution quality.

Adjusting the value of t_e affects *Kino-PAX**'s behavior in two ways. First, it influences λ, which is calculated by:

$$\lambda = \text{floor}\left(\frac{t_e}{|V_A|}\right), \tag{3.1}$$

where $|V_A|$ denotes the cardinality of the active node set. This relation ensures maximal propagation of each active node within a single iteration of the `Propagate` subroutine while guaranteeing that `Propagate` does not generate more than t_e new nodes, which would result in a segmentation fault. Increasing t_e leads to the generation of more trajectory segments (Algorithm 6), potentially yielding lower-cost trajectories. However, higher t_e values require greater computational resources and additional memory to accommodate larger node sets and trees, ultimately increasing iteration times. This trade-off is examined in detail in experiments in Section 3.6.

Secondly, t_e controls the total number of nodes stored within $\mathcal{T}$. For complex problems, particularly those involving high, dimensional systems—a larger t_e is recommended to enable sufficient exploration of the state space and identification of high-quality solutions. Due to *Kino-PAX**'s structured search strategy, hyperparameter tuning is quite straightforward. This is demonstrated in Section 3.6, where we consistently achieve high-quality solutions across a wide range of t_e values and problem configurations.

3.4.1.1 Space Decomposition Tuning

*Kino-PAX** also allows tuning through its spatial decomposition $\mathcal{R}$, where adjusting the fineness of the decomposition changes algorithm behavior.

With a coarser decomposition, fewer nodes are accepted into $\mathcal{T}$, as multiple nodes are more likely to fall within the same region. This configuration improves memory efficiency by enabling solutions to be found with a relatively small t_e. Moreover, a coarser discretization helps prevent *Kino-PAX** from exhausting memory prematurely, allowing the algorithm to run for a sufficient number of iterations to iteratively improve the solution. However, if the discretization is too coarse, it can hinder the algorithm's ability to propagate nodes between regions, potentially preventing the

discovery of feasible solutions.

Additionally, a finer decomposition significantly changes the algorithm's behavior. If the decomposition is excessively fine relative to a small t_e, newly generated nodes frequently populate unvisited regions, rapidly exhausting available memory before sufficient iterations can be executed. In contrast, pairing a fine decomposition with a large t_e enables high-quality solutions and sufficient iterations but increases per-iteration runtime, delaying the discovery of the first solution. As shown in Section 3.6, the spatial decomposition in *Kino-PAX** can be tuned effectively, with quality solutions obtained across a wide range of decompositions and environment complexities for a given dynamical system.

3.5 Analysis

In this section, we establish that *Kino-PAX** achieves asymptotic δ-robust near-optimality (Definition 6) and probabilistic δ-robust completeness (Definition 7). Our analysis extends the results of Li et al. [78], who introduced and proved these properties for the SST algorithm. We begin by recalling Definition 14 from [78], describing a covering ball sequence along a trajectory $\mathbf{x}$:

Definition 8 (Covering Balls)**.** *Given a trajectory $\mathbf{x}(t) : [0, T] \to X_{valid}$, robust clearance $\delta \in \mathbb{R}^+$, and cost increment $C_\Delta > 0$, the covering ball sequence $\mathbb{B}(\mathbf{x}(t), \delta, C_\Delta)$ is a set of $M + 1$ hyper-balls $\{\mathbb{B}(\delta, x_0), \mathbb{B}(\delta, x_1), \ldots, \mathbb{B}(\delta, x_M)\}$, each of radius δ, where the centers x_i are defined such that $Cost(x_i \to x_{i+1}) = C_\Delta$ for all $i = 0, 1, \ldots, M - 1$.*

Next, we reference Theorem 17 from Li et al. [78], which characterizes the reachable set between two covering balls for systems satisfying the dynamics assumptions outlined in Section 3.2. This formally proven result underpins our analysis of *Kino-PAX**:

Theorem 2 (Reachability between Covering Balls)**.** *Given a trajectory $\mathbf{x}$ of duration $t_\mathbf{x}$, the probability that MonteCarlo-Prop generates a δ-similar trajectory $\mathbf{x}'$ from an initial state $\mathbf{x}'(0) \in \mathbb{B}_\delta(\mathbf{x}(0))$ to a final state within the ball $\mathbb{B}_\delta(\mathbf{x}(t_\mathbf{x}))$, with propagation duration $t_{\mathbf{x}'} = T_{prop} > t_\mathbf{x}$, is lower bounded by a strictly positive probability $\rho_\delta > 0$.*

This theorem is essential for our analysis as it guarantees that any state within the initial ball $\mathbb{B}_\delta(\mathbf{x}(0))$ can probabilistically reach any state within the final ball $\mathbb{B}_\delta(\mathbf{x}(t_\mathbf{x}))$ via a trajectory $\mathbf{x}'$ that remains within distance δ of the original trajectory $\mathbf{x}$. Furthermore, it establishes that if the reference trajectory $\mathbf{x}$ is optimal from $\mathbf{x}(0)$ to $\mathbf{x}(t_\mathbf{x})$, then the trajectory $\mathbf{x}'$ meets the criteria of δ-optimality as defined in Definition 5. A visual illustration of this property is provided below.

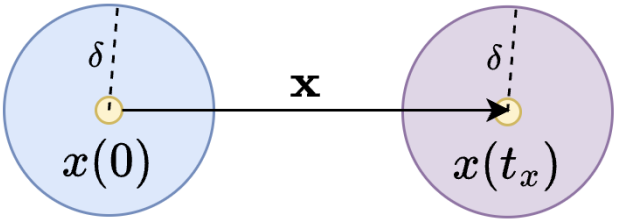


Figure 3.3: Illustration of Theorem 2. The trajectory $\mathbf{x}$ is shown with the starting state $\mathbf{x}(0)$ and the ending state $\mathbf{x}(t_\mathbf{x})$, each surrounded by a δ-sized ball. The blue region represents an arbitrary state within a δ distance of $\mathbf{x}(0)$, and the purple region indicates the states within a δ distance of $\mathbf{x}(t_\mathbf{x})$ that are reachable by $\mathbf{x}'_0$ while remaining δ-near $\mathbf{x}$ throughout the propagation of $\mathbf{x}'$.

Before assessing the properties of *Kino-PAX**, we define the set of regions $\mathcal{R}^*$ associated with a given δ-robust optimal trajectory $\mathbf{x}^*$. This trajectory is characterized by a sequence of covering balls $\mathbb{B}(\mathbf{x}^*(t), \delta, C_\Delta)$, each with radius δ, defined as the diagonal length of its corresponding region. Each ball $\mathbb{B}_i$ in the sequence is centered at state x_i^* located within a region $\mathcal{R}_i^* \in \mathcal{R}^*$. The set $\mathcal{R}^*$ consists of all regions containing the centers of these balls. A figure showing this setup can be found in Figure 3.4

Definition 9 (Optimal Trajectory Regions)**.** *Consider a space decomposition $\mathcal{R}$ where each region $\mathcal{R}_i \in \mathcal{R}$ has diagonal length $\delta \in \mathbb{R}^+$, an optimal trajectory $\mathbf{x}^*(t) : [0, T] \to X_{valid}$ with robust clearance δ, and the corresponding covering ball sequence $\mathbb{B}$. The set $\mathcal{R}^* \subseteq \mathcal{R}$ is defined as the collection of regions where each region $\mathcal{R}_i^* \in \mathcal{R}^*$ contains the center x_i^* of the corresponding covering ball $\mathbb{B}_i \in \mathbb{B}$.*

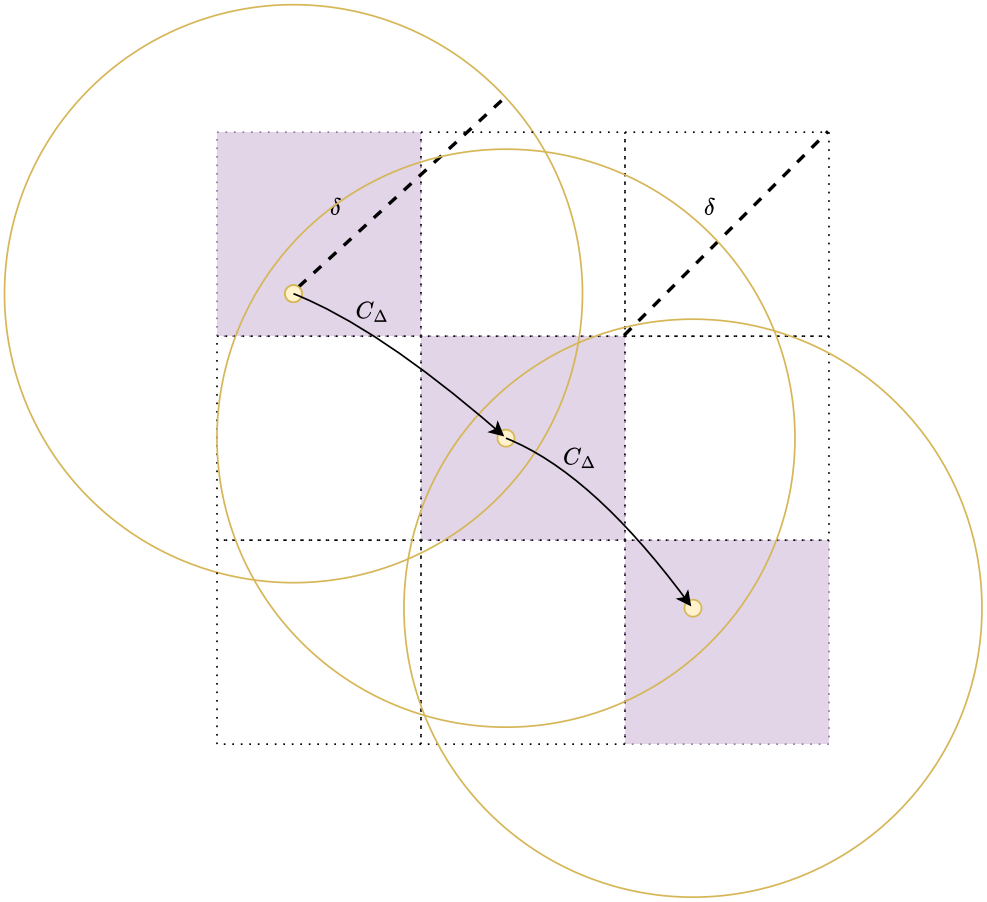


Figure 3.4: An illustration of a δ-robust optimal trajectory $\mathbf{x}^*$ (yellow nodes), with nodes equally spaced in terms of cost increments of C_Δ. Each node lies within a region $\mathcal{R}_i$, and the purple regions collectively form the set $\mathcal{R}^*$.

Using Definition 9, we now show that for any pair of consecutive regions $\mathcal{R}^*_{j-1}, \mathcal{R}^*_j \in \mathcal{R}^*$, any state $x_{j-1} \in \mathcal{R}^*_{j-1}$ can generate a trajectory reaching any state $x_j \in \mathcal{R}^*_j$, while remaining within a δ-distance of the optimal trajectory $\mathbf{x}^*$.

Lemma 2. *Given a δ-robust optimal trajectory $\mathbf{x}^*$ with associated optimal trajectory regions $\mathcal{R}^*$, the probability of successfully propagating an arbitrary state $x_{j-1} \in \mathcal{R}^*_{j-1}$ to an arbitrary state $x_j \in \mathcal{R}^*_j$, while maintaining a trajectory within δ distance of $\mathbf{x}^*$, is bounded below by a positive constant $\rho_{\mathcal{R}} > 0$.*

Proof. From Theorem 2 ([78]), we have that for any state in the covering ball $\mathbb{B}_{j-1}$, centered at

$x^*_{j-1} \in \mathcal{R}^*_{j-1}$, there exists a strictly positive probability ρ_δ of generating a δ-similar trajectory segment to any state within $\mathbb{B}_j$. By definition, the radius δ of the covering ball is equal to the diagonal length of the region $\mathcal{R}^*_{j-1}$, thus guaranteeing:

$$\mathcal{R}^*_{j-1} \subseteq \mathbb{B}_{j-1}.$$

Consequently, any state $x'_{j-1} \in \mathcal{R}^*_{j-1}$ also has a positive probability of generating a trajectory segment to any state within $\mathbb{B}_j$. Since we similarly have $\mathcal{R}^*_j \subseteq \mathbb{B}_j$, it follows that extending any state x'_{j-1} to any state $x'_j \in \mathcal{R}^*_j$ also occurs with strictly positive probability, which we denote by $\rho_{\mathcal{R}} > 0$. □

Finally, we present the main analytical result.

Theorem 3. *Kino-PAX* is asymptotically δ-robustly near-optimal.*

Proof. Given a δ-robust optimal trajectory $\mathbf{x}^*$ with corresponding optimal trajectory regions $\mathcal{R}^*$, we demonstrate that *Kino-PAX** asymptotically generates a δ-optimal trajectory. The proof proceeds by induction, showing that *Kino-PAX** will eventually generate a δ-optimal trajectory reaching any region $\mathcal{R}^*_i \in \mathcal{R}^*$. To achieve this, we analyze two exhaustive scenarios under which *Kino-PAX** extends a δ-optimal trajectory into each region $\mathcal{R}^*_i$ as the number of iterations goes to infinity.

First, define the events:

- $A^{(n)}_j$: On iteration n, *Kino-PAX** generates a trajectory $\mathbf{x}$ terminating in region $\mathcal{R}^*_j$ with cost satisfying $cost(\mathbf{x}_j) \leq h(cost(\mathbf{x}^*_j), \delta)$.
- $E^{(n)}_j$: By iteration n, *Kino-PAX** has generated at least one trajectory satisfying event $A^{(n)}_j$.

We begin by verifying the base cases $E^{(n)}_0$ and $E^{(n)}_1$.

Base Case ($E^{(n)}_0$ and $E^{(n)}_1$):

Since the root node of the tree $\mathcal{T}$ in *Kino-PAX** is $x_0 = x^*_0 \in \mathcal{R}^*_0$, event $E^{(n)}_0$ trivially occurs with probability 1 at the first iteration ($n = 1$). Furthermore, by Lemma 2, node x_0 can extend

δ-optimally to region $\mathcal{R}_1^*$ given infinite sampling opportunities. Since *Kino-PAX** repeatedly adds the lowest-cost node from each visited region to the active set V_A, node x_0 is expanded infinitely many times as $n \to \infty$. Hence:

$$\lim_{n\to\infty} \mathbb{P}(E_1^{(n)}) = 1.$$

Inductive Step (from $E_{j-1}^{(n)}$ to $E_j^{(n)}$):

Assume event $E_{j-1}^{(n)}$ holds; thus, *Kino-PAX** has produced a δ-optimal trajectory $\mathbf{x}_{j-1}$ terminating in $\mathcal{R}_{j-1}^*$, ensuring the presence of a node in $\mathcal{R}_{j-1}^*$ that will be repeatedly selected for expansion in subsequent iterations. By Lemma 2, infinite sampling guarantees that *Kino-PAX** generates a δ-optimal trajectory $\mathbf{x}_j$ containing a segment from $\mathcal{R}_{j-1}^*$ to $\mathcal{R}_j^*$ as $n \to \infty$. Two mutually exclusive scenarios for trajectory acceptance arise:

Case 1 (Acceptance): *Kino-PAX** directly accepts trajectory $\mathbf{x}_j$, as it achieves the lowest cost for its terminal region $\mathcal{R}_j^*$. Acceptance occurs with strictly positive probability $\gamma_1 > 0$.

Case 2 (Rejection): *Kino-PAX** rejects trajectory $\mathbf{x}_j$ due to the existence of another trajectory $\mathbf{x}_j'$ with a lower cost endpoint already in region $\mathcal{R}_j^*$. Since the rejected trajectory $\mathbf{x}_j$ is δ-optimal, the accepted trajectory $\mathbf{x}_j'$ must also be δ-optimal. Thus, even rejection results in having a δ-optimal trajectory in region $\mathcal{R}_j^*$, and this case occurs with strictly positive probability $\gamma_2 > 0$.

These cases exhaust all possibilities, ensuring:

$$\lim_{n\to\infty} \mathbb{P}(E_j^{(n)}) = \gamma_1 + \gamma_2 = 1.$$

By induction, event $E_j^{(n)}$ occurs with probability 1 for all trajectory segments indexed by j as $n \to \infty$. Therefore, *Kino-PAX** asymptotically achieves δ-robust near-optimality.

□

From Theorem 3, we further derive that *Kino-PAX** is probabilistically δ-robust complete, satisfying Definition 7:

Theorem 4. *Kino-PAX* is probabilistically δ-robust complete.*

Proof. Definition 7 states that an algorithm is probabilistically δ-robust complete if the probability that it fails to solve a δ-robustly feasible motion planning problem approaches zero as the number of iterations goes to infinity.

From Theorem 3, we have already demonstrated that if there exists a δ-robust optimal trajectory $\mathbf{x}^*$, then *Kino-PAX** asymptotically finds a δ-optimal solution trajectory $\mathbf{x}$ with probability 1 as $n \to \infty$. Consequently, given a δ-robustly feasible motion planning problem, the probability of *Kino-PAX** failing to find at least one valid solution trajectory approaches zero.

Thus, by satisfying the condition established in Definition 7, we conclude that *Kino-PAX** is probabilistically δ-robust complete.

□

3.6 Experiments

We evaluate *Kino-PAX**'s performance across four 3D environments using three dynamical systems. Three environments were introduced in Section 2.6 and depicted in Fig. 2.3, while a new fourth environment, shown in Fig. 3.5, specifically tests *Kino-PAX**'s capability for planning in tight corridors over extended horizons. The dynamical systems include: (i) a 6D double integrator, (ii) a 6D Dubins airplane [12], and (iii) a 12D nonlinear quadcopter [21].

For each dynamical system and environment, we benchmark *Kino-PAX** against our previous

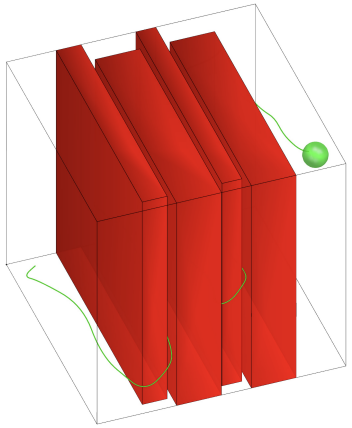


Figure 3.5: Environment d. Corridors

algorithm, *Kino-PAX*, and the traditional serial SBMP algorithm SST [78]. This comparison assesses *Kino-PAX**'s time to initial solution relative to *Kino-PAX* and evaluates its solution quality compared to SST.

We implemented *Kino-PAX** in CUDA C and benchmarked its performance alongside *Kino-PAX* on an NVIDIA RTX 4090 GPU with 16,384 CUDA cores and 24 GB of RAM. The serial SBMP SST, implemented in C++ using OMPL [109], ran on an Intel Core i9-14900K CPU (base clock 4.4 GHz, 128 GB RAM). Our implementation of *Kino-PAX** is publicly available at `https://github.com/aria-systems-group/Kino-PAX`.

All algorithms were tuned for each dynamical system, maintaining consistent parameters across environments. Identical state propagation, validity checking, and node cost calculation methods were used for fairness. Each combination of algorithm, dynamical system, and environment was evaluated in over 100 queries, minimizing the cost of Euclidean distance, with a maximum runtime of 300 seconds per query.

We evaluated two tuning strategies for *Kino-PAX**: *Kino-PAX* -large-δ* and *Kino-PAX* -small-δ*. The *Kino-PAX* -large-δ* strategy uses coarser decomposition and smaller expected tree sizes, emphasizing rapid initial solutions and early termination. Specifically, for the 6D double integrator: $t_e = 100,000$ with 27,000 regions; for the 6D Dubins airplane: $t_e = 300,000$ with 52,000 regions; and for the 12D nonlinear quadcopter: $t_e = 1,000,000$ with 100,000 regions. Conversely, the *Kino-PAX* -small-δ* strategy uses a finer decomposition and larger tree sizes ($t_e = 100,000,000$ with 10,000,000 regions for all systems), focusing on extensive state space exploration.

3.6.1 Benchmark Results

Tables 3.1, 3.2, and 3.3 summarize results, presenting median initial solution time (IT), normalized median initial solution cost (IC), median final solution time (FT), normalized median final solution cost (FC), and the success rate across 100 queries.

Table 3.1: Benchmark results over 100 trials with a 300-second maximum planning time for the 6D double integrator system.

Algorithm	Device	6D Double Integrator				
		IT (ms)	IC (L2)	FT (ms)	FC (L2)	Succ %
Environment a						
SST	CPU	1950.0	1.0	165201.5	0.79	100.0
Kino-PAX	GPU	3.7	0.70	4.0	0.70	100.0
Kino-PAX -large-δ*	GPU	3.0	0.72	6.2	0.68	100.0
Kino-PAX -small-δ*	GPU	1173.0	0.68	180017.5	0.64	100.0
Environment b						
SST	CPU	3525.0	1.0	97060.0	0.78	100.0
Kino-PAX	GPU	2.8	0.5	2.8	0.49	100.0
Kino-PAX -large-δ*	GPU	2.2	0.5	5.3	0.47	100.0
Kino-PAX -small-δ*	GPU	1002.2	0.48	252173.0	0.44	100.0
Environment c						
SST	CPU	5225.5	1.0	131089.5	0.84	100.0
Kino-PAX	GPU	5.1	0.62	5.4	0.62	100.0
Kino-PAX -large-δ*	GPU	4.7	0.62	8.6	0.56	100.0
Kino-PAX -small-δ*	GPU	1556.0	0.57	133854.0	0.48	100.0
Environment d						
SST	CPU	34760.0	1.0	146466.5	0.95	100.0
Kino-PAX	GPU	7.7	0.83	8.5	0.83	100.0
Kino-PAX -large-δ*	GPU	10.6	0.79	19.9	0.73	100.0
Kino-PAX -small-δ*	GPU	2698.1	0.72	91483.6	0.63	100.0

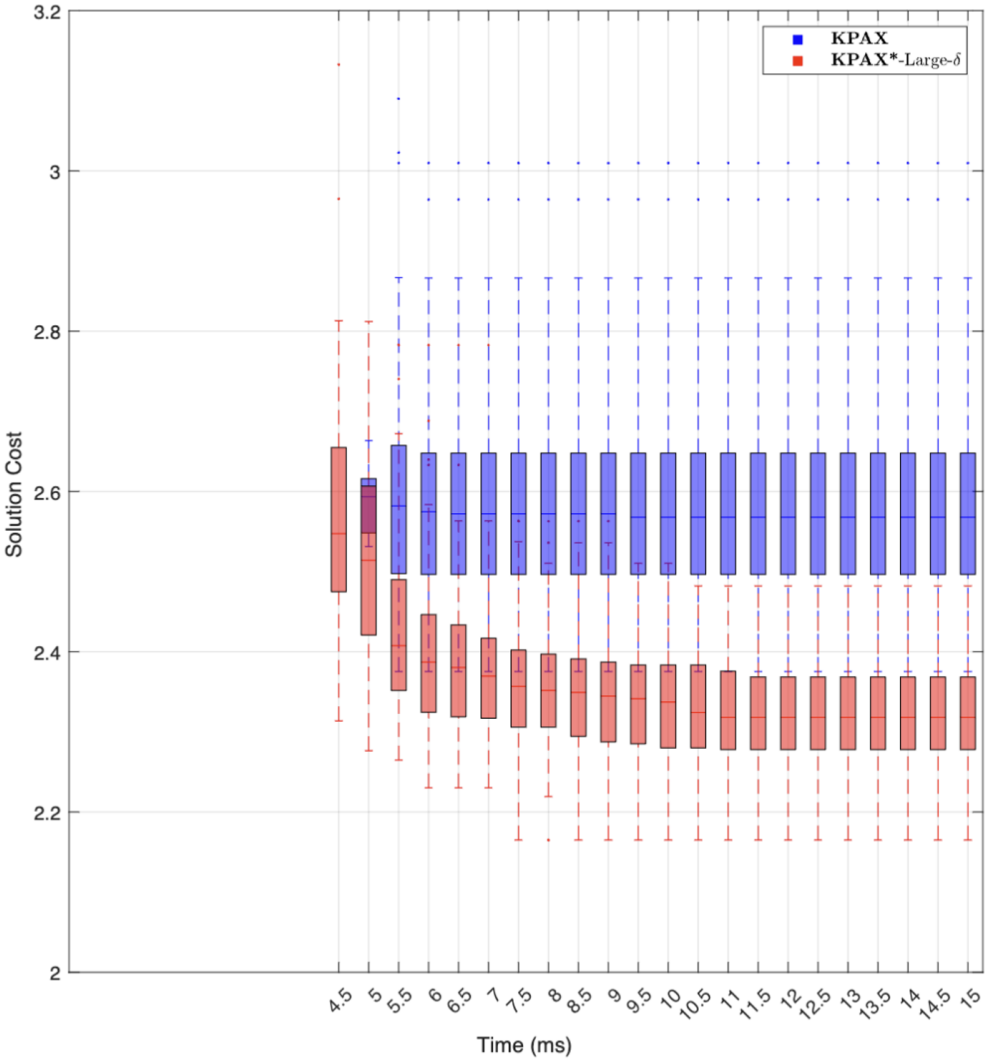


Figure 3.6: Comparison of planning performance between *Kino-PAX* and *Kino-PAX* -large-δ* on the 6D double integrator system in the house environment over a 15 ms planning duration.

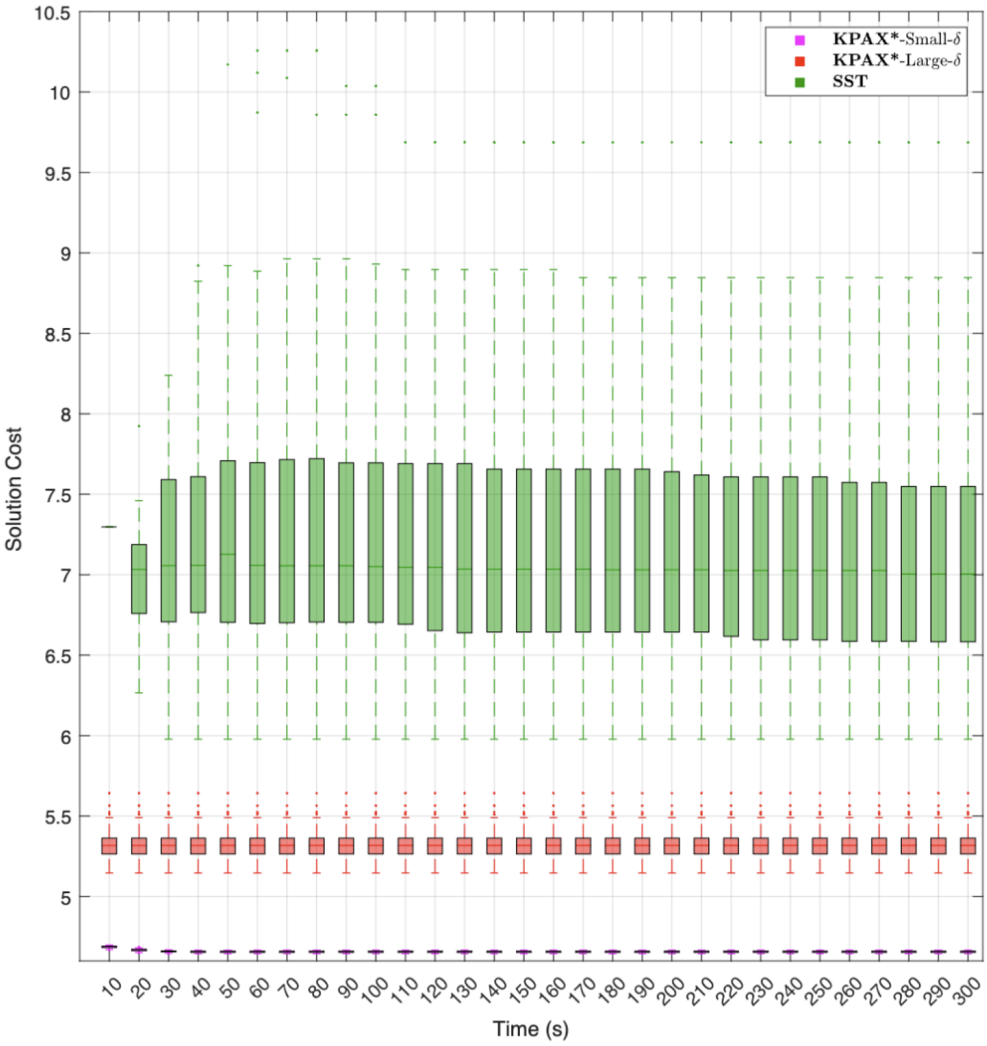


Figure 3.7: Planning performance of *Kino-PAX* -small-δ*, *Kino-PAX* -large-δ*, and SST on the 6D double integrator system in the house environment over a 300 s planning duration.

Table 3.1 shows that, for the 6-D double-integrator, *Kino-PAX* -large-δ* reaches an initial solution in under 11, ms across all environments, comparable to *Kino-PAX*, which does so in under 8, ms. A comparison of their initial solution times and solution quality over time is shown in Figure 3.6. Relative to SST, *Kino-PAX* -large-δ* finds the first solution 650×, 1600×, 1100×, and 3300× faster in environments a–d, respectively. On average, *Kino-PAX* -large-δ*'s initial solution cost is 0.65% of SST's first-solution cost. After 10, ms of planning, *Kino-PAX* -large-δ* converges to a local optimum with an average cost that is 0.61% of SST's initial solution cost. In comparison,

the fine-grained tuning variant, *Kino-PAX* -small-δ*, converges to a local optimum after an average of 160, s of planning, achieving an average cost that is 0.55% of SST's initial cost. Meanwhile, SST, after the full five-minute duration, improves only to 0.84% of its own initial cost. A visualization comparing SST with *Kino-PAX* -large-δ* and *Kino-PAX* -small-δ* is shown in Figure 3.7.

Table 3.2: Benchmark results over 100 trials with a 300-second maximum planning time for the Dubins Airplane system.

Algorithm	Device	Dubins Airplane				
		IT (ms)	IC (L2)	FT (ms)	FC (L2)	Succ %
Environment a						
SST	CPU	10341.0	1.0	182846.0	0.91	100.0
Kino-PAX	GPU	4.2	0.86	4.2	0.86	100.0
Kino-PAX -large-δ*	GPU	6.1	0.87	14.5	0.77	100.0
Kino-PAX -small-δ*	GPU	1559.7	0.81	89458.9	0.74	100.0
Environment b						
SST	CPU	24294.5	1.0	141664.0	0.83	100.0
Kino-PAX	GPU	3.2	0.67	3.2	0.67	100.0
Kino-PAX -large-δ*	GPU	4.7	0.65	12.0	0.61	100.0
Kino-PAX -small-δ*	GPU	1419.5	0.62	56694.8	0.58	100.0
Environment c						
SST	CPU	154149.0	1.0	218667.0	0.97	87.0
Kino-PAX	GPU	7.5	0.94	7.5	0.94	100.0
Kino-PAX -large-δ*	GPU	13.4	0.91	34.8	0.80	100.0
Kino-PAX -small-δ*	GPU	2274.8	0.75	250972.0	0.64	100.0
Environment d						
SST	CPU	NaN	NaN	NaN	NaN	0.0
Kino-PAX	GPU	NaN	NaN	NaN	NaN	0.0
Kino-PAX -large-δ*	GPU	NaN	NaN	NaN	NaN	0.0
Kino-PAX -small-δ*	GPU	4320.7	1.0	71019.9	0.91	100.0

Continuing to the Dubins Airplane system in Table 3.2, we see largely similar results, where

Kino-PAX -large-δ* achieves initial solutions on average 6000× faster than SST, with initial solution costs that are 78% of SST's first solution cost. *Kino-PAX* -large-δ* converges to solutions that are 72% of SST's first solution cost, while *Kino-PAX* -small-δ* converges to solutions that are 65% of the cost. Notably, in environment (d), only *Kino-PAX* -small-δ* is able to find a valid solution within the planning time. This environment posed a particularly challenging problem due to the Dubins Airplane system's large turning radius and the presence of long, narrow passages with tight turns. Thanks to its fine discretization and large expected tree size (t_e), *Kino-PAX* -small-δ* is able to find an initial solution in 4.3 seconds and improve that solution by an additional 9% over the five-minute planning period.

Table 3.3: Benchmark results over 100 trials with a 300-second maximum planning time for the 12D Nonlinear Quadcopter system.

Algorithm	Device	12D Nonlinear Quadcopter				
		IT (ms)	IC (L2)	FT (ms)	FC (L2)	Succ %
Environment a						
SST	CPU	86931.0	1.0	218315.0	0.92	84.0
Kino-PAX	GPU	83.7	0.72	98.2	0.68	100.0
Kino-PAX -large-δ*	GPU	159.3	0.72	1764.5	0.65	100.0
Kino-PAX -small-δ*	GPU	4214.2	0.69	176232.0	0.62	100.0
Environment b						
SST	CPU	132363.0	1.0	221516.0	0.96	53.0
Kino-PAX	GPU	88.8	0.55	99.8	0.52	100.0
Kino-PAX -large-δ*	GPU	145.3	0.55	1396.9	0.48	100.0
Kino-PAX -small-δ*	GPU	4015.4	0.51	185098.5	0.45	100.0
Environment c						
SST	CPU	209123.0	1.0	271112.5	0.93	42.0
Kino-PAX	GPU	147.6	0.65	163.1	0.62	100.0
Kino-PAX -large-δ*	GPU	263.0	0.62	1406.6	0.53	100.0
Kino-PAX -small-δ*	GPU	5631.6	0.59	246833.0	0.47	100.0
Environment d						
SST	CPU	NaN	NaN	NaN	NaN	0.0
Kino-PAX	GPU	287.7	0.98	303.4	0.97	100.0
Kino-PAX -large-δ*	GPU	1038.1	1.0	2918.0	0.88	100.0
Kino-PAX -small-δ*	GPU	11504.0	0.93	221927.0	0.78	100.0

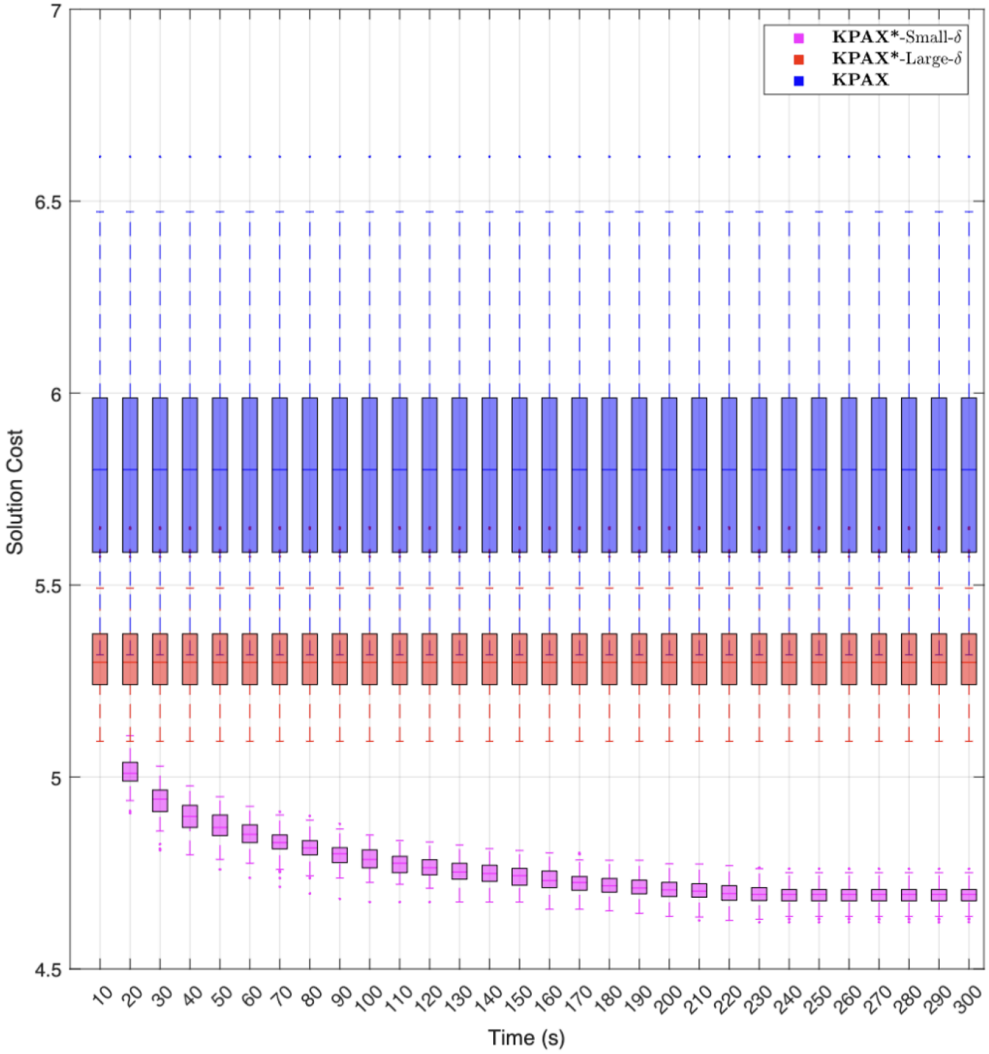
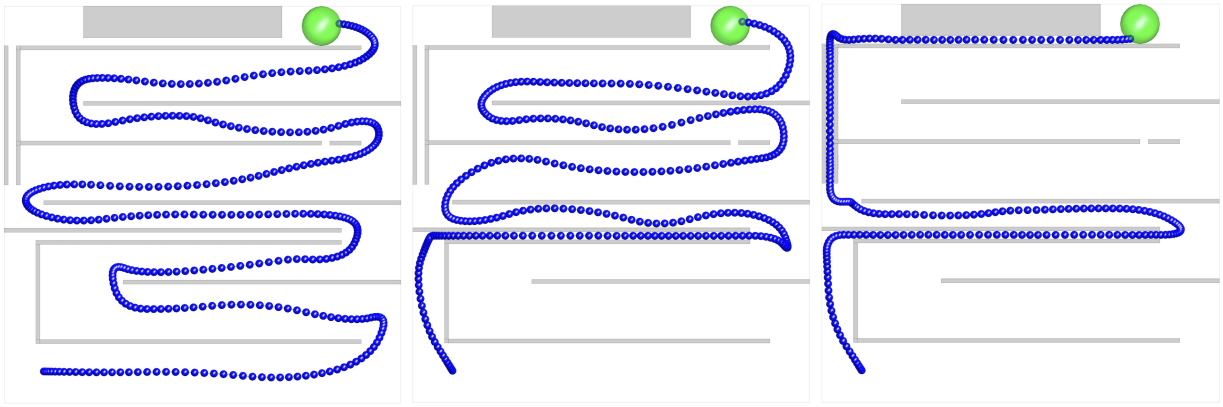


Figure 3.8: Planning performance of *Kino-PAX* -small-δ*, *Kino-PAX* -large-δ*, and *Kino-PAX* on the 12D nonlinear quadcopter system in the corridors environment over a 300 s planning duration.

Lastly, Table 3.3 and Figure 3.8 present the planning results for the 12D nonlinear quadcopter system. Both tuning variants of *Kino-PAX** consistently outperform SST across all evaluated metrics, including time to first solution, initial solution cost, and final solution cost. In these experiments, SST successfully found solutions in only 84%, 53%, 42%, and 0% of trials in environments a–d, respectively. In contrast, both *Kino-PAX* -large-δ* and *Kino-PAX* -small-δ* achieved a 100% success rate across all environments. On average, *Kino-PAX* -large-δ* finds the first solution 750× faster than SST and with a cost that is 63% of SST's first solution cost. It

converges to a final solution that is 55% of SST's first solution cost, while *Kino-PAX* -small-δ* converges to a final solution that is 51% of SST's. In comparison, SST converges to a final solution that is 94% of its own initial solution cost.

As evident from the results, *Kino-PAX** outperforms existing serial asymptotically near-optimal planners in terms of time to first solution, quality of the initial solution, time to convergence, and final solution quality, while remaining robust across a wide range of tuning configurations. *Kino-PAX* -large-δ* demonstrates that it can be tuned to rapidly find high-quality initial solutions, achieving speeds thousands of times faster than existing serial algorithms. Meanwhile, *Kino-PAX* -small-δ* shows that the algorithm remains effective and reliable for extremely challenging problems that were previously infeasible for traditional SBMPs. Finally, our experiments across diverse dynamical systems indicate that *Kino-PAX** maintains strong performance despite changes in state space dimension and dynamic complexity.



(a) First Solution (b) Intermediate Solution (c) Final Solution

Figure 3.9: Three solutions found by *Kino-PAX** using 6D Double Integrator dynamics and minimizing path length cost. Subfigure (a) is the first solution found, subfigure (b) is an intermediate solution, subfigure (c) is the final solution found.

3.7 Conclusion

We have introduced a novel motion planning algorithm for the asymptotically near-optimal kinodynamic motion planning problem. Our design enables significant parallelization of the planning process and leverages recent advancements in many-core computing architectures. Benchmark results show that initial solution times are, on average, three orders of magnitude faster than those of existing asymptotically near-optimal kinodynamic SBMPs, with significantly lower initial solution costs. We also demonstrate that our algorithm scales effectively across diverse dynamical systems and varying tuning parameters. For future work, we plan to conduct more sophisticated simulation studies, followed by deployment in a real-world robotic system.

Chapter 4

GPU Implementation

This chapter details the GPU implementation of both *Kino-PAX* and *Kino-PAX**, expanding on key algorithmic components not covered in Chapters 2 and 3, which are critical for efficient execution on many-core devices. Our discussion focuses primarily on the implementation of *Kino-PAX** for clarity and conciseness; however, much of the implementation applies equally to *Kino-PAX*.

We begin by introducing a naive many-core SBMP to highlight common pitfalls in inefficient GPU implementations. Next, we describe the data structure format and shared subroutines used in both *Kino-PAX* and *Kino-PAX**. Finally, we present a detailed walkthrough of the *Kino-PAX** implementation, which can be readily adapted to *Kino-PAX*.

4.1 Naive Implementation

We first describe a naive GPU implementation of an RRT-like planner to highlight common inefficiencies. In this approach, similar to *Kino-PAX**, we expand a set of tree nodes V_A in parallel and integrate newly generated nodes V_U into the tree. In this implementation, the tree $\mathcal{T}$, obstacles O, and node expansion set V_A reside in CPU memory. For each iteration, the CPU identifies promising nodes within $\mathcal{T}$ and populates V_A. Subsequently, V_A and O are transferred to the GPU, which propagates these nodes in parallel and performs collision checks on resulting trajectory segments. Newly generated nodes V_U are then transferred back to the CPU and integrated into $\mathcal{T}$. A visualization of this data transfer process is provided in Fig. 4.1.

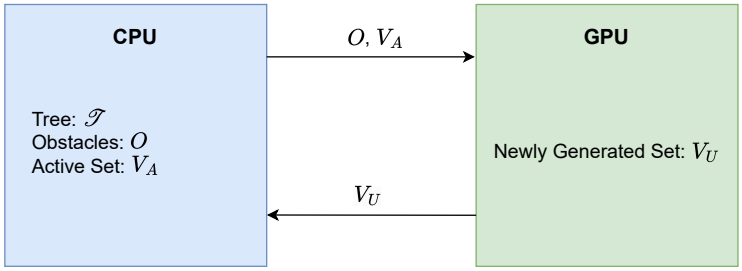


Figure 4.1: Naive GPU implementation of an RRT-like planner.

This design results in poor performance, primarily due to frequent, large-volume data transfers between the CPU and GPU. Such CPU–GPU communication is a high-latency operation [96] that significantly degrades runtime. Minimizing this communication is therefore essential for efficient GPU utilization.

To address this issue, our implementation of *Kino-PAX** adopts a design where all key data structures reside solely on the GPU, significantly reducing data transfer overhead. As illustrated in Fig. 4.2, the planning tree and node sets are constructed and managed directly on the GPU. The CPU's involvement is limited to GPU memory initialization and orchestrating GPU kernel launches.

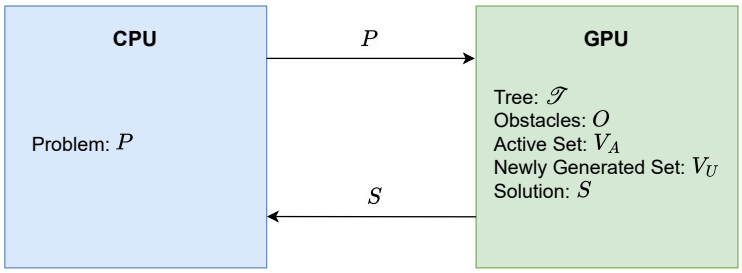


Figure 4.2: Improved GPU design used by *Kino-PAX**, storing all key data structures directly on the GPU.

This revised approach greatly improves efficiency compared to the naive implementation and serves as the basis for *Kino-PAX**. However, maintaining algorithm correctness becomes more challenging, as GPU data structures require unconventional layouts and careful memory

management. The subsequent section describes our GPU data structure representations and details common routines, such as graph searches, performed directly on these data structures.

4.2 Data Structure Representation

Unlike traditional SBMPs, which represent the planning tree using node objects, *Kino-PAX** represents $\mathcal{T}$ as a one-dimensional floating-point array. Prior to planning, the user specifies the expected tree size t_e, determining the GPU memory allocation. Specifically, $\mathcal{T}$ is structured as a floating-point array of size:

$$\mathcal{T}_{\text{size}} = t_e \times (s_{\text{dim}} + c_{\text{dim}} + 2),$$

where s_{dim} denotes the state dimension, c_{dim} the control dimension, and the additional 2 represents the control duration and parent node index. For instance, a node in $\mathcal{T}$ for a 6D Double Integrator is structured as follows:

Figure 4.3: Example node structure in $\mathcal{T}$ for a 6D Double Integrator system.

This representation significantly enhances efficiency for parallel computation. Threads independently read states from assigned indices of interest in $\mathcal{T}$, propagate trajectory segments, and safely write results back to predetermined locations. A key challenge arises in efficiently identifying these indices of interest, essentially a parallelized graph search over active nodes.

To facilitate parallel graph searches, node sets in *Kino-PAX** are represented as boolean mask arrays on the GPU. These arrays have length t_e, where a value of '1' indicates the corresponding node is included in the set. For example, with $t_e = 5$ and nodes 0 and 3 active in V_A, the representation is as follows:

1	0	0	1	0

Figure 4.4: Boolean mask representation for active nodes in V_A.

To efficiently perform a graph search on $\mathcal{T}$ and extract active indices (e.g., nodes 0 and 3 in the example), we utilize two GPU-optimized operations. First, an `exclusiveScan` operation computes the cumulative sum of the V_A mask, storing the results in a separate array, S. Our implementation employs optimized parallel operations provided by CUDA's Thrust library [96]. The result of `exclusiveScan` for the example is:

0	1	1	1	2

Figure 4.5: Resulting cumulative sum array S after applying `exclusiveScan` to V_A.

Next, arrays S and V_A serve as inputs to a second parallel GPU routine, `GraphSearch`, which compacts active indices from V_A into a new integer array, I, also of size t_e. The resulting array I for our example and pseudocode for `GraphSearch` are provided below:

0	3	0	0	0

Figure 4.6: Resulting array I containing active indices after executing `GraphSearch`.

Algorithm 9: `GraphSearch`

Input : V_A, S, I

Output: Updated array I

```
1 tid ← unique thread index;
2 if tid ≥ |V_A| or ¬V_A[tid] then
3 |   return;
4 I[S[tid]] ← tid;
```

The `GraphSearch` subroutine launches t_e threads, each assigned to a unique index in V_A. Each thread determines its unique ID, exiting immediately if its corresponding node is inactive (Algorithm 9, Lines 1–3). Active threads compute their target position in I using $S[tid]$. Since S remains constant until encountering subsequent active nodes, each thread safely writes its unique identifier to a distinct position in I (Algorithm 9, Line 4). After all threads complete execution, array I compactly represents active node indices, and the final value in array S indicates the total number of active nodes. This compact representation provides essential data for initiating the subsequent parallel propagation phase.

4.3 GPU Implementation of *Kino-PAX**

Given our design choices, data structure layout, and maintenance methods, we now detail the workflow of *Kino-PAX**. Assume an expansion set V_A has undergone the parallel `exclusiveScan` and `GraphSearch` operations to populate arrays S and I. To initiate the `Propagate` subroutine Algorithm 6, only a single integer—the count of active nodes in V_A obtained from the final index of S—must be transferred from the GPU to the CPU. Using this integer, the CPU orchestrates the GPU's `Propagate` kernel by launching $|V_A| \cdot \lambda$ threads. This ensures the number of threads never exceeds t_e, thus preventing memory out-of-bounds errors.

We now describe the parallel `Propagate` routine in detail for *Kino-PAX** (Algorithm 10). This GPU kernel takes as inputs $\mathcal{T}$, I, O, $\mathcal{R}$, λ, V_U, and an empty tree structure $\mathcal{U}$ (with identical size to $\mathcal{T}$).

Each thread first calculates its unique identifier (tid), ranging from 0 to $|V_A| \cdot \lambda - 1$. Using this identifier, it obtains its assigned index from I by evaluating $I[\lfloor tid/\lambda \rfloor]$. This indexing scheme ensures exactly λ threads access each active node, while no threads exceed the compact set I. Threads then read their assigned state data x from the array $\mathcal{T}$ using index arithmetic involving the system dimension (e.g., for the 6D double integrator from Fig. 4.3, DIM = 11) (Algorithm 10, Lines 1–3).

After obtaining the node x, each thread randomly generates controls and durations, prop-

agating x to a new state x' (Algorithm 10, Lines 4–5). Threads then verify trajectory validity, determine the corresponding region $\mathcal{R}_i$ for state x', and calculate the cost to reach x' (Lines 6–8).

To prevent race conditions, threads use atomic operations to update the minimum cost of $\mathcal{R}_i$ (Line 9). This atomic operation ensures that if multiple threads attempt to write to the memory location of $\mathcal{R}_i$ simultaneously there is an ordering constraint, preventing data races. If a thread's node x' represents the minimal cost within its region, the thread writes x' into the empty tree structure $\mathcal{U}$ at index $tid \cdot \text{DIM}$, guaranteeing unique write locations for each thread. The thread then marks the corresponding position in the boolean mask array V_U to indicate the addition of a promising node (Lines 10–12). Lastly if a thread's node x' is in the goal region and has a lower cost than the best-known goal satisfying trajectory $\mathbf{x}$ it atomically updates the best-known solution cost (Lines 13–14).

The rationale for using $\mathcal{U}$ rather than directly writing to $\mathcal{T}$ is threefold: (1) Direct writing to $\mathcal{T}$ requires knowledge of empty indices of $\mathcal{T}$ to avoid overwriting critical data. (2) Writing directly to a partially populated $\mathcal{T}$ would produce sparse, inefficient memory usage. (3) Direct writes limit parallelism since at most $t_e - |\mathcal{T}|$ threads could safely run without overwriting. Using a separate, initially empty structure $\mathcal{U}$ allows launching all t_e threads in every iteration, with a subsequent compact transfer of new nodes to $\mathcal{T}$.

The complete pseudocode for the parallel propagation routine in *Kino-PAX** is provided below:

Algorithm 10: `Kino-PAX* Parallel Propagate`

Input : $\mathcal{T}, I, O, \mathcal{R}, \lambda, \mathcal{U}, V_U$

Output: Updated arrays $\mathcal{U}$ and V_U

$tid \leftarrow$ unique thread index;

$treeIdx \leftarrow I[\lfloor tid/\lambda \rfloor]$;

$x \leftarrow \mathcal{T}[treeIdx \cdot DIM : (treeIdx + 1) \cdot DIM]$;

Randomly sample control u and duration dt;

$x' \leftarrow$ `PropagateODE`(x, u, dt);

if trajectory from x **to** x' **is valid then**

  Map x' to region $\mathcal{R}_i$;

  $cost(x') \leftarrow cost(x) +$ `getCost`(x, x');

  $cost(\mathcal{R}_i) \leftarrow$ `atomicMin`$(cost(\mathcal{R}_i), cost(x'))$;

  if $cost(x') = cost(\mathcal{R}_i)$ **then**

    $\mathcal{U}[tid \cdot DIM : (tid + 1) \cdot DIM] \leftarrow x'$;

    $V_U[tid] \leftarrow 1$;

    if $x' \in X_{goal}$ **then**

      $cost(\mathbf{x}) \leftarrow$ `atomicMin`$(cost(\mathbf{x}'), cost(x'))$;

After completing the `Propagate` routine, the unexplored tree structure $\mathcal{U}$ and boolean mask array V_U are populated, and the minimum costs for visited regions in $\mathcal{R}$ are updated. With these updated region costs, we proceed to prune nodes from the tree $\mathcal{T}$. The GPU implementation of node pruning closely follows the serialized pseudocode (Algorithm 7); no atomic operations are necessary since each GPU thread independently inspects a single node at index $tid \cdot DIM$ and updates the corresponding boolean value in $V_A[tid]$ to indicate whether the node remains active or is pruned.

However, a key implementation detail involves filtering nodes added to $\mathcal{U}$ that no longer represent the lowest-cost node within their respective regions—this situation arises when one thread initially adds a node as the region's lowest-cost node, only for a subsequent thread (within the same subprocess) to identify a node with lower cost in the same region. To resolve this, we perform an

additional GPU subprocess using our graph search routine (Algorithm 9) to update array I with active indices from V_U. We then launch $|V_U|$ threads to reevaluate and filter out these higher-cost nodes, ensuring that only the lowest-cost nodes remain marked as active in V_U.

Next, we detail the GPU implementation of the `UpdateTree` subroutine. This step efficiently transfers new nodes from the temporary tree $\mathcal{U}$ to the main tree $\mathcal{T}$ and identifies if an improved solution trajectory has been found. First, we perform another graph search on $\mathcal{U}$ over the boolean set V_U, identifying active indices. Then, we launch a GPU thread for each node in $\mathcal{U}$ and transfer it to a compact position in $\mathcal{T}$ at index $|\mathcal{T}| \cdot DIM + tid \cdot DIM$. This ensures efficient, concurrent, and safe writing without overlapping indices. Lastly, threads check if their newly transferred node is within the goal region and has cost equal to that updated in Algorithm 10. If so, the thread updates a solution integer variable with the tree index of this node, thus efficiently maintaining the optimal solution trajectory.

We then repeat the three core subprocesses—`Node Propagate`, `Node Pruning`, and `Tree Update`. Critically, we designed this cycle to minimize latency between subprocesses by reducing both the frequency and volume of data transfers between the CPU and GPU. Figure 4.7 illustrates this repeated cycle and highlights the latency periods.

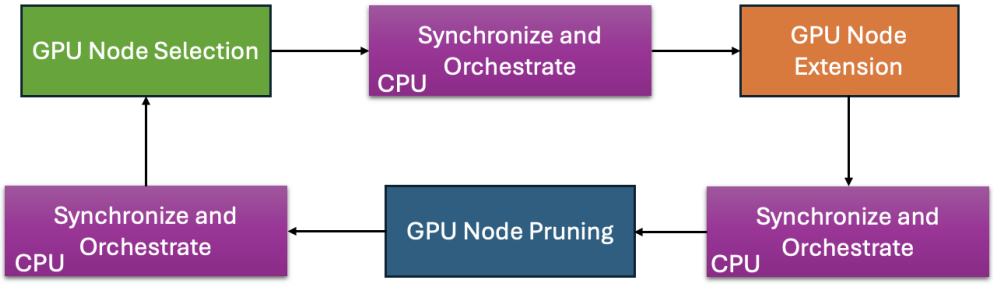


Figure 4.7: Planning cycle of *Kino-PAX**, consisting of repeated GPU subprocesses. A CPU-side synchronization occurs between each subprocess to coordinate data exchange and maintain execution order.

Chapter 5

Conclusions

5.1 Summary

In summary, we have presented two key contributions toward achieving real-time and robust motion planning for robotic systems operating in the real world. These approaches are applicable to both high-dimensional state spaces and systems with complex dynamics, while also providing formal guarantees of completeness and optimality.

This thesis begins in Chapter 1 by introducing the kinodynamic motion planning problem. We then provide an overview of existing serial and parallel algorithms developed for both CPU and GPU platforms to solve various classes of motion planning problems. From this analysis, we identify a critical gap in the literature: the lack of efficient many-core algorithms for kinodynamic sampling-based motion planning (SBMP).

In Chapter 2, we introduce *Kinodynamic Parallel Accelerated eXpansion* (*Kino-PAX*), a novel many-core kinodynamic SBMP designed to efficiently leverage modern computing hardware. We describe the algorithm in detail and provide a proof of probabilistic completeness. *Kino-PAX* is shown to solve challenging kinodynamic motion planning problems in under 10 ms—an order-of-magnitude improvement over parallelized versions of existing serial SBMPs. It is also demonstrated to be robust across a range of dynamical systems and tuning hyperparameters.

Chapter 3 presents an extension of *Kino-PAX*: *Asymptotically Near Optimal Kinodynamic Parallel Accelerated eXpansion* (*Kino-PAX**). This algorithm is designed to rapidly compute initial solutions in milliseconds and then refine them toward near-optimal trajectories over longer planning

durations. In this chapter, we provide a detailed description of *Kino-PAX**, present two theoretical guarantees—probabilistic completeness and asymptotic near-optimality—and report extensive benchmarking results. *Kino-PAX** is shown to compute initial solutions thousands of times faster than previous asymptotically near-optimal planners, with significantly lower initial costs. It also scales efficiently across a variety of dynamical systems and maintains strong performance across a wide range of tuning parameters.

Lastly, in Chapter 4, we provide insight into the implementation of both *Kino-PAX* and *Kino-PAX**, focusing on components that are critical for computational efficiency and correctness. We begin by presenting a naive GPU-based SBMP to highlight common pitfalls in designing algorithms for many-core architectures. We then describe the data structure format used by *Kino-PAX* and *Kino-PAX**, along with the common subroutines that operate on these structures. Finally, we provide a detailed discussion of the core computational procedures of *Kino-PAX** and explain how they are executed at the level of individual GPU threads.

5.2 Future Direction

The contributions of this thesis open several promising directions for future research. One such direction is extended simulation testing followed by real-world deployment. Verifying and integrating the proposed algorithms on a physical robotic system introduces numerous challenges, such as dealing with sensor uncertainty, hardware constraints, and ensuring system-level safety. These real-world experiments may also reveal new research opportunities in areas such as safety guarantees.

Another compelling direction is the extension of this work to multi-agent planning. Coordinating motion plans for multiple robots in real time poses significant computational challenges, especially in unstructured environments. The substantial computational speedups demonstrated in this thesis may enable scalable and efficient multi-agent kinodynamic planning, which has been traditionally limited by algorithmic complexity and runtime constraints.

Bibliography

[1] Steven Adams, Andrea Patane, Morteza Lahijanian, and Luca Laurenti. Bnn-dp: Robustness certification of bayesian neural networks via dynamic programming. In International Conference on Machine Learning (ICML), pages 133–151, Honolulu, HI, USA, July 2023. Association for Computing Machinery (ACM).

[2] Steven Adams Adams, Morteza Lahijanian, and Luca Laurenti. Formal control synthesis for stochastic neural network dynamic models. IEEE Control Systems Letters (L-CSS), 2022.

[3] Shaull Almagor, Justin Kottinger, and Morteza Lahijanian. Temporal segmentation in multi-agent path finding with applications to explainability. Artificial Intelligence, 330:104087, 2024.

[4] Shaull Almagor and Morteza Lahijanian. Explainable multi agent path finding. In Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '20, pages 34—-42, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems.

[5] Nancy M Amato and Lucia K Dale. Probabilistic roadmap methods are embarrassingly parallel. In Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C), volume 1, pages 688–694. IEEE, 1999.

[6] Peter Amorese and Morteza Lahijanian. Optimal cost-preference trade-off planning with multiple temporal tasks. In 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 2071–2077, Detroit, MI, USA, October 2023. IEEE.

[7] Sean B Andersson, Dimitris Hristu-Varsakelis, Morteza Lahijanian, et al. Observers in language-based control. Communications in Information & Systems, 8(2):85–106, 2008.

[8] Guillaume O Berger, Monal Narasimhamurthy, Kandai Watanabe, Morteza Lahijanian, and Sriram Sankaranarayanan. An algorithm for learning switched linear dynamics from data. In Advances in Neural Information Processing Systems (NeurIPS), pages 30419–30431, New Orleans, Louisiana, USA, Nov. 2022. Curran Associates, Inc.

[9] Amit Bhatia, Lydia E Kavraki, and Moshe Y Vardi. Sampling-based motion planning with temporal goals. In Int'l Conf. on Robotics and Automation, pages 2689–2696. IEEE, 2010.

[10] S. Caselli and M. Reggiani. Randomized motion planning on parallel and distributed architectures. In Euromicro Workshop on Parallel and Distributed Processing, pages 297–304, 1999.

[11] Nathalie Cauchi, Luca Laurenti, Morteza Lahijanian, Alessandro Abate, Marta Kwiatkowska, and Luca Cardelli. Efficiency through uncertainty: Scalable formal synthesis for stochastic hybrid systems. In Proceedings of the 2019 22nd ACM International Conference on Hybrid Systems: Computation and Control, Montreal, QC, Canada, Apr. 2019. ACM.

[12] Hamidreza Chitsaz and Steven M LaValle. Time-optimal paths for a dubins airplane. In 46th IEEE Conference on Decision and Control, pages 2379–2384. IEEE, 2007.

[13] W.L. Chow. Über systeme von linearen partiellen differentialgleichungen erster ordnung. Mathematische Annalen, 117:98–105, 1940/1941.

[14] Igor Cizelj, Xu Chu Dennis Ding, Morteza Lahijanian, Alessandro Pinto, and Calin Belta. Probabilistically safe vehicle control in a hostile environment. In IFAC Proceedings Volumes, volume 44, pages 11803–11808. Elsevier, 2011.

[15] Giannis Delimpaltadakis, Morteza Lahijanian, Manuel Mazo Jr., and Luca Laurenti. Interval markov decision processes with continuous action-spaces. In International Conference on Hybrid Systems: Computation and Control (HSCC), San Antonio, TX, USA, May 2023. ACM.

[16] Xu Chu Ding, Jing Wang, Morteza Lahijanian, Ioannis Ch. Paschalidis, and Calin Belta. Temporal logic motion control using actor-critic methods. In Proceedings of the IEEE International Conference on Robotics and Automation, pages 4687–4692, Saint Paul, MN, May 2012. IEEE.

[17] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. Journal of the ACM, 40(5):1048–1066, 1993.

[18] S. Edelkamp, M. Lahijanian, D. Magazzeni, and E. Plaku. Integrating temporal reasoning and sampling-based motion planning for multigoal problems with dynamics and time windows. IEEE Robotics and Automation Letters, 3(4):3473–3480, 2018.

[19] Francisco Eiras and Morteza Lahijanian. Towards provably correct driver assistance systems through stochastic cognitive modeling. In Robotics: Science and Systems Workshop on Robust Autonomy: Tools for Safety in Real-World Uncertain Environments, June 2019.

[20] Francisco Eiras, Morteza Lahijanian, and Marta Kwiatkowska. Correct-by-construction advanced driver assistance systems based on a cognitive architecture. In 2019 IEEE 2nd Connected and Automated Vehicles Symposium, pages 1–7. IEEE, 2019.

[21] Bernard Etkin and Lloyd Duff Reid. Dynamics of flight: stability and control. John Wiley & Sons, 1995.

[22] Aleksandra Faust, Oscar Ramirez, Marek Fiser, Kenneth Oslund, Anthony G. Francis, James Davidson, and Lydia Tapia. PRM-RL: long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. CoRR, abs/1710.03937, 2017.

[23] Eduardo Figueiredo, Andrea Patane, Morteza Lahijanian, and Luca Laurenti. Uncertainty propagation in stochastic systems via mixture models with error quantification. In IEEE Conference on Decision and Control (CDC). IEEE, Dec. 2024. (accepted).

[24] Anthony G. Francis, Aleksandra Faust, Hao-Tien Lewis Chiang, Jasmine Hsu, J. Chase Kew, Marek Fiser, and Tsang-Wei Edward Lee. Long-range indoor navigation with PRM-RL. CoRR, abs/1902.09458, 2019.

[25] Ibon Gracia, Dimitris Boskos, Luca Laurenti, and Morteza Lahijanian. Data-driven strategy synthesis for stochastic systems with unknown nonlinear disturbances. In Learning for Dynamics and Control Conference (L4DC), Proceedings of Machine Learning Research, Oxford, UK, July 2024. PMLR.

[26] Ernst Moritz Hahn, Vahid Hashemi, Holger Hermanns, Morteza Lahijanian, and Andrea Turrini. Multi-objective robust strategy synthesis for interval Markov decision processes. In International Conference on Quantitative Evaluation of Systems, pages 207–223, Berlin, Germany, Sep. 2017. Springer.

[27] Ernst Moritz Hahn, Vahid Hashemi, Holger Hermanns, Morteza Lahijanian, and Andrea Turrini. Interval Markov decision processes with multiple objectives: From robust strategies to pareto curves. ACM Transactions on Modeling and Computer Simulation, 29(4):1–31, 2019.

[28] Keliang He, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. Towards manipulation planning with temporal logic specifications. In Proceedings of the IEEE International Conference on Robotics and Automation, pages 346–352, Seattle, WA, May 2015. IEEE.

[29] Keliang He, Morteza Lahijanian, Lydia E Kavraki, and Moshe Y Vardi. Automated abstraction of manipulation domains for cost-based reactive synthesis. IEEE Robotics and Automation Letters, 4(2):285–292, 2018.

[30] Keling He, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. Reactive synthesis for finite tasks under resource constraints. In Int. Conf. on Intelligent Robots and Systems (IROS), pages 5326–5332, Vancouver, BC, Canada, Sep. 2017. IEEE.

[31] Qi Heng Ho, Tyler Becker, Benjamin Kraske, Zakariya Laouar, Martin S. Feather, Federico Rossi, Morteza Lahijanian, and Zachary N. Sunberg. Recursively-constrained partially observable markov decision processes. In Conference on Uncertainty in Artificial Intelligence (UAI), Proceedings of Machine Learning Research, Barcelona, Spain, July 2024. PMLR.

[32] Qi Heng Ho, Martin S. Feather, Federico Rossi, Zachary N. Sunberg, and Morteza Lahijanian. Sound heuristic search value iteration for undiscounted pomdps with reachability objectives. In Conference on Uncertainty in Artificial Intelligence (UAI), Proceedings of Machine Learning Research, Barcelona, Spain, July 2024. PMLR.

[33] Qi Heng Ho, Roland B. Ilyes, Zachary Sunberg, and Morteza Lahijanian. Automaton-guided control synthesis for signal temporal logic specifications. In IEEE Conference on Decision and Control (CDC), Cancun, Mexico, Dec. 2022. IEEE.

[34] Qi Heng Ho, Zachary Sunberg, and Morteza Lahijanian. Gaussian belief trees for chance constrained asymptotically optimal motion planning. In Int'l Conf. on Robotics and Automation, pages 11029–11035. IEEE, May 2022.

[35] Qi Heng Ho, Zachary N. Sunberg, and Morteza Lahijanian. Planning with SiMBA: Motion planning under uncertainty for temporal goals using simplified belief guides. In Int'l Conf. on Robotics and Automation, pages 5723–5729, London, UK, 2023. IEEE.

[36] Qi Heng Ho, Zachary N. Sunberg, and Morteza Lahijanian. Sampling-based reactive synthesis for nondeterministic hybrid systems. Robotics and Automation Letters (RA-L), December 2023. (accepted).

[37] David Hsu, J-C Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In Proceedings of international conference on robotics and automation, volume 3, pages 2719–2726. IEEE, 1997.

[38] Vu Anh Huynh, Sertac Karaman, and Emilio Frazzoli. An incremental sampling-based algorithm for stochastic optimal control. In Int'l Conf. on Robotics and Automation, pages 2865–2872. IEEE, 2012.

[39] cJeffrey Ichnowski and cRon Alterovitz. Parallel sampling-based motion planning with superlinear speedup. In Int. Conf. on Intelligent Robots and Systems, pages 1206–1212. IEEE, 2012.

[40] Jeffrey Ichnowski and Ron Alterovitz. Concurrent nearest-neighbor searching for parallel sampling-based motion planning in so (3), se (3), and euclidean spaces. In Workshop on the Algorithmic Foundations of Robotics, pages 69–85. Springer, 2020.

[41] Brian Ichter, Edward Schmerling, Ali-akbar Agha-mohammadi, and Marco Pavone. Real-time stochastic kinodynamic motion planning via multiobjective search on gpus. In Int'l Conf. on Robotics and Automation, pages 5019–5026. IEEE, 2017.

[42] Brian Ichter, Edward Schmerling, and Marco Pavone. Group marching tree: Sampling-based approximately optimal motion planning on gpus. In Int'l Conf. on Robotic Computing, pages 219–226. IEEE, 2017.

[43] Roland B. Ilyes, Qi Heng Ho, and Morteza Lahijanian. Stochastic robustness interval for motion planning with signal temporal logic. In International Conference on Robotics and Automation (ICRA), pages 5716–5722, London, England, UK, May 2023. IEEE.

[44] John Jackson, Luca Laurenti, Eric Frew, and Morteza Lahijanian. Safety verification of unknown dynamical systems via gaussian process regression. In 2020 IEEE 59th Conference on Decision and Control (CDC). IEEE, 2020.

[45] John Jackson, Luca Laurenti, Eric Frew, and Morteza Lahijanian. Towards data-driven verification of unknown dynamical systems. In Workshop on Robust Autonomy: Tools for Safety in Real-World Uncertain Environments, July 2020.

[46] John Jackson, Luca Laurenti, Eric Frew, and Morteza Lahijanian. Strategy synthesis for partially-known switched stochastic systems. In Conference on Hybrid Systems: Computation and Control (HSCC), pages 1–11, 2021.

[47] John Jackson, Luca Laurenti, Eric Frew, and Morteza Lahijanian. Synergistic offline-online control synthesis via local gaussian process regression. In IEEE Conference on Decision and Control (CDC). IEEE, Dec 2021.

[48] John Jackson, Luca Laurenti, Eric Frew, and Morteza Lahijanian. Towards safe, abstraction-based online learning and synthesis for unknown systems. In Workshop on Integrating Planning and Learning, July 2021.

[49] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. Journal of Robotics Research, 34(7):883–921, 2015.

[50] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE transactions on Robotics and Automation, 12(4):566–580, 1996.

[51] David B Kirk and W Hwu Wen-Mei. Programming massively parallel processors: a hands-on approach. Morgan kaufmann, 2016.

[52] Michal Kleinbort, Kiril Solovey, Zakary Littlefield, Kostas E Bekris, and Dan Halperin. Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation. IEEE Robotics and Automation Letters, 4(2):i–vii, 2018.

[53] Justin Kottinger, Shaull Almagor, and Morteza Lahijanian. MAPS-X: Explainable multi-robot motion planning via segmentation. In Int'l Conference on Robotics and Automation (ICRA), Xi'an, China, May 2021. IEEE.

[54] Justin Kottinger, Shaull Almagor, and Morteza Lahijanian. Conflict-based search for explainable multi-agent path finding. In Int'l Conference on Automated Planning and Scheduling (ICAPS), Singapore, June 2022. AAAI Press.

[55] Justin Kottinger, Shaull Almagor, and Morteza Lahijanian. Conflict-based search for multi-robot motion planning with kinodynamic constraints. In Int'l Conference on Intelligent Robots and Systems (IROS), Kyoto, Japan, Oct. 2022. IEEE.

[56] Justin Kottinger, Tzvika Geft, Shaull Almagor, Oren Salzman, and Morteza Lahijanian. Introducing delays in multi-agent path finding. In International Symposium on Combinatorial Search (SoCS), Alberta, Canada, June 2024. AAAI Press.

[57] Justin Kottinger, Shaull Shaull Almagor, and Morteza Lahijanian. Explainable multi-agent path planning. In Workshop on Explainable and Trustworthy Robot Decision Making for Scientific Data Collection, July 2020.

[58] Hadas Kress-Gazit, Morteza Lahijanian, and Vasumathi Raman. Synthesis for robots: Guarantees and feedback for robot behavior. Annual Review of Control, Robotics, and Autonomous Systems, 1:211–236, 2018.

[59] Andrew M Ladd and Lydia E Kavraki. Fast tree-based exploration of state space for robots with dynamics. Algorithmic Foundations of Robotics VI, pages 297–312, 2005.

[60] M. Lahijanian, S.B. Andersson, and C. Belta. A probabilistic approach for control of a stochastic system from LTL specifications. In Proceedings of the IEEE Conference on Decision and Control (CDC), pages 2236–2241. IEEE, 2009.

[61] M Lahijanian, SB Andersson, and C Belta. Control of Markov decision processes from PCTL specifications. In Proceedings of the 2011 American Control Conference, pages 311–316. IEEE, 2011.

[62] M. Lahijanian, J. Wasniewski, S.B. Andersson, and C. Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees,. In International Conference on Robotics and Automation, pages 1050–4729, Anchorage, Alaska, 2010. IEEE.

[63] Morteza Lahijanian, Shaull Almagor, Dror Fried, Lydia E. Kavraki, and Moshe Y. Vardi. This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, pages 3664–3671, Austin, TX, Jan. 2015. AAAI Press.

[64] Morteza Lahijanian, Sean B. Andersson, and Calin Belta. Approximate Markovian abstractions for linear stochastic systems. In Proceedings of the IEEE Conference on Decision and Control, pages 5966–5971. IEEE, Dec 2012.

[65] Morteza Lahijanian, Sean B. Andersson, and Calin Belta. Temporal logic motion planning and control with probabilistic satisfaction guarantees. IEEE Transactions on Robotics, 28(2):396–409, Apr. 2012.

[66] Morteza Lahijanian, Sean B. Andersson, and Calin Belta. Formal verification and synthesis for discrete-time stochastic systems. IEEE Transactions on Automatic Control, 60(8):2031–2045, Aug. 2015.

[67] Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. A sampling-based strategy planner for nondeterministic hybrid systems. In IEEE Conference on Robotics and Automation, pages 3005–3012, Hong Kong, China, May 2014. IEEE.

[68] Morteza Lahijanian, Marius Kloetzer, Sara Itani, Calin Belta, and Sean B Andersson. Automatic deployment of autonomous cars in a robotic urban-like environment (rule). In IEEE International Conference on Robotics and Automation, pages 2055–2060. IEEE, 2009.

[69] Morteza Lahijanian and Marta Kwiatkowska. Social trust: a major challenge for the future of autonomous systems. In AAAI Fall Symposium on Cross-Disciplinary Challenges for Autonomous Systems, AAAI Fall Symposium. AAAI Press, 2016.

[70] Morteza Lahijanian and Marta Kwiatkowska. Specification revision for Markov decision processes with optimal trade-off. In Proceedings of the IEEE 55th Conference on Decision and Control, pages 7411–7418. IEEE, Dec. 2016.

[71] Morteza Lahijanian, Matthew R. Maly, Dror Fried, Lydia E. Kavraki, Hadas Kress-Gazit, and Moshe Y. Vardi. Iterative temporal planning in uncertain environments with partial satisfaction guarantees. IEEE Transactions on Robotics, 32(3):538–599, May 2016.

[72] Morteza Lahijanian, Maria Svorenova, Akshay A Morye, Brian Yeomans, Dushyant Rao, Ingmar Posner, Paul Newman, Hadas Kress-Gazit, and Marta Kwiatkowska. Resource-performance tradeoff analysis for mobile robots. IEEE Robotics and Automation Letters, 3(3):1840–1847, 2018.

[73] J-P Laumond. Controllability of a multibody mobile robot. IEEE Trans. on Robotics and Automation, 9(6):755–763, 1993.

[74] Luca Laurenti, Morteza Lahijanian, Alessandro Abate, Luca Cardelli, and Marta Kwiatkowska. Formal and efficient synthesis for continuous-time linear stochastic hybrid processes. IEEE Transactions on Automatic Control, 66(1):17–32, 2020.

[75] Steven M LaValle. Planning algorithms. Cambridge university press, 2006.

[76] Steven M LaValle and James J Kuffner. Rapidly-exploring random trees: Progress and prospects. Alg. and comp. robotics, pages 303–307, 2001.

[77] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. Int. J. of robotics research, 20(5):378–400, 2001.

[78] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Asymptotically optimal sampling-based kinodynamic planning. The International Journal of Robotics Research, 35(5):528–564, 2016.

[79] Brandon Luders, Mangal Kothari, and Jonathan How. Chance constrained RRT for probabilistic robustness to environmental uncertainty. In AIAA Guidance, Navigation, and Control Conference, 2010.

[80] Ryan Luna, Morteza Lahijanian, Mark Moll, and Lydia E. Kavraki. Fast stochastic motion planning with optimality guarantees using local policy reconfiguration. In IEEE Conference on Robotics and Automation, pages 3013–3019, Hong Kong, China, May 2014. IEEE.

[81] Ryan Luna, Morteza Lahijanian, Mark Moll, and Lydia E. Kavraki. Optimal and efficient stochastic motion planning in partially-known environments. In Artificial Intelligence, pages 2549–2555, Quebec City, 2014. AAAI.

[82] Ryan Luna, Morteza Lahijanian, Mark Moll, and Lydia E. Kavraki. Asymptotically optimal stochastic motion planning with temporal goals. In Int'l Workshop on the Alg. Found. of Robotics, volume 107, pages 335–352. Springer, Aug. 2015.

[83] Matthew R. Maly, Morteza Lahijanian, Lydia E. Kavraki, Hadas Kress-Gazit, and Moshe Y. Vardi. Iterative temporal motion planning for hybrid systems in partially unknown environments. In Hybrid Systems: Computation and Control, pages 353–362, Philadelphia, PA, Apr. 2013. ACM.

[84] Frederik Baymler Mathiesen, Morteza Lahijanian, and Luca Laurenti. IntervalMDP.jl: Accelerated Value Iteration for Interval Markov Decision Processes. In IFAC Conference on Analysis and Design of Hybrid Systems (ADHS). IFAC, July 2024.

[85] Rayan Mazouz, Karan Muvvala, Akash Ratheesh Babu, Luca Laurenti, and Morteza Lahijanian. Safety guarantees for neural network dynamic systems via stochastic barrier functions. In Advances in Neural Information Processing Systems (NeurIPS), volume 35, pages 9672–9686, New Orleans, Louisiana, USA, Nov. 2022. Curran Associates, Inc.

[86] Rayan Mazouz, John Skovbekk, Frederik Baymler Mathiesen, Eric Frew, Luca Laurenti, and Morteza Lahijanian. Data-driven permissible safe control with barrier certificates. In IEEE Conference on Decision and Control (CDC). IEEE, Dec. 2024. (accepted).

[87] Jay McMahon, Nisar Ahmed, Morteza Lahijanian, Peter Amorese, Taralicin Deka, Karan Muvvala, Kian Shakerin, Trevor Slack, and Shohei Wakayama. Reason-recourse software for science operations of autonomous robotic landers. In IEEE Aerospace Conference (AERO), pages 1–11, Big Sky, MT, USA, 2023. IEEE.

[88] Jay McMahon, Nisar Ahmed, Morteza Lahijanian, Peter Amorese, Taralicin Deka, Karan Muvvala, Trevor Slack, and Shohei Wakayama. Expert-informed autonomous science planning for in-situ observations and discoveries. In 2022 IEEE Aerospace Conference (AERO), pages 1–11. IEEE, March 2022.

[89] Duane Merrill, Michael Garland, and Andrew Grimshaw. Scalable gpu graph traversal. ACM Sigplan Notices, 47(8):117–128, 2012.

[90] R. Moazzez Estanjini, X.C. Ding, M. Lahijanian, J. Wang, C.A. Belta, and I.Ch. Paschalidis. Least squares temporal difference actor-critic methods with applications to robot motion control. In Proceedings of the IEEE Conference on Decision and Control, pages 704–709. IEEE, 2011.

[91] Karan Muvvala, Peter Amorese, and Morteza Lahijanian. Let's collaborate: Regret-based reactive synthesis for robotic manipulation. In 2022 IEEE Conference on Robotics and Automation (ICRA), pages 4340–4346. IEEE, May 2022.

[92] Karan Muvvala and Morteza Lahijanian. Reactive synthesis for human-aware robotic manipulation using regret games. In RSS Workshop on Robotics for People: Perspectives on Interaction, Learning and Safety, July 2021.

[93] Karan Muvvala and Morteza Lahijanian. Efficient symbolic approaches for quantitative reactive synthesis with finite tasks. In 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 8666–8672, Detroit, MI, USA, October 2023. IEEE.

[94] Karan Muvvala, Andrew Wells, Morteza Lahijanian, Lydia Kavraki, and Moshe Vardi. Stochastic games for interactive manipulation domains. In 2024 IEEE Conference on Robotics and Automation (ICRA), Yokohama, Japan, May 2024. IEEE.

[95] Islam Nazmy, Andrew Harris, Morteza Lahijanian, and Hanspeter Schaub. Shielded deep reinforcement learning for multi-sensor spacecraft imaging. In American Control Conference (ACC), Atlanta, GA, USA, Jun. 2022. IEEE.

[96] NVIDIA. CUDA C++ Programming Guide, 2024. Version 12.6.

[97] Michael Otte and Nikolaus Correll. C-forest: Parallel shortest path planning with superlinear speedup. IEEE Transactions on Robotics, 29(3):798–806, 2013.

[98] Èric Pairet, Juan David Hernández, Marc Carreras, Yvan Petillot, and Morteza Lahijanian. Online mapping and motion planning under uncertainty for probabilistically safe autonomous navigation. IEEE Transactions on Automation Science and Engineering, 2021.

[99] Èric Pairet, Juan David Hernández, Morteza Lahijanian, and Marc Carreras. Uncertainty-based online mapping and motion planning for marine robotics guidance. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2367–2374. IEEE, 2018.

[100] Jeff M Phillips, Nazareth Bedrossian, and Lydia E Kavraki. Guided expansive spaces trees: A search strategy for motion-and cost-constrained state spaces. In Int'l Conf. on Robotics and Automation, volume 4, pages 3968–3973. IEEE, 2004.

[101] Erion Plaku, Kostas E Bekris, Brian Y Chen, Andrew M Ladd, and Lydia E Kavraki. Sampling-based roadmap of trees for parallel motion planning. IEEE Transactions on Robotics, 21(4):597–608, 2005.

[102] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. Motion planning with dynamics by a synergistic combination of layers of planning. IEEE Transactions on Robotics, 26(3):469–482, 2010.

[103] Robert Reed, Luca Laurenti, and Morteza Lahijanian. Promises of deep kernel learning for control synthesis. IEEE Control Systems Letters, pages 1–1, December 2023.

[104] Robert Reed, Luca Laurenti, and Morteza Lahijanian. "error bounds for gaussian process regression under bounded support noise with applications to safety certification. In AAAI Conference on Artificial Intelligence (AAAI). AAAI, 2025. (submitted).

[105] Robert Reed, Hanspeter Schaub, and Morteza Lahijanian. Shielded deep reinforcement learning for complex spacecraft specifications. In American Control Conference (ACC), Toronto, Canada, July 2024. IEEE.

[106] John H Reif. Complexity of the mover's problem and generalizations. In Symp. on Foundations of Comp. Science, pages 421–427. IEEE, 1979.

[107] John Skovbekk, Luca Laurenti, Eric Frew, and Morteza Lahijanian. Formal abstraction of general stochastic systems via noise partitioning. IEEE Control Systems Letters, pages 1–1, December 2023.

[108] Ioan A Sucan and Lydia E Kavraki. A sampling-based tree planner for systems with complex dynamics. IEEE Transactions on Robotics, 28(1):116–131, 2011.

[109] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. IEEE Robotics & Automation Magazine, 19(4):72–82, December 2012. `https://ompl.kavrakilab.org`.

[110] Wen Sun, Sachin Patil, and Ron Alterovitz. High-frequency replanning under uncertainty using parallel sampling-based motion planning. IEEE Transactions on Robotics, 31(1):104–116, 2015.

[111] Anne Theurkauf, Nisar Ahmed, and Morteza Lahijanian. Pareto optimal strategies for event triggered estimation. In IEEE Conference on Decision and Control (CDC), Cancun, Mexico, Dec. 2022. IEEE.

[112] Anne Theurkauf, Qi Heng Ho, Roland Ilyes, Nisar Ahmed, and Morteza Lahijanian. Chance-constrained motion planning with event-triggered estimation. In International Conference on Robotics and Automation (ICRA), pages 7944–7950, London, England, UK, May 2023. IEEE.

[113] Anne Theurkauf, Justin Kottinger, Nisar Ahmed, and Morteza Lahijanian. Chance-constrained multi-robot motion planning under gaussian uncertainties. IEEE Robotics and Automation Letters, 9(1):835–842, Dec. 2023.

[114] Wil Thomason, Zachary Kingston, and Lydia E Kavraki. Motions in microseconds via vectorized sampling-based planning. In Int'l Conf. on Robotics and Automation, pages 8749–8756. IEEE, 2024.

[115] Jing Wang, Xu Chu Ding, Morteza Lahijanian, Ioannis Ch. Paschalidis, and Calin Belta. Temporal logic motion control using actor-critic methods. International Journal of Robotics Research, 34(10):1329–1344, Aug. 2015.

[116] Kandai Watanabe, Georgios Fainekos, Bardh Hoxha, Morteza Lahijanian, Hideki Okamoto, and Sriram Sankaranarayanan. Optimal planning for timed partial order specifications. In 2024 IEEE Conference on Robotics and Automation (ICRA), Yokohama, Japan, May 2024. IEEE.

[117] Kandai Watanabe, Bardh Hoxha, Danil Prokhorov, Georgios Fainekos, Morteza Lahijanian, Sriram Sankaranarayanan, and Tomoya Yamaguchi. Timed partial order inference algorithm. In International Conference on Automated Planning and Scheduling (ICAPS), pages 639–647, Prague, Czech Republic, July 2023. AAAI.

[118] Kandai Watanabe, Nicholas Renninger, Sriram Sankaranarayanan, and Morteza Lahijanian. Probabilistic specification learning for planning with safety constraints. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, Sep 2021.

[119] Kandai Watanabe, Nicholas Renninger, Sriram Sankaranarayanan, and Morteza Lahijanian. Task learning with preferences for planning with safety constraints. In Workshop on Accessibility of Robot Programming and the Work of the Future, Sep 2021.

[120] Nathan A Wedge and Michael S Branicky. On heavy-tailed runtimes and restarts in rapidly-exploring random trees. In Artificial Intelligence, pages 127–133, 2008.

[121] Andrew M. Wells, Z. Kingston, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. Finite horizon synthesis for probabilistic manipulation domains. In International Conference on Robotics and Automation (ICRA), Xi'an, China, May 2021. IEEE.

[122] Andrew M Wells, Morteza Lahijanian, Lydia E Kavraki, and Moshe Y Vardi. Ltlf synthesis on probabilistic systems. In International Symposium on Games, Automata, Logics, and Formal Verification (GandALF), 2020.

[123] Min Wu, Tyron Louw, Morteza Lahijanian, Wenjie Ruan, Xiaowei Huang, Natasha Merat, and Marta Kwiatkowska. Gaze-based intention anticipation over driving manoeuvres in semi-autonomous vehicles. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, pages 6210–6216. IEEE, 2019.